Trabajo Fin de Máster
Máster Universitario en Ingeniería Industrial

Clasificador de modelos de movilidad mediante Deep Learning

Autora: Manuela Grau Romero
Tutores: Sergio Toral Marín
Daniel Guitierrez Reina

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Máster
Máster Universitario en Ingeniería Industrial

Clasificador de modelos de movilidad mediante Deep Learning

Autora:
Manuela Grau Romero

Tutores:
Sergio Toral Marín
Daniel Gutierrez Reina

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Máster: Clasificador de modelos de movilidad mediante Deep Learning

Autora: Manuela Grau Romero

Tutor: Sergio Toral Marín
Daniel Gutierrez Reina

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

Agradecimientos

Antes de comenzar, me gustaría agradecer a todos los que me han apoyado durante estos años de formación académica. A mi familia y amigos en general, y especialmente a mis padres y a Jorge que han sabido animarme en todo momento.

También agradecer a mis directores del Trabajo de Fin de Máster su activa colaboración en la ejecución de este trabajo.

No olvidar a mis amigos de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla, que han sido de gran ayuda, tanto académicamente como en lo personal y han hecho más amenos los momentos más duros de la etapa estudiantil.

Manuela Grau Romero

Sevilla, 2020

Resumen

El objetivo de este proyecto es el desarrollo de una red neuronal capaz de clasificar hasta cuatro patrones de movimientos con la mejor precisión posible.

Se utilizan modelos sintetizados de movimientos, dado que no se disponen de datos reales de patrones de movimiento, generados con la herramienta Bonn Motion. Esta herramienta nos permite simular diferentes modelos de movimiento. Para este proyecto se utilizan los modelos Random waypoint, Truncated Levy Walk, Gauss Markov y Manhattan.

Se entrenan redes neuronales de tres tipos: *Fully connected*, Recurrentes y Convolucionales. Las redes neuronales convolucionales y las recurrentes tienen en cuenta la secuencia temporal del movimiento, por lo que se espera que aporten mejores resultados.

La sintonización de las redes neuronales se comienza con un modelo básico de las mismas y se hacen experimentos que nos permiten ajustarlas hasta dar con su mejor resultado, observando la pérdida y la precisión. Por otro lado, se analiza también la influencia del set de datos en los resultados. Se entrenan las redes neuronales variando, por un lado, el tiempo de las simulaciones de los patrones de movimiento y por otro, el número de patrones movimiento de entrada.

Para el desarrollo de dichas redes neuronales, se utiliza Python 3.7 y las librerías Tensor Flow y Keras. Salvo para las redes neuronales recurrentes más complejas, se trabaja con la CPU del ordenador. Para el resto, se trabaja con la GPU. La GPU permite realizar cálculos matriciales a gran velocidad, lo que reduce el tiempo de entrenamiento.

Con los resultados obtenidos, las conclusiones principales son las siguientes:

- Entrenando redes recurrentes y convolucionales se obtienen mejores resultados que con las redes *fully connected*, aunque estas últimas requieren menor coste computacional (el tiempo de entrenamiento es considerablemente menor).
- Las redes neuronales recurrentes presentan mejores resultados que las convolucionales para patrones de movimiento de corta duración.

Abstract

The aim of this project is to develop a neural network capable of classifying up to four movement patterns with the best possible accuracy.

In this case, since no real movement pattern data is available, synthesized movement models, generated with the Bonn Motion tool, are used. This tool allows us to simulate different movement models. For this project, Random waypoint, Truncated Levy Walk, Gauss Markov and Manhattan models are used.

Three types of neural networks are trained: *Fully connected*, Recurrent and Convolutional. Convolutional and recurrent neural networks take into account the time sequence of movement, so they are expected to provide better results.

The tuning of the neuronal networks starts with a basic model of them and experiments are made that allow us to adjust them until we find the best result, observing the loss and precision. On the other hand, the influence of the data set on the results is also analysed. The neural networks are trained by varying, on the one hand, the time of the movement pattern simulations and, on the other hand, the number of input movement patterns.

Python 3.7 and the Tensor Flow and Keras libraries are used for the development of these neural networks. Except for the more complex recurrent neural networks, we work with the computer CPU. For the rest, we work with the GPU. The GPU enables high-speed matrix computing, which reduces training time.

With the results obtained, the main conclusions are the following:

- Training recurrent and convolutional networks provides better results than *fully connected* networks, although the latter require less computational cost (training time is considerably less).

- Recurrent neural networks show better results than convolutional networks for short duration movement patterns.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

SW	Software
HW	Hardware
RNN	Red Neuronal Recurrente
FCNN	Red Neuronal *Fully connected* (Densa)
CNN	Red Neuronal Convolucional
FC	Función de coste
TFM	Trabajo Fin de Máster
CPU	Unidad Central de Procesamiento
GPU	Unidad Gráfica de Procesamiento
relu	Unidad Lineal Rectificada

1 INTRODUCCIÓN

Actualmente, con el uso generalizado de dispositivos inteligentes y el reducido coste de equipos de almacenamiento, disponemos de una gran cantidad de datos. Si se tratan de manera adecuada, se pueden obtener grandes beneficios.

En este proyecto se desarrolla un clasificador de modelos de movimiento basado en una red neuronal. Las redes neuronales son modelos matemáticos que relacionan una serie de datos de entrada con unas salidas determinadas. En este caso, se introducen la posición en el tiempo (x(t), y(t)) y se obtiene el tipo de movimiento.

Las redes neuronales presentan mejores resultados ante problemas no lineales y con ruido que los modelos estadísticos convencionales. Además, con el desarrollo y la utilización de redes neuronales para resolución de todo tipo de problemas, se han creado herramientas informáticas que facilitan su desarrollo. Estos son los motivos por los que en este proyecto se utilizan redes neuronales. El desarrollo se realiza en Python 3.7 con la ayuda de las librerías Keras y Tensor Flow.

En situaciones normales, el movimiento de los cuerpos no es aleatorio, sino que suele seguir un patrón, repetitivo o no, pero con unas características que suelen ser comunes para una serie de objetos o situaciones con características similares. En el caso de las personas, el movimiento que realiza una persona que va a trabajar, generalmente no es el mismo que el que realiza una persona que está de paseo. También se puede diferenciar según el movimiento si una persona camina sola o con un grupo de personas. Según el movimiento, se puede identificar también el tipo de móvil, si es un coche, una persona o un avión. Estos patrones de movimientos pueden analizarse como series temporales para resolver diferentes tipos de necesidades.

El objetivo de este proyecto es hacer un clasificador de patrones de movimientos, desarrollar un sistema que sea capaz de distinguir ciertos patrones de movimientos frente a otros. En este caso, se desarrolla un clasificador de modelos de movimiento mediante redes neuronales.

Para trabajar con herramientas de Machine Learning, es necesario contar con datos anotados. Existen bases de datos públicas, aunque las que se encuentran gratuitas están relacionadas con temas ambientales, empresariales u organismos públicos, como el registro histórico meteorológico, la evolución de empresas en bolsa o la tasa de natalidad. No se han encontrado bases de datos públicas que registen datos de movimientos. Por esto, en este proyecto, se utilizan modelos de movimiento sintetizado, modelos de movimiento generados con la herramienta Bonn Motion.

El objetivo de este proyecto es desarrollar una red neuronal que aporte buenos resultados en la clasificación de los modelos de movimiento. Para ello se entrenan tres tipos distintos de redes neuronales: *Fully connected*, Convolucionales y Recurrentes. Se hacen ensayos variando su morfología, comparando los resultados y encontrando una solución de compromiso entre la red que mejores resultados aporte y el coste computacional.

Dado que no se dispone de datos reales de patrones de movimiento, se utilizan modelos sintetizados de movimientos, generados con la herramienta Bonn Motion. Esta herramienta nos permite simular diferentes modelos de movimiento. Para este proyecto se utilizan los modelos Random waypoint, Truncated Levy Walk, Gauss Markov y Manhattan.

Con la propagación del COVID-19 estamos viviendo una de las crisis sanitarias más importantes del siglo. Actualmente todo el mundo dispone de dispositivos inteligentes con GPS que permite saber la localización histórica del individuo. Pues bien, con ayuda de esta red neuronal, si dispusiéramos de datos históricos de localizaciones de pacientes, podría predecirse si una persona se ha podido contagiar o no, en función de los lugares en los que ha estado. Esto ayudaría a las personas a autoaislarse y extremar las condiciones de higiene cuando aún no se tienen síntomas, evitando la propagación del virus.

Este clasificador puede servir también en comercios, donde las cámaras detectaran los movimientos de las personas identificando potenciales clientes o ladrones. De la misma manera, en el ámbito de la seguridad, ya sea en la calle o en las viviendas, donde se pudiera identificar personas sospechosas, potenciales ladrones o vándalos.

En el ámbito de las redes Ad Hoc, donde cada nodo forma parte del canal de comunicación. Si se clasifican los movimientos, se podrían descartar nodos para canales de comunicación concreto facilitandos el encuentro de la

ruta.

En la seguidad automovilística, podrían detectarse los movimientos que el conductor realiza con el coche, pudiéndose identificar la indisposición del conductor para la conducción.

En el área ambiental, clasificando los movimientos de las aves migratorias se pueden detectar cambios en los ecosistemas, zonas de contaminación, secado de lagunas, cambios climáticos u otros riesgos ambientales. Estos datos son interesantes también para adelantarse a problemas en la agricultura.

2 GENERADOR DE MODELOS DE MOVILIDAD

En este proyecto no se dispone de datos reales de patrones de movimiento, por lo que se utiliza una herramienta para simularlos y tener los datos de entrada para el entrenamiento de la red. En este caso se utiliza Bonn Motion.

Bonn Motion es un software desarrollado en Java por las universidades de Bonn y Osnabrück y la Colorado Mines School en Estados Unidos. Esta aplicación genera modelos de movimiento con las características que el usuario determine.

2.1 Instalación de Bonn Motion

La versión de Bonn Motion que se utiliza es la 3.0.1.

En primer lugar es necesario instalarse un kit de desarrollo de java (JDK). Para este proyecto, se utilizó la versión 1.8.0 _211.

A continuación, se reconfigura el archivo compile, que se encuentra en la carpeta bin, de manera que se indiquen las rutas correctas de los archivos.

En el archivo install.bat, se modifican las definiciones de las variables JAVAPATH y BONNMOTION, indicando la carpeta raíz del kit de desarrollo java y la carpeta raíz de Bonn Motion respectivamente.

Seguidamente, se ejecuta el script install.bat.

En la carpeta bin se habrá generado el archivo bm, con el que se van a compilar los modelos de movimiento.

2.2 Modelos de movimiento

- **Boundless (Sin límites):** Se convierte el área de simulación rectangular 2D en un área de simulación en forma de toro, donde no existen límites.
- ***Chain scenario*** **(escenario encadenado):** Concatena diferentes modelos simulados con la herramienta.
- **Columna:** Es un modelo de movilidad de grupos. En el que el grupo se mueve en una dirección de manera recta.
- **Área de desastre:** Es un modelo utilizado para simular situaciones de emergencia, en los que existen grupos de personas que siguen el movimiento de un líder. También se puede simular el movimiento de transporte de heridos y otros vehículos.
- **Gauss-Markov:** Simula movimientos correlados. Tanto en posición como en velocidad no hay cambios bruscos.
- **Manhattan Grid:** Es un movimiento en cuadrículas. Sus giros son de 90º. Simulan el movimiento de una persona en una ciudad moderna, donde todas las calles son rectas.
- **Basados en mapa:** Simula rutas óptimas en un mapa. Para poder simularlo se requiere un archivo OSM y la dirección URL de un servidor que ejecute una instancia de Open Source Routing.
- **Comunidad nómada:** Un conjunto de nodos se mueven juntos de un lugar a otro. Todos los nodos se mueven de un lugar a otro y cada uno vagarían por cada lugar en particular de forma individual.
- **Probabilístico de Caminata Aleatoria:** Utiliza un conjunto de probabilidades para determinar la siguiente posición de un nodo.
- **Persecución:** Representa nodos que rastrean un único nodo objetivo.

- **Dirección aleatoria:** El nodo se mueve dentro del área de movimiento y cambia de dirección cuando se topa con el borde. El cambio de dirección es aleatorio.
- **Caminata aleatoria:** El nodo se mueve desde el origen al destino con velocidades y direcciones aleatorias. Si topa con un límite del área, el nodo vuelve a entrar con un ángulo de entrada determinado.
- **Waypoint aleatorio:** Es un modelo aleatorio de movimiento de usuarios móviles. Cada vez que elige un nuevo waypoint, toma una velocidad, dirección aleatoria. Permite movimientos en tres dimensiones.
- **Grupo de puntos de referencia ("RPGM"):** Un grupo de nodos sigue a un líder. En este modelo se implementa también la posibilidad de pausa, de manera que los nodos harían la pausa cuando el líder la hiciera.
- **SMOOTH:** Modelo de movilidad que es a la vez realista y sencillo. Crea trazas que coinciden con los movimientos humanos. Los movimientos generados pueden predecirse, ya que son también influenced por sus movimientos anteriores.
- **Estático:** Por defecto, los nodos están distribuidos de manera homogénea.
- **Estático con deriva:** El usuario tiene que introducir un archivo con las posiciones iniciales de los nodos. Las posiciones se desplazan a lo largo de intervalos ajustables.
- **Waypoint aleatorio de estado estacionario:** Utiliza el modelo de waypoints aleatorios, en el que un nodo elige un punto aleatorio en el área de simulación y una velocidad aleatoria y luego viaja a ese punto a la velocidad elegida.
- **Modelo Táctico de Movilidad Interior ("TIMM"):** Simula movimientos en el interior de edificios.
- **Truncated Lévy Walk**: Simula movimientos de corta distancia, como los que una persona realiza generalmente. Utiliza un modelo de velocidad específico, combinado con un modelo de caminata aleatoria simple en el que un caminante sigue una secuencia de pasos.

Los modelos de movimiento que se utilizan para entrenar la red neuronal en este proyecto son: Waypoint aleatorio, Manhattan, Truncated Levy Walk y Gauss Markov. Se utilizan estos modelos porque son los que más se utilizan para la simulación de movimientos de personas.

2.2.1 Generación de modelos de movimiento

Una vez instalado el software, para generar los modelos de movimientos se llama al script bm seguido de una serie de parámetros y comandos que se explican a continuación.

El número de nodos, la duración de la simulación, el tiempo inicial de simulación que no se almacena en la salida, la definición del archivo y las medidas del área de movimiento son parámetros comunes para todos los modelos de movimiento. Se llaman de la siguiente manera:

- -n seguido del número de nodos.
- -d seguido de la duración de la simulación en segundos.
- -i seguido del tiempo de simulación inicial que no se guardará en la salida.
- -f seguido del nombre del archivo.
- -x seguido del número de metros en el eje X del plano.
- -y seguido del número de metros en el eje Y del plano.

A continucación se especifican los comandos para los modelos de movimiento utilizados en este proyecto.

