Universidad Zaragoza

Trabajo Fin de Master

Reconocimiento de acciones deportivas en secuencias de vídeo mediante técnicas de aprendizaje automático

Recognition of sports actions in video sequences by means of machine learning techniques

Autor

Fernando Sierra Murillo

Director

Carlos Miguel Orrite Uruñuela

Escuela de Ingeniería y Arquitectura

2020

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe entregarse en la Secretaría de la EINA, dentro del plazo de depósito del TFG/TFM para su evaluación).

D./Dª. FERNANDO SIERRA MURILLO ,en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza, Declaro que el presente Trabajo de Fin de (Grado/Máster) Master en Ingeniería Electrónica (Título del Trabajo)

Reconocimiento de acciones deportivas en secuencias de vídeo mediante técnicas de aprendizaje automático

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 20 de Junio de 2020

Fdo: Fernando Sierra Murillo

Reconocimiento de acciones deportivas en secuencias de vídeo mediante técnicas de aprendizaje automático

RESUMEN

Se va a tratar de desarrollar una herramienta que sea capaz de reconocer acciones de tenis en secuencias de vídeo mediante técnicas de *machine learning*. Se va a crear una base de datos adecuada a las ontologías relativas a la actividad elegida (tenis), a partir de ella, se van a estudiar diferentes metodologías de tratamiento de imágenes con el fin de extraer características para, posteriormente, analizarlas mediante técnicas de *machine learning*. La validez de los resultados obtenidos junto al estado del arte nos permitirá llegar a unas conclusiones y nos descubrirá nuevos enfoques sobre el reconocimiento de acciones y personas.

Tabla de contenidos

Glosario

2D: 2 Dimensiones

3D: 3 Dimensiones

API: Application Programming Interface

BdD: Base de Datos

CNN: Convolutional Neural Networks

fps: frames per second

JSON: JavaScript Object Notation

LSTM: Long Short-Term Memory

MSE: Mean Squared Error

NTU RGB-D: Nanyang Technological University's Red Blue Green and Depth information

PCA: Principal Component Analysis

RAE: Real Academia Española

ReLU: Rectified Linear Unit

RNN: Recurrent Neural Networks

ROC: Receiver Operating Characteristic

SE: Special Euclidean group

SVM: Support Vector Machines

1. Introducción

En este capítulo se van a describir los objetivos y alcance del proyecto, el trabajo previo en el que se apoya, el contexto y la forma en la que se van a abordar los problemas que surgen a la hora de la realización del mismo.

1.1 Motivación y Contexto

El trabajo parte de la base del proyecto ASTRO: Análisis Semántico y Comprensión del Comportamiento del Jugador en Aplicaciones Deportivas [50], financiado por el Ministerio de Ciencia, Innovación y Universidades. En dicho proyecto se establecen una serie de retos como la detección automática de la postura del jugador basada en la imagen, utilizando para ello técnicas de *Deep Learning* que permitan establecer el modelo de articulaciones en 3D del jugador y el tipo de acción realizada mediante el análisis temporal de dichas imágenes.

1.2 Objetivos

El objetivo es desarrollar una herramienta que sea capaz de reconocer diferentes gestos llevados a cabo en una actividad deportiva, como es el tenis, mediante el análisis de secuencias de vídeo, así como la generación de una base de datos de ejemplo para futuras bases de datos más completas de vídeos de tenis utilizables por dicha herramienta.

Las cuestiones a abordar son:

-La generación de una base de datos de golpes en partidos de tenis, así como el estudio de la posibilidad de utilizar bases de datos disponibles de los repositorios de Internet con el fin de poder valorar de manera más completa los resultados

-La detección del sujeto y la segmentación de las distintas partes del cuerpo mediante herramientas basadas en visión por computador disponibles basadas en detección de esqueleto

-La normalización de los datos y la aplicación de técnicas de aprendizaje neuronal en secuencias temporales para poder reconocer que gesto se está realizando en función de las coordenadas de los puntos extraídos por las herramientas de detección.

1.3 Alcance

Para conseguir estos objetivos se va a generar una base de datos fundamentada en una serie de ontologías, de tal manera que queden definidos completamente los conceptos asociados a cada video. Se van a crear varios scripts en *Python* que se encargarán de procesar los datos obtenidos de las diferentes herramientas e implementar los diferentes algoritmos, y se obtendrán una serie de resultados que al analizarlos permitirán establecer unas conclusiones adecuadas al contexto del proyecto.

1.4 Contenidos

Siguiendo las fases definidas en el cronograma Anexo XI, el contenido de la memoria se estructura de la siguiente manera:

Cap.2 Revisión del estado del arte: Se abordarán de manera más exhaustiva los contenidos de la introducción, así como alternativas a los métodos y técnicas utilizados.

Cap.3 Creación de bases de datos deportivas: Se definirán los conceptos de ontologías y se aplicarán para obtener una base de datos utilizable por los métodos de análisis posteriores. Además, se seleccionarán otras bases de datos externas a las que aplicar dichas ontologías.

Cap. 4 Segmentación de esqueleto y normalización: A partir de diferentes técnicas se van a procesar las imágenes de las bases de datos para extraer puntos característicos que, una vez depurados, se utilizaran para el reconocimiento de acciones.

Cap. 5 Algoritmos de *Machine Learning*: Se implementarán varios algoritmos neuronales capaces de procesar las bases de datos para reconocer acciones o generar nuevos datos.

Cap. 6 Resultados: Una vez probados dichos algoritmos, se extraerán los resultados obtenidos de cada método y se establecerán las conclusiones.

2. Estado del Arte

A la hora de realizar una herramienta capaz de reconocer acciones en imágenes y secuencias de video, es necesario el seguimiento de una metodología específica que nos permita obtener unos resultados y un rendimiento óptimo. Es necesaria la creación de una base de datos completa y de contenido relevante al objetivo a alcanzar, que cuente con suficientes ejemplos representativos de las diferentes ontologías del campo a tratar. También es necesario elegir las técnicas de procesamiento de información basadas en algoritmos de aprendizaje automático entre el gran abanico de posibilidades de que disponemos y el soporte físico que las ejecutará. A continuación, se va a realizar una revisión del estado del arte de este procedimiento con el fin de tener una amplia perspectiva sobre el reconocimiento de acciones mediante técnicas de aprendizaje automático en imágenes.

2.1 Introducción

El reconocimiento en imágenes mediante técnicas de aprendizaje automático ha adquirido gran relevancia en los últimos años, siendo una de las causas principales la disponibilidad masiva de datos de entrada. Imágenes y secuencias de video de infinidad de situaciones proporcionan suficientes ejemplos tanto para cada aplicación específica como para la generación de diferentes modelos [1], [2] del contenido de la imagen. Otra de las causas es el crecimiento de la demanda de seguridad en el entorno personal o el bienestar, ganando protagonismo las aplicaciones capaces de reconocer patrones o comportamientos, lo que acaba por aumentar el desarrollo de los algoritmos de aprendizaje automático capaces de realizar dicho reconocimiento.

Para que un algoritmo de aprendizaje automático opere correctamente, es necesario reunir una serie de requisitos relacionados con el objetivo a cumplir y con el propio algoritmo. Se ha de disponer de una base de datos acorde al tamaño y complejidad del algoritmo utilizado, que cuente con los suficientes ejemplos como para entrenar y testear el algoritmo, y que a su vez tenga un reparto de ejemplos equitativo para todas las clases a reconocer.

Un segundo requisito es la obtención de características de dicha base de datos, un preprocesamiento necesario para facilitar el posterior análisis realizado por el algoritmo neuronal, el cual se fundamenta en la extracción de características o de un modelo en 2D o 3D del objeto a reconocer en la imagen.

Una vez entramos en la fase de desarrollo del algoritmo, las posibilidades de utilizar diferentes técnicas y sus variantes aumentan considerablemente [3]. Todas ellas se basan en los modelos fundamentales de aprendizaje automático y su estudio en estos últimos años ha dado como resultado un mejor conocimiento de su comportamiento en función de las necesidades de la aplicación, por lo que su elección será fundamental para la correcta consecución de los objetivos a alcanzar.

2.2 Bases de datos

Se trata del punto de partida y parte fundamental a la hora del desarrollo de todo el procedimiento. Aunque bien es cierto que Internet nos ha proporcionado acceso a ingentes cantidades de datos, tanto de imagen como de secuencias de video, contamos con desventajas asociadas como la necesidad de depuración de datos que no son representativos [4]. Para ello, existen soluciones como la utilización de algoritmos de reconocimiento de mayor nivel de abstracción [5] a partir de redes neuronales preentrenadas, las cuales realizan un filtrado "grueso" de un gran conjunto de datos.

Una vez encontrado un conjunto de datos representativo, se debe plantear una estructura jerárquica que defina las relaciones existentes entre los conceptos (sujetos o propiedades) del área del conocimiento a tratar, como indica la RAE en su definición de **ontología.** Se debe poder etiquetar la base de datos en función de dicha ontología si se desea realizar un entrenamiento supervisado. Este proceso dependerá en gran medida de la complejidad de la ontología, la cual puede componerse por subconjuntos anidados, lo que implica la necesidad de disponer de ejemplos etiquetados más específicos en la base de datos. Cuanta más compleja sea la ontología creada que se desee reconocer, mayor esfuerzo se deberá realizar en el etiquetado de las bases de datos, tanto si se realiza manualmente como si es a través de algoritmos automáticos. Bertini y Del Bimbo [6] plantean como ejemplo la obtención de ontologías automáticamente a partir de videos deportivos.

2.3 Preprocesamiento y Segmentación

El siguiente paso, antes de enviar los datos a los algoritmos neuronales, es la extracción de características con el fin de "retirar" la información innecesaria que pueda estorbar en su procesamiento posterior u obtener nuevas características que nos permitan analizar otros conjuntos de datos. Al tratarse de imágenes, la extracción directa permite únicamente obtener modelos en 2D mientras que, mediante técnicas más avanzadas, se pueden obtener a partir de dichos modelos 2D reconstrucciones más precisas en 3D [7], [8]. El uso de unos modelos u otros viene determinado por las necesidades de la aplicación. En el caso de analizar movimientos con un grado de libertad en cuanto a rotación, el uso de modelos 2D reduce el coste computacional al necesitar menos información por ser modelos más simples. En cambio, una aplicación en la que el sujeto pueda encontrarse en diferentes orientaciones, necesitará de un modelo 3D complejo que modele más fielmente el comportamiento.

Una de las técnicas de extracción más estudiadas es la segmentación del esqueleto [9]. Esta técnica consigue una gran reducción de dimensionalidad mediante la extracción de los puntos correspondientes a las articulaciones corporales de la imagen, de manera que se modela la estructura ósea. Esta técnica es aplicable tanto para su uso en modelos 2D como 3D. Se fundamenta en redes neuronales preentrenadas capaces de reconocer las diferentes articulaciones y partes del cuerpo humano y realizar un mapeado espacial en

2D de las mismas. Para la obtención de modelos 3D a partir del esqueleto existen diferentes técnicas, como la interpolación de modelos desde uno o varios puntos de vista [10] o la reconstrucción de la profundidad a partir de algoritmos de cálculo [11].

Existen también sensores y cámaras que permiten obtener información sobre la profundidad de la imagen y así conseguir modelos más precisos en 3D [12]. Estas características adicionales pueden constituir tanto modelos independientes, con los que realizar el reconocimiento, como complementos a la segmentación del esqueleto y generar así modelos más robustos y complejos [13].

Una vez extraídos los puntos característicos, es conveniente analizar si se requiere de un procesamiento posterior de dichos datos, ya que la utilización de modelos de características implica la generación de una nueva base de datos. Se trata de un paso indispensable ya que las imágenes pueden ser tomadas a diferentes distancias y contener mayor o menor resolución. Este proceso de normalización [14] y escalado es habitualmente realizado en base a métodos de análisis estadístico como el análisis cinemático [15] de los datos de los modelos o la transformación de los ejes de coordenadas [16].

2.4 Algoritmos neuronales

En este capítulo se van a analizar algunos de los problemas y técnicas utilizadas en el reconocimiento de acciones a partir de imágenes. No existen categorías definidas a la hora de etiquetar el comportamiento de los métodos, ya que prácticamente la totalidad de algoritmos se encuentran dentro del marco teórico de las redes neuronales y el aprendizaje automático. Únicamente se destacará el objetivo principal de cada algoritmo, o la solución que plantea al problema de reconocimiento abordado.

2.4.1 Datos no etiquetados

Ante la gran cantidad de datos necesarios para entrenar los algoritmos, es preciso realizar grandes esfuerzos para crear bases de datos completas y etiquetadas, por lo que una solución muy utilizada es la obtención previa de características de manera automática mediante técnicas de entrenamiento no supervisado [17]. Una práctica habitual en estos casos es el uso de *autoencoders* ya que realizan la extracción de características o patrones de manera autónoma. Estos patrones no son específicos y pueden compartirse entre clases, por lo que es necesario el uso complementario de algoritmos supervisados para la clasificación final. La ventaja reside en que este algoritmo clasificador necesita de menos entrenamiento y por tanto menos datos etiquetados ya que ha habido una reducción de dimensionalidad previa.

Otro de los retos abordados con respecto a las imágenes y datos de entrada es el marco temporal en el que se encuentran organizados. Se pueden realizar métodos dirigidos a detección en imágenes estáticas, como en [18], donde, además se trata el problema de la similitud de dos acciones distintas que comparten el mismo contexto. En este caso,

Xin propone el uso de una red auxiliar que detecte el contexto de la imagen, lo que permite distinguir entre varias clases que comparten ontologías en imágenes estáticas.

Por otro lado, existen algoritmos destinados al procesamiento de series de datos, como las redes *Long-Short-Term Memory* (LSTM) [19], en las cuales, las unidades de procesamiento reciben los datos almacenados de sus vecinas y de sus instantes anteriores, lo que le permite reconocer patrones temporales. Las *Convolutional Neural Networks* (CNN) [20] permiten reconocer patrones espaciales a partir de los filtros convolucionales que aplican en cada capa de procesamiento, o una combinación de ambas [21], donde se ha comprobado que la combinación de varias *CNN* y *LSTMs* permite una mejora sustancial en el reconocimiento de secuencias temporales.

Uno de los problemas relacionados con las series temporales es la decisión de cambio de clase en una misma trama de datos. Una solución planteada por Dawar [22] es el análisis de los cambios de energía potencial de las imágenes para detectar segmentos donde el sujeto se encuentra en pausa o en movimiento.

Dependiendo del tipo de información que proporcionan los datos de entrada (esqueleto, mapas de profundidad, puntos característicos…), se pueden aplicar diferentes métodos de reconocimiento, siendo posible usar cualquiera de los algoritmos actuales. Muchos de estos métodos están enfocados en el preprocesamiento de dichos datos y cuentan con varios pasos intermedios y transformaciones entre la imagen y la clasificación final, es por eso que dicho análisis puede quedar definido tanto como preprocesamiento como por algoritmo de reconocimiento.

2.4.2 Imágenes y mapas de profundidad

Estos algoritmos utilizan la imagen en crudo y/o mapas de profundidad [23] con una extracción simple de puntos característicos u otras técnicas de preprocesamiento. En [24] se analizan series temporales de mapas de profundidad para reconocer movimiento mediante el mapeo de 16 fotogramas superpuestos, y establece la clasificación mediante *SVM*.

2.4.3 Transformación de imágenes y mapas de profundidad

A partir de los datos anteriores, se pueden obtener otros nuevos aplicándoles transformaciones geométricas mediante el cálculo de los parámetros de la cámara, permitiendo así obtener información complementaria. En [25], se aplican 2 rotaciones en los ejes *y* y *z*, a los mapas de profundidad obtenidos de la cámara *Kinect* para tener así representadas 3 orientaciones (*top*, *side*, *front*). Cada orientación se procesa en una *CNN* y el resultado se fusiona obteniendo la clasificación final. Otra solución se aplica en [26] mediante *SVM* (*Support Vector Machine*) y *transfer learning* analizando diferentes puntos de vista del sujeto a reconocer.

2.4.4 Esqueletos 3D

Como se ha visto anteriormente, la segmentación de imágenes mediante la extracción de esqueleto proporciona grandes ventajas en el reconocimiento como la robustez de los datos o la versatilidad de la información. En [27] y [28] se presentan dos métodos en los que el esqueleto es dividido en 5 partes con una relación estructural entre sus componentes, se codifican las características extraídas de dichas partes y éstas alimentan a *RNNs* y *CNNs* respectivamente.

2.4.5 Transformación de esqueletos 3D

De manera similar a las imágenes y mapas de profundidad, los esqueletos, gracias a la información espacio-temporal que mantienen de la realidad, pueden ser transformados para obtener nuevas características que ayuden al proceso de reconocimiento.

En [29] se lleva a cabo una representación del esqueleto en la que se establecen relaciones geométricas entre varias partes del cuerpo sin necesidad de que estén conectadas (dar palmas es más intuitivo representarlo como la relación entre las dos manos). Para ello se define el *Special Euclidean group* (SE), modelado como las transformaciones geométricas necesarias (rotación, traslación) para llevar un punto del cuerpo al otro. Mediante *SVM*, esta nueva representación permite mejorar el reconocimiento del comportamiento.

2.4.6 Combinación de imágenes y esqueletos

Finalmente, es posible combinar la información obtenida por ambos medios, siempre que los datos correspondan a las mismas imágenes. De este modo, Tang [30] propone una estructura de datos basada en la composición del mapa de profundidad y el esqueleto extraído de una imagen para alimentar los algoritmos neuronales. Este enfoque está realizado en base a las nuevas tecnologías de captura de imagen, las cuales cada vez permiten extraer más información, ya que es necesario poder establecer relaciones entre los datos para obtener nuevas y mejoradas características.

3. Base de datos y Ontología

3.1 Bases de datos

Para nuestra herramienta se han utilizado varias bases de datos de personas disponibles en los repositorios de la web y también se ha creado una a partir de videos extraídos de la plataforma *Youtube*. La elección de la base de datos vendrá dada por varios parámetros a considerar:

-Tamaño: Es necesario que la base de datos cuente con un número suficiente de muestras, no suponiendo esto un problema cuando tratamos con archivos de video ya que estos cuentan con gran cantidad de imágenes.

-Variedad: Por contraposición, tener muchas imágenes invariantes en el tiempo como puede ser un video estático puede perjudicar en el proceso de análisis generando redundancia.

-Calidad de las muestras: Un video con poca resolución o con muchas oclusiones tanto ambientales como las generadas por la propia estructura humana pueden hacer que la obtención de características sea incompleta. En ese caso, el contar con gran variedad de posturas u orientaciones será necesario para compensar dichas características perdidas.

-Etiquetado: Para realizar el entrenamiento de la red neuronal es necesario proporcionarle el valor de salida al que corresponde cada entrada, y es este punto el que mayores esfuerzos requiere y el que diferencia una buena base de datos de otra.

Con estos parámetros, las bases estudiadas son las siguientes:

Nombre	Tipo	Nº Muestras	Formato	Etiquetada
Thetis	Videos de Tenis	196.048	2D	Si
NTU RGB-D	Videos de Acciones	3.325.519	3D	Si
Tenis	Videos de Tenis	57.056	2D	Si

Tabla 1:Bases de datos

Thetis [31] cuenta con 1.980 secuencias de gestos de tenis, realizados por diferentes personas, desde una posición frontal a la cámara. En total se computan 196.048 *fotogramas*.

En la Figura 1 podemos ver un fotograma de dicha BdD.

Figura 1: Fotograma de la BdD Thetis

La base de datos *Tenis*, está constituida por varios fragmentos de video en los que aparecen jugadores profesionales disputando un partido, en ellos, la cámara está situada en uno de los fondos de la pista. Se ha creado mediante el software de edición de video *Filmora9* [49] a partir de la extracción de aquellas secuencias en las que solo aparecían los dos jugadores disputando el punto. Cuenta con 57.056 fotogramas de video etiquetados en 672 secuencias de golpes de longitud variable. Su etiquetado es el mismo al utilizado en *Thetis.*

En la Figura 2 se muestra un fotograma extraído de dicha BdD.

Figura 2: Fotograma de la BdD Tenis

Adicionalmente, se van a testear los algoritmos de reconocimiento sobre la base de datos *NTU RGB+D*. Esta base de datos se compone de coordenadas *X*, *Y* y Z (3D) de las articulaciones obtenidas al extraer los esqueletos de una serie de videos de diferentes acciones cotidianas. Es una base de datos muy completa que cuenta con numerosas muestras muy variadas y con diferentes acciones. Sin embargo, no se adapta al dominio deportivo que buscamos, por tanto, se utilizará como validación adicional tanto de la fiabilidad de la red como del tratamiento de los datos. Se van a usar cuatro clases de

acciones: Dos de ellas muy similares (leer y escribir) y dos muy diferentes (saltar y atarse un zapato).

En la Figura 3 se muestra un fotograma de dicha BdD junto con la representación del esqueleto obtenido.

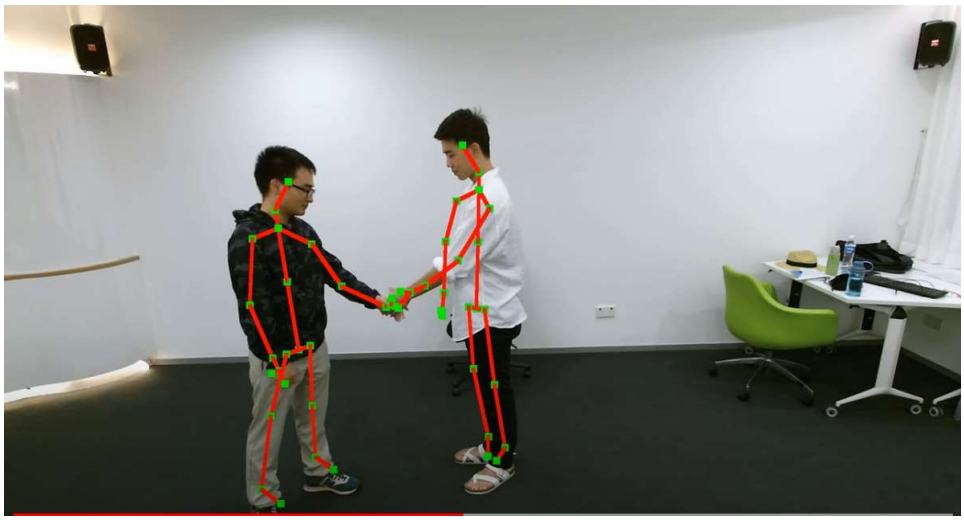


Figura 3: Fotograma y esqueleto de la BdD NTU RGB+D

A continuación, se muestra en la Tabla 2 la distribución de fotogramas de cada base de datos.