2.2.1.1 Manhattan

Además de los parámetros comunes se definen una serie de parámetros especiífico para este tipo de movimiento. Se definen de la siguiente manera:

- -u seguido del número de cuadrículas que van a haber en el eje X, número de columnas.
- -v seguido del número de cuadrículas que van a haber en el eje Y, número de filas.
- -c seguido de la probabilidad de cambio. Probabilidad con la que cambiará el sentido en el próximo punto.
- -e seguido de la velocidad mínima.
- -m seguido de la velocidad máxima
- -o seguido de la pausa máxima. En el caso de que el punto siguiente sea el mismo, este parámetro restringe el número máximo de veces que el nodo puede estar en la misma posición.
- -p seguido de la probabilidad de pausa. Probabilidad con la que el punto siguiente del nodo sea el mismo en el que se encuentra.
- -q seguido de la distancia de actualización
- -s seguido de la desviación estándar de la velocidad
- -t seguido de la probabilidad de giro.

A continuación, se muestra la línea de código que se ejecuta para generar el modelo con un nodo, una duración de tres mil seiscientos segundos, en una superficie de 1000 x 1000, dividido en cinco columnas y siete filas, con una probabilidad de cambio del 35%, velocidad mínima de 0 y máxima de 1.02m/s. La desviación estándar de la velocidad es de 2.01 m/s. La probabilidad de pausa es del 20% y se establece una pausa máxima de 5.5 segundos. La distancia de actualización es de 1000 metros y la probabilidad de giro es del 25%.

Los archivos que se generan toman el nombre de *manhattan*, tal como se indica al principio de la línea.

```
 bm -f manhattan ManhattanGrid -n 1 -d 3600 -x 1000 -y 1000 -u 5 -v 7 -
c 0.35 -e 0 -m 1.02 -s 2.01 -p 0.2 -o 5.5 -q 1000 -t 0.25
```

2.2.1.2 Gauss Markov

Además de los parámetros comunes se definen una serie de parámetros especiífico para este tipo de movimiento. Se definen de la siguiente manera:

- -a seguido del módulo de la velocidad media.
- -w seguido del parámetro de memoria del patrón. Toma valores entre 0 y 1.
- -q seguido de la frecuencia de actualización
- -s seguido de la desviación estándar de la velocidad

A continuación, se muestra la línea de código que se ejecuta para generar el modelo con un nodo, una duración de tres mil seiscientos segundos en un área de 1000 x 1000, con una velocidad de 4.05 m/s con una desciación estándar de 8 m/s. Si la velocidad se definiera muy elevada, se multiplicaría por un escalar que la redujera a una velocidad máxima. Un parámetro de memoria de 0.78 con una frecuencia de actualización de 20Hz.

Los archivos que se generan toman el nombre de *modeloGauss*, tal como se indica al principio de la línea.

```
 bm -f modeloGauss OriginalGaussMarkov -d 3600 -x 1000 -y 1000 -n 1 -a
4.05 -w 0.78 -q 20 -s 8
```

2.2.1.3 Random Waypoint

Además de los parámetros comunes se definen una serie de parámetros especiífico para este tipo de movimiento. Se definen de la siguiente manera:

- -h seguido de la velocidad máxima.
- -l especifica la variación de la pausa.

- -p especifica la variación de la velocidad.

A continuación, se muestra la línea de código que se ejecuta para generar el modelo con un nodo, una duración de tres mil seiscientos segundos en un área de 1000 x 1000 m, con una velocidad máxima de 2.3 m/s, con una variación de 4.2 m/s y una variación de la pausa de 0s.

Los archivos que se generan toman el nombre de *modeloRandom*, tal como se indica al principio de la línea.

```
 bm -f modeloRandom RandomWaypoint -d 3600 -x 1000 -y 1000 -n 1 -h 2.3
-l 0 -p 4.2
```

2.2.1.4 Truncated Levy Walk

Este modelo de movimiento es una generalización del modelo de movimiento aleatorio, en el que la longitud del paso durante la caminata (también llamada vuelo) sigue una distribución probabilística de "cola paseada".

Además de los parámetros comunes se definen una serie de parámetros específicos para este tipo de movimiento. Se definen de la siguiente manera:

- -a coeficiente de la longitud de vuelo.
- -b coeficiente de la pausa de vuelo.
- -s distancia máxima del vuelo.
- -m número mínimo de pasos.
- -M número máximo de pasos.
- -l ángulo mínimo de giro de un vuelo a otro.
- -L ángulo máximo de giro de un vuelo a otro.
- -c factor de escala de la longitud de vuelo.

A continuación se muestra la línea de código que se ejecuta para generar el modelo con un nodo, una duración de tres mil seiscientos segundos en un área de 1000 x 1000 m, con un coeficiente de la longitud de vuelo de 1.02, un coeficiente de pausa de 1.84, un tamaño máximo del vuelo de 1000m, con un mínimo de 0 pasos y un máximo de 1200..

```
 bm -f modeloTLW TLW -d 3600 -x 1000 -y 1000 -n 1 -a 1.02 -b 1.84 -s
1000 -m 0 -M 1200 -l 10 -L 100 -c 2
```

Para generar el modelo, se llama al script bm, seguido del nombre del archivo, el nombre de la función del modelo de movimiento, el número de nodos, la duración de la simulación

Se realizan 4000 muestras de cada tipo de movimiento. Se acotan todos los movimientos en un área de 1000 x 1000 y en un tiempo de 3600 segundos.

En las figuras 1, 2, 3 y 4 se pueden ver los movimientos Manhattan, Random Waypoint, Truncated Levy Walk y Gauss Markov respectivamente.

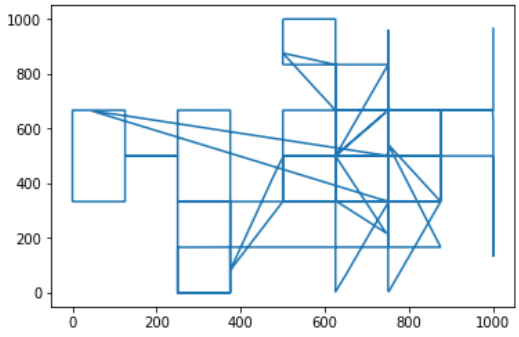

Figura 1. Movimiento Manhattan

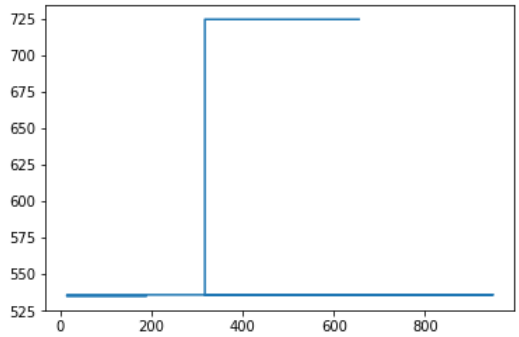

Figura 2. Movimiento Random Waypoint

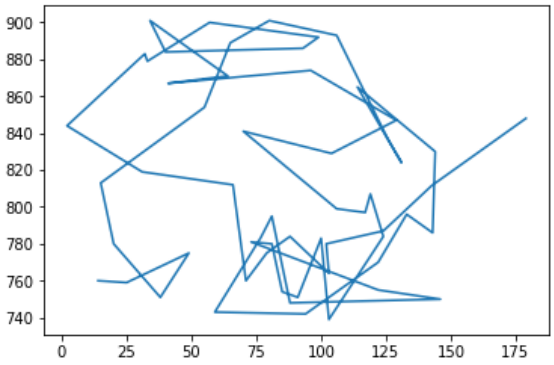

Figura 3. Movimiento Truncated Levy Walk

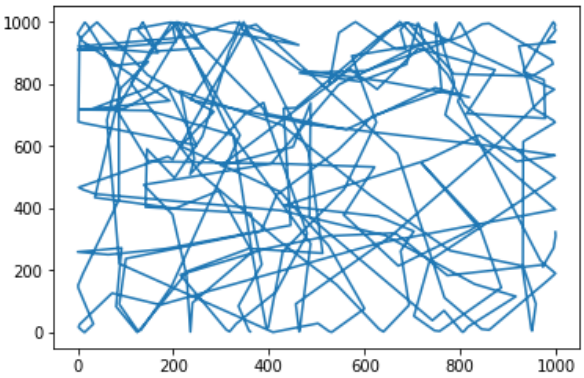

Figura 4. Movimiento Gauss Markov

3 REDES NEURONALES

3.1 Conceptos básicos

En este apartado se va a exponer brevemente los conceptos fundamentales básicos para la comprensión del funcionamiento de las redes neuronales, desde su nivel más bajo.

3.1.1 La neurona

Una neurona no es más que una función matemática que toma como entrada diferentes estímulos generando una salida. Es una suma ponderada de sus entradas, sesgada y distorsionada por la función de activación. En la figura 5 podemos ver un esquema de la neurona.

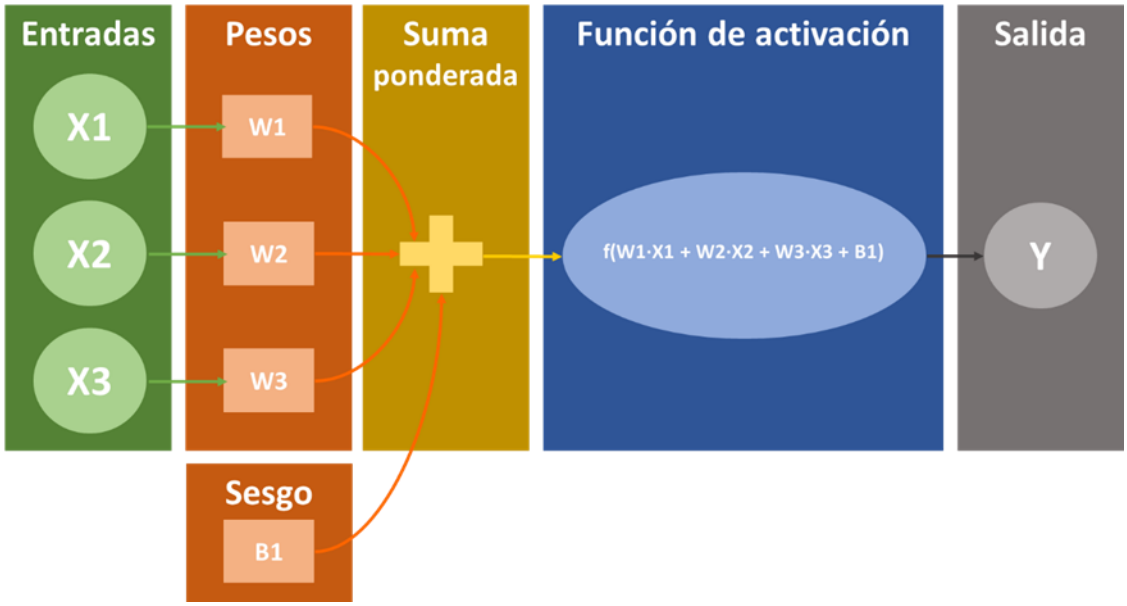

Figura 5. Estructura de la neurona

3.1.2 La función de activación

Es la que determina el valor de la salida una vez se ha calculado la suma ponderada. Las funciones de activación más comunes son:

- **Escalonada**: Transforma los valores a 0 ó 1. No favorece el aprendizaje, por lo que no se suele usar.
- **Sigmoide**: La función transforma los valores a una escala de entre 0 y 1. Los valores muy bajos tienden asintóticamente a 0 y los muy altos a 1. Tiene una lenta converjencia. Tiene buen rendimiento en la última capa.
- **Tangente hiperbólica:** Transforma los valores de entrada a una escala entre -1 y 1. Se utiliza para decidir entre una opción o la contraria. Tiene una lenta convergencia. Funciona bien en redes recurrentes.
- **Unidad rectificada lineal (relu)**: Anula los valores negarivos y saca a los positivos sin modificar. Funciona bien en redes convolucionales.
- **Softmax**: Transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas de 1. Se utiliza para normalizar multiclases. Tiene buen rendimiento en las últimas capas.

3.1.3 Función de coste

Es la función que calcula el error a partir de la comparación de la salida obtenida en la red con la esperada.

Las más importantes son:

- **Error cuadrático medio:** Calcula la media de los errores al cuadrado. Se utiliza para cuando la salida puede tomar un rango de números reales.
- **Entropía cruzada binaria:** Se utiliza para calcular errores de salidas binarias.
- **Entropía cruzada categórica:** Se utiliza para calcular errores de salidas discretas.

3.1.4 El algoritmo de back-propagation

El algoritmo de back-propagation es un algoritmo de parendizaje supervisado, es decir, necesita comparar el resultado obtenido en la red con el resultado real que debería tomar. Por esto, durante el entrenamiento se introducen estímulos de entrada y las salidas esperadas.

Las salidas obtenidas se comparan con las salidas reales y se calcula el error mediante la función de coste. A continuación, se calculan las derivadas parciales del error con respecto a los pesos de las neuronas. Se comienza por la última capa y se va repitiendo el algoritmo en las capas anteriores. El objetivo de hacerlo de atrás hacia adelante es que el error de las capas anteriores depende del error de las posteriores.

Las derivadas parciales que aquí se calculan son utilizadas posteriormente por el algoritmo del Descenso del Gradiente para la minimización del error.

En este proyecto no se realiza manualmente este algoritmo, ya que se encuentra implementado en las librerías de tensor flow.

3.1.5 El descenso del gradiente

Es un algoritmo que consite en calcular mínimos de la función de coste.

Al iniciar el entrenamiento, los parámetros de la red toman valores aleatorios, por lo que el punto de inicio del cálculo es aleatorio. A continuación, se calcula el gradiente, es decir, la dirección en la que aumenta la pendiente y se avanza en sentido contrario. El avance se conoce como ratio de aprendizaje. Ya en el nuevo punto, Vuelve a calcular el gradiente y se continúa avanzando, hasta que el gradiente se hace cero. Esto indica que se ha llegado a un mínimo.

Este algoritmo se repite para cada una de las capas de la red neuronal.

Este algoritmo, al igual que el de back propagation, no se realizan manualmente en este proyecto, ya que para ello, disponemos de la librería Keras en la que lo que se define es el Optimizador.

3.1.6 Overfitting

El overfitting es un fenómeno que ocurre cuando sobreentrenamos nuestro modelo. Lo que ocurre es que nuestro modelo de red neuronal considera el ruido de los datos de entrenamiento como parte del algoritmo. Es decir, encuentra un modelo que se ajusta a todos los datos del entrenamiento, pero no generaliza bien para otros datos.

3.2 Tipos de redes neuronales

3.2.1 Red Neuronal Monocapa

Es la red neuronal más simple. Tiene dos capas, la de entrada y la de salida. En la de entrada se toman las entradas y se realizan los cálculos. En la de salida se tratan los resultados para disponerlos según las salidas esperadas. Se utilizan para problemas sencillos.

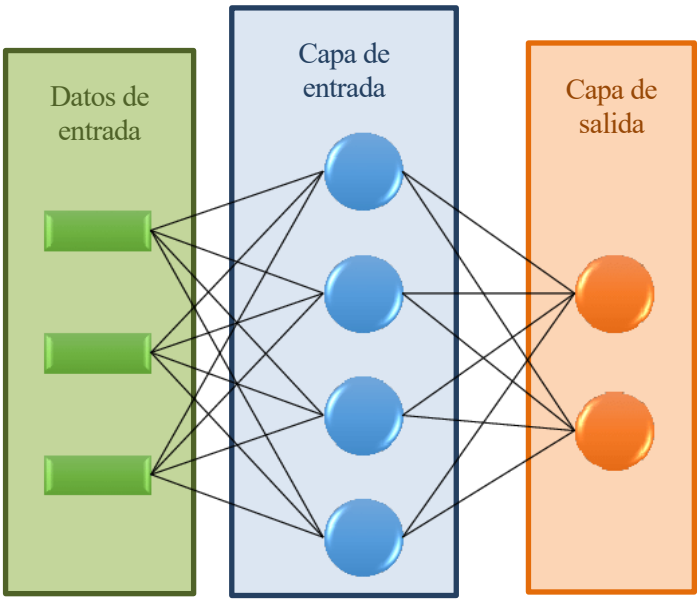

Figura 6. Estructura de la red monocapa

3.2.2 Red Neuronal *Fully connected*

Es un tipo de red neuronal multicapas. Con respecto a la anteriror tiene la ventaja de que es capaz de aprender de manera jerarquizada, donde en las primeras capas se aprenden términos más concretos y en las posteriores términos más abstractos.

En este tipo de redes, las neuronas de cada capa están conectada con todas las neuronas de la capa siguiente.

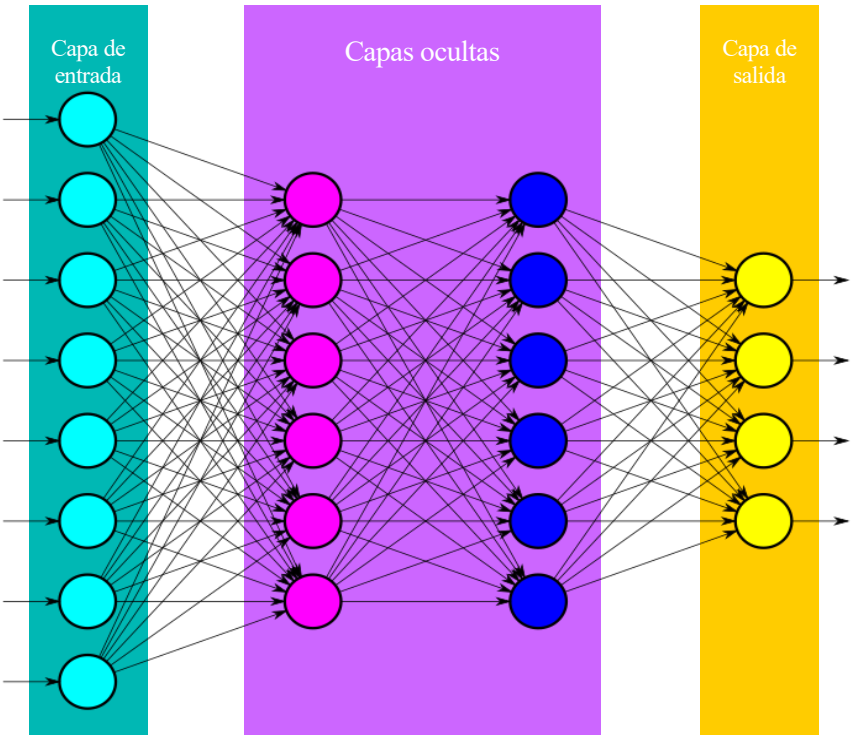

Figura 7. Estructura de la red *Fully connected*-

3.2.3 Red Neuronal Convolucional

Es una red neuronal, cuya principal función es la interpretación de imágenes. Las partes que la component son las siguientes.

1. Datos de entrada: En el caso que ocupa este proyecto, los datos de entrada son bidimiensionales, ya que son coordenadas XY. En la imagen que se muestra a continucaión, los datos de entradan tienen una profundidad de tres ya que toma como entrada una imagen, cuyos pixels están compuestos por tres colores (RGB).
2. Capa de convolución: Se barren los datos de entrada, se van combinando y se genera un dato de menor dimension XY, pero de mayor profundidad. En esta capa, hay que tener en cuenta lo siguiente:
    - El tamaño del filtro, el tamaño que tiene la ventana con la que barremos los datos de entrada.
    - El número de filtros: Si tuvieramos una imagen, los distintos filtros podrían ser para la detección de bordes, sombras, texturas, etc. Por cada capa de entrada, una vez pasado el filtro el número de filtros determinará la profundidad de la salida.
    - El paso: Cuanto avanza la ventana a lo largo de la entrada.
3. Pooling: Reduce el tamaño de los datos sin aumentar su profundidad. Similar a la capa anterior pero sin diferentes filtros.
4. Flattern: Para obtener una clasificación, necesitamos finalmente unas capas densas, que son las que realizan la clasificación. La capa convolucional saca los datos tridimensionales. Con esta capa, lo que se hace es aplanar los datos para poder introducirlos en las capas densas.
5. *Fully connected*: Al final de estas redes se añade una red neuronal *fully connected*. En esta se realiza la clasificación.

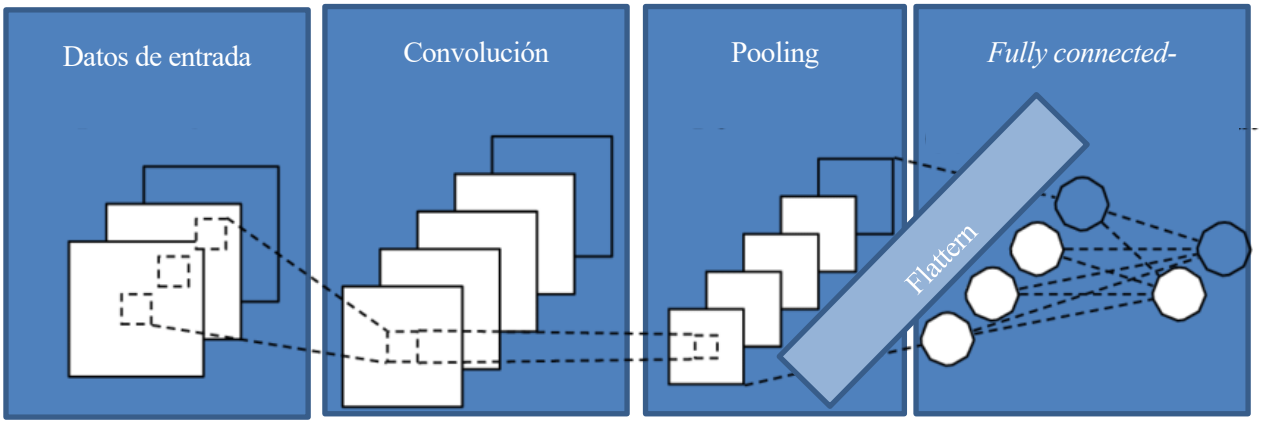

Figura 8. Estructura de la red Convolucional

Las capas convolucionales pueden ser de una dimensión (Conv1D) o de dos (Conv2D). La diferencia es que lasde una dimensión se realiza en uno de los ejes en lugar de los dos. En Keras, las capas convolucionales incluyen la convolución y la capa *pooling*.

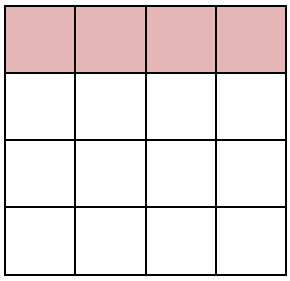

Figura 9. Convolucional 1D

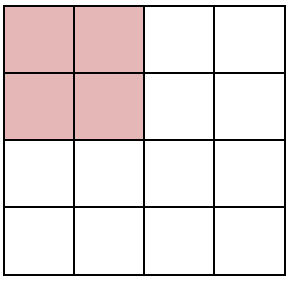

Figura 10. Convolucional 2D

3.2.4 Red Neuronal Recurrente

En este tipo de redes neuronales, la diferencia reside en que el flujo de datos no solo va en dirección entrada-salida, sino que, además, la salida de una neurona puede ser la entrada de otra neurona de su misma capa, permitiendo esto un aprendizaje temporal.

Este tipo de redes toma la entrada ordenada en el tiempo, leyendo los valores de manera secuencial. Esto permite que las muestras de entrada puedan tener distintas duraciones. Es decir, podríamos tener simulaciones de movimiento de cinco minutos o de una hora sin tener que modificar la red.

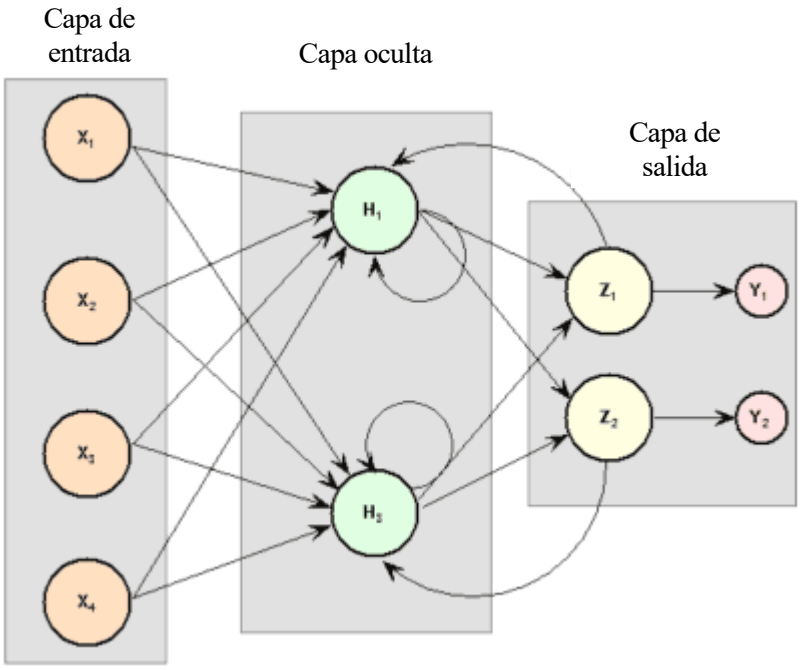

Figura 11. Estructura de la red neuronal Recurrente

Las redes recurrentes simples tienen una memoria de corto plazo, por lo que a medida que se alarga la simulación, pierden precisión. Para solventar este problema se desarrollaron las redes LSTM (Long Short Term Memory) un tipo de red neuronal recurrente diseñado específicamente para recordar información durante largos períodos de tiempo.

Todas las redes recurrentes tienen siguen una forma secuencial. En las redes recurrentes estándar, este módulo repetitivo tiene una estructura muy sencilla, en la que el módulo repetitivo solo tiene una capa con una función de activación de tangente hiperbólica. Para aclarar esto, véase la Figura 12.

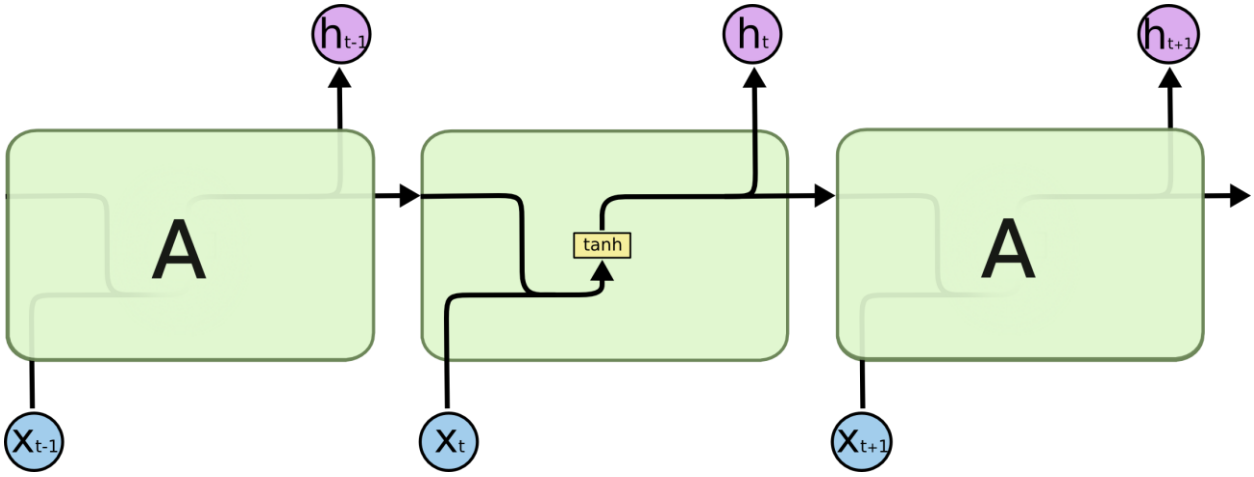

Figura 12. Módulo repetitivo de red recurrente.

Las LSTM también tienen esta estructura tipo secuencial, la diferencia recae en el módulo de repetición, que tiene una estructura más compleja. En lugar de tener una sola capa, hay cuatro que interactúan de la siguiente

manera:

- En la primera fase se decide si la información de histórica es relevante o no. De manera que, si no lo es, se elimina la información almacenada.
- En la segunda fase, se calcula la actualización del estado anterior.
- Por último, se genera la salida de la capa.

En la Figura 13 se puede ver un esquema del módulo de repetición de una red neuronal recurrente LSTM.

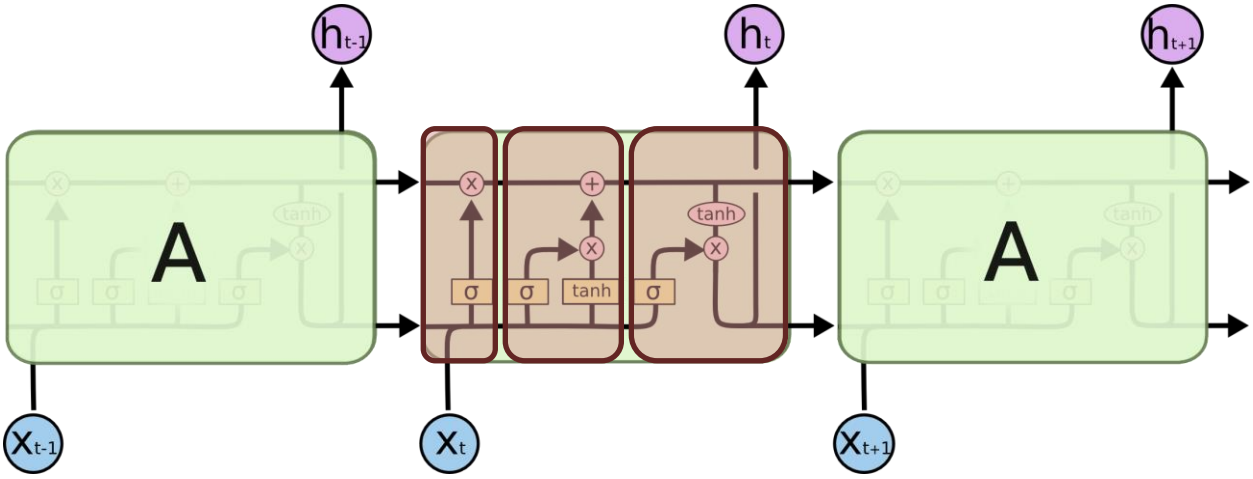

Figura 13. Módulo repetitivo de red LSTM

3.2.5 Red Neuronal de Base Radial

Son redes de tipo multicapa en las que hay una capa de entrada, una occulta y una de salida. Calculan la salida de la función en función de la distancia a un punto denominado centro. La salida es una combinación lineal de las funciones de activación radiales utilizadas por las neuronas individuales.