	Thetis	Tenis		NTU
0-Drive	35.497	18.838	0-Leer	84.728
1-Reves	36.932	18.310	1-Escribir	84.324
2-Servicio	39.559	8.358	2-Saltar	96.903
3-Volea	22.101	7.510	3-Calzarse	40.998
4-Remate	11.644	3.570		
5-Rechazo	0	470		

Tabla 2: Distribución de datasets

3.2 Ontología

Una vez analizados los *datasets* y dada su naturaleza como conjunto de datos, se va a estudiar el uso de ontologías que modelen el dominio del conocimiento concreto en el que nos encontramos, en este caso el tenis.

Como se expone en [32], la base para una ontología es la conceptualización junto con un vocabulario para referirse a las entidades de un dominio particular. Es decir, las ontologías para representar el conocimiento precisan de los componentes determinados en el *Anexo I-Ontologías*.

Una vez analizada la ontología del tenis, podemos utilizarla para desarrollar bases de datos más completas y etiquetarlas conforme a la estructura creada, o también podemos mejorar futuras versiones de la herramienta, teniendo en cuenta ya no un solo gesto, sino conectando y reconociendo secuencias de varios gestos.

4. Extracción de características y preprocesado

Una vez creada la base de datos, comienza el proceso de tratamiento de los datos y extracción de características. En este caso, para determinar qué acción se está llevando a cabo habrá que tener en cuenta dos variables fundamentales: El tiempo, y la postura corporal. De acuerdo a esto, se elige como característica principal a extraer el esqueleto del individuo que aparece en la imagen. Existen múltiples herramientas disponibles que realizan esta segmentación, como es el caso de la aplicación utilizada en nuestra herramienta: *OpenPose* [9], [33], [34], [35], [36]. Mediante una serie de redes neuronales profundas, *OpenPose* combina la detección de rostro, cuerpo y extremidades, tanto las visibles, como realiza una predicción de los defectos de imagen como las oclusiones. Todo ello en una aplicación gráfica con requerimientos de recursos hardware dedicados en su fase de inferencia. También permite elegir entre varias fuentes de imagen (imagen, video, webcams…) y ajustar la configuración de salida de la red.

La red cuenta con un tiempo de inferencia bajo para el hardware utilizado (*Nvidia* GTX 1070), con una frecuencia de unos 20 *fps* para los videos e imágenes de las bases de datos.

Una vez configurados los parámetros adecuados, *OpenPose* genera un archivo json en el directorio designado por cada fotograma, en el que se indican, de cada individuo detectado, las coordenadas *X* e *Y*, y el grado de confidencia de cada articulación (*Anexo II: Inferencia de OpenPose; Archivos JSON)*.

En la Figura 4 vemos como quedan representados los índices de cada una de las articulaciones, teniendo en cuenta que, aunque el esqueleto sea simétrico, se ha probado como *OpenPose* detecta la orientación del individuo (p. ej. el punto 5 corresponde siempre al hombro izquierdo).

4.1 Extracción de esqueletos

A la hora de aplicar *OpenPose* sobre las diferentes bases de datos se ha observado que, en el caso de *Tenis,* no es capaz de reconocer a ambos jugadores debido a la lejanía de uno de ellos. Por tanto, se ha limitado a una única detección de esqueleto por fotograma. Esto limita el aprendizaje de los algoritmos a jugadores de espaldas para esta base de datos.

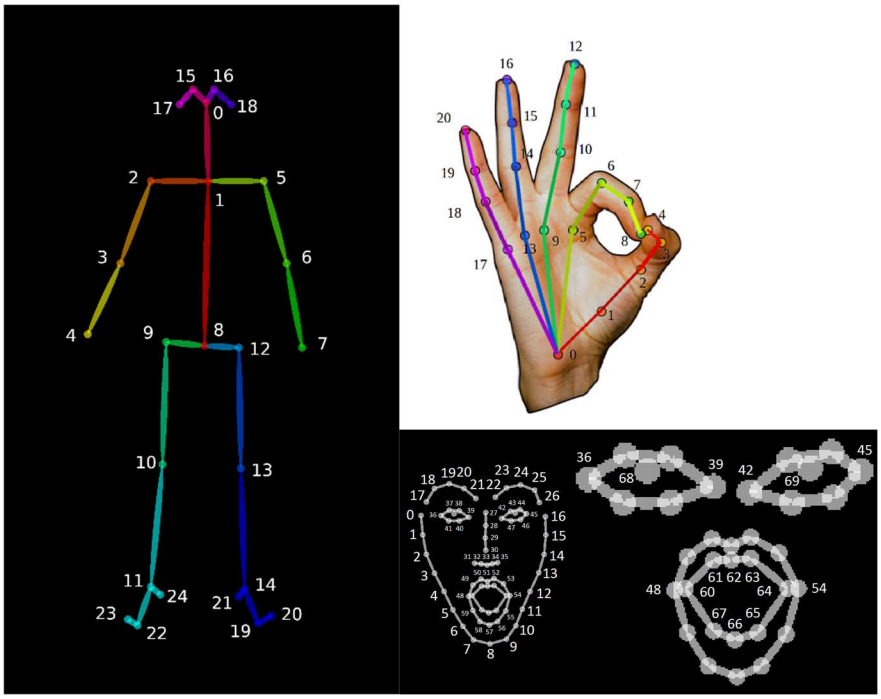


Figura 4: Representación de salida de OpenPose

Una vez obtenidos los esqueletos, se han de almacenar en estructuras de datos que sean interpretables por los algoritmos neuronales, para ello, se han desarrollado varios scripts en Python debido a que cada base de datos tiene un formato diferente:

-*Lee_videos_Tenis.py* (Anexo III)

-*Lee_videos_Thetis*.py (Anexo IV)

Ambas tienen una función común *Read_JSON ()* encargada de:

-Leer cada uno de los archivos *json* y almacenarlos en una variable del tipo *numpy* que será exportada al directorio de trabajo. Se guardan solamente los valores de coordenadas y se descartan los de confidencia.

-Desechar los fotogramas incompletos o que no dispongan de un número suficiente de articulaciones, contando el número de puntos detectados en cada esqueleto y rechazando aquellos que no superen un valor umbral.

-Normalizar los esqueletos tomando como distancia unitaria las articulaciones 0 y 8 y escalarlos tomando como referencia la articulación de origen 1.

-Generar los vectores de etiquetas en caso de haberlas.

Ambos *scripts* nos permiten procesar y representar los datos provenientes de las BdD 2D (*Tenis* y *Thetis*), así pues, serán necesarias otras funciones para visualizar correctamente los datos 3D (*NTU*).

El uso de esqueletos en 2D plantea limitaciones a la hora del reconocimiento de gestos o posturas corporales debido a la falta de información sobre la profundidad de la imagen. Por ello, se van a plantear varios métodos para la generación de modelos 3D a partir de los esqueletos en 2D, lo que nos permitirá analizar el impacto de dicha profundidad en los métodos de reconocimiento.

Otra de las ventajas que nos aporta la tercera dimensión, es la posibilidad de rotar los esqueletos en torno al eje vertical, lo cual permite aplicar una nueva normalización a los datos con el fin de mejorar su procesamiento.

4.2 Reconstrucción de datos

Uno de los recursos que proveen las redes neuronales es la generación de *autoencoders*, los cuales nos permiten, mediante una estructura simétrica, codificar la información de la BdD en una capa latente que almacena características obtenidas durante el entrenamiento. La finalidad de esta red es doble, pues al codificar la información conseguimos una reducción de dimensionalidad, por lo que se necesita menor espacio en disco. Por otra parte, al obtener en la salida el mismo dato de entrada, se puede utilizar como un método de reconstrucción en el caso hipotético de no disponer de la totalidad de los datos de entrada en la fase de inferencia o como generación de nuevos datos a partir de una semilla en la capa de entrada. Ésta primera característica es la más relevante en nuestra aplicación, ya que las imágenes de las cámaras convencionales solo proporcionan dos dimensiones, mientras que la tercera (la profundidad) es clave a la hora de reconocer posturas corporales.

Como vemos en la Figura 5, el *autoencoder* se conforma mediante dos redes independientes, pero invertidas. Por un lado, tenemos el *encoder,* encargado de codificar en una dimensión menor los datos de entrada y extracción de características, similar a la función de un *PCA* [42]. Por otro lado, tenemos la parte del *decoder*, encargado de devolver los datos a su dimensión original.

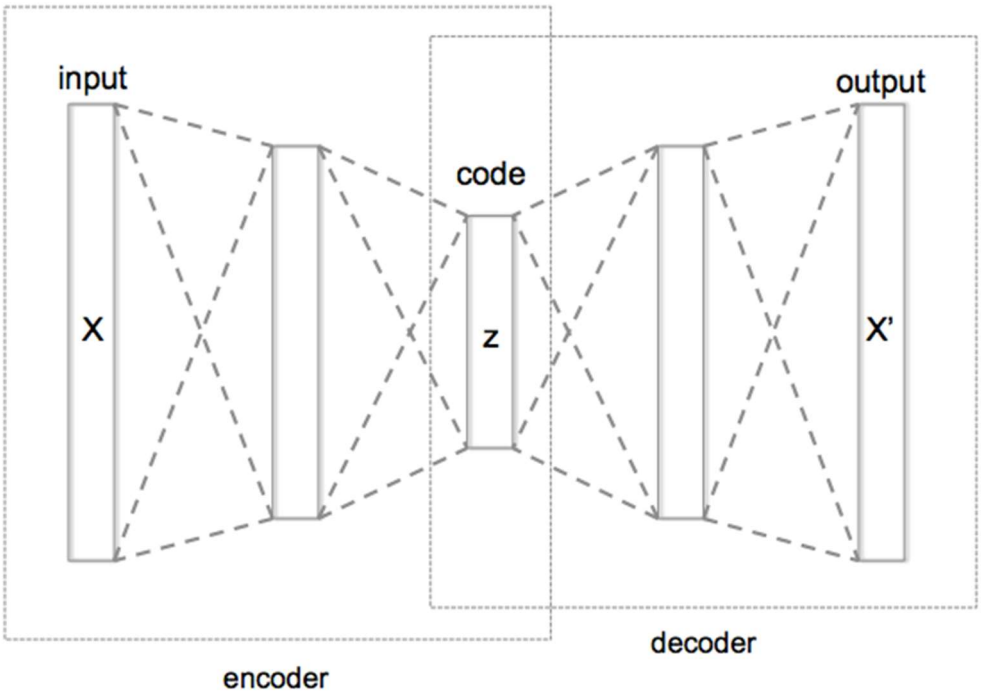


Figura 5: Estructura de un autoencoder

Así pues, en este capítulo se va a estudiar la implementación y utilización de algoritmos capaces de realizar dicha reconstrucción de la tercera dimensión de una imagen a partir de *autoencoders*. Además, se estudiarán otros algoritmos que complementen al *autoencoder* con redes recurrentes para, como se ha explicado anteriormente, tratar series temporales de datos (videos).

Otra de las ventajas del uso de estas redes es que no necesitan de bases de datos etiquetadas, por lo que su entrenamiento es no supervisado y se puede aumentar su tamaño con el fin de lograr un algoritmo robusto que recoja en su capa latente muchas características de las imágenes.

4.2.1 Autoencoder 2D

Para probar la validez del uso de autoencoders como método de reconstrucción, se va a implementar un autoencoder de varias capas que será alimentado por imágenes 2D.

Las BdD a utilizar serán: *NTU* para el entrenamiento y *Tenis* para el test. La elección de cada una de ellas viene dada en primer lugar, por la gran cantidad de esqueletos que proporciona *NTU*, y en segundo lugar, para probar la capacidad de adaptación y la robustez del autoencoder testeando sobre otra base de datos diferente.

La implementación de esta red se ha realizado en *Google Colab*, se encuentra en el *Anexo VII: Autoencoders.py* y su estructura queda definida en la Tabla 3.

Autoencoder 2D

Hiperparámetro	Valor
Capas ocultas	5
Dimensiones	50D-25D-15D-10D-15D-25D-50D
Regularización	None
Activación	relu
Error	*mean_squared_error*
Entrenamiento	*Adam*
Ratio de aprendizaje	0.01
Épocas	10
Tamaño de batch	10%
Otros	

*D=Dense

Tabla 3: Estructura del Autoencoder

Para su entrenamiento se ha descartado la tercera dimensión que proporciona la BdD *NTU,* y se ha testeado sobre la BdD *Tenis.* Se utilizará el *MSE* (*Mean Squared Error*) como parámetro de optimización, ya que en este caso el error vendrá dado por la diferencia de cada punto entre la entrada y la salida.

4.2.2 Autoencoder 3D

Una vez visto el carácter reconstructor de la señal de entrada que proporciona el *autoencoder* cuando ésta es completa (caso 2D), se va a probar su funcionamiento frente a datos incompletos que se desean generar. Así pues, con la misma estructura se entrenará sobre la BdD completa de *NTU* (3D) y se tratará de reconstruir *Tenis* con valores nulos en su tercera dimensión.

4.2.3 VideoPose3D

Desde la plataforma *Facebook Research*, se ha desarrollado *VideoPose3D* [43], una herramienta que permite reconstruir la tercera dimensión y la verdadera magnitud de secuencias de video. Gracias al uso de redes neuronales profundas, dicha herramienta es capaz de transformar secuencias de video (segmentadas mediante esqueletos 2D) en esqueletos 3D.

Para la extracción de esqueletos 2D se utiliza otra red neuronal desarrollada por el mismo grupo, *Detectron* [44]. Esta red utiliza las librerías *Caffe2* y *PyTorch.* Permite tanto la creación de un modelo de segmentación, como utilizar los disponibles para procesar las secuencias de video.

Para su implementación es necesario utilizar una distribución de Linux. En este caso, se han utilizado *Ubuntu* 16.04, y *Google Colab*.

Una vez instaladas todas las dependencias y librerías se ha probado la herramienta sobre la BdD *Thetis* y se han analizado las limitaciones y puntos fuertes.

Por un lado, ofrece una reconstrucción bastante fiable de la tercera dimensión (Figuras 6 y 7) y la *API* permite realizar un renderizado de la salida con la que comprobar la precisión y formato de salida.

Por otro lado, la herramienta está limitada a un solo actor por secuencia y su comportamiento es impreciso ante objetivos lejanos, por lo que será necesario adaptar las secuencias de entrada para maximizar su rendimiento. De este último punto nace la elección de la BdD *Thetis*. Otra solución sería el desarrollo de una herramienta de *tracking* de los distintos actores de la secuencia [45] [46] [47].

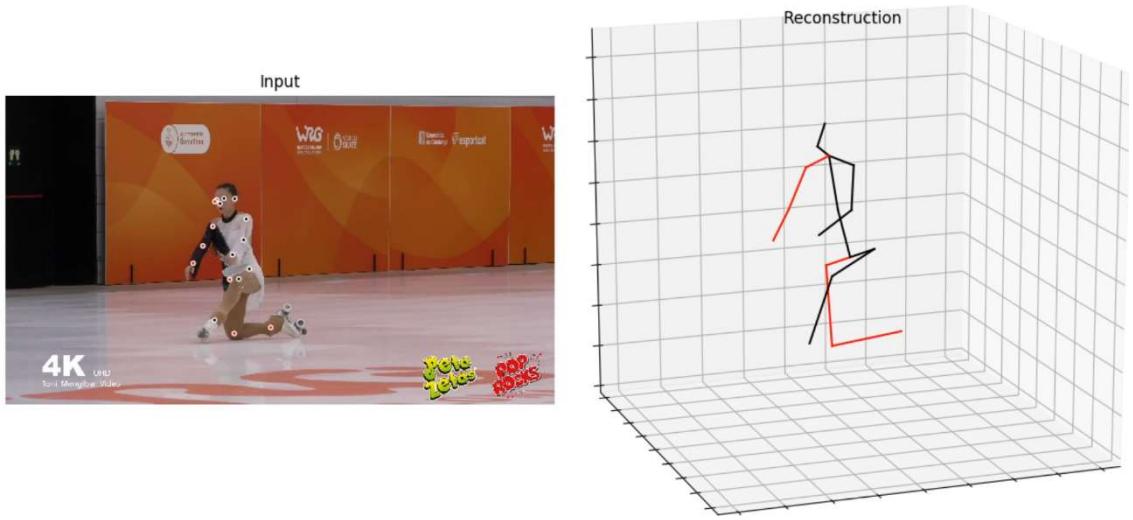


Figura 6: Reconstrucción VideoPose3D

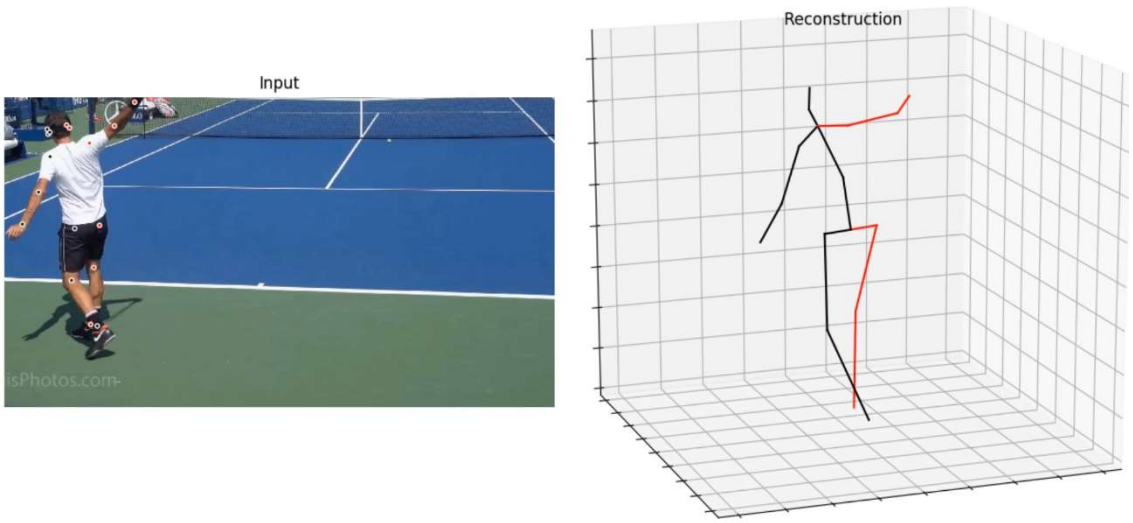


Figura 7: Reconstrucción VideoPose3D

5. Reconocimiento de acciones mediante Redes Neuronales

Una vez procesadas cada una de las bases de datos, se procede a testear algunos de los algoritmos neuronales ya existentes, combinándolos y modificando su estructura con el fin de explorar sus ventajas y desventajas, así como de obtener resultados óptimos para la herramienta de reconocimiento.

Se van a implementar redes con capacidades espacio-temporales para las bases de datos de secuencias de video, y posteriormente, técnicas de reconstrucción y generación de datos mediante algoritmos neuronales.

Estos algoritmos están implementados en *Python*, utilizando las librerías de *TensorFlow* [37] y *Keras* [38] sobre *scripts* ejecutados tanto de manera local como mediante recursos *cloud* como Google Colab [39]. Se conocen como redes recurrentes RNN (*Recurrent Neural Networks*) a aquellas que se caracterizan por almacenar un estado interno que les permite procesar secuencias temporales de entrada. Dentro de esta tipología de redes se encuentran las redes LSTM (*Long Short-Term Memory*) [40], las cuales son capaces de encontrar las dependencias temporales entre los diferentes fotogramas de una secuencia, además, se estructuran de manera que los vectores de entradas deben contener la secuencia temporal completa. En el *Anexo XII: Redes LSTM* se detalla su funcionamiento interno.

Uno de los pasos más importantes a la hora de alimentar las redes *LSTM* es generar el formato de entrada adecuado para el entrenamiento de la red, ya que las secuencias de las bases de datos no tienen la misma longitud y las dimensiones de entrada de la red deben ser constantes. Para ello, se han estudiado varias alternativas:

-Submuestreo de cada secuencia al valor de menor longitud del total, mediante la extracción de fotogramas intermedios (Figura 8).

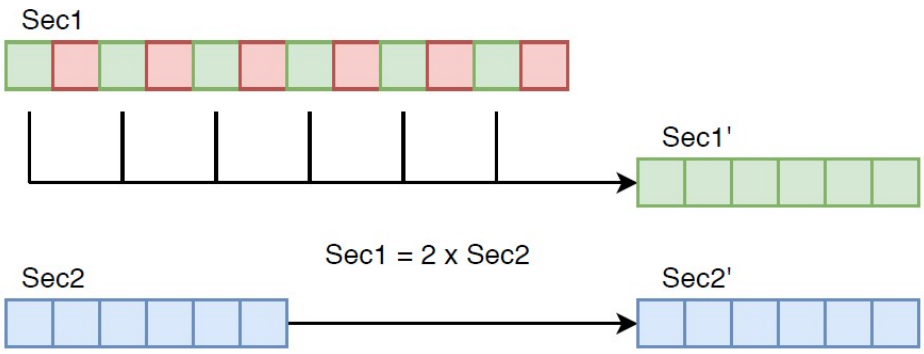


Figura 8: Procesado mediante submuestreo

-Ampliar cada secuencia al valor mayor de longitud del total, añadiendo ceros en los fotogramas excedentes (Figura 9).