Las redes de base radial tienen la ventaja de que no presentan mínimos locales donde la retropropagación pueda quedarse bloqueada.

Las funciones de base radial tienen un carácter local, ya que son funciones que alcanzan un nivel cercano al máximo de su recorrido cuando la muestra de entrada está próximo al centro de la neurona. A medida que la muestra se aleja del centro, el valor de la función va tendiendo al valor mínimo de su recorrido.

3.3 Software utilizado

Este proyecto se ha realizado en Python 3.7 con la ayuda de las librerías Keras y Tensor Flow.

- **Python**: Es un lenguaje de programación interpretado que facilita la legibilidad del código. Esto permite que la escritura del código se asemeje a lo que estamos programando. Este es le principal motivo por el que python es el principal lenguaje para la programación de redes neuronales. Además, texisten numerosas librerías matemáticas y específicas para la programación de redes neuronales, que facilitan el desarrollo de las mismas.
- **Tensor Flow:** Es una librería de diferenciación automática. Se encarga de realizar todas las derivadas parciales que necesitemos para optimizar la fucnión de coste para cualquier estructura de redes neuronales que programemos. Durante el entrenamiento, se encargan de calcular los pesos de la red y optimizarlos.

Para ello, lo que hace es minimizar la función de coste (pérdidas) mediante el método del descenso del gradiente.

- **Keras:** Es una biblioteca de código abierto escrita en python. Sumado a Tensor Flow, logra abstraer el código que temenos que generar al nivel de capa. Es decir, en lugar de programar las distintas operaciones que se realizan en la neurona, basta con indicar el tipo de capa y sus parámetros.

3.4 Hardware utilizado

La GPU permite realizar cálculos matriciales a gran velocidad, lo que reduce el tiempo de entrenamiento. Esto es necesario para el entrenamiento de redes neuronales complejas. En este proyecto se utiliza una tarjeta gráfica Nvidia Gforce GTX 850 M.

4 FORMULACIÓN DEL PROBLEMA

El objetivo de este proyecto es obtener un clasificador de modelos de movimiento. Para ello, se entrenan redes neuronales *fully connected*, recurrentes y convolucionales. Se entrenan variando el número de capas y el número de neuronas de cada capa. Además, se varían los tiempos de simulación de los datos de entrada así como el número de muestras. Finalmente se realiza un análisis de los resultados obtenidos y se indica cuál es la red neuronal óptima para este problema.

En este proyecto, se clasifican cuatro movimientos: *Manhattan*, *Gauss-Markov*, *Truncated Levy Walk* y *Random Aleatory Waypoint.* Son cuatro salidas discretas posibles. Por esto mismo se utiliza como función de pérdida la entropía cruzada categórica para todos los tipos de redes ensayadas y se analizan los resultados en función de la precisión y la pérdida.

4.1 Datos de entrada

Una vez generados los modelos de movimiento, los tratamos para estructurarlos de manera que sea compatible con las entradas de las redes neuronales que vamos a entrenar.

En la figura 14 se puede ver la estructura del set de datos que vamos a introducir.

Su estructura es la siguiente:

- Tiempo de movimiento: Es el tiempo de la simulación de cada muestra de movimiento.
- Número de coordenadas: En este caso, es un movimiento bidimensional, con coordenadas X, Y.
- Número de muestras: Es el número de ejemplos de movimientos que vamos a usar para entrenar la red.

El set de datos completo que hemos obtenido de Bon Motion contiene un total de 16000 muestras (4000 de cada tipo de movimiento). Cada simulación tiene una duración de 3600 segundos en los que se indican la coordenada X e Y en cada segundo.

Figura 14. Esquema de la estructura de datos de entrada.

4.2 Tipos de redes utilizadas.

4.2.1 Red Neuronal *Fully connected*

Es el tipo de red neuronal más básico que se entrena en este proyecto.

Las capas de estas redes neuronales se llaman capas densas. Para entrenar esta red con el set de datos indicado anteriormente, antes de las capas densas introducimos una capa en Keras llamada Flattern, que lo que hace es

transformarnos los datos tridimensionales (posición X, Y en el tiempo, repetido el número de muestras) matriz de [número de muestras x tiempo de simulación x 2] en una matriz de [número de muestras x (tiempo de simulación · 2)]. Los datos de entrada, a las capas Dense tienen la siguiente forma: [número de muestras x (tiempo de simulación · 2)].

En las pruebas que se realizan, se estudia el comportamiento de la red para distintos tiempos de simulación y para distintos números de muestras.

En las capas de entrada y ocultas, se utiliza la función de activación Relu. Para la capa de salida, considerando el propósito de clasificación de este proyecto, la función Softmax.

Para este tipo de redes, se analizan los resultados considerando distintos números de capas y distinto número de neuronas para cada capa. También se hacen pruebas intercalando capas *Dropout* para mejorar el *overfitting*.

Se compila la red neuronal con la función de pérdida entropía cruzada categórica, el optimizador Adam y se toma como medida la precisión. El optimizador Adam es un optimizador que va actualizando el ratio de aprendizaje de manera automática.

Por último, ajusta la red con los datos de entrenamiento y se valida con los datos de prueba. El entrenamiento se hace con un batch size de 50 y se realizan 250 repeticiones (epochs). Para la validación se toma el mismo batch size que para el entrenamiento.

En la figua X se puede ver un diagrama de flujo de la programación de la red neuronal. En la figura se expone el caso en el que se ensaya con seis capas densas de 512, 256, 128, 64, 32 y 4 neuronas por capa.

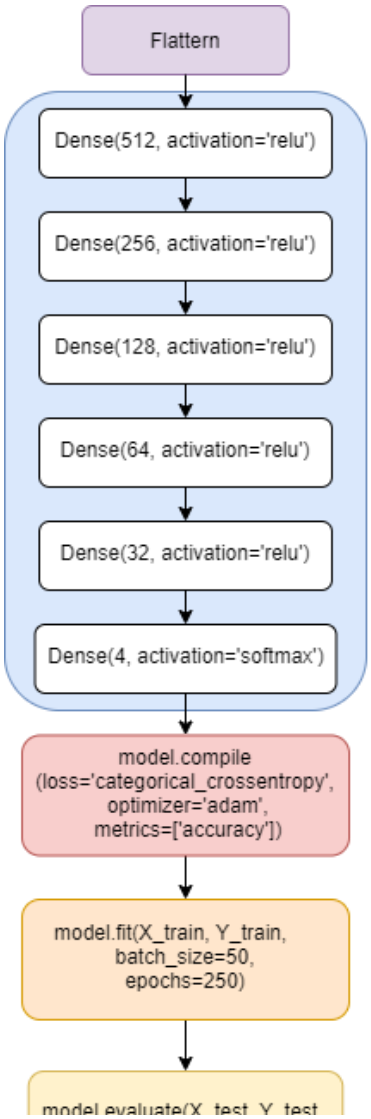

Figura 15. Esquema red neuronal *fully connected*-

4.2.2 Red Neuronal Convolucional

Cuando se estudia clasificar el movimiento con una red neuronal convolucional, se estudia como un mapa de coordenadas, en los que el tiempo no influye, pero sí la condición de coordenada X o Y.

Los datos de entrada tienen las dimensiones: [número de muestras x tiempo de simulación x 2]. En las pruebas que se realizan, se estudia el comportamiento de la red para distintos tiempos de simulación y para distintos números de muestras.

En esta red, la capa de entrada es la capa convolucional, con función de activación relu, seguida de la capa *Flattern*, para adaptar los datos a las capas Densas, cuyo fin es obtener la clasificación.

En este tipo de redes se estudia el comportamiento para distintos tamaños de filtros, con capas de 16 filtros.

Se compila la red neuronal con la función de pérdida entropía cruzada categórica, el optimizador Adam y se toma como medida la precisión. El optimizador Adam es un optimizador que va actualizando el ratio de aprendizaje de manera automática.

Por último ajusta la red con los datos de entrenamiento y se valida con los datos de prueba. El entrenamiento se hace con un batch size de 50 y se realizan 250 repeticiones (epochs). Para la validación se toma el mismo batch size que para el entrenamiento.

En la figua X se puede ver un diagrama de flujo de la programación de la red neuronal. En la figura se expone el caso en el que se ensaya con 16 filtros de tamaño 8 con simulaciones de entrada de 3600 segundos y dos variables, las posiciones X e Y.

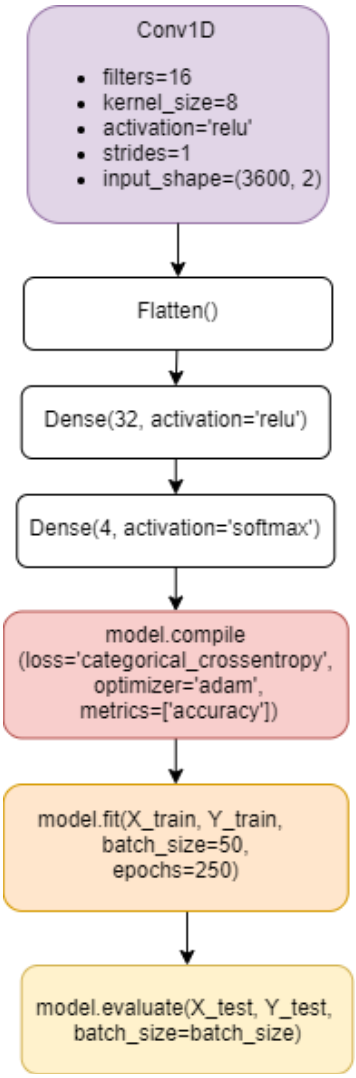

Figura 16. Esquema red neuronal convolucional

4.2.3 Red Neuronal Recurrente

En el estudio de un movimiento, es muy importante la secuencia temporal, por eso se estudia el comportamiento de esta red. Antes de realizar los ensayos, es la más prometedora.

Los datos de entrada tienen las dimensiones: [número de muestras x tiempo de simulación x 2]. En las pruebas que se realizan, se estudia el comportamiento de la red para distintos tiempos de simulación y para distintos números de muestras.

Se utilizan capas de tipo LSTM, ya que su memoria temporal es mayor que las capas simples. Presentan mejores resultados para tiempos de simulación más largos.

Las capas de entrada y ocultas son de tipo LSTM, que incluyen la función de activación tangente hiperbólica. La de salida es de tipo Densa con función de activación Softmax para convertir los resultados en una de las cuatro categorías.

Para este tipo de redes, se analizan los resultados considerando distintos números de capas y distinto número de neuronas para cada capa. También se hacen pruebas intercalando capas *Dropout* para mejorar el *overfitting*.

En la Figura 17 se puede ver el diagrama de flujo del código Python. En la capa de entrada, input_shape indica la forma de los datos de entradas, que es del tiempo de simulación x 2, donde dos es el número de variables, en este caso, posición X, Y. El dropout se añade en algunos de los ensayos, con un valor de 0.05.

Durante el entrenamiento de la red se tienen problemas debido al elevado coste computacional que conllevan. Hasta ahora, todos los modelos se han ejecutado desde la CPU. Al entrenar este tipo de redes, el tiempo estimado por *epoch* es de cerca de 10 minutos. Imposobilitando el entrenamiento de las mismas. Para reducir tiempos de entrenamiento, se installa Tensor Flow para la GPU. Hay un paquete en anaconda que se instala con todas las librerías necesarias. El paquete se llama keras-gpu.

Para poder entrenar redes neuronales desde la GPU, es necesario que la tarjeta gráfica sea Nvidia y de un modelo relativamente reciente. En este caso se ha utilizado una tarjeta Nvidia Gforce GTX 850 M.

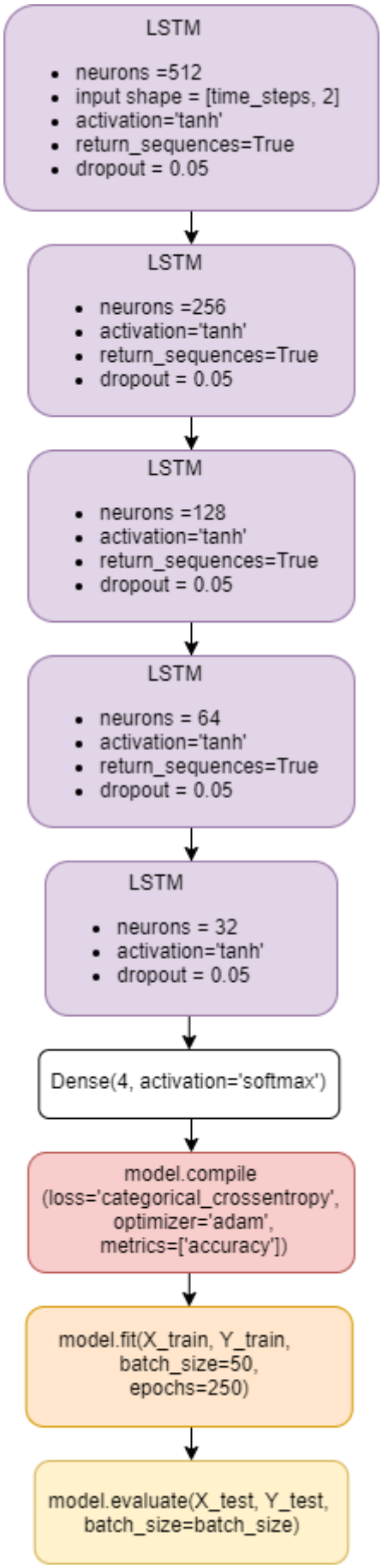

Figura 17. Esquema red neuronal recurrente

5 RESULTADOS

5.1 Red neuronal *Fully connected*

5.1.1 Red neuronal sin capa oculta.

La red neuronal está compuesta por una capa de entrada densa con función de activación relu de 32 neuronas y una capa de salida densa con función de activación softmax de 4.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

5.1.1.1 Ensayos variando el tiempo de la simulación

Se introduce un set de datos de 16000 muestras, 12000 para el entrenamiento y 4000 para la validación.

Observando de la Figura 18 a la Figura 24, podemos comprobar que para el caso de las redes *Fully connected*, los mejores resultados se obtienen para patrones de movimiento de 100 segundos de duración.

5.1.1.1.1 Simulación de 50 segundos.

- Tiempo por epoch: 0s
- Pesos: 3364.
- Resultados de entrenamiento:
  - Pérdida: 0.8510
  - Precisión: 0.6375
- Resultados de validación:
  - Pérdida: 0.8589
  - Precisión: 0.6366

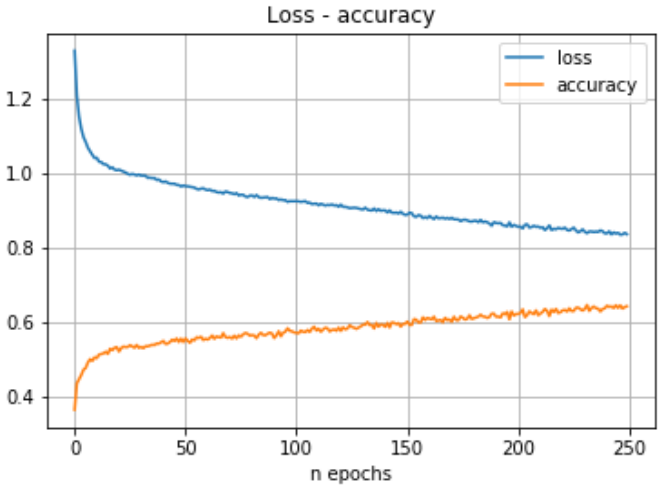

Figura 18. Red neuronal *Fully connected.* Ensayo 1

5.1.1.1.2 Simulación de 100 segundos.

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.7970
  - Precisión: 0.6858
- Resultados de validación:
  - Pérdida: 0.8270
  - Precisión: 0.6806

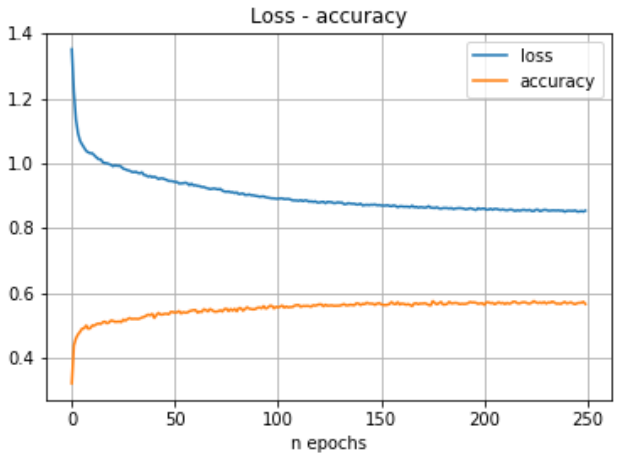

Figura 19. Red neuronal *Fully connected.* Ensayo 2

5.1.1.1.3 Simulación de 250 segundos.

- Tiempo por epoch: 0s
- Pesos: 16164
- Resultados de entrenamiento:
  - Pérdida: 0.8148
  - Precisión: 0.6776
- Resultados de validación:
  - Pérdida: 0.8706
  - Precisión: 0.6629

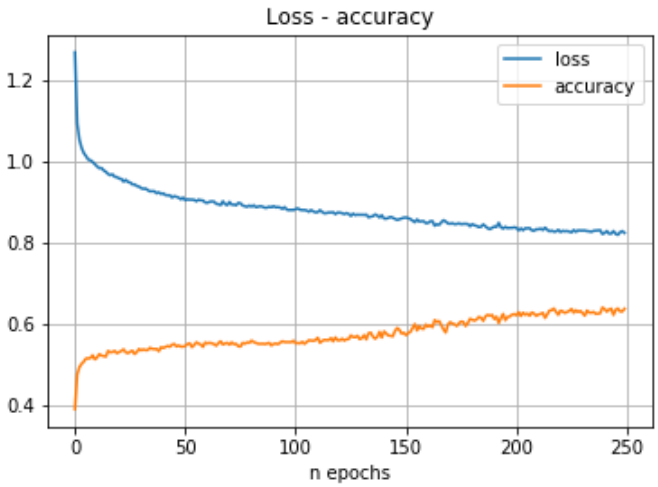

Figura 20. Red neuronal *Fully connected.* Ensayo 3

5.1.1.1.4 Simulación de 500 segundos.

- Tiempo por epoch: 1s
- Pesos: 32164
- Resultados de entrenamiento:
  - Pérdida: 0.8550
  - Precisión: 0.6553
- Resultados de validación:
  - Pérdida: 0.9161
  - Precisión: 0.6380

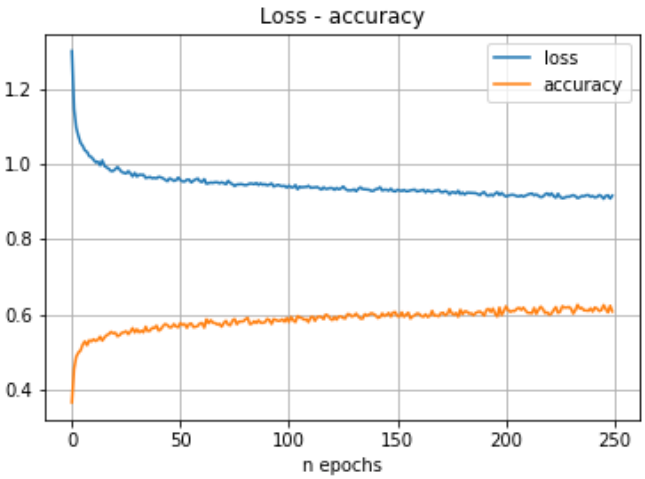

Figura 21. Red neuronal *Fully connected.* Ensayo 4

5.1.1.1.5 Simulación de 750 segundos.

- Tiempo por epoch: 1s
- Pesos: 48164
- Resultados de entrenamiento:
  - Pérdida: 0.9697
  - Precisión: 0.5785
- Resultados de validación:
  - Pérdida: 1.0048
  - Precisión: 0.5740

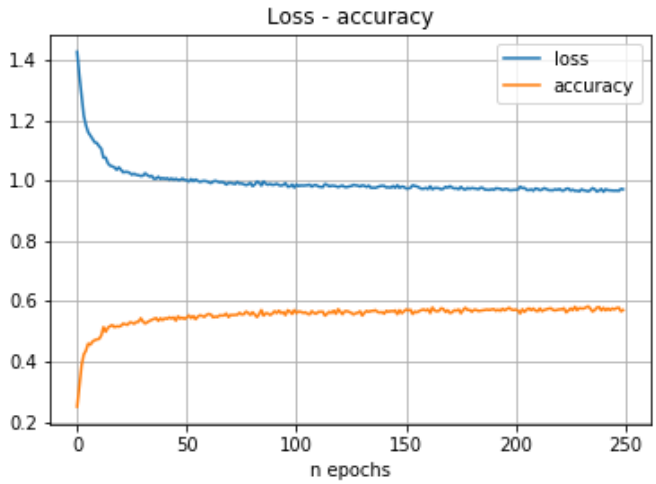

Figura 22. Red neuronal *Fully connected.* Ensayo 5

5.1.1.1.6 Simulación de 1000 segundos.

- Tiempo por epoch: 1s
- Pesos: 64164
- Resultados de entrenamiento:
  - Pérdida: 1.1399
  - Precisión: 0.4546
- Resultados de validación:
  - Pérdida: 1.1771
  - Precisión: 0.4500

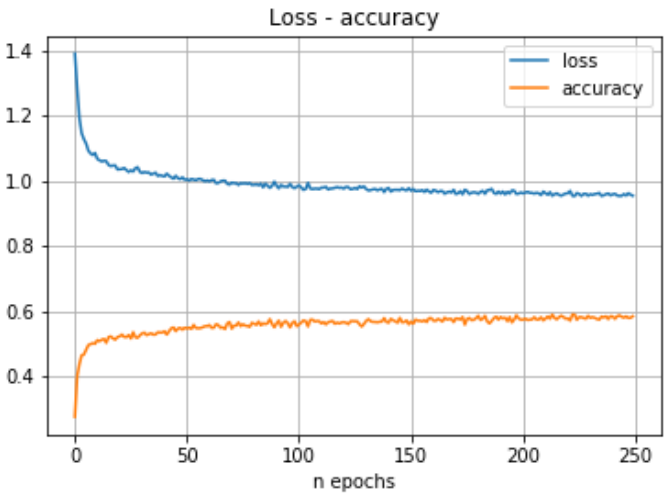

Figura 23. Red neuronal *Fully connected.* Ensayo 6

5.1.1.1.7 Simulación de 3600 segundos.

- Tiempo por epoch: 2s
- Pesos: 230564
- Resultados de entrenamiento:
  - Pérdida: 1.3864
  - Precisión: 0.2435
- Resultados de validación:
  - Pérdida: 1.3863
  - Precisión: 0.2490

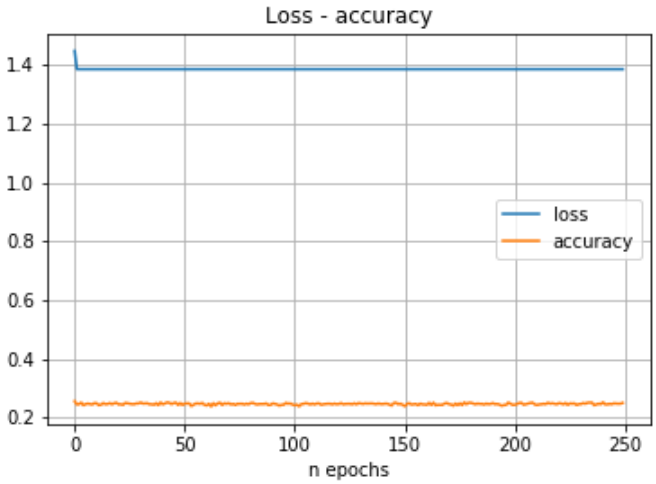

Figura 24. Red neuronal *Fully connected.* Ensayo 7

5.1.1.2 Ensayos variando el número de muestras

El tiempo de la simulación que tomamos en este ensayo es el que mejores resultados ha dado en el ensayo anterior, 100 segundos.

Observando de la Figura 25 a la Figura 32, podemos comprobar que para el caso de las redes *Fully connected*, los resultados mejoran cuanto más set de datos de entrada se aportan.

5.1.1.2.1 Muestras de entrenamiento: 1500, muestras de validación: 500

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.9194
  - Precisión: 0.5937
- Resultados de validación:
  - Pérdida: 0.9803
  - Precisión: 0.546

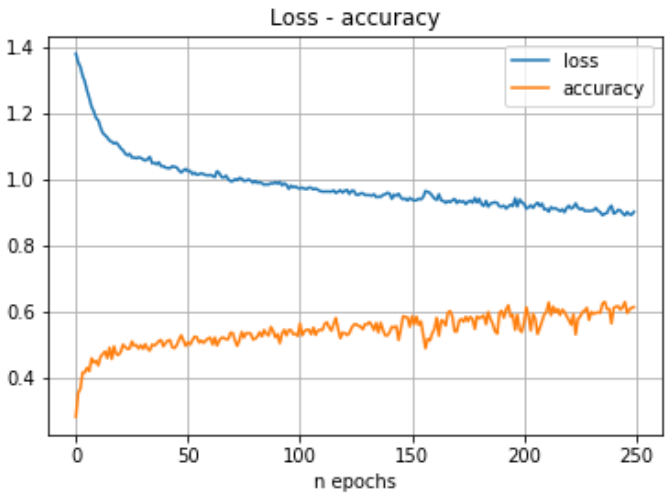

Figura 25. Red neuronal *Fully connected.* Ensayo 8

5.1.1.2.2 Muestras de entrenamiento: 3000, muestras de validación: 1000

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.8784
  - Precisión: 0.6403
- Resultados de validación:
  - Pérdida: 0.8828
  - Precisión: 0.6365

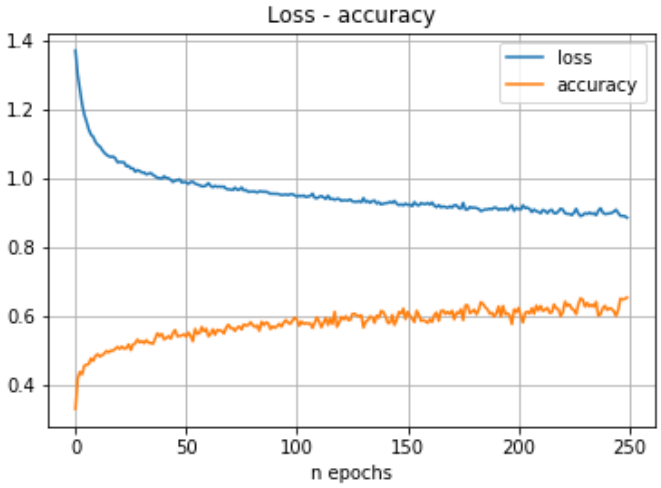

Figura 26. Red neuronal *Fully connected.* Ensayo 9

5.1.1.2.3 Muestras de entrenamiento: 4500, muestras de validación: 1500

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.8423
  - Precisión: 0.6514
- Resultados de validación:
  - Pérdida: 0.9118
  - Precisión: 0.6302

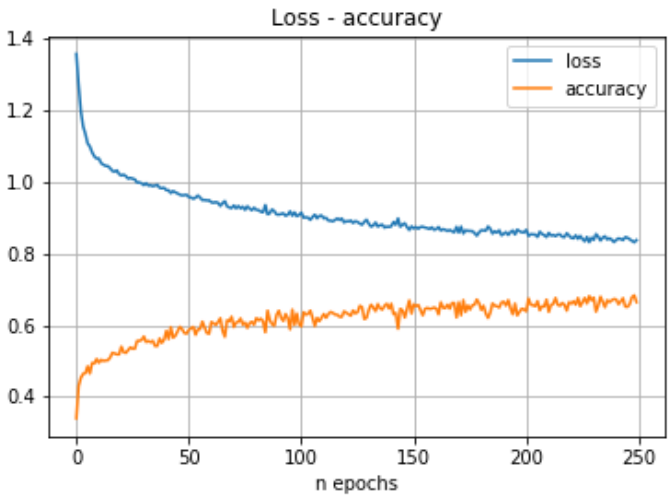

Figura 27. Red neuronal *Fully connected.* Ensayo 10

5.1.1.2.4 Muestras de entrenamiento: 6000, muestras de validación: 2000

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.8437
  - Precisión: 0.6528
- Resultados de validación:
  - Pérdida: 0.8914
  - Precisión: 0.6430

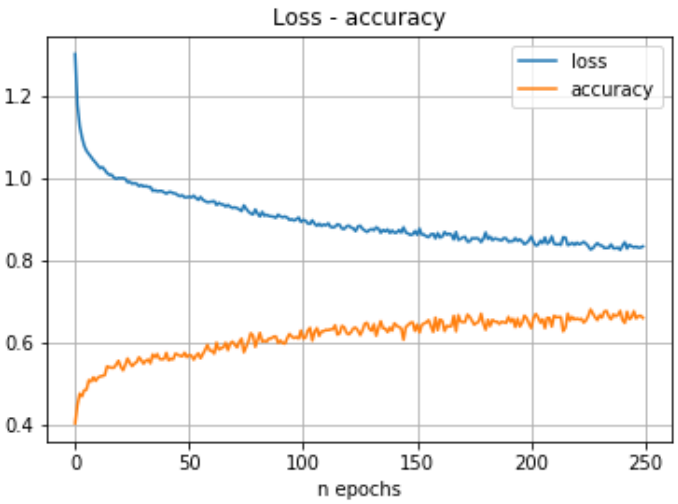

Figura 28. Red neuronal *Fully connected.* Ensayo 11

5.1.1.2.5 Muestras de entrenamiento: 7500, muestras de validación: 2500

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.8185
  - Precisión: 0.6687
- Resultados de validación:
  - Pérdida: 0.8649
  - Precisión: 0.6752