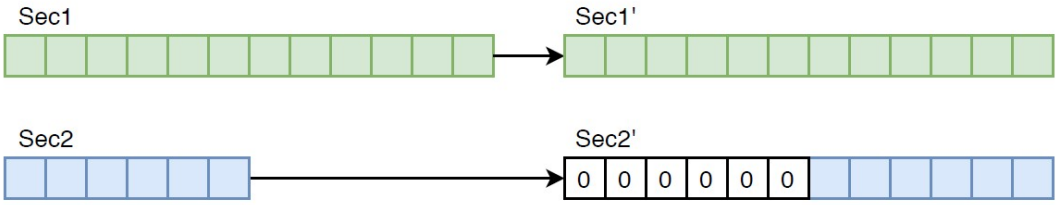


Figura 9: Normalizado de secuencias

-Generar una ventana deslizante que se desplace a lo largo de todas las secuencias solapándose mediante un valor de paso conocido (Figura 10).

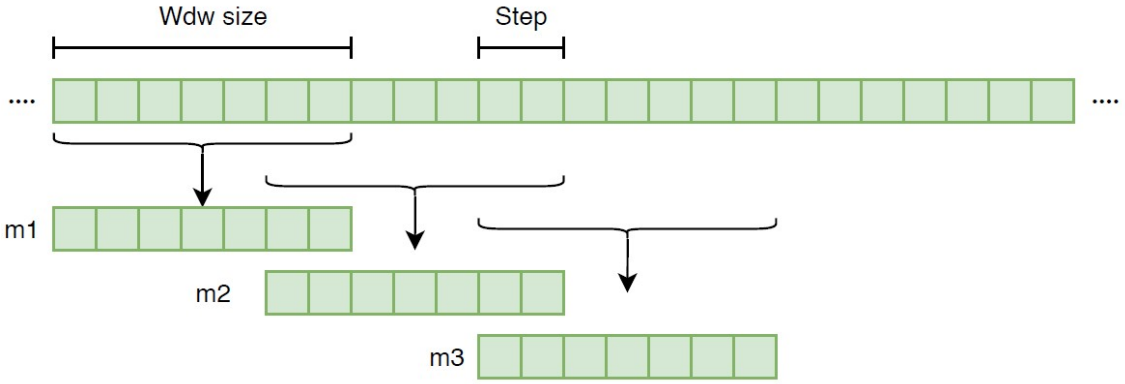


Figura 10: Procesado mediante ventana deslizante

La primera opción se va a aplicar en la BdD *Tenis* ya que tenemos información precisa del número de fotogramas por secuencia, mientras que de la de *Thetis* solo contamos con una aproximación (4s por secuencia). La segunda es la menos efectiva, ya que implica entrenar la red con valores nulos que distorsionan el proceso. La tercera se utilizará para la BdD *Thetis* tomando como tamaño de ventana 40 fotogramas (aproximadamente 1 segundo de video son 80 fotogramas) y un paso de 10 muestras.

Un valor de paso demasiado pequeño genera demasiada redundancia, mientras que uno muy grande limita el número de muestras de entrenamiento.

De esta manera, aunque no se conozca el tamaño de secuencia, el *LSTM* aprende fragmentos de cada uno de las secuencias y es capaz de reconocerlas.

Una vez generados los vectores de entrenamiento y test adaptados a las dimensiones del *LSTM* se van a probar diferentes estructuras.

5.1 LSTM

Esta red consta de varias capas *LSTM* y varias capas *Dense*. Las primeras analizan el comportamiento temporal de las muestras y las segundas realizan el reconocimiento en base a las anteriores. En la Tabla 4 se observa un esquema de su estructura.

LSTM

Hiperparámetro	Valor
Capas ocultas	2
Dimensiones	200L-100L-50D-4D*
Regularización	None
Activación	Tanh
Error	*categorical_crossentropy*
Entrenamiento	Adadelta
Ratio de aprendizaje	1.0
Épocas	30
Tamaño de batch	1%
Otros	Dropout=0.2, EarlyStopping=0.005

*L=LSTM, D=Dense

Tabla 4: Estructura LSTM

Se utilizará como red de control, a partir de la cual se buscarán distintas configuraciones (capas y/o parámetros) que mejoren el rendimiento de la misma.

5.2 LSTM + CNN

Esta red sustituye la capa de entrada del Modelo 1 por una capa convolucional que aplica un *kernel* al conjunto de datos de entrenamiento para extraer una serie de características que serán analizadas posteriormente por la capa *LSTM*.

En la Tabla 5 se indica su estructura.

LSTM+Conv1D

Hiperparámetro	Valor
Capas ocultas	2
Dimensiones	200C-100L-50D-4D*
Regularización	None
Activación	relu
Error	*categorical_crossentropy*
Entrenamiento	Adadelta
Ratio de aprendizaje	1.0
Épocas	30
Tamaño de batch	1%
Otros	EarlyStopping=0.005

*L=LSTM, D=Dense, C=Conv1D

Tabla 5: Modelo LSTM Convolucional

5.3 LSTM + CNN + MaxPooling

Intercalando capas *MaxPooling1D* entre los *kernels* convolucionales, como se realiza en [41], se puede reducir el tamaño de la dimensión temporal del conjunto de datos de entrada. Así pues, cada capa *MaxPooling* reduce a la mitad las muestras de la secuencia

de la capa anterior, por lo que realizaría un filtrado similar al submuestreo llevado a cabo en el proceso de tratamiento de los datos.

En la Tabla 6 se muestra su estructura.

LSTM+Conv1D+MaxPooling

Hiperparámetro	Valor
Capas ocultas	4
Dimensiones	25CMP-50CMP-100CMP-100LSTM-50D-4D
Regularización	None
Activación	relu
Error	*categorical_crossentropy*
Entrenamiento	Adadelta
Ratio de aprendizaje	1.0
Épocas	30
Tamaño de batch	1%
Otros	EarlyStopping=0.005

*L=LSTM, D=Dense, CMP=Conv-MaxPooling1D

Tabla 6: Modelo LSTM Convolucional con capas MaxPooling

Al utilizar esta capa podemos reducir considerablemente la cantidad de parámetros de la red, pero se puede perder el carácter temporal de las series de datos si reducimos demasiado dicha dimensión.

6. Resultados

Una vez definidos todos los métodos a utilizar, esta sección recoge los resultados obtenidos de las simulaciones realizadas, así como datos estadísticos que permitan analizar el comportamiento y rendimiento de los algoritmos descritos anteriormente.

El hardware utilizado para evaluar los algoritmos de forma local es un procesador Intel i7-8700K junto con un procesador gráfico *Nvidia* GTX1070, así pues, los tiempos de computo recogidos serán en función de esta configuración y pueden variar si se utiliza otra diferente.

Las pruebas realizadas parten de la estructura descrita en el capítulo anterior. Su configuración se describe en el *Anexo V: Librería Keras donde* se detallan los parámetros y argumentos de las diferentes capas y funciones provistas por las librerías de *Keras* y *TensorFlow*. La implementación se ha desarrollado el script *LSTM.py (Anexo VI: LSTM.py).*

Dado que la combinación de hiperparámetros es infinita, se han seleccionado aquellos que modifican la respuesta de la red en mayor medida, los cuales se describen a continuación.

-Tipo de Dato: Existen diferencias entre los datos en 3D y 2D, pero la ordenación de los puntos en formato vector no afecta al entrenamiento ya que cada coordenada se conecta a una neurona de entrada

-Formato de entrada: Ya se ha explicado cómo se pueden elegir varios formatos de datos de entrada (submuestreo y ventana deslizante). El uso de uno u otro afecta de manera significativa al resultado del entrenamiento, siendo favorable o desfavorable dependiendo de la BdD utilizada.

-Tamaño de ventana deslizante: Partiendo de la hipótesis de que un movimiento de tenis puede durar unos pocos segundos se utilizarán valores de 20 y 40 fotogramas para capturar en cada muestra al menos una parte completa del gesto.

-Tamaño de capas ocultas: El número de celdas depende de la complejidad de las características que se puedan extraer de cada secuencia. Se han probado valores de 50, 100 y 200 neuronas para dichas capas.

-Épocas: Para las BdD más grandes, el tiempo de entrenamiento es significativo, por lo que el efecto del *EarlyStopping* puede facilitar el proceso. En cambio, algunas pruebas pueden reducir su porcentaje de acierto si se detiene el entrenamiento de manera prematura.

Los siguientes hiperparámetros pueden ser modificados de igual forma, pero su impacto en la respuesta de la red es mínimo o simplemente afectan a otros procesos intrínsecos al entrenamiento que para nosotros resultan transparentes. Su elección no es al azar,

puesto que un hiperparámetro mal elegido puede invalidar todo el entrenamiento. Así pues, siguiendo la información del *Anexo V: Librería Keras*, tenemos:

-Funciones de activación: Cada tipo de capa neuronal tiene una activación dependiendo de su función global en la red. Para nuestro caso utilizaremos *ReLu* para las capas de obtención de características, *Sigmoid* para las capas densas que preceden a la salida, y *Softmax* para las capas de salida o de toma de decisión.

-Función de error (*loss)*: De los varios tipos de error que existen, nos centraremos en la entropía cruzada (*categorical_crossentropy*) para los entrenamientos con datos etiquetados y en el erro cuadrático (*MSE)* para los no etiquetados

-Optimizador: Se han utilizado *Adam* y *Adadelta.*

-Tamaño de *batch*: Se ha utilizado un 1% del tamaño del conjunto de entrenamiento.

6.1 Reconstrucción de datos

A la hora de evaluar una red encargada de reconstruir datos, a diferencia del clasificador, nos fijaremos en valores obtenidos del entrenamiento como el error entre la entrada y la salida, definido como parámetro a optimizar en dicho proceso.

También se puede ver si la reconstrucción es válida mediante la representación de los esqueletos obtenidos a la salida.

6.1.1 Reconstrucción 2D y Generación 3D

En cuanto al *autoencoder2D* encargado de la reconstrucción, debido al gran número de muestras con las que entrenar y al uso de las librerías de *TensorFlow* (ya obsoletas), se ha necesitado de aproximadamente 20 minutos de entrenamiento, siendo una red de unas pocas capas.

El *MSE* obtenido al final del entrenamiento ha sido de: **0.01047913**

Una vez representadas la entrada y la salida de la red se muestran a continuación en las Figuras 11 y 12 respectivamente

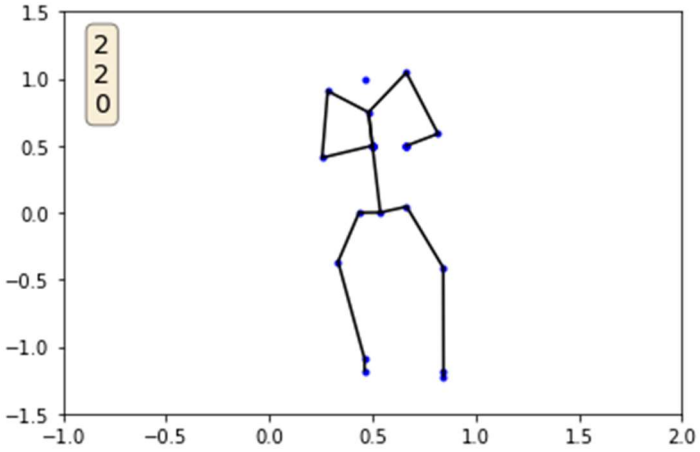


Figura 11: Esqueleto de Entrada del Autoencoder

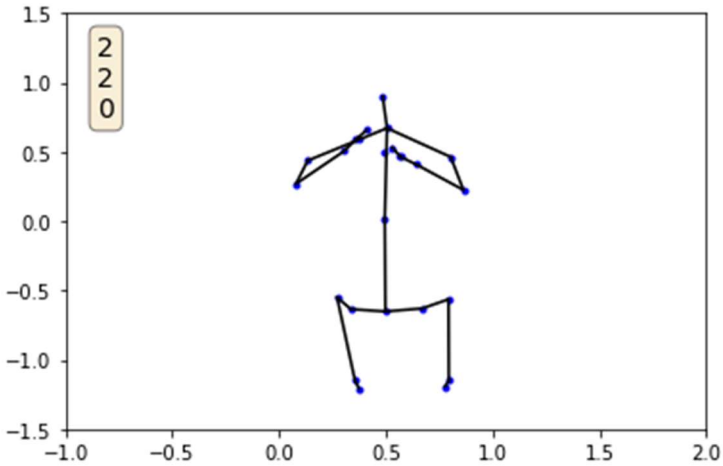


Figura 12: Esqueleto reconstruido en la salida del Autoencoder

Como se observa, se mantiene una cierta estructura coherente con el esqueleto humano, aunque la posición exacta no se corresponda. Esto se debe a que las posturas con las que se ha entrenado (NTU no contiene gestos específicos de tenis) no se asemejan a las que se dan en las acciones de tenis. Una posible solución sería la distribución de las diferentes partes del cuerpo en diferentes subredes, como ya se ha comentado en el estado del arte, para no depender de acciones completas.

De manera similar, se ha evaluado el autoencoder encargado de la generación de la tercera dimensión, obteniéndose en este caso un *MSE* de **0.021548256**, valor que duplica al obtenido en la reconstrucción 2D y que nos indica una peor reconstrucción.

En las Figuras 13 y 14 se observan la entrada y la salida de *autoencoder3D*, donde se aprecia, de la misma manera que en 2D, como la estructura de la salida cuenta con cierta coherencia estructural, pero vemos como la tercera dimensión queda completamente distorsionada y en general se aleja bastante de ser una reconstrucción fiel al esqueleto de entrada.

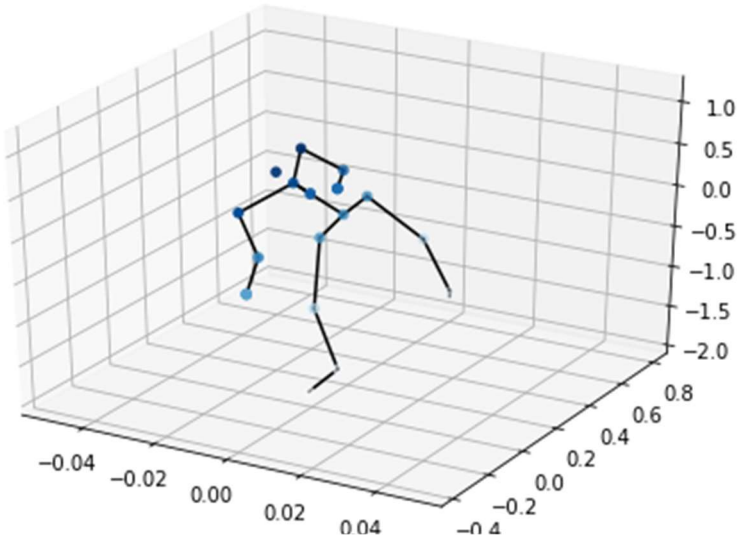


Figura 13: Esqueleto 2D de entrada del autoencoder 3D

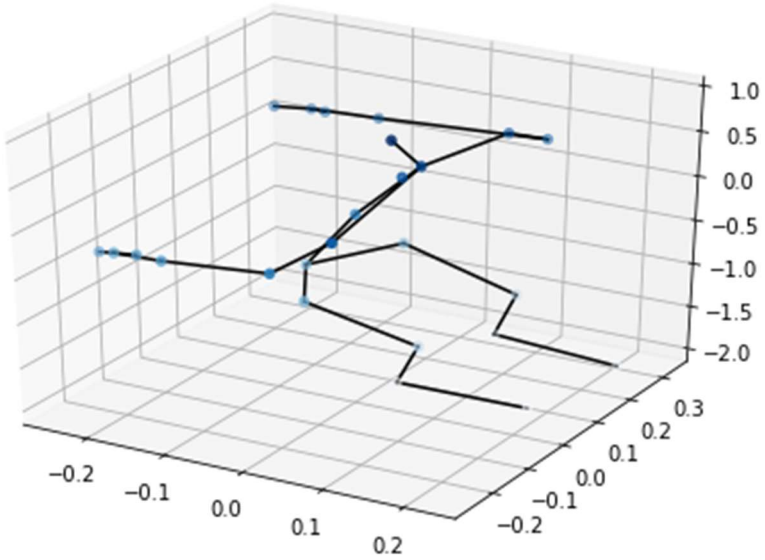


Figura 14: Reconstrucción de la tercera dimensión

6.1.2 VideoPose3D

Una vez instaladas las librerías necesarias, se seleccionan el conjunto de secuencias y se procesan mediante *Detectron*. De esta primera herramienta se obtiene un archivo *npz* que contiene los puntos 2D correspondientes al modelo que se haya pasado como parámetro. Una vez obtenido dicho archivo, *VideoPose3D* se encarga de generar a partir de esos datos la estimación de la tercera dimensión del esqueleto y generar un renderizado del conjunto de esqueletos de la secuencia.

El formato de salida de *VideoPose3D* agrupa todos los esqueletos en un vector de 3 dimensiones (muestra, articulación, coordenada) mientras que los algoritmos implementados trabajan con otro tipo de formato, por lo que se han de adaptar previamente. Los esqueletos se encuentran ya escalados en el rango [-1,1] y centrados,

como se puede ver en la Figura 15. Por otro lado, al haber una tercera dimensión, se puede rotar el esqueleto para poder entrenar con secuencias de jugadores en múltiples orientaciones.

Para aplicar la rotación se ha tomado como ángulo de referencia (θ) el formado por las articulaciones del hombro izquierdo y el derecho en el plano XZ con respecto al eje X. Una vez obtenido el ángulo se aplica la matriz de rotación (1) a cada uno de los puntos del esqueleto para hallar su transformación.

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \operatorname{sen}\theta \\ 0 & 1 & 0 \\ -\operatorname{sen}\theta & 0 & \cos\theta \end{bmatrix} \quad (1)$$

Estas adaptaciones y la generación de los vectores de la base de datos *Thetis3D* se encuentran en *Anexo X*, *Procesado 3D.*

Una vez generados los datos en 3D, se analizarán mediante los algoritmos de reconocimiento de la misma manera que los 2D y se compararán sus resultados.

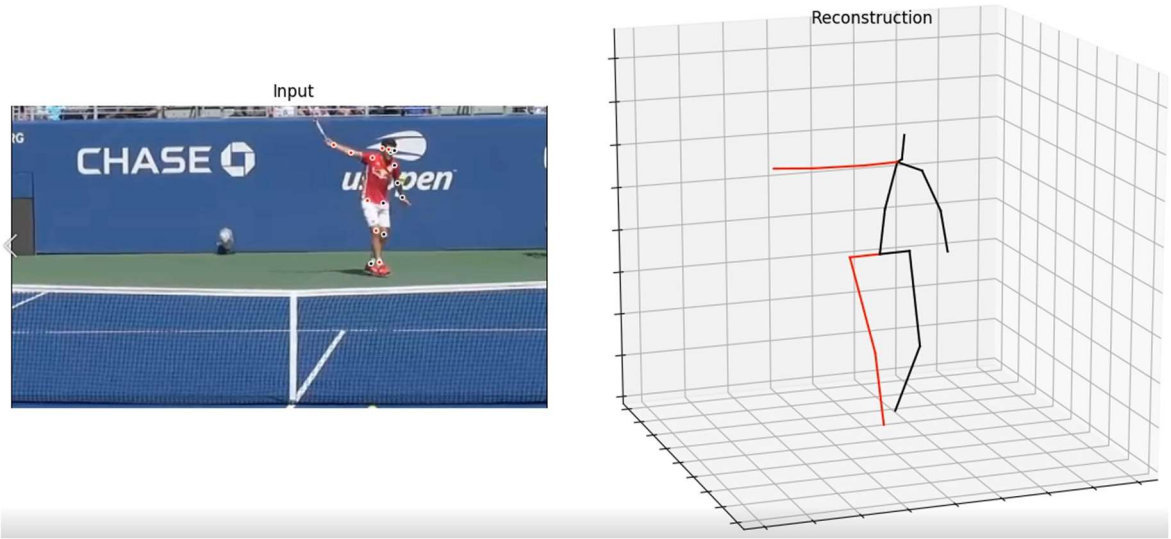


Figura 15: Renderizado de VideoPose3D

6.2 Reconocimiento de gestos

En el *Anexo VIII: Resultados LSTM* se encuentran los principales marcadores obtenidos del entrenamiento y validación de los algoritmos descritos en el capítulo anterior para cada base de datos, tanto en 2D como 3D y los diferentes formatos de entrada. De ella se extraen los mejores resultados de exactitud (*accuracy*) en la Tabla 7, y se evaluarán a continuación:

Red	Dim.	1.LSTM			2. LSTM+Conv1D			3. LSTM+Conv+MaxPool		
Formato		SW(40)	SW(20)	SF	SW(40)	SW(20)	SF	SW(40)	SW(20)	SF
NTU (%)	**3D**	**64.98**	**65.10**	**76.21**	**72.20**	**68.67**	**72.12**	**68.00**	**65.59**	**74.72**
Tenis(%)	**2D**	**67.13**	**61.15**	**57.35**	**60.49**	**58.71**	**63.24**	**60.84**	**52.76**	**69.12**
Thetis(%)	**2D**	**79.15**	**75.06**		**86,75**	**85.72**		**83.69**	**77.49**	
Thetis3D(%)	**3D**	**49.80**	**48.63**	**48.00**	**43.51**	**49.61**	**45.00**	**48.22**	**46.66**	**46.00**

Tabla 7: Accuracy del reconocimiento (SF = SUBMUESTREO, SW = SLIDING WINDOW)

Se observa como las redes más complejas funcionan de manera más eficiente con BdD más grandes, como ocurre con *NTU* y *Thetis* entre las redes 1 y 2, mientras que para BdD más pequeñas, el uso de redes más complejas no implica un mejor resultado.

También se puede ver como la red 3 en combinación con el formato de datos submuestreado genera muy buenos resultados, aunque como contrapartida, no se puede verificar con *Thetis*, al no disponer de la información necesaria para su generación.

La máxima puntuación la obtiene la red 2 con la BdD *Thetis*, con formato de entrada de ventana deslizante y con independencia del tamaño de ventana.

Finalmente, se han extraído ciertos datos estadísticos que nos ayudarán a analizar el rendimiento del clasificador con mayor *accuracy*. Para ello, se han utilizado las librerías que proporciona *Sklearn* [48], con las cuales se pueden obtener tanto valores promedios de las principales métricas (*accuracy*, *precision*, *recall*...) como representaciones gráficas de éstas o sus interacciones (*precision-recall*, *ROC*…). En el *Anexo IX: Datos Estadísticos* se encuentra el script utilizado para la obtención de dichos datos, así como las gráficas obtenidas para cada *dataset*.

6.2.1 NTU

En la figura 16 se muestra la matriz de confusión. En ella se observa la alta tasa de falsos negativos entre las dos primeras clases (leer y escribir), las cuales son muy parecidas entre sí, mientras que prácticamente la totalidad de las otras dos (saltar y atarse el zapato) son reconocidas. Otro valor que nos indica que no se distinguen bien estas dos últimas clases es el valor de la salida para ambas, manteniéndose siempre en un valor en torno a 0.5 en ambas neuronas de salida.

	Leer	Escribir	Saltar	Calzarse
Leer	0,656	0,281	0,0312	0,0312
Escribir	0,416	0,574	0,0099	0
Saltar	0,0282	0	0,901	0,0704
Calzarse	0,0308	0	0,0154	0,954

Figura 16: Matriz de confusión del Dataset NTU

6.2.2 Tenis

En la figura 17 se observa como para este *dataset* en el que sí se ha tenido en cuenta la clase de rechazo, la mayor tasa de verdaderos positivos la encontramos en el servicio por ser el gesto más diferenciado. En cambio, las clases de rechazo (*rejection*) y remate se ven distorsionadas debido a la poca cantidad de muestras, por lo que su entrenamiento no se ha realizado correctamente. Se observa además como tenemos altos valores de falsos positivos y verdaderos negativos entre las clases revés y volea.

	Drive	Revés	Servicio	Volea	Remate	Rejection
Drive	0,739	0,174	0	0,087	0	0
Revés	0,0526	0,632	0	0,211	0,105	0
Servicio	0	0,0769	0,846	0,0769	0	0
Volea	0	0,182	0	0	0,182	0
Remate	0	1	0	0	0	0
Rejection	0	0	1	0	0	0

Figura 17:Matriz de confusión Tenis Dataset

6.2.3 Thetis 2D

Como se muestra en la figura 18, en este *dataset* sí existe una distribución uniforme de todas las clases y no es necesaria la inclusión de la clase de rechazo. A diferencia de la BdD *Tenis*, en la que la clase remate no estaba presente, en este caso se aprecia como el clasificador confunde las clases servicio y remate en mayor medida que otras clases. La clase mejor clasificada es el revés.

	Drive	Revés	Servicio	Volea	Remate
Drive	0,81	0,076	0,014	0,095	0,0088
Revés	0,0025	0,99	0	0,01	0
Servicio	0,0034	0,0067	0,9	0	0,093
Volea	0,05	0,15	0,013	0,78	0,0033
Remate	0	0	0,16	0,0096	0,83

Figura 18: Matriz de Confusión Thetis Dataset 2D

6.2.4 Thetis 3D

El objetivo principal de añadir la tercera coordenada (Z) es poder proporcionar a la red más información sobre la postura del jugador y así mejorar el rendimiento del

reconocimiento. Para ello, se ha reconstruido la tercera dimensión de todo el conjunto de datos contenidos en *Thetis* mediante la herramienta VideoPose3D, obteniendo una nueva base de datos que llamaremos *Thetis3D.* Con el fin de poder utilizar el formato de submuestreo se han procesado cada una de las secuencias de *Thetis* individualmente, por lo que se ha podido extraer la información del tamaño de cada secuencia necesaria para dicho formato. A continuación, se ha comprobado que los esqueletos se han generado correctamente analizando si el valor de las coordenadas está dentro de los rangos esperados y si la representación gráfica del conjunto de articulaciones es consistente con las formas del esqueleto humano.

Una vez entrenados y testeados los algoritmos de reconocimiento, como muestra la figura 19, el *dataset Thetis3D* no mejora el reconocimiento si no que lo empeora considerablemente. Ya que la base de datos *NTU* es 3D y su reconocimiento es bueno podemos descartar que la causa provenga de la imposibilidad de añadir esta tercera dimensión, pudiendo focalizarla en la estructura de los datos del esqueleto a raíz de su procesado mediante *Detectron* y *VideoPose3D*.

	Drive	Revés	Servicio	Volea	Remate
Drive	0,407	0,260	0,112	0,179	0,039
Revés	0,305	0,453	0,043	0,148	0,049
Servicio	0,070	0,076	0,660	0,038	0,153
Volea	0,239	0,374	0,000	0,386	0,000
Remate	0,043	0,000	0,608	0,043	0,304

Figura 19: Matriz de Confusión Thetis 3D

De la matriz de confusión podemos extraer como el servicio y remate son confundidos en mayor medida, debido al parecido en su ejecución, siendo el servicio el mejor reconocido.

Podemos observar en el *Anexo VIII*: *Resultados LSTM,* como los valores de *accuracy* del entrenamiento son similares a los otros *datasets*, mientras que la fase de test se ve claramente deteriorada.

Ya que *Thetis* cuenta únicamente con jugadores colocados frente a la cámara, se ha probado a analizar los datos sin aplicar el normalizado en rotación. Se han obteniendo similares resultados lo que nos permite descartar errores en dicho procedimiento.

Se ha realizado un entrenamiento suprimiendo del vector de entrada la tercera dimensión reconstruida, de forma que se obtiene una clasificación 2D. Los resultados alcanzan valores del 45%, por lo que se puede afirmar que la reconstrucción no es la causa del mal rendimiento.

7. Conclusiones

En este trabajo se ha conseguido, aplicado los procedimientos y técnicas necesarios, generar una herramienta capaz de reconocer acciones de tenis en secuencias de video.

Inicialmente se ha estudiado el estado actual del reconocimiento de acciones en imágenes y vídeos. A continuación, se ha creado una base de datos de vídeos a partir de una ontología concreta sobre gestos de tenis que ha servido para extraer características de los participantes que aparecen en cada fotograma. También se han obtenido otras bases de datos de los repositorios *web* disponibles y se han adaptado a dicha ontología.

Se han probado y ajustado métodos de reconocimiento de acciones mediante algoritmos de *machine learning*, obteniendo en la mayor parte de los casos un buen rendimiento y así determinar que gesto está realizando el jugador.

Inicialmente, se plantearon las técnicas de reconstrucción de datos mediante autoencoders para obtener nuevas características a partir de las extraídas con el fin de mejorar el reconocimiento posterior. Sin embargo, se encontró la herramienta *VideoPose3D* que era capaz de realizar dicha tarea en secuencias de video, por lo que se utilizó esta última para generar los datos en 3D.

Tras las pruebas realizadas sobre los datos reconstruidos y analizados los resultados fallidos en el reconocimiento, se ha podido llegar a la conclusión de que, aunque los datos se han generado correctamente, se han perdido detalles de cada elemento individual. Como ya se observaba en la reconstrucción mediante autoencoders, *VideoPose3d* trata de reproducir un modelo único sobre el esqueleto de entrada, lo que produce un filtrado en los detalles más particulares de cada clase. Así pues, los esqueletos reconstruidos son tan similares entre sí que el clasificador no es capaz de distinguir las diferentes clases.

Esta es la razón principal por la cual las bases de datos extraídas directamente de las imágenes han logrado un mejor rendimiento que la reconstruida. Así pues, se ha descartado el uso de la técnica de reconstrucción y reconocimiento en la herramienta debido a su baja eficacia. En un futuro será necesaria la aplicación de otras técnicas de reconstrucción para poder realizar el reconocimiento correctamente.

Como líneas futuras de trabajo se ha comentado en este documento, por un lado, la posibilidad de implementar una herramienta de *tracking* para agilizar la obtención de secuencias de tenis, con el fin de crear BdD más completas. Por otro lado, la implementación de otras estructuras en los algoritmos neuronales que puedan mejorar la eficiencia de los mismos. La viabilidad de la reconstrucción 3D permitirá el uso de otras técnicas de procesado de datos para la extracción de características adicionales.

Finalmente, la implementación completa de la ontología creada sobre las bases de datos permitirá aumentar el nivel de reconocimiento a un plano superior, como la posibilidad de realizar un seguimiento de las acciones realizadas no solo en una secuencia, sino a lo

largo de un conjunto de varias de ellas, con el fin de determinar otros aspectos estadísticos de la actividad deportiva (preferencia de gestos de un jugador, porcentaje de puntos ganados o perdidos, …)

8. Bibliografía

[1] C. Zhu and S. Satoh, "Evaluation of visual object retrieval datasets," 2013 IEEE International Conference on Image Processing, Melbourne, VIC, 2013, pp. 3954-3958.

[2] Y. Yang and D. Ramanan, "Articulated pose estimation with flexible mixtures-of-parts," CVPR 2011, Colorado Springs, CO, USA, 2011, pp. 1385-1392.

[3] X. Li, S. Deng, S. Wang, Z. Lv and L. Wu, "Review of Small Data Learning Methods," 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, 2018, pp. 106-109.

[4] S. R. Kunze and S. Auer, "Dataset Retrieval," 2013 IEEE Seventh International Conference on Semantic Computing, Irvine, CA, 2013, pp. 1-8.

[5] I. V. Chugunkov, D. V. Kabak, V. N. Vyunnikov and R. E. Aslanov, "Creation of datasets from open sources," 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Moscow, 2018, pp. 295-297.

[6] M. Bertini, A. del Bimbo and C. Torniai, "Soccer Video Annotation Using Ontologies Extended with Visual Prototypes," 2007 International Workshop on Content-Based Multimedia Indexing, Bordeaux, 2007, pp. 212-218.

[7] T. S. Kumar and P. B. Rakesh, "3D Reconstruction of facial structures from 2D images for cosmetic surgery," 2011 International Conference on Recent Trends in Information Technology (ICRTIT), Chennai, Tamil Nadu, 2011, pp. 743-748.

[8] S. Chang, Y. Tsai, H. Hsu and K. Li, "3D Skeleton Construction by Multi-view 2D Images and 3D Model Segmentation," 2011 Fourth International Conference on Ubi-Media Computing, Sao Paulo, 2011, pp. 168-173.

[9] Z. Cao, T. Simon, S. Wei and Y. Sheikh, "Realtime Multi-person 2D Pose Estimation Using Part Affinity Fields," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 1302-1310.

[10] C. Wang, Y. Wang, Z. Lin, A. L. Yuille and W. Gao, "Robust Estimation of 3D Human Poses from a Single Image," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 2369-2376.

[11] F. Gong and C. Kang, "3D Mesh Skeleton Extraction Based on Feature Points," 2009 International Conference on Computer Engineering and Technology, Singapore, 2009, pp. 326-329.

[12] D. Cheng and X. Li, "Extracting human body feature measurement based on 3D point cloud data," 2011 4th International Congress on Image and Signal Processing, Shanghai, 2011, pp. 898-901.

[13] C. Changhong and G. Zongliang, "Action recognition from a different view," in China Communications, vol. 10, no. 12, pp. 139-148, Dec. 2013.

[14] B. Kwon, J. Kim, K. Lee, Y. K. Lee, S. Park and S. Lee, "Implementation of a Virtual Training Simulator Based on 360° Multi-View Human Action Recognition," in IEEE Access, vol. 5, pp. 12496-12511, 2017.

[15] E. Ghorbel, R. Boutteau, J. Bonnaert, X. Savatier and S. Lecoeuche, "A fast and accurate motion descriptor for human action recognition applications," 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, 2016, pp. 919-924.

[16] J. P. Vox and F. Wallhoff, "Preprocessing and Normalization of 3D-Skeleton-Data for Human Motion Recognition," 2018 IEEE Life Sciences Conference (LSC), Montreal, QC, 2018, pp. 279-282.

[17] J. Qin, Z. Zhang and Y. Wang, "Cross-view action recognition via transductive transfer learning," 2013 IEEE International Conference on Image Processing, Melbourne, VIC, 2013, pp. 3582-3586.

[18] Q. Xiao and Y. Si, "Human action recognition using autoencoder," 2017 3rd IEEE International Conference on Computer and Communications (ICCC), Chengdu, 2017, pp. 1672-1675.

[19] M. Xin, H. Zhang, D. Yuan and M. Sun, "Learning discriminative action and context representations for action recognition in still images," 2017 IEEE International Conference on Multimedia and Expo (ICME), Hong Kong, 2017, pp. 757-762.

[20] J. Liu, A. Shahroudy, D. Xu, A. C. Kot and G. Wang, "Skeleton-Based Action Recognition Using Spatio-Temporal LSTM Network with Trust Gates," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 12, pp. 3007-3021, 1 Dec. 2018.

[21] Z. Rostami, M. Afrasiabi and H. Khotanlou, "Skeleton-based action recognition using spatio-temporal features with convolutional neural networks," 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, 2017, pp. 0583-0587.

[22] Chuankun Li, Pichao Wang, Shuang Wang, Yonghong Hou and Wanqing Li, "Skeleton-based action recognition using LSTM and CNN," 2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW), Hong Kong, 2017, pp. 585-590.

[23] N. Dawar and N. Kehtarnavaz, "Continuous detection and recognition of actions of interest among actions of non-interest using a depth camera," 2017 IEEE International Conference on Image Processing (ICIP), Beijing, 2017, pp. 4227-4231.

[24] A. Jalal, Y. Kim, S. Kamal, A. Farooq and D. Kim, "Human daily activity recognition with joints plus body features representation using Kinect sensor," 2015 International Conference on Informatics, Electronics & Vision (ICIEV), Fukuoka, 2015, pp. 1-6.

[25] L. Liao, G. Cao and W. Cao, "Action recognition based on depth image sequence," 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Kansas City, MO, 2017, pp. 1583-1587.

[26] Wang, Pichao & Li, Wanqing & Gao, Zhimin & Zhang, Jing & Tang, Chang & Ogunbona, Philip. (2015). Deep Convolutional Neural Networks for Action Recognition Using Depth Map Sequences.

[27] Y. Du, Y. Fu and L. Wang, "Representation Learning of Temporal Dynamics for Skeleton-Based Action Recognition," in IEEE Transactions on Image Processing, vol. 25, no. 7, pp. 3010-3022, July 2016.

[28] Zewei Ding, Pichao Wang, P. O. Ogunbona and Wanqing Li, "Investigation of different skeleton features for CNN-based 3D action recognition," 2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW), Hong Kong, 2017, pp. 617-622.

[29] D. Xu, X. Xiao, X. Wang and J. Wang, "Human action recognition based on Kinect and PSO-SVM by representing 3D skeletons as points in lie group," 2016 International Conference on Audio, Language and Image Processing (ICALIP), Shanghai, 2016, pp. 568-573.

[30] Keyu Li, Z. Liu, Liqin Liang and Yanan Song, "Human action recognition using associated depth and skeleton information," 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, 2016, pp. 418-422.

[31] S. Gourgari, G. Goudelis, K. Karpouzis and S. Kollias, "THETIS: Three-Dimensional Tennis Shots a Human Action Dataset," 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops, Portland, OR, 2013, pp. 676-681, doi: 10.1109/CVPRW.2013.102.

[32] Barchini G, Álvarez M, Herrera S, Trejo M. El rol de las ontologías en los sistemas de información. Revista Ingeniería Informática 2007;14. Disponible en: http://www.inf.udec.cl/revista/ediciones/edicion14/barchini.pdf

[33] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields", 2019 IEEE Transactions on Pattern Analysis and Machine Intelligence

[34] Tomas Simon and Hanbyul Joo and Iain Matthews and Yaser Sheikh, "Hand Keypoint Detection in Single Images using Multiview Bootstrapping, 2017 CVPR

[35] Shih-En Wei and Varun Ramakrishna and Takeo Kanade and Yaser Sheikh, "Convolutional pose machines", 2016 CVPR

[36] Zhe Cao and Tomas Simon and Shih-En Wei and Yaser Sheikh, "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields", 2017 CVPR

[37] 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, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon 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.

[38] Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras

[39] https://colab.research.google.com/

[40] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[41] Juan C. Nez, Ral Cabido, Juan J. Pantrigo, Antonio S. Montemayor, and Jos F. Vlez. 2018. Convolutional Neural Networks and Long Short-Term Memory for skeleton-based human activity and hand gesture recognition. Pattern Recogn. 76, C (April 2018), 80–94. DOI: https://doi.org/10.1016/j.patcog.2017.10.033

[42] Ladjal, Saïd & Newson, Alasdair & Pham, Chi-Hieu. (2019). A PCA-like Autoencoder.

[43] Pavllo, Dario & Feichtenhofer, Christoph & Grangier, David & Auli, Michael. (2019). 3D Human Pose Estimation in Video with Temporal Convolutions and Semi-Supervised Training. 7745-7754. 10.1109/CVPR.2019.00794.

[44] Ross Girshick and Ilija Radosavovic and Georgia Gkioxari and Piotr Dollr and Kaiming He, 2018, Detectron. Retrieved from https://github.com/facebookresearch/detectron

[45] Ezzahout, Abderrahmane and Rachid Oulad Haj Thami. "Conception and development of a video surveillance system for detecting, tracking and profile

analysis of a person." 2013 3rd International Symposium ISKO-Maghreb (2013): 1-5.

[46] M. Doi, A. Tsujimoto, A. Kimachi, S. Nishi and N. Ikoma, "Color Based Person Tracking with Illumination Compensation by using Multiple Statistics of Scene," 2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS), Toyama, Japan, 2018, pp. 427-430, doi: 10.1109/SCIS-ISIS.2018.00081.

[47] Luo, Xinghan et al. "Multi-person tracking based on vertical reference lines and dynamic visibility analysis." 2011 18th IEEE International Conference on Image Processing (2011): 1877-1880.

[48] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E., Scikit-learn: Machine Learning in Python, 2011 Journal of Machine Learning Research: Volume 12, pp. 2825-2830.

[49] Wondershare (2020), Wondershare Filmora9 (Version 9.2.9.13). Descargado de: https://filmora.wondershare.com/es/editor-de-video/

[50] Orrite C., Herrero E., Valencia M. (2019) From Features to Attribute Graphs for Point Set Registration. In: Morales A., Fierrez J., Sánchez J., Ribeiro B. (eds) Pattern Recognition and Image Analysis. IbPRIA 2019. Lecture Notes in Computer Science, vol 11867. Springer, Cham

Anexos

Anexo I: Ontologías

Las ontologías quedan determinadas por los siguientes componentes:

- **Conceptos**: son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.
- **Relaciones**: representan la interacción y enlace entre los conceptos, formando la taxonomía del dominio. Las relaciones básicas son: sub-clase-de, parte-de, conectada-a.
- **Funciones**: son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.
- **Instancias**: se utilizan para representar objetos determinados de un concepto.
- **Axiomas**: son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Especifican las definiciones de los términos en la ontología y las restricciones de sus interpretaciones. Los axiomas deben proveerse para definir la semántica o el significado de los términos.

Bajo estas premisas se elaboran nuestras ontologías, mediante la herramienta *Protègé* [49] de la universidad de Stanford, que nos facilita la organización y visualización de las ontologías. Así pues, quedan definidos dos superclases o conceptos, **Acción** y **Característica**, de los que cuelgan varias subclases (flechas azules), y las propiedades o relaciones (flechas amarilla, naranja y marrón). Dentro de las propiedades, existen de dos tipos: De objeto, las cuales son las que relacionan las clases, y las de datos, que simplemente aportan más información. La diferencia entre las propiedades de datos y las propiedades de objetos radica en que las primeras pueden cuentan con sentido propio e inherente a la acción deportiva. Por ejemplo, podemos ver una secuencia de un golpe de derecha (acción-golpe-Derecha) o podemos ver un video de un golpe ganador (característica-resultado-ganador), mientras que no tiene significado propio, referido al tenis, un video de una vista lateral del jugador o una ubicación del jugador en el fondo de la pista.

En la Figura 1 se muestra una representación del grafo de ontologías creado.

La relación de las características y las acciones, aunque estas puedan ser separables, queda definida mediante las propiedades del objeto, ya que a cada ejemplo le corresponden al menos una acción y alguna de las características. Por defecto, cada secuencia corresponderá a una acción y serán las relaciones con las diferentes características lo que permitirá asociar el ejemplo a las otras subclases. Para ello usamos el término **propiedad**, definida como la relación entre la acción y la característica. Cada acción lleva asociada una propiedad que la relaciona con una o varias características, así

pues, aparecen los conceptos **dominio** de la propiedad, es decir, en qué acción o acciones se aplica dicha característica y el **rango**, que indica el resultado que se extrae de la aplicación de una característica a la acción.

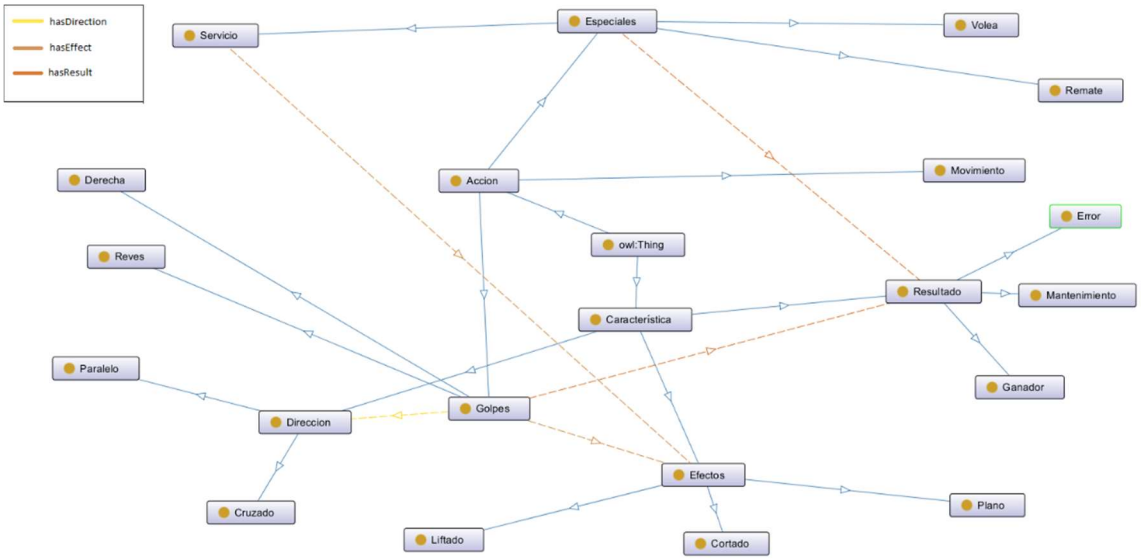


Figura 1: Grafo de clases y relaciones del tenis

Como propiedades de datos relevantes, y que pueden ser interesantes a la hora de profundizar en el conocimiento, se establecen las siguientes:

- Origen: Ubicación en la pista del jugador al golpear (Fondo, medio, red).
- Mano Dominante (Zurda o Diestra).
- Destino: Lugar de la pista al que va dirigido el golpe (Fondo, medio, red).
- Vista: Posición de la cámara (Frontal, lateral, cenital).