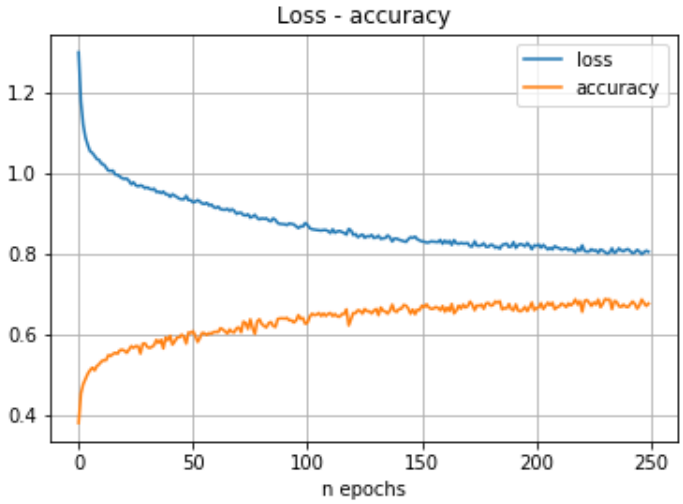

Figura 29. Red neuronal *Fully connected.* Ensayo 12

5.1.1.2.6 Muestras de entrenamiento: 9000, muestras de validación: 3000

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.8226
  - Precisión: 0.6598
- Resultados de validación:
  - Pérdida: 0.8597
  - Precisión: 0.6529

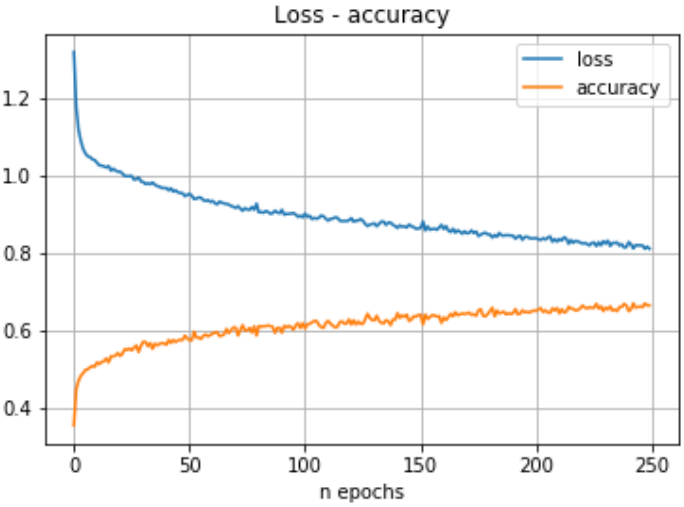

Figura 30. Red neuronal *Fully connected.* Ensayo 13

5.1.1.2.7 Muestras de entrenamiento: 10500, muestras de validación: 3500

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.8111
  - Precisión: 0.6727
- Resultados de validación:
  - Pérdida: 0.8355
  - Precisión: 0.6707

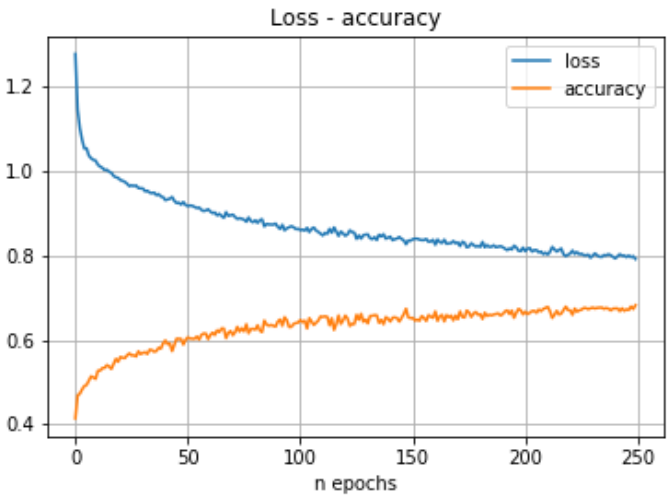

Figura 31. Red neuronal *Fully connected.* Ensayo 14

5.1.1.2.8 Muestras de entrenamiento: 12000, muestras de validación: 4000

- Tiempo por epoch: 0s
- Pesos: 6564
- Resultados de entrenamiento:
  - Pérdida: 0.7817
  - Precisión: 0.6804
- Resultados de validación:
  - Pérdida: 0.8024
  - Precisión: 0.6812

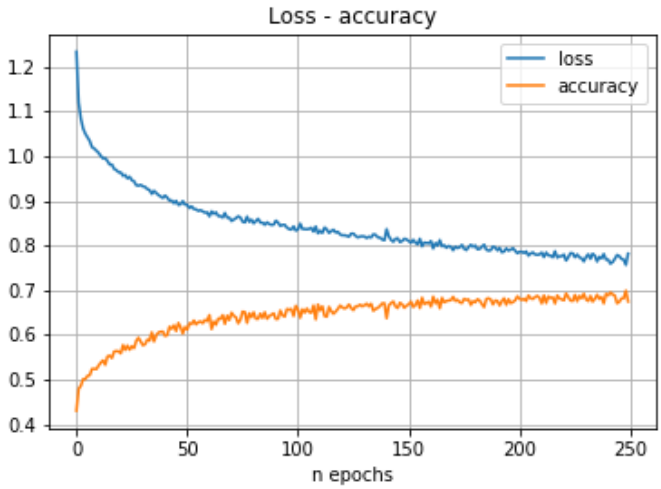

Figura 32. Red neuronal *Fully connected.* Ensayo 15

5.1.2 Ensayos variando la morfología de la red

Observando de la Figura 33 a la Figura 37, podemos comprobar que para el caso de las redes *Fully connected*, los mejores resultados se obtienen para una red neuronal de una capa de entrada de 256 neurnas, tres capas ocultas, de 128, 64 y 32 neuronas respectivamente y una capa de salida softmax de 4.

5.1.2.1 Red neuronal de una capa oculta

La red neuronal está compuesta por una capa de entrada densa de 64 neuronas , una capa oculta de 32 y una capa de salida de 4. Tanto la capa de entrada como las ocultas tienen función de activación relu. La se dalida softmax.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 0s
- Pesos: 15076.
- Resultados de entrenamiento:
  - Pérdida: 0.5426
  - Precisión: 0.7816
- Resultados de validación:
  - Pérdida: 0.7218
  - Precisión: 0.7543

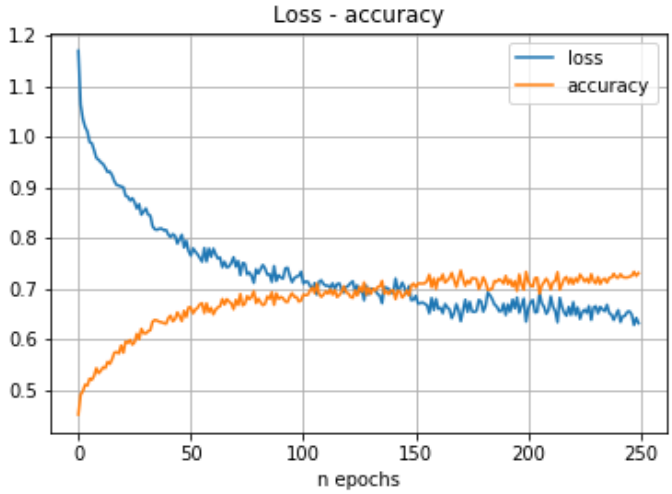

Figura 33. Red neuronal *Fully connected.* Ensayo 16

5.1.2.2 Red neuronal de dos capas ocultas

La red neuronal está compuesta por una capa de entrada de 128 neuronas, dos capas ocultas de 64 y 32 y una capa de salida de 4. Tanto la capa de entrada como las ocultas tienen función de activación relu. La se dalida softmax.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 0s
- Pesos: 36196.
- Resultados de entrenamiento:
    - Pérdida: 0.3753
    - Precisión: 0.8506
- Resultados de validación:
    - Pérdida: 0.8605
    - Precisión: 0.8132

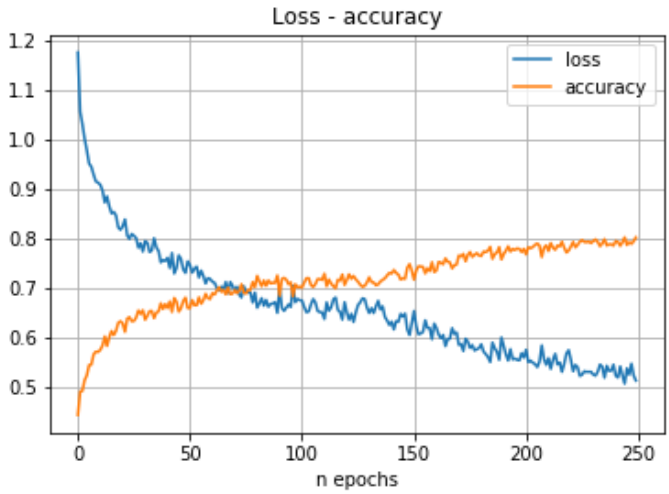

Figura 34. Red neuronal *Fully connected.* Ensayo 17

5.1.2.3 Red neuronal de tres capas ocultas

La red neuronal está compuesta por una capa de entrada de 256 neuronas, tres capas ocultas de 128, 64 y 32 y una capa de salida de 4. Tanto la capa de entrada como las ocultas tienen función de activación relu. La se dalida softmax.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 0s
- Pesos: 94820.
- Resultados de entrenamiento:
    - Pérdida: 0.2923
    - Precisión: 0.8799
- Resultados de validación:
    - Pérdida: 0.8536
    - Precisión: 0.8342

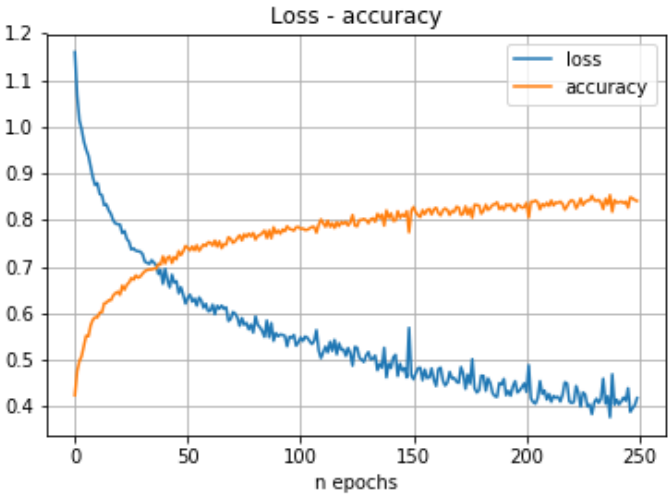

Figura 35. Red neuronal *Fully connected.* Ensayo 18

5.1.2.4 Red neuronal de cuatro capas ocultas

La red neuronal está compuesta por una capa de entrada de 512 neuronas, cuatro capas ocultas de 256, 128, 64 y 32 y una capa de salida de 4. Tanto la capa de entrada como las ocultas tienen función de activación relu. La

se dalida softmax.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 1s
- Pesos: 277604.
- Resultados de entrenamiento:
  - Pérdida: 0.24428
  - Precisión: 0.8979
- Resultados de validación:
  - Pérdida: 0.7918
  - Precisión: 0.8260

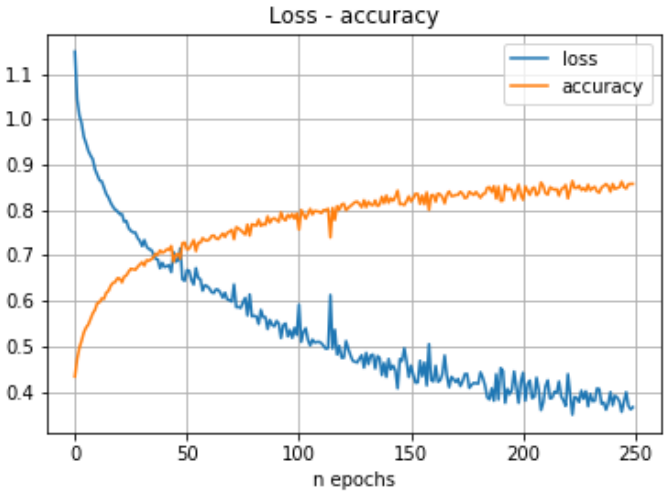

Figura 36. Red neuronal *Fully connected.* Ensayo 19

5.1.2.5 Red neuronal de tres capas ocultas con dropout

La red neuronal está compuesta por una capa de entrada de 256 neuronas, cuatro capas ocultas de 128, 64 y 32 y una capa de salida de 4 y dropout intermedios de 0.2. Tanto la capa de entrada como las ocultas tienen función de activación relu. La se dalida softmax.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 2s
- Pesos: 94820.
- Resultados de entrenamiento:
  - Pérdida: 0.2976
  - Precisión: 0.8804
- Resultados de validación:
  - Pérdida: 0.6363
  - Precisión: 0.8110

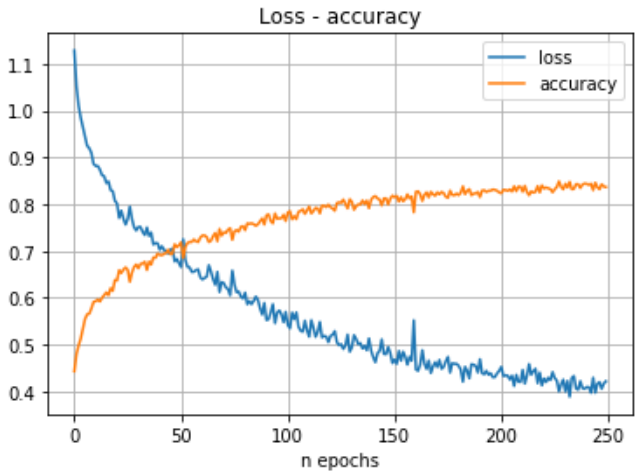

Figura 37. Red neuronal *Fully connected.* Ensayo 20

5.2 Red neuronal Convolucional

La red neuronal está compuesta por una capa de entrada convolucional de una dimensión, con 16 filtros de tamaño 8. Para adaptar la salida, tiene una capa densa con función de activación relu de 32 neuronas y una capa de salida densa con función de activación softmax de 4.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

5.2.1 16 filtros de tamaño 8

La red neuronal está compuesta por una capa de entrada de 32 neuronas y una capa de salida softmax de 4.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

5.2.1.1 Ensayos variando el tiempo de simulación.

Se introduce un set de datos de 16000 muestras, 12000 para el entrenamiento y 4000 para la validación.

Observando de la Figura 38 a la Figura 44Figura 18Figura 24, podemos comprobar que para el caso de las redes Convolucionales, los resultados mejoran cuanto mayor duración tengan los patrones de movimientos.