Anexo II: Inferencia de OpenPose; Archivos JSON

La fase de inferencia de OpenPose proporciona para cada fotograma evaluado, un archivo json con el siguiente formato:

{"version":1.2,"people":[{"pose_keypoints_2d":[0,0,0,0.747892,0.709014,0.915865,0.758598,0.709086,0.876088,0.764777,0.744495,0.867656,0.755546,0.731016,0.21817,0.737188,0.709032,0.909648,0.72949,0.73377,0.867578,0.729535,0.714624,0.416824,0.747876,0.774467,0.889172,0.754069,0.774458,0.896886,0.764801,0.804538,0.849572,0.770885,0.85618,0.824358,0.740194,0.774457,0.924045,0.727944,0.807187,0.864491,0.715782,0.85082,0.887558,0,0,0,0,0,0,0.753978,0.684678,0.851093,0.741723,0.684656,0.762693,0.711155,0.845414,0.61379,0.706554,0.850862,0.722726,0.715793,0.858988,0.915111,0.772401,0.856289,0.600693,0.776996,0.856331,0.698591,0.770874,0.859046,0.833035],"face_keypoints_2d":[],"hand_left_keypoints_2d":[],"hand_right_keypoints_2d":[],"pose_keypoints_3d":[],"face_keypoints_3d":[],"hand_left_keypoints_3d":[],"hand_right_keypoints_3d":[]},{"pose_keypoints_2d":[0.165736,0.207664,0.195105,0.164235,0.213272,0.572776,0.164369,0.213295,0.654828,0.165854,0.229548,0.50795,0.173441,0.229748,0.25831,0.162676,0.213237,0.438417,0.161251,0.229492,0.179101,0.16585,0.232301,0.0991945,0.164227,0.24598,0.351431,0.164285,0.248556,0.38377,0.173457,0.256879,0.135574,0,0,0,0.162742,0.245947,0.321829,0.167381,0.259559,0.0937203,0,0,0,0.165771,0.205162,0.196456,0,0,0,0.165722,0.205107,0.61723,0.164221,0.20525,0.102741,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],"face_keypoints_2d":[],"hand_left_keypoints_2d":[],"hand_right_keypoints_2d":[],"pose_keypoints_3d":[],"face_keypoints_3d":[],"hand_left_keypoints_3d":[],"hand_right_keypoints_3d":[]}]}

Anexo III: Lee_videos_Tenis.py

```
import json
import numpy as np
from os import listdir
import math
import matplotlib.pyplot as plt
import pandas as pd


threshold = 13      #puntos del esqueleto necesarios para reconocer persona
numpyFrame=np.empty([1,50])
Crop_np=np.empty([1,50])
N=0
labels=[]
erase =[]
#############################################################################
def my_range(start, end, step):
    while start <= end:
        yield start
        start += step

#############################################################################
def connectpoints(x,y,p1,p2):
    x1, x2 = x[p1], x[p2]
    y1, y2 = y[p1], y[p2]
    plt.plot([x1,x2],[y1,y2],'k-')

#############################################################################
def dibuja_esqueleto(x,y):
    connectpoints(x,y,0,1)
    connectpoints(x,y,1,2)
    connectpoints(x,y,2,3)
    connectpoints(x,y,3,4)
    connectpoints(x,y,1,5)
    connectpoints(x,y,5,6)
    connectpoints(x,y,6,7)
    connectpoints(x,y,1,8)
    connectpoints(x,y,8,9)
    connectpoints(x,y,9,10)
    connectpoints(x,y,10,11)
    connectpoints(x,y,8,12)
    connectpoints(x,y,12,13)
    connectpoints(x,y,13,14)
#############################################################################
# ===========================================================================
#   tomando como referencia el vertice de la columna, donde su distancia sea 1
#   escalamos todos los puntos con respecto a  esa medida
# ===========================================================================
def escala_ejeX(p1x,p1y,p2x,p2y):
    ##p1 (3,4) corresponde al cuello p2(24,25) a cintura
    dist= math.sqrt((p2x-p1x)**2+(p2y-p1y)**2)
    if(dist!=0.0):
        return 1.0/dist
    else:
        return -1.0
#############################################################################
#############################################################################
```

```
# ============================================================================
#     Lee un elemento JSON proveniente de OpenPose y almacena la variable
# ============================================================================
def Read_JSON(archivo):
    print(N)
    global Cropframe,numpyFrame,frame,data,escala,plotx,ploty,numpylabels,labels_np
    with open(archivo) as f:
        data = json.load(f)
        if(len(data["people"])!=0):
            size=len(data["people"])
            for ppl in my_range(0,size-1,1):
                Cropframe=[]
                x2=data["people"][0]["pose_keypoints_2d"][24]       #punto [8]
                y2=data["people"][0]["pose_keypoints_2d"][25]
                x1=data["people"][0]["pose_keypoints_2d"][3]       #punto [0]
                y1=data["people"][0]["pose_keypoints_2d"][4]
                if(x1!=0.0 and y1!=0.0 and x2!=0.0 and y2!=0.0):
                    cont=0
                    xC=(x1+x2)/2.0
                    yC=(y1+y2)/2.0
                    for el in my_range(0,72,3):
                        if((data["people"][ppl]["pose_keypoints_2d"][el]!=0.0) &
(data["people"][ppl]["pose_keypoints_2d"][el+1]!=0.0)):

Cropframe.append((data["people"][ppl]["pose_keypoints_2d"][el]-xC))

Cropframe.append((data["people"][ppl]["pose_keypoints_2d"][el+1]-yC))
                        else:
                            Cropframe.append(0.0)
                            Cropframe.append(0.0)
                            cont=cont+1
                    escala = 1.0/(math.sqrt(pow(x2-x1,2)+pow(y2-y1,2)))
                    for el in my_range(0,48,2):
                        Cropframe[el]=Cropframe[el]*escala
                        Cropframe[el+1]=Cropframe[el+1]*-escala
                    if(cont<=threshold):
                        Crop_np = np.array([Cropframe])
                        numpyFrame=np.concatenate((numpyFrame,Crop_np),axis=0)
                    else:
                        #labels_np = np.delete(labels_np,N)
                        erase.append(N)
                else:
                    #labels_np = np.delete(labels_np,N)
                    erase.append(N)
        else:
            #labels_np = np.delete(labels_np,N)
            erase.append(N)
################################################################################
################################################################################
#Read the .xlsx file
file_NUMPY = "x_Tenis.npy"
folder_JSON="E:/UNIVERSIDAD/MASTER_ELECTRONICA/TFM/Database/Videos/JSON"
#Carpeta dentro de root para lectura JSON
file_nplabels = "y_Tenis.npy"
file_frames = "frames_Tenis.npy"
file_labels = "labels_Tenis.npy"
file = 'E:/UNIVERSIDAD/MASTER_ELECTRONICA/TFM/Database/Videos/DATABASE.xlsx'
x1 = pd.ExcelFile(file)
df1 = x1.parse('Hoja1')
```

```
M=0

labels_np = np.zeros((1,1),dtype = int)
#Generamos el np que contendrá el vector de labels
for el in range(df1['FRAME'].size):
    accion = np.array(df1['ACCION'][el],ndmin=2)
    for el2 in range(df1['n frames'][el]):
        labels_np = np.concatenate((labels_np,accion),axis=0)
        print(M)

        M=M+1

frames_np = np.array(df1['n frames'])
acciones_np = np.array(df1['ACCION'])
#Recorremos la carpeta de JSON y generamos x e y si cumplen threshold
for el in listdir(folder_JSON):
    Read_JSON(folder_JSON+"/"+el)
    N=N+1

numpyFrame=np.delete(numpyFrame,0,axis=0)
labels_np=np.delete(labels_np,0,axis=0)

##BORRADO DE LOS DATOS QUE NO PROPORCIONAN INFORMACION DE LOS 3 .NPY
#LOS JSON YA HAN SIDO ELIMINADOS (NO CONTABILIZADOS)
for el in range(len(erase)-1,-1,-1):   #eliminamos lAS LABELS
    labels_np=np.delete(labels_np,erase[el],axis=0)

#LOS FRAMES SE ELIMINAN RECORRIENDO CADA POSICION DEL VECTOR DE FRAMES
    i=0
for x in range(frames_np.shape[0]):
    timesteps=frames_np[x]
    for y in range(timesteps):
        for el in erase:
            if(i+y==el):
                frames_np[x]=frames_np[x]-1
                print("Eliminado")
    i=i+timesteps

##GUARDADO DE LAS VARIBLES
np.save(file_NUMPY,numpyFrame)      #Datos de entrada.npy
np.save(file_nplabels,labels_np)  #Datos de salida.npy
np.save(file_frames,frames_np)    #Datos de frames.npy
np.save(file_labels,acciones_np)  #Datos de etiquetas.npy

numpyFrame = np.load(file_NUMPY)
labels_np = np.load(file_nplabels)

##REPRESENTACION DE UNO DE LOS DATOS PARA COMPROBAR
for el in range(10):
    plotx=[]
    ploty=[]
    for x in my_range(0,48,2):
        plotx.append(numpyFrame[el,x])
        ploty.append(numpyFrame[el,x+1])

    plt.plot(plotx,ploty,'b.')
    dibuja_esqueleto(plotx,ploty)
    plt.pause(0.1)
    plt.clf()
```

Anexo IV: Lee_videos_Thetis.py

```
import json
import numpy as np
from os import listdir
import math
import matplotlib.pyplot as plt
import pandas as pd


threshold = 13      #puntos del esqueleto necesarios para reconocer persona
numpyFrame=np.empty([1,50])
Crop_np=np.empty([1,50])
N=0
labels=[]
erase =[]
#############################################################################
def my_range(start, end, step):
    while start <= end:
        yield start
        start += step

#############################################################################
def connectpoints(x,y,p1,p2):
    x1, x2 = x[p1], x[p2]
    y1, y2 = y[p1], y[p2]
    plt.plot([x1,x2],[y1,y2],'k-')

#############################################################################
def dibuja_esqueleto(x,y):
    connectpoints(x,y,0,1)
    connectpoints(x,y,1,2)
    connectpoints(x,y,2,3)
    connectpoints(x,y,3,4)
    connectpoints(x,y,1,5)
    connectpoints(x,y,5,6)
    connectpoints(x,y,6,7)
    connectpoints(x,y,1,8)
    connectpoints(x,y,8,9)
    connectpoints(x,y,9,10)
    connectpoints(x,y,10,11)
    connectpoints(x,y,8,12)
    connectpoints(x,y,12,13)
    connectpoints(x,y,13,14)
#############################################################################
# ===========================================================================
#   tomando como referencia el vertice de la columna, donde su distancia sea 1
#   escalamos todos los puntos con respecto a  esa medida
# ===========================================================================
def escala_ejeX(p1x,p1y,p2x,p2y):
    ##p1 (3,4) corresponde al cuello p2(24,25) a cintura
    dist= math.sqrt((p2x-p1x)**2+(p2y-p1y)**2)
    if(dist!=0.0):
        return 1.0/dist
    else:
        return -1.0
#############################################################################
def get_label(string):
```

```
    #Dependerá del nombre que tenga el archivo JSON en el que aparece la etiqueta:
Control_test/Viper/MVC/Pose
    if(string.find('drive_')!=-1): #Ori1 o _0 o _0 o 0_
        return 0
    elif(string.find('reves_')!=-1):#Ori2 o _45 o _1 o 1_
        return 1
    elif(string.find('servicio_')!=-1):#Ori2 o _45 o _1 o 1_
        return 2
    elif(string.find('volea_')!=-1):#Ori2 o _45 o _1 o 1_
        return 3
    elif(string.find('remate_')!=-1):#Ori2 o _45 o _1 o 1_
        return 4
    else:
        return -1

################################################################################
# ==============================================================================
#     Lee un elemento JSON proveniente de OpenPose y almacena la variable
# ==============================================================================
def Read_JSON(archivo):
    print(N)
    global
Cropframe,numpyFrame,frame,data,escala,plotx,ploty,numpylabels,labels_np,label
    label=np.zeros((1,1))
    with open(archivo) as f:
        data = json.load(f)
        if(len(data["people"])!=0):
            size=len(data["people"])
            for ppl in my_range(0,size-1,1):
                Cropframe=[]
                x2=data["people"][0]["pose_keypoints_2d"][24]       #punto [8]
                y2=data["people"][0]["pose_keypoints_2d"][25]
                x1=data["people"][0]["pose_keypoints_2d"][3]       #punto [0]
                y1=data["people"][0]["pose_keypoints_2d"][4]
                if(x1!=0.0 and y1!=0.0 and x2!=0.0 and y2!=0.0):
                    cont=0
                    xC=(x1+x2)/2.0
                    yC=(y1+y2)/2.0
                    for el in my_range(0,72,3):
                        if((data["people"][ppl]["pose_keypoints_2d"][el]!=0.0)       &
(data["people"][ppl]["pose_keypoints_2d"][el+1]!=0.0)):

Cropframe.append((data["people"][ppl]["pose_keypoints_2d"][el]-xC))

Cropframe.append((data["people"][ppl]["pose_keypoints_2d"][el+1]-yC))
                        else:
                            Cropframe.append(0.0)
                            Cropframe.append(0.0)
                            cont=cont+1
                    escala = 1.0/(math.sqrt(pow(x2-x1,2)+pow(y2-y1,2)))
                    for el in my_range(0,48,2):
                        Cropframe[el]=Cropframe[el]*escala
                        Cropframe[el+1]=Cropframe[el+1]*-escala
                    if(cont<=threshold):
                        Crop_np = np.array([Cropframe])
                        numpyFrame=np.concatenate((numpyFrame,Crop_np),axis=0)
                        label[0,0]=get_label(archivo)
                        labels_np = np.concatenate((labels_np,label),axis=0)
                    else:
```

```
                        #labels_np = np.delete(labels_np,N)
                        erase.append(N)
                else:
                    #labels_np = np.delete(labels_np,N)
                    erase.append(N)
        else:
            #labels_np = np.delete(labels_np,N)
            erase.append(N)
################################################################################
################################################################################
file_NUMPY = "x_Thetis.npy"
folder_JSON="E:/UNIVERSIDAD/MASTER_ELECTRONICA/TFM/Datasets/Thetis/JSON"      #Carpeta
dentro de root para lectura JSON
file_nplabels = "y_Thetis.npy"
M=0

labels_np = np.zeros((1,1),dtype = int)

#Recorremos la carpeta de JSON y generamos x e y si cumplen threshold
for el in listdir(folder_JSON):
    for el2 in listdir(folder_JSON+"/"+el):
        Read_JSON(folder_JSON+"/"+el+"/"+el2)
        N=N+1

numpyFrame=np.delete(numpyFrame,0,axis=0)
labels_np=np.delete(labels_np,0,axis=0)


##GUARDADO DE LAS VARIBLES
np.save(file_NUMPY,numpyFrame)      #Datos de entrada.npy
np.save(file_nplabels,labels_np)  #Datos de salida.npy


##REPRESENTACION DE UNO DE LOS DATOS PARA COMPROBAR
for el in range(10):
    plotx=[]
    ploty=[]
    for x in my_range(0,48,2):
        plotx.append(numpyFrame[el,x])
        ploty.append(numpyFrame[el,x+1])

    plt.plot(plotx,ploty,'b.')
    dibuja_esqueleto(plotx,ploty)
    plt.pause(0.1)
    plt.clf()
################################################################################
```

Anexo V: *Librería Keras*

Dentro de la documentación web quedan definidos los miembros de las diferentes clases que utiliza esta librería dedicada a la implementación, entrenamiento y validación de redes neuronales. De las distintas técnicas de implementación se ha utilizado el método secuencial, por el cual se añade cada capa de manera ordenada desde la entrada hasta la salida parametrizando en cada una de ellas sus hiperparámetros.

Una vez definida la red, se añaden los métodos de optimización y se genera el entrenamiento e inferencia.

Los diferentes tipos de capas utilizadas se definen a continuación

-Dense (): Se trata de una capa *densely-connected* genérica, la cual aplica la función de activación a la entrada y el *bias* tras pasar por la matriz de pesos (*kernel*).

Se definen el número de neuronas de la capa y la función de activación de las mismas.

-Conv1D (): Se compone de una capa convolucional para el tratamiento temporal de datos que implementa filtrado convolucional al dato de entrada para generar la salida de una dimensión definida.

Se definen el tamaño de la dimensión de salida, la función de activación, y el tamaño del kernel.

-Dropout (): Esta capa tiene como cometido la regularización de la red mediante la desactivación aleatoria de la salida de ciertas neuronas con el fin de evitar el sobreajuste de la red.

Esta capa toma como dimensiones de entrada y salida la de la capa anterior, y queda definida mediante el valor de la ratio de desactivación de neuronas.

-LSTM (): Se trata de una capa recurrente dedicada al tratamiento de datos temporales basada en celdas, las cuales engloban los datos de entrada, salida y los estados anteriores.

Para definir esta capa se establecerá el número de celdas.

-MaxPooling1D (): Se trata de una capa que reduce la dimensionalidad de los datos de entrada a partir del valor máximo definido en una ventana de datos.

Se define dicha capa con sus valores por defecto, lo que reduce a la mitad la dimensión temporal de los datos.

Una vez definidas las capas de la red, el siguiente proceso a definir es el relativo al entrenamiento del modelo. Para ello, la librería utiliza las funciones *compile* () y *fit* (), a la cual se le proporcionan diferentes parámetros.

-Datos: Tanto de entrada como la salida asociada para el entrenamiento

-Tamaño de batch: Número de muestras por cada actualización del gradiente

-Épocas: Número de veces que se realiza el entrenamiento

-Callbacks: Es posible definir ciertos subprocesos dentro del entrenamiento que afectan al rendimiento del mismo, como por ejemplo el EarlyStopping, que finaliza el entrenamiento cuando el rendimiento deja de mejorar.

-Optimizador: Se encarga de gestionar el proceso de entrenamiento, mediante la aplicación de métodos para el descenso por el gradiente.

-Loss: Se define como el error cometido por cada fase del entrenamiento y el parámetro por el cual se optimizará la red.

-Métricas: A la hora de visualizar el proceso de entrenamiento, estos valores muestran los diferentes valores extraídos del entrenamiento para su monitorización y control.

Una vez entrenada la red, se evalúa sobre un set de test mediante las funciones *evaluate* () y *predict*(). Ambas funciones realizan la misma función, aplicar la entrada a la red entrenada y predecir las salidas, devolviendo la primera el *score* obtenido y la segunda los valores predichos de cada muestra de entrada.