5.2.1.1.1 Simulación de 50 segundos

- Tiempo por epoch: 3s
- Pesos: 22,452.
- Resultados de entrenamiento:
    - Pérdida: 0.5313
    - Precisión: 0.8481
- Resultados de validación:
    - Pérdida: 0.3576
    - Precisión: 0.8318

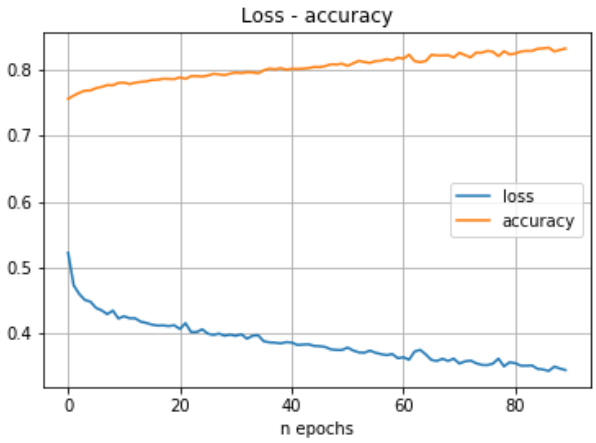

Figura 38. Red neuronal Convolucional. Ensayo 1

5.2.1.1.2 Simulación de 100 segundos

- Tiempo por epoch: 3s
- Pesos: 48,052.
- Resultados de entrenamiento:
  - Pérdida: 0.5139
  - Precisión: 0.8726
- Resultados de validación:
  - Pérdida: 0.3180
  - Precisión: 0.8642

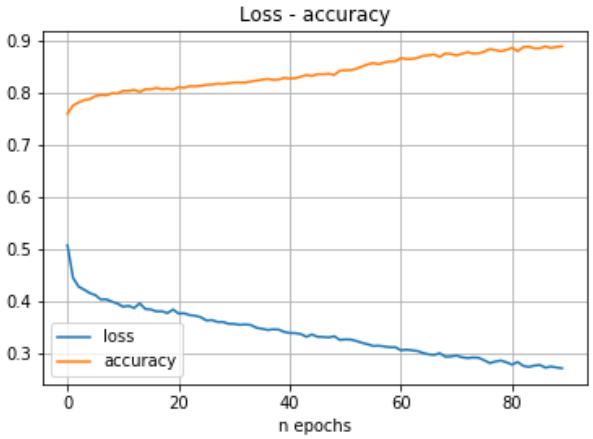

Figura 39. Red neuronal Convolucional. Ensayo 2

5.2.1.1.3 Simulación de 250 segundos

- Tiempo por epoch: 4s
- Pesos: 124852.
- Resultados de entrenamiento:
  - Pérdida: 0.5153
  - Precisión: 0.8935
- Resultados de validación:
  - Pérdida: 0.3045
  - Precisión: 0.8764

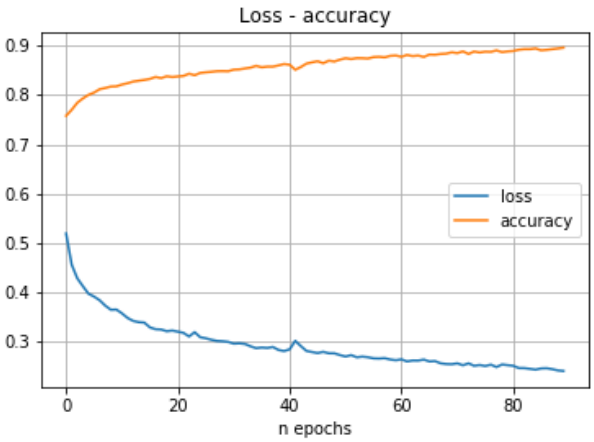

Figura 40. Red neuronal Convolucional. Ensayo 3

5.2.1.1.4 Simulación de 500 segundos

- Tiempo por epoch: 6s
- Pesos: 252852.
- Resultados de entrenamiento:
  - Pérdida: 0.5096
  - Precisión: 0.8920
- Resultados de validación:
  - Pérdida: 0.2846
  - Precisión: 0.8729

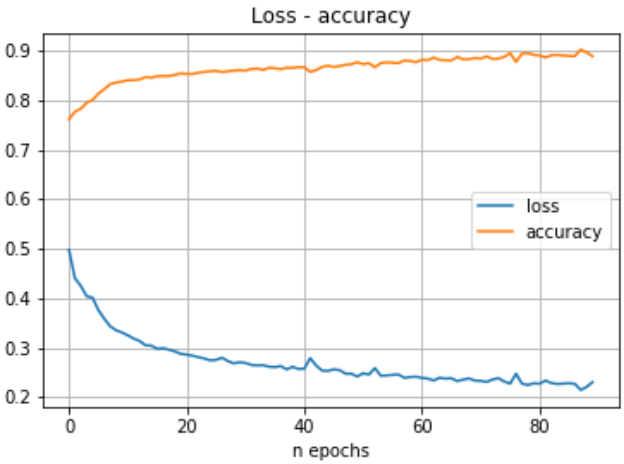

Figura 41. Red neuronal Convolucional. Ensayo 4

5.2.1.1.1 Simulación de 750 segundos

- Tiempo por epoch: 7s
- Pesos: 380852.
- Resultados de entrenamiento:
  - Pérdida: 0.4994
  - Precisión: 0.8919
- Resultados de validación:
  - Pérdida: 0.2721
  - Precisión: 0.8782

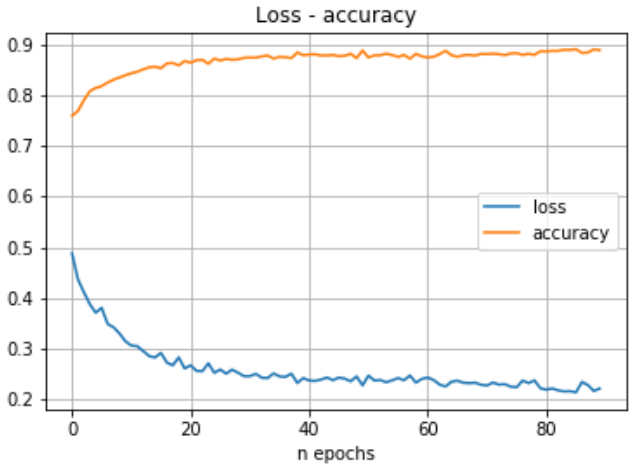

Figura 42. Red neuronal Convolucional. Ensayo 5

1.2.1.1.1 Simulación de 1000 segundos

- Tiempo por epoch: 9s
- Pesos: 508852.
- Resultados de entrenamiento:
  - Pérdida: 0.4844
  - Precisión: 0.8987
- Resultados de validación:
  - Pérdida: 0.2579
  - Precisión: 0.8896

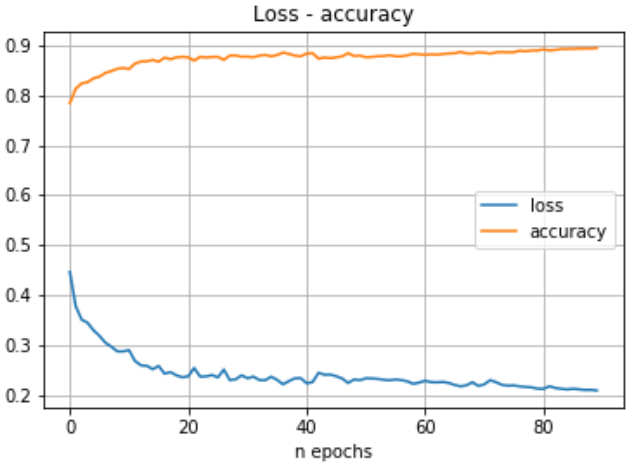

Figura 43. Red neuronal Convolucional. Ensayo 6

5.2.1.1.2 Simulación de 3600 segundos

- Tiempo por epoch: 23s
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.5644
  - Precisión: 0.9449
- Resultados de validación:
  - Pérdida: 0.2086
  - Precisión: 0.9235

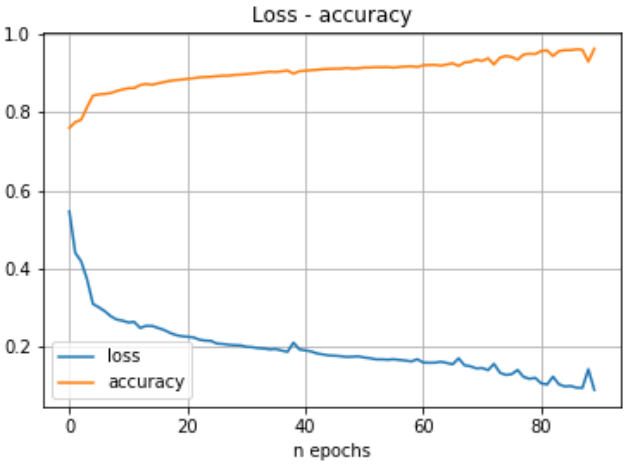

Figura 44. Red neuronal Convolucional. Ensayo 7

5.2.1.2 Ensayos variando el número de muestras

El tiempo de la simulación que tomamos en este ensayo es el que mejores resultados ha dado en el ensayo anterior, 3600 segundos.

Observando de la Figura 45Figura 25 a la Figura 52, podemos comprobar que para el caso de las redes Convolucionales, los resultados mejoran cuanto más set de datos de entrada se aportan.

5.2.1.2.1 Muestras de entrenamiento: 1500, muestras de validación: 500

- Tiempo por epoch: 6s
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.8406
  - Precisión: 0.8660
- Resultados de validación:
  - Pérdida: 0.5526
  - Precisión: 0.7973

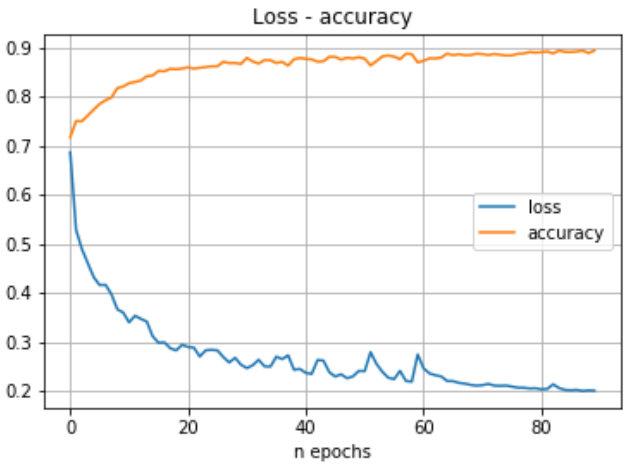

Figura 45. Red neuronal Convolucional. Ensayo 8

5.2.1.2.2 Muestras de entrenamiento: 3000, muestras de validación: 1000

- Tiempo por epoch: 9s
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.6654
  - Precisión: 0.9022
- Resultados de validación:
  - Pérdida: 0.3780
  - Precisión: 0.8639

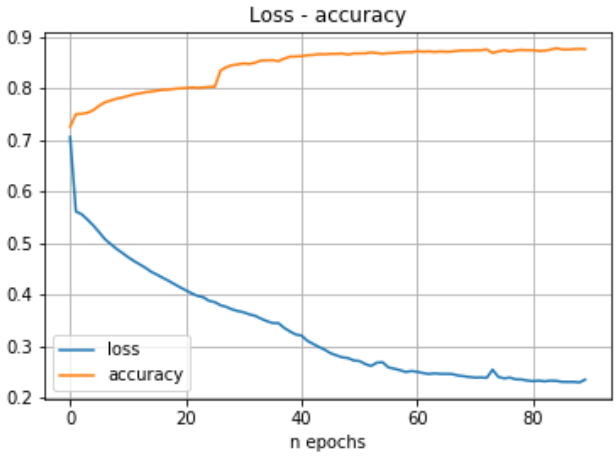

Figura 46. Red neuronal Convolucional. Ensayo 9

5.2.1.2.3 Muestras de entrenamiento: 4500, muestras de validación: 1500

- Tiempo por epoch: 12s
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.6271
  - Precisión: 0.9264
- Resultados de validación:
  - Pérdida: 0.3952
  - Precisión: 0.8790

Figura 47. Red neuronal Convolucional. Ensayo 10

5.2.1.2.4 Muestras de entrenamiento: 6000, muestras de validación: 2000

- Tiempo por epoch: 14s
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.5775
  - Precisión: 0.9190
- Resultados de validación:
  - Pérdida: 0.3570
  - Precisión: 0.8840

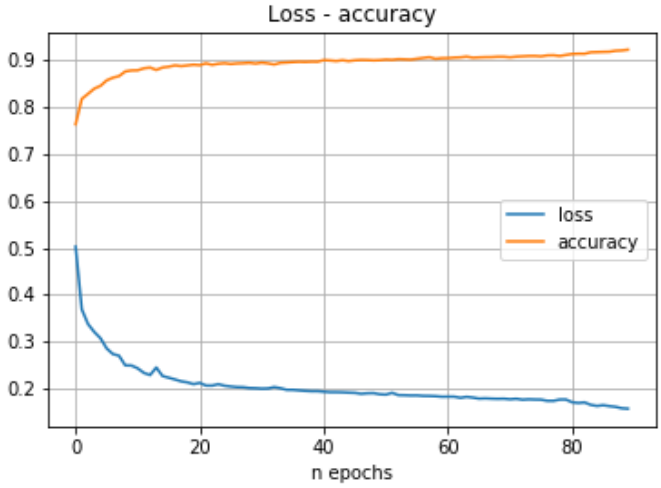

Figura 48. Red neuronal Convolucional. Ensayo 11

5.2.1.2.5 Muestras de entrenamiento: 7500, muestras de validación: 2500

- Tiempo por epoch: 16
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.5709
  - Precisión: 0.9349
- Resultados de validación:
  - Pérdida: 0.3018
  - Precisión: 0.9042

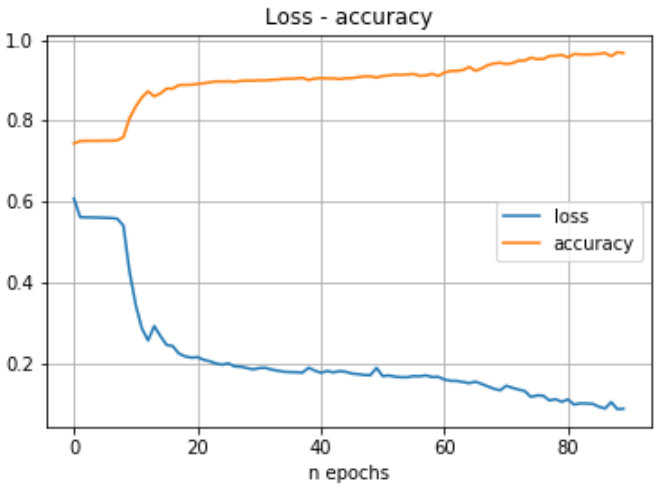

Figura 49. Red neuronal Convolucional. Ensayo 12

5.2.1.2.6 Muestras de entrenamiento: 9000, muestras de validación: 3000

- Tiempo por epoch: 19
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.5697
  - Precisión: 0.9425
- Resultados de validación:
  - Pérdida: 0.2620
  - Precisión: 0.9216