Anexo VI: LSTM.py

```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import statistics as sta
import keras.utils
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Dense, LSTM, Dropout, TimeDistributed
from keras.layers.embeddings import Embedding
from keras.callbacks import EarlyStopping
from keras import regularizers
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score,
precision_score,recall_score, roc_auc_score
from itertools import cycle

SW = 40
################################################################################
################################################################################
##################       Auxiliary Functions            ########################
################################################################################
################################################################################

'''Delete data with values out of range [-4,+4]'''
def delete_spureous(data):
    result = data
    for x in range(data.shape[0]):
        for y in range(data.shape[1]):
            if((data[x,y] > 4.0) or (data[x,y] < -4.0)):
                #print(cnt,": Eliminado en ",x,", ",y," el valor: ",data[x,y])
                result[x,y]=0.0
    return result
################################################################################
'''Normalize data between 0 and 1 respect max_value'''
def normalization(data,max_value):
    result = data/max_value
    return result
################################################################################
'''Randomize 2 arrays (a & b)  with the same index order'''
def randomize(a,b):
    # Generate the permutation index array.
    indices = np.arange(a.shape[0])
    np.random.shuffle(indices)
    # Shuffle the arrays by giving the permutation in the square brackets.
    shuffled_a = a[indices]
    shuffled_b = b[indices]
    return shuffled_a, shuffled_b
################################################################################
'''Create a sequence of numbers from start, incremented by step, until end
 (including end value, instead of range() inherited function)'''
def my_range(start, end, step):
    while start <= end:
        yield start
        start += step
################################################################################
################################################################################
##############       Data Arrangement Functions            ####################
```

```
################################################################################
################################################################################

''' Extract the action arrays from NTU Database'''
def Extrae_NTU(data,acciones,frames,labels,name,num,index):
    #Extraemos las secuencias correspodientes a las acciones 11,12,16 y 27
    #acc_np = np.zeros((75,1))
    sec = []

    #Acciones (realizar solo una vez ya que lleva mucho tiempo)
    '''for el in range(0,data.shape[(1)]):
        if(labels[0,el]==num):
            acc_np = np.concatenate((acc_np, data[:,el].reshape(75,1)),axis=1)
    acc_np=np.transpose(acc_np)
    #np.save(name,acc_np)'''

    #Secuencias
    for el in range(0,acciones.shape[(1)]):
        if(acciones[0,el]==num):
            sec.append(frames[0,el])
    sec_np = np.array(sec,dtype='uint8').reshape((1,len(sec)))

    #Labels
    labels_total=np.zeros((data.shape[0],1))
    idx=0
    for el in range(sec_np.shape[1]):
        for i in range (sec_np[0,el]):
            labels_total[idx+i,0]=index
        idx = idx+sec_np[0,el]
    return sec_np,labels_total  #acc_np
################################################################################
'''Train and Test data: Sliding Window with constant size and step'''
def Genera_3D_slideW(n_samples, np_data, labels,Wdw,step):
    num=int(n_samples/step)
    n_train= int(num*0.9)
    n_test=num-n_train
    datos_train = np.zeros((n_train, Wdw, np_data.shape[1]))
    datos_test = np.zeros((n_test+1, Wdw, np_data.shape[1]))
    labels_train = np.zeros((n_train,1))
    labels_test = np.zeros((n_test+1,1))
    i=0
    for el in my_range (0,n_samples-Wdw,step):
        if(i < (n_train)):
            for x in range(Wdw):
                datos_train[i,x,:]=np_data[el+x,:]
            labels_train[i,0]=labels[el+int(Wdw/2),0]
        else:
            for x in range(Wdw):
                datos_test[i-(n_train),x,:]=np_data[el+x,:]
            labels_test[i-(n_train),0]=labels[el+int(Wdw/2),0]
        i=i+1
    return datos_train,datos_test,labels_train,labels_test
################################################################################
'''Train and Test data: Subframe to lowest duration sequence'''
def Genera_3D_subframe(n_samples, np_data, labels,seq,steps_min):
    n_train=int(seq.shape[1]*0.9)
    n_test=seq.shape[1]-n_train

    datos_train = np.zeros((n_train, steps_min+1, np_data.shape[1]))
```

```
    datos_test = np.zeros((n_test, steps_min+1, np_data.shape[1]))
    labels_train = np.zeros((n_train,1))
    labels_test = np.zeros((n_test,1))
    i=0
    for el in range (seq.shape[1]):
        steps = seq[0,el]
        ratio = int(np.ceil(seq[0,el]/steps_min))
        j=0
        if(el < (n_train)):
            for x in my_range(0,steps,ratio):
                datos_train[el,j,:]=np_data[i+x,:]
                j=j+1
            labels_train[el,0]=labels[i,0]
        else:
            for x in my_range(0,steps-1,ratio):
                datos_test[el-(n_train),j,:]=np_data[i+x,:]
                j=j+1
            labels_test[el-(n_train),0]=labels[i,0]
        i=i+steps
    return datos_train,datos_test,labels_train,labels_test
################################################################################
################################################################################
#######################        Data Section          ###########################
################################################################################
################################################################################


'''NTU Database (Skeletons + Labels + N.Frames)'''

mat = sio.loadmat('E:/UNIVERSIDAD/MASTER_ELECTRONICA/TFM/LSTM/seleccion.mat')
acciones_NTU = np.array(mat['accionstrain'])
actortrain_NTU = np.array(mat['actortrain'])
data_NTU = np.array(mat['data'])
labels_NTU = np.array(mat['labelstrain'])
frames_NTU = np.array(mat['n_frames_train'])

acc11_NTU=np.transpose(np.load('ACC11.npy'))
acc12_NTU=np.transpose(np.load('ACC12.npy'))
acc27_NTU=np.transpose(np.load('ACC27.npy'))
acc16_NTU=np.transpose(np.load('ACC16.npy'))

sec11_NTU,labels_total11_NTU =
Extrae_NTU(acc11_NTU,acciones_NTU,frames_NTU,labels_NTU,'ACC11.npy',11,0)
sec12_NTU,labels_total12_NTU =
Extrae_NTU(acc12_NTU,acciones_NTU,frames_NTU,labels_NTU,'ACC12.npy',12,1)
sec16_NTU,labels_total16_NTU =
Extrae_NTU(acc16_NTU,acciones_NTU,frames_NTU,labels_NTU,'ACC16.npy',16,2)
sec27_NTU,labels_total27_NTU =
Extrae_NTU(acc27_NTU,acciones_NTU,frames_NTU,labels_NTU,'ACC27.npy',27,3)

labels_total16_NTU =2*np.ones((labels_total16_NTU.shape),float)
#Train and test data generation

'''Sliding Window Method'''
X11W_train,X11W_test,Y11W_train,Y11W_test =
Genera_3D_slideW(acc11_NTU.shape[0],acc11_NTU,labels_total11_NTU,SW,int(SW/4))
X12W_train,X12W_test,Y12W_train,Y12W_test =
Genera_3D_slideW(acc12_NTU.shape[0],acc12_NTU,labels_total12_NTU,SW,int(SW/4))
```

```
X16W_train,X16W_test,Y16W_train,Y16W_test =
Genera_3D_slideW(acc16_NTU.shape[0],acc16_NTU,labels_total16_NTU,SW,int(SW/4))
X27W_train,X27W_test,Y27W_train,Y27W_test =
Genera_3D_slideW(acc27_NTU.shape[0],acc27_NTU,labels_total27_NTU,SW,int(SW/4))

X_trainW_NTU = np.concatenate((X11W_train,X12W_train,X16W_train,X27W_train)) #ACC11-
ACC12-ACC16-ACC27
X_testW_NTU = np.concatenate((X11W_test,X12W_test,X16W_test,X27W_test))
Y_trainW_NTU =
keras.utils.to_categorical(np.concatenate((Y11W_train,Y12W_train,Y16W_train,Y27W_trai
n)))
Y_testW_NTU =
keras.utils.to_categorical(np.concatenate((Y11W_test,Y12W_test,Y16W_test,Y27W_test)))

'''Subframe Method'''
steps_min_NTU =
min(np.min(sec11_NTU),np.min(sec12_NTU),np.min(sec16_NTU),np.min(sec27_NTU))
X11S_train,X11S_test,Y11S_train,Y11S_test = Genera_3D_subframe(acc11_NTU.shape[0],
acc11_NTU, labels_total11_NTU, sec11_NTU,steps_min_NTU)
X12S_train,X12S_test,Y12S_train,Y12S_test = Genera_3D_subframe(acc12_NTU.shape[0],
acc12_NTU, labels_total12_NTU, sec12_NTU,steps_min_NTU)
X16S_train,X16S_test,Y16S_train,Y16S_test = Genera_3D_subframe(acc16_NTU.shape[0],
acc16_NTU, labels_total16_NTU, sec16_NTU,steps_min_NTU)
X27S_train,X27S_test,Y27S_train,Y27S_test = Genera_3D_subframe(acc27_NTU.shape[0],
acc27_NTU, labels_total27_NTU, sec27_NTU,steps_min_NTU)

X_trainS_NTU = np.concatenate((X11S_train,X12S_train,X16S_train,X27S_train)) #ACC11-
ACC12-ACC16-ACC27
X_testS_NTU = np.concatenate((X11S_test,X12S_test,X16S_test,X27S_test))
Y_trainS_NTU =
keras.utils.to_categorical(np.concatenate((Y11S_train,Y12S_train,Y16S_train,Y27S_trai
n)))
Y_testS_NTU =
keras.utils.to_categorical(np.concatenate((Y11S_test,Y12S_test,Y16S_test,Y27S_test)))

##############################################################################
##############################################################################

'''TENIS Database (Skeletons + Labels + N.Frames)'''

data_tenis=np.load('x_Tenis.npy')
frames_tenis=np.load('frames_Tenis.npy')
labels_tenis=np.load('labels_Tenis.npy')

labels_tenis=np.reshape(labels_tenis,(1,labels_tenis.shape[0]))
frames_tenis=np.reshape(frames_tenis,(1,frames_tenis.shape[0]))

labels_total_tenis=np.zeros((data_tenis.shape[0],1))
idx=0
for el in range (frames_tenis.shape[1]):
    for i1 in range(frames_tenis[0,el]):
        labels_total_tenis[idx+i1,0]=labels_tenis[0,el]
    idx=idx+frames_tenis[0,el]

#Train and test data generation

'''Sliding Window Method'''
X_trainW_TENIS,X_testW_TENIS,Y_trainW_TENIS,Y_testW_TENIS =
Genera_3D_slideW(data_tenis.shape[0], data_tenis, labels_total_tenis, SW, int(SW/4))
```

```
'''Subframe Method'''
steps_min_TENIS = np.min(frames_tenis)
X_trainS_TENIS,X_testS_TENIS,Y_trainS_TENIS,Y_testS_TENIS =
Genera_3D_subframe(data_tenis.shape[0],data_tenis,labels_total_tenis,frames_tenis,ste
ps_min_TENIS)

Y_trainS_TENIS = keras.utils.to_categorical(Y_trainS_TENIS,6)
Y_testS_TENIS = keras.utils.to_categorical(Y_testS_TENIS,6)
Y_trainW_TENIS = keras.utils.to_categorical(Y_trainW_TENIS,6)
Y_testW_TENIS = keras.utils.to_categorical(Y_testW_TENIS,6)

################################################################################
################################################################################

'''THETIS Database (Skeletons + Labels +3D )'''

data_thetis=np.load('x_Thetis.npy')
labels_total_thetis = np.load("y_Thetis.npy")

#Reconstrucción 3D
x_drive3d = np.load("3d_Thetis/data_drive.npy")
y_drive3d = np.zeros((x_drive3d.shape[0],1))
frames_drive3d= np.load("3d_Thetis/frames_drive.npy")

x_reves3d = np.load("3d_Thetis/data_reves.npy")
y_reves3d = np.ones((x_reves3d.shape[0],1))
frames_reves3d= np.load("3d_Thetis/frames_reves.npy")

x_servicio3d = np.load("3d_Thetis/data_servicio.npy")
y_servicio3d = 2 * np.ones((x_servicio3d.shape[0],1))
frames_servicio3d= np.load("3d_Thetis/frames_servicio.npy")

x_volea3d = np.load("3d_Thetis/data_volea.npy")
y_volea3d = 3 * np.ones((x_volea3d.shape[0],1))
frames_volea3d= np.load("3d_Thetis/frames_volea.npy")

x_remate3d = np.load("3d_Thetis/data_remate.npy")
y_remate3d = 4 * np.ones((x_remate3d.shape[0],1))
frames_remate3d= np.load("3d_Thetis/frames_remate.npy")

#Train and test data generation 3D
'''Subframe method'''
steps_min_THETIS =
min(np.min(frames_drive3d),np.min(frames_reves3d),np.min(frames_servicio3d),np.min(fr
ames_volea3d),np.min(frames_remate3d))

XdriveS_train,XdriveS_test,YdriveS_train,YdriveS_test =
Genera_3D_subframe(x_drive3d.shape[0], x_drive3d, y_drive3d,
frames_drive3d,steps_min_THETIS)
XrevesS_train,XrevesS_test,YrevesS_train,YrevesS_test =
Genera_3D_subframe(x_reves3d.shape[0], x_reves3d, y_reves3d,
frames_reves3d,steps_min_THETIS)
XservicioS_train,XservicioS_test,YservicioS_train,YservicioS_test =
Genera_3D_subframe(x_servicio3d.shape[0], x_servicio3d, y_servicio3d,
frames_servicio3d,steps_min_THETIS)
XvoleaS_train,XvoleaS_test,YvoleaS_train,YvoleaS_test =
Genera_3D_subframe(x_volea3d.shape[0], x_volea3d, y_volea3d,
frames_volea3d,steps_min_THETIS)
```

```
XremateS_train,XremateS_test,YremateS_train,YremateS_test =
Genera_3D_subframe(x_remate3d.shape[0], x_remate3d, y_remate3d,
frames_remate3d,steps_min_THETIS)

X_trainS_THETIS3D =
np.concatenate((XdriveS_train,XrevesS_train,XservicioS_train,XvoleaS_train,XremateS_t
rain)) #ACC11-ACC12-ACC16-ACC27
X_testS_THETIS3D =
np.concatenate((XdriveS_test,XrevesS_test,XservicioS_test,XvoleaS_test,XremateS_test)
)
Y_trainS_THETIS3D =
keras.utils.to_categorical(np.concatenate((YdriveS_train,YrevesS_train,YservicioS_tra
in,YvoleaS_train,YremateS_train)))
Y_testS_THETIS3D =
keras.utils.to_categorical(np.concatenate((YdriveS_test,YrevesS_test,YservicioS_test,
YvoleaS_test,YremateS_test)))

'''Sliding Window'''
XdriveW_train,XdriveW_test,YdriveW_train,YdriveW_test =
Genera_3D_slideW(x_drive3d.shape[0],x_drive3d,y_drive3d,SW,int(SW/4))
XrevesW_train,XrevesW_test,YrevesW_train,YrevesW_test =
Genera_3D_slideW(x_reves3d.shape[0],x_reves3d,y_reves3d,SW,int(SW/4))
XservicioW_train,XservicioW_test,YservicioW_train,YservicioW_test =
Genera_3D_slideW(x_servicio3d.shape[0],x_servicio3d,y_servicio3d,SW,int(SW/4))
XvoleaW_train,XvoleaW_test,YvoleaW_train,YvoleaW_test =
Genera_3D_slideW(x_volea3d.shape[0],x_volea3d,y_volea3d,SW,int(SW/4))
XremateW_train,XremateW_test,YremateW_train,YremateW_test =
Genera_3D_slideW(x_remate3d.shape[0],x_remate3d,y_remate3d,SW,int(SW/4))

X_trainW_THETIS3D =
np.concatenate((XdriveW_train,XrevesW_train,XservicioW_train,XvoleaW_train,XremateW_t
rain)) #ACC11-ACC12-ACC16-ACC27
X_testW_THETIS3D =
np.concatenate((XdriveW_test,XrevesW_test,XservicioW_test,XvoleaW_test,XremateW_test)
)
Y_trainW_THETIS3D =
keras.utils.to_categorical(np.concatenate((YdriveW_train,YrevesW_train,YservicioW_tra
in,YvoleaW_train,YremateW_train)))
Y_testW_THETIS3D =
keras.utils.to_categorical(np.concatenate((YdriveW_test,YrevesW_test,YservicioW_test,
YvoleaW_test,YremateW_test)))


#Train and test data generation 2D
'''Sliding Window Method'''
X_trainW_THETIS,X_testW_THETIS,Y_trainW_THETIS,Y_testW_THETIS =
Genera_3D_slideW(data_thetis.shape[0],data_thetis,labels_total_thetis,SW,int(SW/4))

'''Concatenate vectors-randomize-separate'''
X_THETIS = np.concatenate((X_trainW_THETIS,X_testW_THETIS))
Y_THETIS = np.concatenate((Y_trainW_THETIS,Y_testW_THETIS))

X_THETIS,Y_THETIS = randomize(X_THETIS, Y_THETIS)

X_trainW_THETIS=X_THETIS[0:X_trainW_THETIS.shape[0],:,:]
X_testW_THETIS=X_THETIS[X_trainW_THETIS.shape[0]:X_THETIS.shape[0],:,:]
Y_trainW_THETIS=Y_THETIS[0:Y_trainW_THETIS.shape[0],:]
Y_testW_THETIS=Y_THETIS[Y_trainW_THETIS.shape[0]:Y_THETIS.shape[0],:]
```

```
Y_trainW_THETIS = keras.utils.to_categorical(Y_trainW_THETIS,5)
Y_testW_THETIS = keras.utils.to_categorical(Y_testW_THETIS,5)


'''DATA 3D'''


################################################################################
################################################################################
####################       Neural Algorithms          ##########################
################################################################################
################################################################################

'''Control LSTM'''
def evaluate_model1(trainX, trainy, testX, testy, n_timesteps, n_outputs,
n_features):
    epochs, batch_size = 30, int(trainX.shape[0]/100)

    model = Sequential()
    model.add(LSTM(200,input_shape=(n_timesteps,n_features),return_sequences = True,
dropout=0.2))
    model.add(LSTM(100))
    model.add(Dense(50, activation='sigmoid'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adadelta',
metrics=['accuracy'])
    # fit network
    es=EarlyStopping(monitor='loss',patience=1, min_delta = 0.005)
    print(model.summary())
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,callbacks =[])
    # evaluate model
    accuracy = model.evaluate(testX, testy, batch_size=batch_size)
    prediction = model.predict(testX,batch_size=batch_size)
    return accuracy,prediction
################################################################################

'''LSTM + Conv1D'''
def evaluate_model2(trainX, trainy, testX, testy, n_timesteps, n_outputs,
n_features):
    epochs, batch_size = 30, int(trainX.shape[0]/100)

    model = Sequential()

model.add(Conv1D(filters=200,kernel_size=3,activation='relu',input_shape=(n_timesteps
,n_features)))
    model.add(LSTM(100,input_shape=(n_timesteps,n_features)))
    model.add(Dense(50, activation='sigmoid'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adadelta',
metrics=['accuracy'])
    # fit network
    #es=EarlyStopping(monitor='loss',patience=1, min_delta = 0.005)
    print(model.summary())
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,callbacks =[])
    # evaluate model
    accuracy = model.evaluate(testX, testy, batch_size=batch_size)
    prediction = model.predict(testX,batch_size=batch_size)
    return accuracy,prediction
################################################################################
```

```
'''conv/Pool profundo 20 50 10 0 + lstm100'''
def evaluate_model3(trainX, trainy, testX, testy, n_timesteps, n_outputs,
n_features):
    epochs, batch_size = 40, int(trainX.shape[0]/100)

    model = Sequential()

model.add(Conv1D(filters=100,kernel_size=3,activation='relu',input_shape=(n_timesteps
,n_features)))
    model.add(MaxPooling1D())
    model.add(Conv1D(filters=50,kernel_size=3,activation='relu'))
    model.add(MaxPooling1D())
    model.add(Conv1D(filters=25,kernel_size=3,activation='relu'))
    #model.add(MaxPooling1D())

    model.add(LSTM(100))
    model.add(Dense(50, activation='sigmoid'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adadelta',
metrics=['categorical_accuracy','acc'])
    # fit network
    es=EarlyStopping(monitor='loss',patience=1, min_delta = 0.005)
    print(model.summary())
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,callbacks =[])

    # evaluate model
    accuracy = model.evaluate(testX, testy, batch_size=batch_size)
    prediction = model.predict(testX,batch_size=batch_size)
    return accuracy,prediction

################################################################################
######################      Run Experiments          ##########################
################################################################################

def run_experiment(X_train,X_test,Y_train,Y_test):


    loss, output = evaluate_model1(X_train, Y_train, X_test,
Y_test,X_train.shape[1],Y_train.shape[1],X_train.shape[2])


    ok=0
    for el in range(Y_test.shape[0]):
        out1=np.where(output[el]==np.max(output[el]))
        out1=out1[0][0]
        out2=np.where(Y_test[el]==np.max(Y_test[el]))
        out2=out2[0][0]
        if( out1 == out2):
           ok=ok+1

    score = (ok*100.0)/float(Y_test.shape[0])

matrix=confusion_matrix(np.argmax(output,axis=1),np.argmax(Y_test,axis=1),normalize='
true')
    print("Aciertos: %2d/%2d (%.2f%%)" % (ok,Y_test.shape[0],score))
    return output,score,matrix
################################################################################
#                           LSTM
```

```
################################################################################
'''LSTM NTU'''
#NTU_SW
#output1,score1, matrix1 =
run_experiment(X_trainW_NTU,X_testW_NTU,Y_trainW_NTU,Y_testW_NTU)

#NTU_SF
#output2,score2, matrix2 =
run_experiment(X_trainS_NTU,X_testS_NTU,Y_trainS_NTU,Y_testS_NTU)


'''LSTM TENIS'''
#TENIS_SW
#output3,score3, matrix3 =
run_experiment(X_trainW_TENIS,X_testW_TENIS,Y_trainW_TENIS,Y_testW_TENIS)

#TENIS_SF
#output4,score4, matrix4 =
run_experiment(X_trainS_TENIS,X_testS_TENIS,Y_trainS_TENIS,Y_testS_TENIS)

'''LSTM THETIS'''
#THETIS_SW 2D
#output5,score5, matrix5 =
run_experiment(X_trainW_THETIS,X_testW_THETIS,Y_trainW_THETIS,Y_testW_THETIS)

#TEHTIS_3D_SF
output6,score6, matrix6 =
run_experiment(X_trainS_THETIS3D,X_testS_THETIS3D,Y_trainS_THETIS3D,Y_testS_THETIS3D)

#THETIS_3D_SW
output7,score7, matrix7 =
run_experiment(X_trainW_THETIS3D,X_testW_THETIS3D,Y_trainW_THETIS3D,Y_testW_THETIS3D)
```

Anexo VII: *Autoencoders.py*

```
# -*- coding: utf-8 -*-
"""Autoencoders.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1ekDiQtKU0jf1TMVQOa00Q3lxiCsXhaRr
"""

from google.colab import drive
drive.mount('/content/drive/', force_remount=True)

import csv
import sys
import numpy as np
import matplotlib.pyplot as plt
from functools import partial
import tensorflow.compat.v1 as tf
import numpy.random as rnd
import joblib
from sklearn.metrics import confusion_matrix
from mpl_toolkits.mplot3d import axes3d, Axes3D
from keras.layers import Input, Dense, Conv1D, MaxPooling1D, UpSampling1D
from keras.models import Model
from keras import regularizers
from keras import backend as K

tf.compat.v1.disable_eager_execution()
tf.disable_v2_behavior()

"""DATOS"""

#DATA3D
NTU_3D=np.load("/content/drive/My Drive/TFM/acciones.npy")##3325519 muestras NTU 3D
Tenis_3D=np.load("/content/drive/My  Drive/TFM/tenisNTU.npy")##57056  muestras  x_Tenis
2D formato NTU

MPII_2D = np.load("/content/drive/My Drive/TFM/x_MPII.npy")##23738 muestras x_Tenis  2D
formato NTU)
Pose_2D  =  np.load("/content/drive/My  Drive/TFM/x_Pose.npy")##8  muestras  x_Tenis   2D
formato NTU)

NTU_3D =np.transpose(NTU_3D)
Tenis_3D =np.transpose(Tenis_3D)
print(NTU_3D.shape)
print(Tenis_3D.shape)

print(MPII_2D.shape)
print(Pose_2D.shape)

"""Auxiliary Functions"""

#################################################################################
def connectpoints(x,y,z,p1,p2):
    x1, x2 = x[p1], x[p2]
    y1, y2 = y[p1], y[p2]
    z1, z2 = z[p1], z[p2]
```

```
    plt.plot([x1,x2],[y1,y2],[z1,z2],'k-')
################################################################################
def dibuja_esqueleto2(x,y,z):
    connectpoints(x,y,z,0,1)
    connectpoints(x,y,z,1,2)
    connectpoints(x,y,z,2,3)
    connectpoints(x,y,z,3,4)
    connectpoints(x,y,z,1,5)
    connectpoints(x,y,z,5,6)
    connectpoints(x,y,z,6,7)
    connectpoints(x,y,z,1,8)
    connectpoints(x,y,z,8,9)
    connectpoints(x,y,z,9,10)
    connectpoints(x,y,z,10,11)
    connectpoints(x,y,z,8,12)
    connectpoints(x,y,z,12,13)
    connectpoints(x,y,z,13,14)
################################################################################
def dibuja_esqueleto(x,y,z):
    connectpoints(x,y,z,0,1)
    connectpoints(x,y,z,1,2)
    connectpoints(x,y,z,2,3)
    connectpoints(x,y,z,2,4)
    connectpoints(x,y,z,2,8)
    connectpoints(x,y,z,8,9)
    connectpoints(x,y,z,9,10)
    connectpoints(x,y,z,4,5)
    connectpoints(x,y,z,5,6)
    connectpoints(x,y,z,0,16)
    connectpoints(x,y,z,0,12)
    connectpoints(x,y,z,16,17)
    connectpoints(x,y,z,17,18)
    connectpoints(x,y,z,12,13)
    connectpoints(x,y,z,13,14)
    connectpoints(x,y,z,14,15)
    connectpoints(x,y,z,18,19)
    connectpoints(x,y,z,10,11)
    connectpoints(x,y,z,11,24)
    connectpoints(x,y,z,11,23)
    connectpoints(x,y,z,6,7)
    connectpoints(x,y,z,7,22)
    connectpoints(x,y,z,7,21)

################################################################################
def my_range(start, end, step):
    while start <= end:
        yield start
        start += step
################################################################################
def dibuja_accion_3D(data):
    fig = plt.figure()
    #ax = fig.add_subplot(111,projection='3d')
    ax = Axes3D(fig)
    plotx=[]
    ploty=[]
    plotz=[]
    for x in my_range(0,72,3):
      plotx.append(data[x+2])
      ploty.append(data[x+1])
```

```
      plotz.append(data[x])

    #plt.figure(1)
    ax.scatter3D(plotx,ploty,plotz,c=plotz,cmap='Blues')
    dibuja_esqueleto(plotx,ploty,plotz)

    plt.show()
    plt.close()
    #plt.plot(plotx,ploty,'b.')
################################################################################
def dibuja_accion_2D(data):
    fig = plt.figure()
    #ax = fig.add_subplot(111,projection='3d')
    ax = Axes3D(fig)
    plotx=[]
    ploty=[]
    plotz=[]
    for x in my_range(0,48,2):
      plotx.append(0)
      ploty.append(data[x])
      plotz.append(data[x+1])

    #plt.figure(1)
    ax.scatter3D(plotx,ploty,plotz,c=plotz,cmap='Blues')
    dibuja_esqueleto(plotx,ploty,plotz)

    plt.show()
    plt.close()
    #plt.plot(plotx,ploty,'b.')

################################################################################
def dibuja_accion_Conv(data):
    fig = plt.figure()
    #ax = fig.add_subplot(111,projection='3d')
    ax = Axes3D(fig)
    plotx=[]
    ploty=[]
    plotz=[]
    for x in range(25):
        ploty.append(data[x,0])
        plotz.append(data[x,1])
        plotx.append(data[x,2])

    #plt.figure(1)
    ax.scatter3D(plotx,ploty,plotz,c=plotz,cmap='Blues')
    dibuja_esqueleto(plotx,ploty,plotz)

    plt.show()
    plt.close()
    #plt.plot(plotx,ploty,'b.')

################################################################################
def ResizeArray(D2): #from (n,75) to (n,25,3)
  D3 = np.zeros((D2.shape[0],28,3))
  for el in range(D2.shape[0]):
    j=0
    for i in range(0,D2.shape[1],3):
      D3[el][j][0] = D2[el][i]
      D3[el][j][1] = D2[el][i+1]
```

```
      D3[el][j][2] = D2[el][i+2]
      j=j+1
  return D3
#################################################################################
def D3toD2(D3):
  D2 = np.zeros((D3.shape[0],50))
  for el in range(D3.shape[0]):
    j=0
    for i in range(0,D3.shape[1],3):
      D2[el][j]=D3[el][i+1]+0.5
      j=j+1
      D2[el][j]=D3[el][i]
      j=j+1
  return D2
#################################################################################
def reshape3D(D3):
  D33 = np.zeros((D3.shape[0],75))
  for el in range(D3.shape[0]):
    j=0
    for i in range(0,D3.shape[1],3):
      D33[el][j]=D3[el][i+1]+0.5
      j=j+1
      D33[el][j]=D3[el][i]
      j=j+1
      D33[el][j]=D3[el][i+2]
      j=j+1

  return D33

"""DATA RESHAPE FROM 3D(n,(25x3)) to 2D(n,(25x2))"""

NTU_2D = D3toD2(NTU_3D)
Tenis_3D = reshape3D(Tenis_3D)
Tenis_2D =D3toD2(Tenis_3D)

Tenis_3D = reshape3D(Tenis_3D)

"""AUTOENCODER"""

def Autoencoder(train_AE, data_rec):
  tf.reset_default_graph()

  dataX= train_AE     #Datos de entrenamiento
  n_input = train_AE.shape[1]
  hidden1 = 25
  hidden2 =15
  hidden3=10
  hidden4=hidden2
  hidden5=hidden1
  n_output=n_input

  learning_rate = 0.01
  l2_reg = 0.0005

  X = tf.placeholder(tf.float32, [None, n_input])

  #he_init = tf.contrib.layers.variance_scaling_initializer()
  #l2_regularizer = tf.contrib.layers.l2_regularizer(l2_reg)
```

```
  hidden1 = tf.layers.dense(X, hidden1, activation=tf.nn.relu, name="hidden1")
  hidden2 = tf.layers.dense(hidden1, hidden2, activation=tf.nn.relu, name="hidden2")
  hidden3 = tf.layers.dense(hidden2, hidden3, activation=tf.nn.relu, name="hidden3")
  hidden4 = tf.layers.dense(hidden3, hidden4, activation=tf.nn.relu, name="hidden4")
  hidden5 = tf.layers.dense(hidden4, hidden5, activation=tf.nn.relu, name="hidden5")
  outputs = tf.layers.dense(hidden5, n_output,activation=None, name="output")

  reconstruction_loss = tf.reduce_mean(tf.square(outputs - X))

  reg_losses = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
  loss = tf.add_n([reconstruction_loss] + reg_losses)

  optimizer = tf.train.AdamOptimizer(learning_rate)
  training_op = optimizer.minimize(loss)

  init = tf.global_variables_initializer()
  saver = tf.train.Saver() # not shown in the book

  n_epochs = 15
  batch_size = int(train_AE.shape[0]*0.01)

  with tf.Session() as sess:
      init.run()
      for epoch in range(n_epochs):
          n_batches = len(dataX) // batch_size
          for iteration in range(n_batches):
              print("\r{}%".format(100 * iteration // n_batches), end="")
              sys.stdout.flush()
              sess.run(training_op, feed_dict={X: dataX})
          loss_train = reconstruction_loss.eval(feed_dict={X: dataX})
          print("\r{}".format(epoch), "Train MSE:", loss_train)

      outputs_val = outputs.eval(feed_dict={X: data_rec})
  return outputs_val

"""2D & 3D RECONSTRUCTION"""

reconstruction_2D = Autoencoder(NTU_2D,Tenis_2D)
reconstruction_3D = Autoencoder(NTU_3D,Tenis_3D)

rec_Pose = Autoencoder(MPII_2D,Pose_2D)

"""2D Reconstruction vs 2D Origin"""

i=4
#dibuja_accion_2D(Pose_2D[i,:])
#dibuja_accion_2D(rec_Pose[i,:])

i=60
dibuja_accion_2D(NTU_2D[i,:])
dibuja_accion_2D(Tenis_2D[i,:])
dibuja_accion_2D(reconstruction_2D[i,:])

"""3D Reconstruction vs 3D Origin"""

i=7000
dibuja_accion_3D(NTU_3D[i,:])
dibuja_accion_3D(Tenis_3D[i,:])
dibuja_accion_3D(reconstruction_3D[i,:])
```

Red	Dim	1.LSTM			2. LSTM+Conv1D			3. LSTM+Conv+MaxPool		
Method		SW(40)	SW(20)	SF	SW(40)	SW(20)	SF	SW(40)	SW(20)	SF
NTU (%)	**3D**	**64.98**	**65.10**	**76.21**	**72.20**	**68.67**	**72.12**	**68.00**	**65.59**	**74.72**
Train acc (%)		81.99	87.79	76.74	97.82	93.10	95.33	96.77	94.22	93.21
Train loss		0.4058	0.2967	0.5014	0.0594	0.045	0.1061	0.0854	0.1462	0.1652
epochs		18	34	14	40	40	40	40	40	40
tSim (s)		414	782	42	800	800	120	120	80	40
Tenis(%)	**2D**	**67.13**	**61.15**	**57.35**	**60.49**	**58.71**	**63.24**	**60.84**	**52.76**	**69.12**
Train acc (%)		77.41	80.72	85.93	98.64	96.49	100	94,82	89.73	99.67
Train loss		0.63	0.5220	0.4338	0.0431	0.1053	0.0024	0.1616	0.2915	0.0561
epochs		13	25	12	40	40	40	40	40	40
tSim(s)		78	125	25	280	160	80	40	40	20
Thetis(%)	**2D**	**79.15**	**75.06**		**86,75**	**85.72**		**83.69**	**77.49**	
Train acc (%)		79.52	75.95		95.99	92.31		89.66	81.41	
Train loss		0.4898	0.5844		0.1087	0.1945		0.2518	0.4477	
epochs		30	33		40	40		40	40	
tSim(s)		420	495		800	600		80	40	
Thetis3D(%)	**3D**	**49.80**	**48.63**	**48.00**	**43.51**	**49.61**	**45.00**	**48.22**	**46.66**	**46.00**
Train acc (%)		81.90	82.38	85.11	92.75	90.90	0.9348	94.24	88.02	94.78
Train loss		0.4164	0.4157	0.3531	0.1802	0.2241	0.1717	0.1507	0.2879	0.1522
epochs		30	30	30	30	30	30	40	40	30
tSim(s)		360	360	60	180	150	30	80	40	30

Otro de los métodos de análisis de errores es la comparativa de los valores de salida con la salida real, como se muestra en la Figura 1, se observa que en determinados puntos del set de test no existe ningún tipo de cohesión que nos permita deducir que está ocurriendo con los datos.

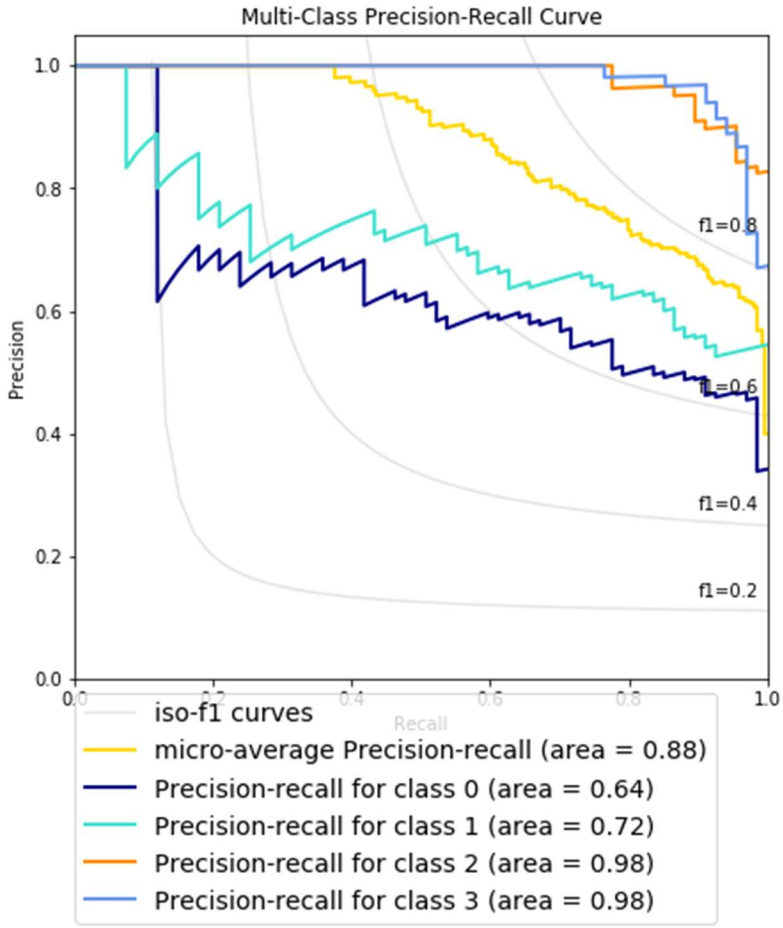
	0	1	2	3	4
33	0.146679	0.846538	0.00130214	0.00494572	0.000535765
34	0.0171392	0.968835	0.00701713	0.00530122	0.00170789
35	0.013738	0.660425	0.259363	0.0478526	0.0186219
36	0.00669458	0.315428	0.255936	0.418213	0.00372843
37	0.00463097	0.27667	0.0329571	0.684948	0.000793762
38	0.0292269	0.184266	0.0688924	0.715886	0.00172895
39	0.92132	0.0571463	0.0103077	0.00855732	0.00266846
40	0.95203	0.0221639	0.00183878	0.0228453	0.00112161
41	0.984294	0.0119208	0.000593313	0.00241059	0.000781235
42	0.740741	0.245888	0.00185801	0.0110731	0.000440034
43	0.0281714	0.247036	0.595762	0.00520052	0.12383
44	0.0177277	0.0697038	0.785797	0.00469907	0.122073
45	0.0824925	0.028064	0.770114	0.0121291	0.107201

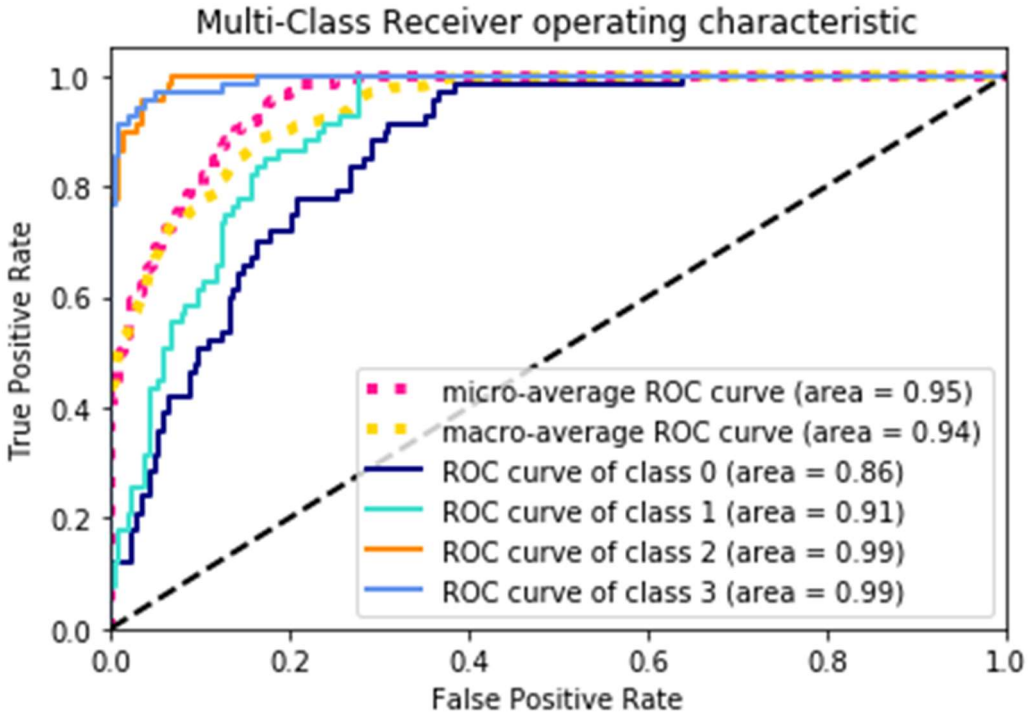
	0	1	2	3	4
33	1	0	0	0	0
34	1	0	0	0	0
35	1	0	0	0	0
36	1	0	0	0	0
37	1	0	0	0	0
38	1	0	0	0	0
39	1	0	0	0	0
40	1	0	0	0	0
41	1	0	0	0	0
42	1	0	0	0	0
43	1	0	0	0	0
44	1	0	0	0	0
45	1	0	0	0	0

Figura. 1: Salida predicha/Salida real conjunto de test

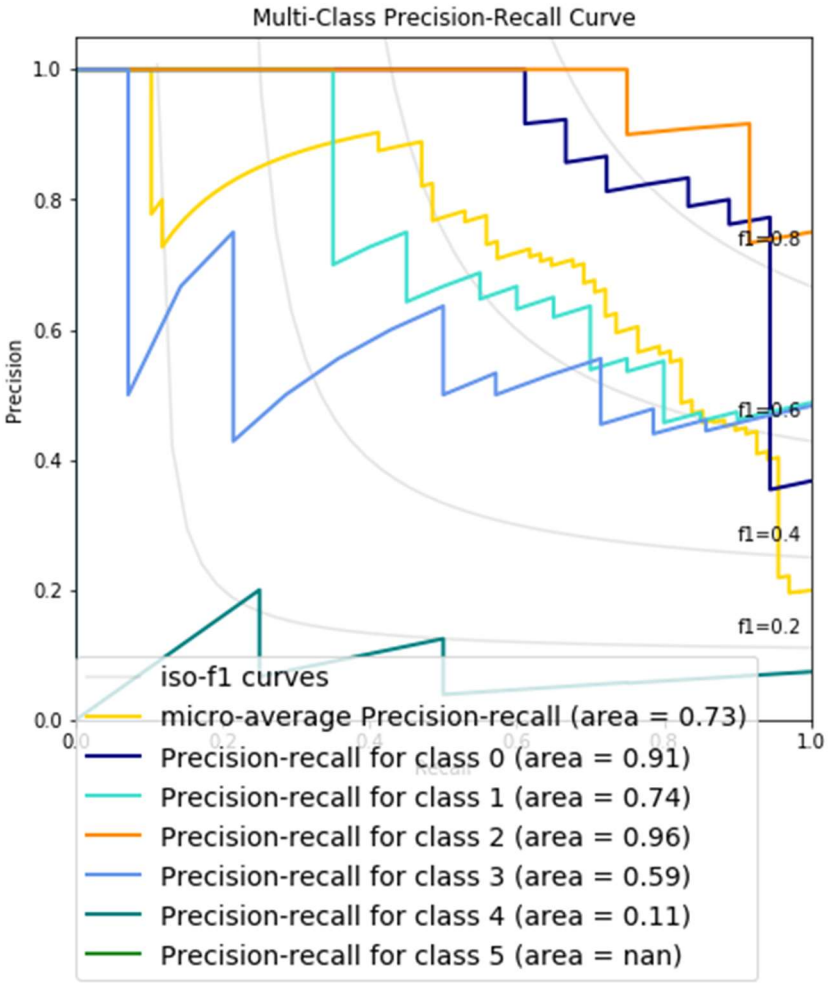
Anexo IX: Datos Estadísticos

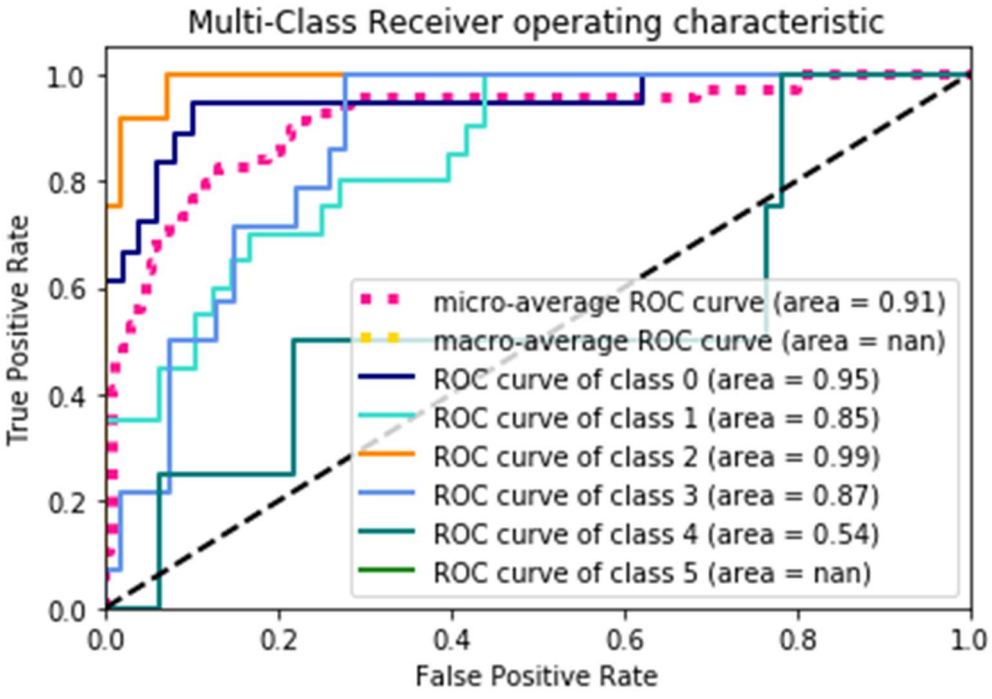
NTU DATABASE:



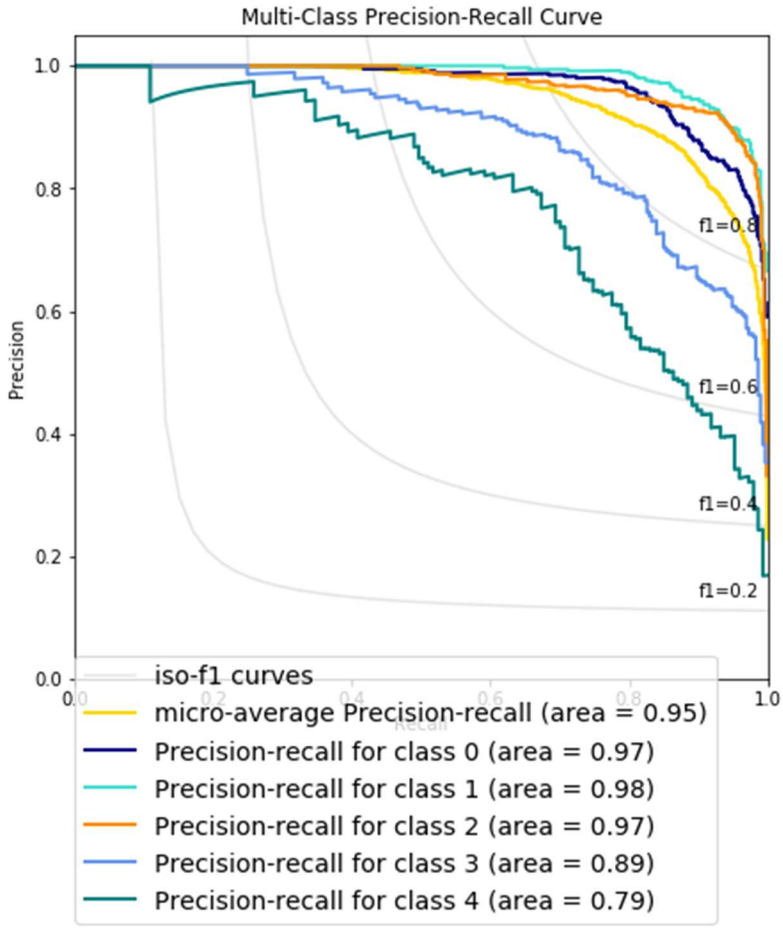


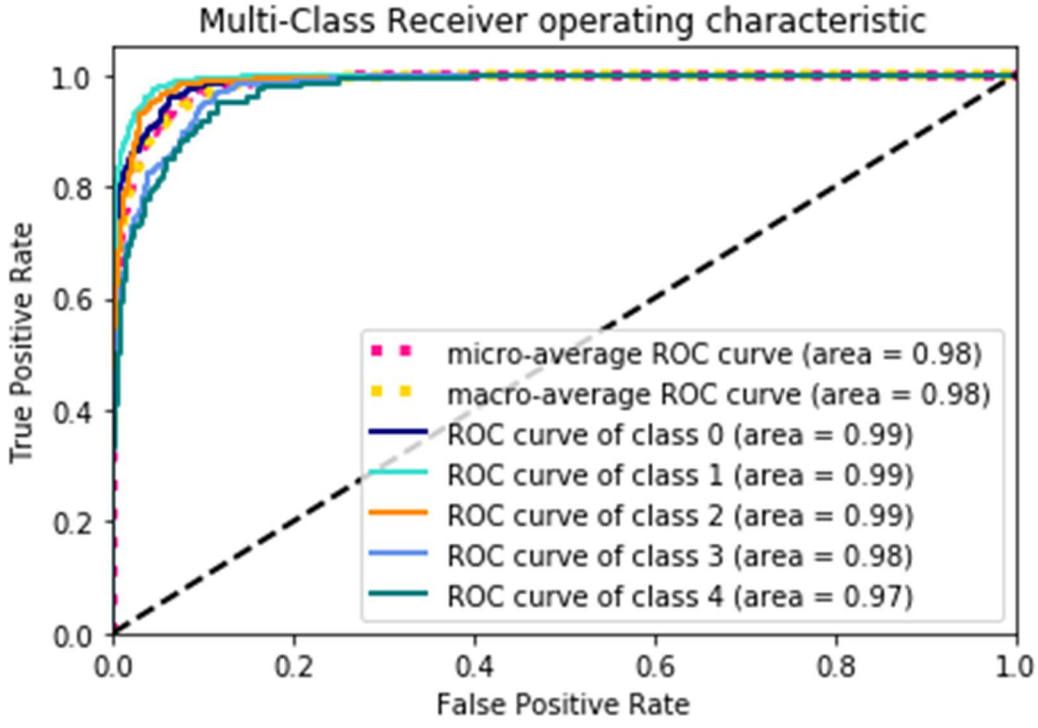
TENIS DATASET:





THETIS DATASET





Datos_estadisticos.py

```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import statistics as sta
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import confusion_matrix,average_precision_score,
precision_recall_curve
from sklearn.metrics import f1_score,plot_roc_curve, accuracy_score,
precision_score,recall_score
from sklearn.metrics import roc_auc_score,roc_curve,auc
from itertools import cycle
from scipy import interp

#Load Data
true2 =np.load('true2.npy')
true4 =np.load('true4.npy')
true5 =np.load('true5.npy')

pred2 =np.load('pred2.npy')
pred4 =np.load('pred4.npy')
pred5 =np.load('pred5.npy')

#############################################################################
#                           Result Analisys
#############################################################################
def output_analisys(true, prediction):

    true_ = np.reshape(np.argmax(true,axis=1),(true.shape[0],1))
    prediction_=np.reshape(np.argmax(prediction,axis=1),(prediction.shape[0],1))
    #Accuracy
    acc =  accuracy_score(true_, prediction_, normalize = True)
    #Precision
    prec =  precision_score(true_, prediction_, average=None)
    avg_prec = precision_score(true_, prediction_, average='micro')
    #Recall
    rec =  recall_score(true_, prediction_, average=None)
    avg_rec =  recall_score(true_, prediction_, average='micro')

    #Precision-recall Curve
    PrecisionRecallCurve(true, prediction)

    #ROC Curve
    ROCCurve(true,prediction)

    return acc, avg_prec,avg_rec,prec,rec
#############################################################################
def PrecisionRecallCurve(true,prediction):
    #Precision-Recall Values
    rec = dict()
    prec = dict()
    avg_prec = dict()
    for i in range(true.shape[1]):
        prec[i], rec[i], _ = precision_recall_curve(true[:, i], prediction[:, i])
        avg_prec[i] = average_precision_score(true[:, i], prediction[:, i])

    prec["micro"], rec["micro"], _ = precision_recall_curve(true.ravel(),
    prediction.ravel())
```

```
    avg_prec["micro"] = average_precision_score(true, prediction,average="micro")

#Plot de precision recall curve
    colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue',
'teal','forestgreen', 'red', 'blueviolet'])

    plt.figure(figsize=(7, 8))
    f_scores = np.linspace(0.2, 0.8, num=4)
    lines = []
    labels = []
    for f_score in f_scores:
        x = np.linspace(0.01, 1)
        y = f_score * x / (2 * x - f_score)
        l, = plt.plot(x[y >= 0], y[y >= 0], color='gray', alpha=0.2)
        plt.annotate('f1={0:0.1f}'.format(f_score), xy=(0.9, y[45] + 0.02))

    lines.append(l)
    labels.append('iso-f1 curves')
    l, = plt.plot(rec["micro"], prec["micro"], color='gold', lw=2)
    lines.append(l)
    labels.append('micro-average Precision-recall (area = {0:0.2f})'
              ''.format(avg_prec["micro"]))

    for i, color in zip(range(true.shape[1]), colors):
        l, = plt.plot(rec[i], prec[i], color=color, lw=2)
        lines.append(l)
        labels.append('Precision-recall for class {0} (area = {1:0.2f})'
                  ''.format(i, avg_prec[i]))

    fig = plt.gcf()
    fig.subplots_adjust(bottom=0.25)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Multi-Class Precision-Recall Curve')
    plt.legend(lines, labels, loc=(0, -.38), prop=dict(size=14))


    plt.show()
##############################################################################

def ROCCurve(true,prediction):

    lw=2
    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(true.shape[1]):
        fpr[i], tpr[i], _ = roc_curve(true[:, i], prediction[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(true.ravel(), prediction.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(true.shape[1])]))
```

```
    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(true.shape[1]):
        mean_tpr += interp(all_fpr, fpr[i], tpr[i])

    # Finally average it and compute AUC
    mean_tpr /= true.shape[1]

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    # Plot all ROC curves
    plt.figure()
    plt.plot(fpr["micro"], tpr["micro"],
             label='micro-average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["micro"]),
             color='deeppink', linestyle=':', linewidth=4)

    plt.plot(fpr["macro"], tpr["macro"],
             label='macro-average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["macro"]),
             color='gold', linestyle=':', linewidth=4)

    colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue',
'teal','forestgreen', 'red', 'blueviolet'])
    for i, color in zip(range(true.shape[1]), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=lw,
                 label='ROC curve of class {0} (area = {1:0.2f})'
                 ''.format(i, roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=lw)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Multi-Class Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
##############################################################################################

acc2, avg_prec2, avg_rec2 ,prec2, rec2 = output_analisys(true2, pred2)
acc4, avg_prec4, avg_rec4 ,prec4, rec4 = output_analisys(true4, pred4)
acc5, avg_prec5, avg_rec5 ,prec5, rec5 = output_analisys(true5, pred5)
```

Anexo X: Procesado datos 3D

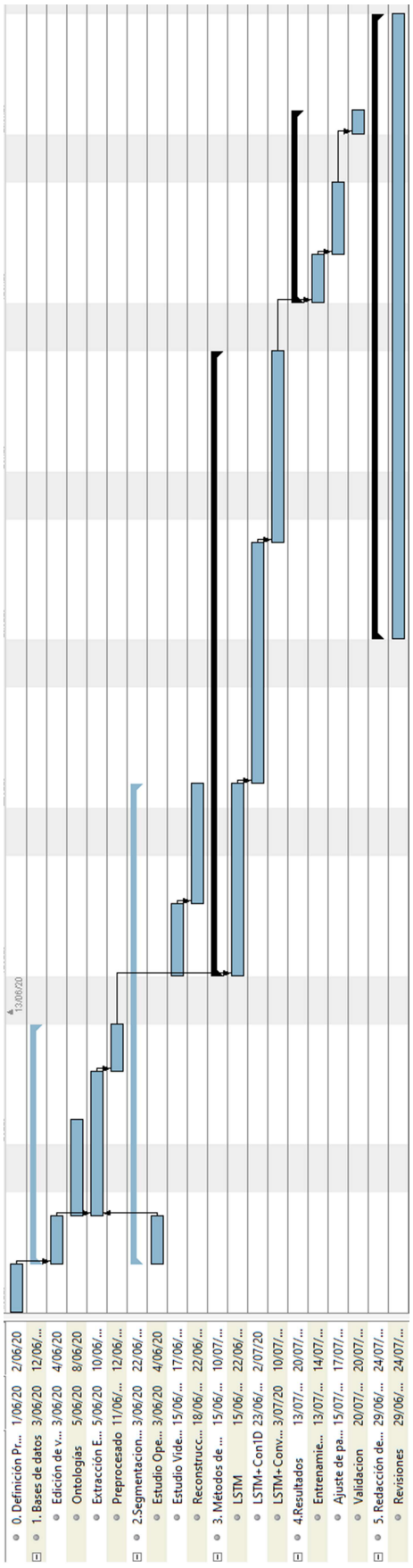
```
import numpy as np
import pandas as pd
from os import listdir
import math
import time

path = 'E:/UNIVERSIDAD/MASTER_ELECTRONICA/TFM\Datasets/Thetis/VideoPose3D'
folder = ['drive','reves','servicio','volea','remate']
data_drive = np.zeros((1,51))
data_reves = np.zeros((1,51))
data_servicio = np.zeros((1,51))
data_volea = np.zeros((1,51))
data_remate = np.zeros((1,51))

frames_drive =[]
frames_reves = []
frames_servicio = []
frames_volea = []
frames_remate = []

def resize_npy(np_data):
    size = np_data.shape[0]
    result = np.zeros((size,np_data.shape[1]*np_data.shape[2]))
    for el in range(size):
        j=0
        for i in range(np_data.shape[1]):
            result[el,j]= np_data[el,i,0]
            result[el,j+1]= np_data[el,i,1]
            result[el,j+2]= np_data[el,i,2]
            j=j+3
    return result, size
#################################################################################
#Angle Normalization
def rotate_skeleton(data):
    result = np.zeros((data.shape[0],data.shape[1]))
    for el in range(data.shape[0]):
        angle = -1.0*get_angle(data[el,33],data[el,35],data[el,42],data[el,44])  #Left
shoulder-Right shoulder
        #Rotate all points in X-Z this angle
        for x in my_range(0,data.shape[1]-3,3):
            result[el,x]= data[el,x]*math.cos(angle) - data[el,x+2]*math.sin(angle)
            result[el,x+1]=data[el,x+1]
            result[el,x+2]= data[el,x+2]*math.cos(angle) + data[el,x]*math.sin(angle)
    return result

def get_angle(x1,y1,x2,y2):
    d1 = [x2-x1,y2-y1]
    d2 = [1.0,0.0]
    angle            =            math.acos(abs(d1[0]*d2[0]+d1[1]*d2[1])            /
((math.sqrt(d1[0]*d1[0]+d1[1]*d1[1]))*(math.sqrt(d2[0]*d2[0]+d2[1]*d2[1]))))
    #angle in radians
    print(angle*180/math.pi)
    return angle

##Representation
import matplotlib.pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D

def connectpoints(x,y,z,p1,p2):
    x1, x2 = x[p1], x[p2]
    y1, y2 = y[p1], y[p2]
    z1, z2 = z[p1], z[p2]
    plt.plot([x1,x2],[y1,y2],[z1,z2],'k-')
################################################################################
def dibuja_esqueleto(x,y,z):
    connectpoints(x,y,z,0,1)
    connectpoints(x,y,z,1,2)
    connectpoints(x,y,z,2,3)
    connectpoints(x,y,z,0,4)
    connectpoints(x,y,z,4,5)
    connectpoints(x,y,z,5,6)
    connectpoints(x,y,z,0,7)
    connectpoints(x,y,z,7,8)
    connectpoints(x,y,z,8,9)
    connectpoints(x,y,z,9,10)
    connectpoints(x,y,z,11,12)
    connectpoints(x,y,z,12,13)
    connectpoints(x,y,z,11,14)
    connectpoints(x,y,z,14,15)
    connectpoints(x,y,z,15,16)


################################################################################
def my_range(start, end, step):
    while start <= end:
        yield start
        start += step
################################################################################
def dibuja_accion(data,idx):
    fig = plt.figure()
    ax = fig.add_subplot(111,projection='3d')
    plotx=[]
    ploty=[]
    plotz=[]
    for x in my_range(0,data.shape[1]-3,3):
        plotx.append(data[idx,x])
        ploty.append(data[idx,x+1])
        plotz.append(data[idx,x+2])

    #plt.figure(1)
    ax.scatter3D(plotx,ploty,plotz,c=plotz,cmap='Blues')
    dibuja_esqueleto(plotx,ploty,plotz)

    plt.show()
    plt.close()
    #plt.plot(plotx,ploty,'b.')
################################################################################


'''DRIVE'''
for el in listdir(path+'/drive'):
    file =np.load('drive/'+el)
    aux_data,aux_frames = resize_npy(file)
    data_drive = np.append(data_drive,aux_data,axis=0)
    frames_drive.append(aux_frames)
```

```
data_drive = np.delete(data_drive,0,axis=0)
data_drive_rot = rotate_skeleton(data_drive)
sec_drive = np.array(frames_drive)
sec_drive =np.reshape(np.transpose(sec_drive),(1,sec_drive.shape[0]))


np.save('data_drive.npy',data_drive_rot)
np.save('frames_drive.npy',sec_drive)

'''REVES'''
for el in listdir(path+'/reves'):
    file =np.load('reves/'+el)
    aux_data,aux_frames = resize_npy(file)
    data_reves = np.append(data_reves,aux_data,axis=0)
    frames_reves.append(aux_frames)

data_reves = np.delete(data_reves,0,axis=0)
data_reves_rot = rotate_skeleton(data_reves)

sec_reves = np.array(frames_reves)
sec_reves =np.reshape(np.transpose(sec_reves),(1,sec_reves.shape[0]))


np.save('data_reves.npy',data_reves_rot)
np.save('frames_reves.npy',sec_reves)

'''SERVICIO'''
for el in listdir(path+'/servicio'):
    file =np.load('servicio/'+el)
    aux_data,aux_frames = resize_npy(file)
    data_servicio = np.append(data_servicio,aux_data,axis=0)
    frames_servicio.append(aux_frames)

data_servicio = np.delete(data_servicio,0,axis=0)
data_servicio_rot = rotate_skeleton(data_servicio)
sec_servicio = np.array(frames_servicio)
sec_servicio =np.reshape(np.transpose(sec_servicio),(1,sec_servicio.shape[0]))


np.save('data_servicio.npy',data_servicio_rot)
np.save('frames_servicio.npy',sec_servicio)


'''VOLEA'''
for el in listdir(path+'/volea'):
    file =np.load('volea/'+el)
    aux_data,aux_frames = resize_npy(file)
    data_volea = np.append(data_volea,aux_data,axis=0)
    frames_volea.append(aux_frames)

data_volea = np.delete(data_volea,0,axis=0)
data_volea_rot = rotate_skeleton(data_volea)
sec_volea = np.array(frames_volea)
sec_volea =np.reshape(np.transpose(sec_volea),(1,sec_volea.shape[0]))


np.save('data_volea.npy',data_volea_rot)
np.save('frames_volea.npy',sec_volea)
```

```
'''REMATE'''
for el in listdir(path+'/remate'):
    file =np.load('remate/'+el)
    aux_data,aux_frames = resize_npy(file)
    data_remate = np.append(data_remate,aux_data,axis=0)
    frames_remate.append(aux_frames)

data_remate = np.delete(data_remate,0,axis=0)
data_remate_rot = rotate_skeleton(data_remate)
sec_remate = np.array(frames_remate)
sec_remate =np.reshape(np.transpose(sec_remate),(1,sec_remate.shape[0]))


np.save('data_remate.npy',data_remate_rot)
np.save('frames_remate.npy',sec_remate)
```

Anexo XI: Cronograma



Anexo XII: Redes LSTM

De manera similar a como los humanos aprendemos a leer o escribir, donde a medida que aparece una palabra la asociamos a las anteriores para generar estructuras con sentido propio, las redes recurrentes (*RNN*) tratan de emular dicho comportamiento implementado bucles internos en el interior de la propia neurona, de manera que la información persiste en el tiempo, Figura 1.

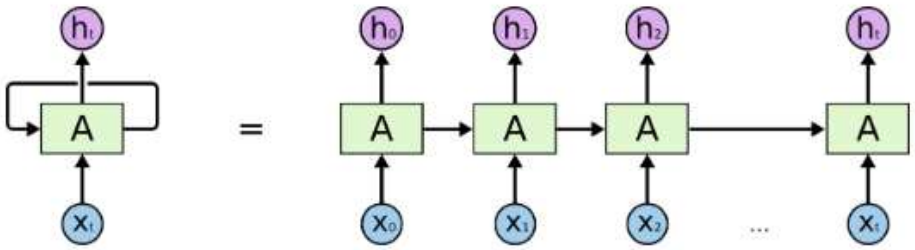


Figura 1: Neurona recurrente

El problema de estas redes viene dado por las dependencias a largo plazo, es decir, como la red debe asociar entradas pasadas con la salida cuando ha pasado un periodo de tiempo considerable. Las redes LSTM están preparadas para solucionar dicho problema ya que son capaces de recordar dichas dependencias a largo plazo.

Para ello la estructura de una celda LSTM se muestra en la Figura 2, donde ese bucle interno queda y su funcionamiento se puede desglosar en varias secciones.

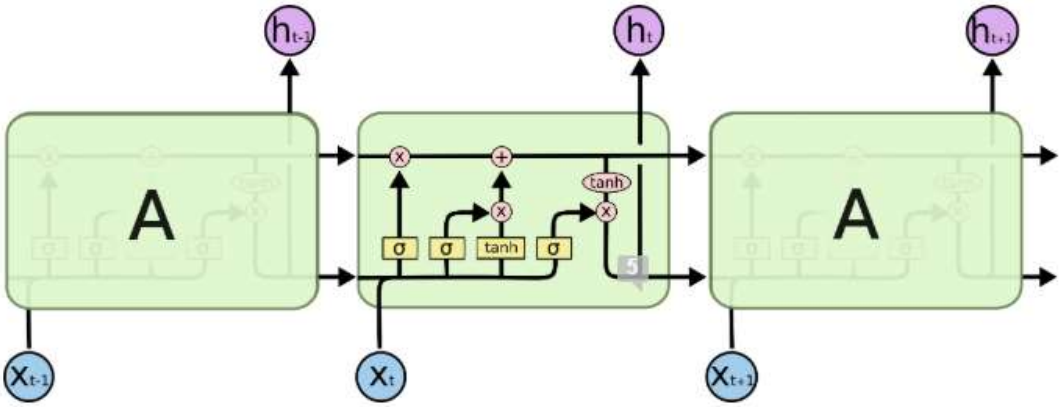


Figura 2: Estructura celda LSTM

Por un lado, tenemos el vector de estado, representado como la línea horizontal de la parte superior de la Figura 1, y es el encargado de recopilar la información sobre los cambios producidos por las entradas y transmitirlos a la siguiente celda.

Esta información se regula mediante las distintas puertas (*gates*), formadas por una capa neuronal con activación sigmoide y una multiplicación de sus elementos punto a punto. En una red LSTM convencional existen 3 de estas puertas encargadas del control del estado.

La primera es conocida como *"forget gate layer"*, y se encarga de decidir qué información es relevante y pasa al estado y cual no.

La segunda se llama "*input gate layer"* y consta de dos partes, las cuales combinadas se encargan de decidir qué información del estado se va a actualizar.

Una vez generado el estado actual a partir de estas dos puertas, se genera la salida de la celda neuronal de la misma manera que una capa neuronal convencional. A partir de una función de activación, pero en este caso aplicada al vector de estado, obtenemos la salida de la celda LSTM que se propagará a la siguiente capa.

Como vemos, el flujo de información tiene dos vías principales en estas redes, longitudinal mediante las entradas y salidas de cada capa, y transversal entre los estados de cada celda, entendida esta última como una realimentación del estado.