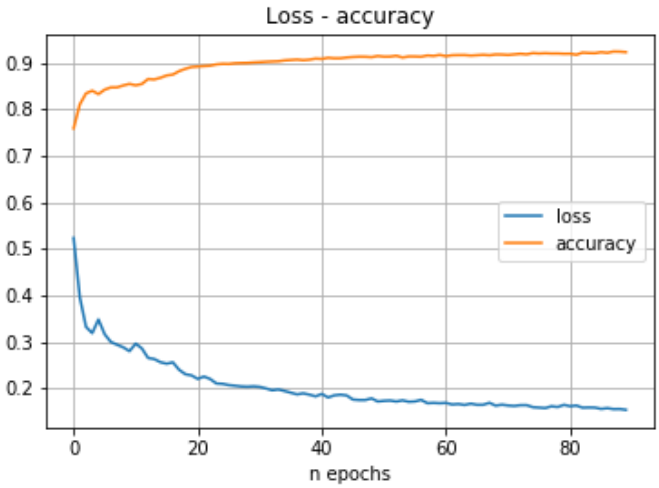

Figura 50. Red neuronal Convolucional. Ensayo 13

5.2.1.2.7 Muestras de entrenamiento: 10500, muestras de validación: 3500

- Tiempo por epoch: 22s
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.5974
  - Precisión: 0.9243
- Resultados de validación:
  - Pérdida: 0.2601
  - Precisión: 0.8974

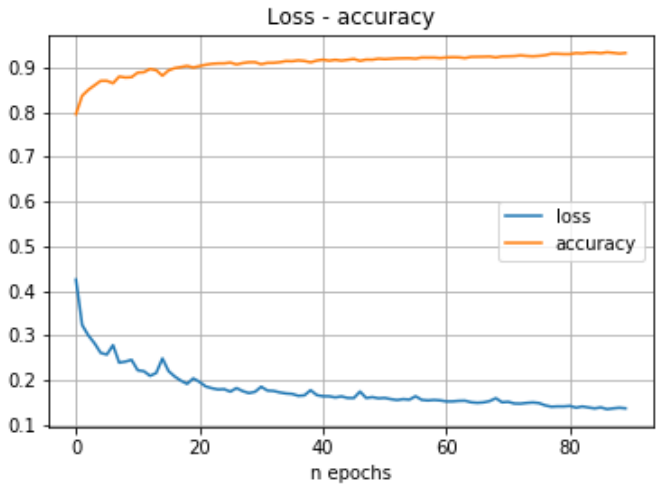

Figura 51. Red neuronal Convolucional. Ensayo 14

5.2.1.2.8 Muestras de entrenamiento: 12000, muestras de validación: 3000

- Tiempo por epoch: 24s
- Pesos: 1840052.
- Resultados de entrenamiento:
  - Pérdida: 0.5462
  - Precisión: 0.9328
- Resultados de validación:
  - Pérdida: 0.2318
  - Precisión: 0.9120

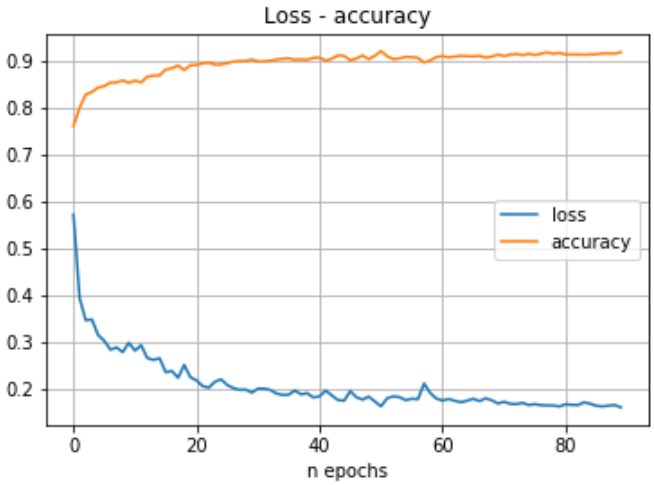

Figura 52. Red neuronal Convolucional. Ensayo 15

5.3 Red neuronal recurrente

5.3.1 Red neuronal sin capa oculta.

La red neuronal está compuesta por una capa de entrada LSTM de 32 neuronas y una capa de salida densa con función de activación softmax de 4 neuronas.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

5.3.1.1 Ensayos variando el tiempo de simulación

Se introduce un set de datos de 16000 muestras, 12000 para el entrenamiento y 4000 para la validación.

Observando de la Figura 18Figura 53 a la Figura 56Figura 24, podemos comprobar que para el caso de las redes Recurrentes, los mejores resultados se obtienen para patrones de movimiento de 100 segundos de duración.

5.3.1.1.1 Simulación de 50 segundos.

- Tiempo por epoch: 9s.
- Pesos: 4612.
- Resultados de entrenamiento:
    - Pérdida: 0.8718
    - Precisión: 0.5969
- Resultados de validación:
    - Pérdida: 0.6982
    - Precisión: 0.6978

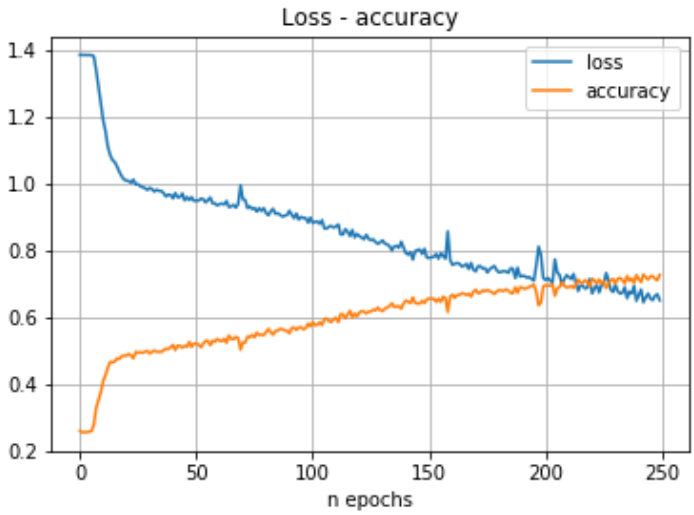

Figura 53. Red neuronal recurrente. Ensayo 1

5.3.1.1.2 Simulación de 100 segundos.

- Tiempo por epoch: 17s.
- Pesos: 4612
- Resultados de entrenamiento:
  - Pérdida: 0.9876
  - Precisión: 0.5022
- Resultados de validación:
  - Pérdida: 0.6328
  - Precisión: 0.7325

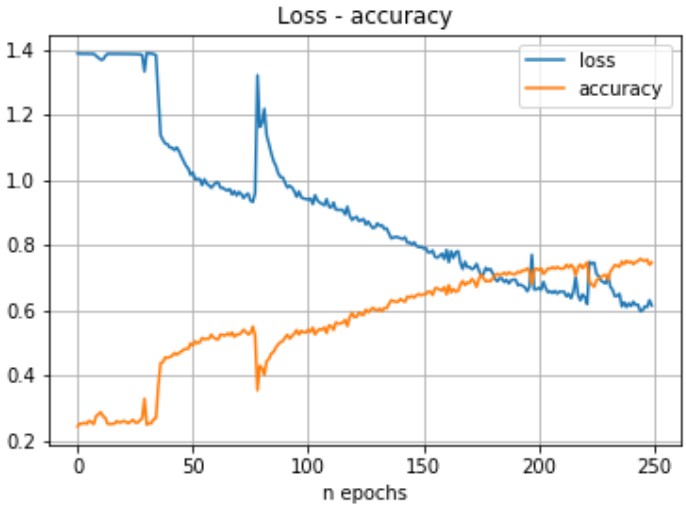

Figura 54. Red neuronal recurrente. Ensayo 2

5.3.1.1.3 Simulación de 250 segundos.

- Tiempo por epoch: 40s.
- Pesos: 4612
- Resultados de entrenamiento:
  - Pérdida: 1.3267
  - Precisión: 0.3262
- Resultados de validación:
  - Pérdida: 0.8600
  - Precisión: 0.6018

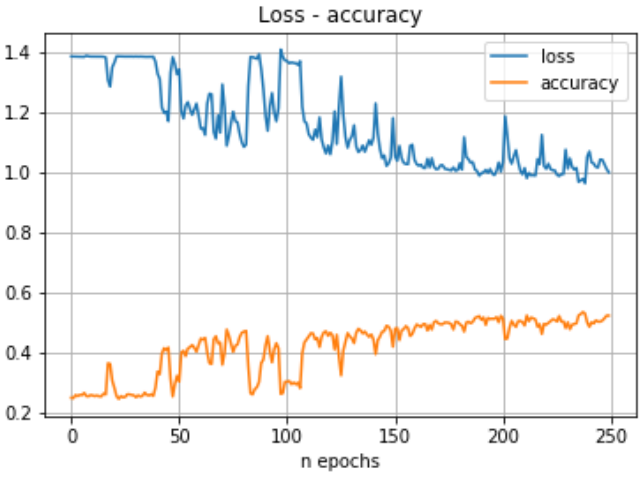

Figura 55. Red neuronal recurrente. Ensayo 3

5.3.1.1.4 Simulación de 500 segundos.

- Tiempo por epoch: 80s
- Pesos: 4612
- Resultados de entrenamiento:
  - Pérdida: 1.2986
  - Precisión: 0.3279
- Resultados de validación:
  - Pérdida: 0.9668
  - Precisión: 0.5222

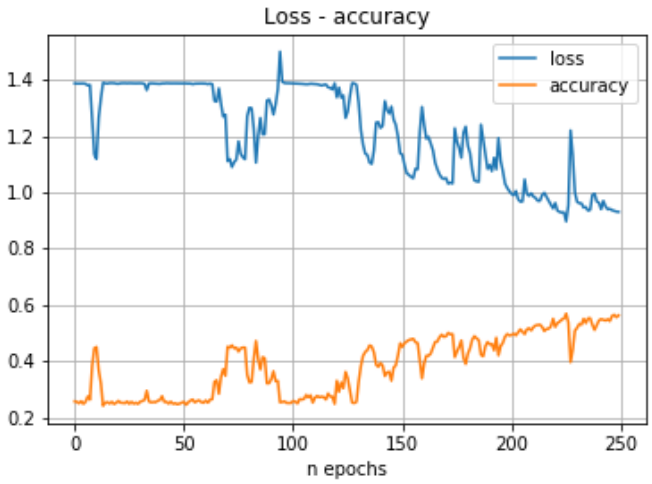

Figura 56. Red neuronal recurrente. Ensayo 4

5.3.1.2 Ensayos variando el número de muestras

El tiempo de la simulación que tomamos en este ensayo es el que mejores resultados ha dado en el ensayo anterior, 100 segundos.

Observando de la Figura 57 a la Figura 64, podemos comprobar que para el caso de las redes Recurrentes, los resultados mejoran cuanto más set de datos de entrada se aportan.

5.3.1.2.1 Muestras de entrenamiento: 1500, muestras de validación: 500

- Tiempo por epoch: 3s
- Pesos: 4612.
- Resultados de entrenamiento:
  - Pérdida: 0.8344
  - Precisión: 0.5979
- Resultados de validación:
  - Pérdida: 0.9562
  - Precisión: 0.5340

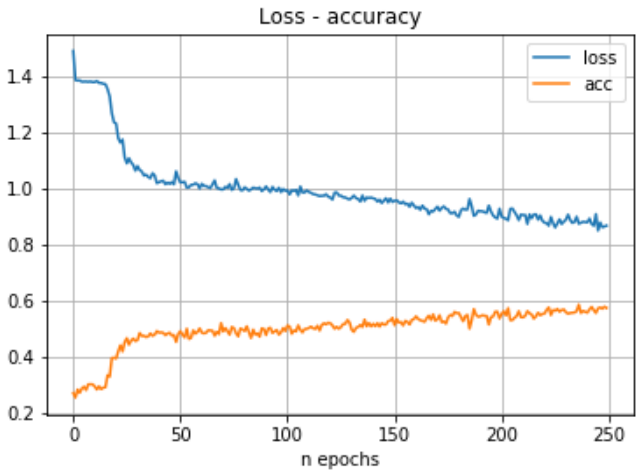

Figura 57. Red neuronal recurrente. Ensayo 5

5.3.1.2.2 Muestras de entrenamiento: 3000, muestras de validación: 1000

- Tiempo por epoch: 5s
- Pesos: 4612.
- Resultados de entrenamiento:
  - Pérdida: 0.6730
  - Precisión: 0.7234
- Resultados de validación:
  - Pérdida: 0.6990
  - Precisión: 0.7120

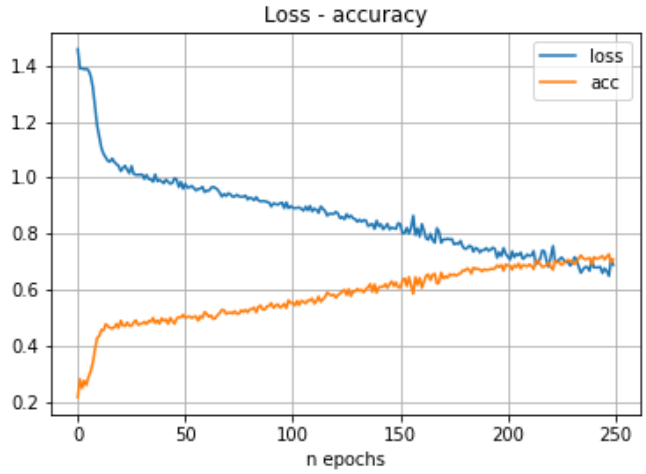

Figura 58. Red neuronal recurrente. Ensayo 6

5.3.1.2.3 Muestras de entrenamiento: 4500, muestras de validación: 1500

- Tiempo por epoch: 6
- Pesos: 4612.
- Resultados de entrenamiento:
  - Pérdida: 0.6385
  - Precisión: 0.7355
- Resultados de validación:
  - Pérdida: 0.7674
  - Precisión: 0.6693

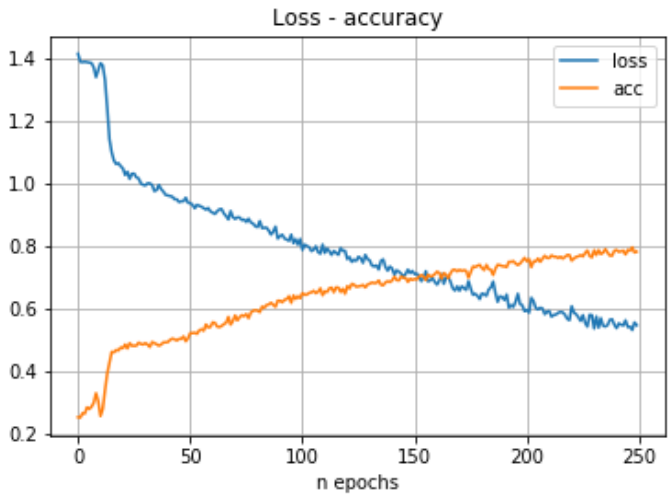

Figura 59. Red neuronal recurrente. Ensayo 7

5.3.1.2.4 Muestras de entrenamiento: 6000, muestras de validación: 2000

- Tiempo por epoch: 8s
- Pesos: 4612.
- Resultados de entrenamiento:
  - Pérdida: 0.5538
  - Precisión: 0.7796
- Resultados de validación:
  - Pérdida: 0.5936
  - Precisión: 0.7650

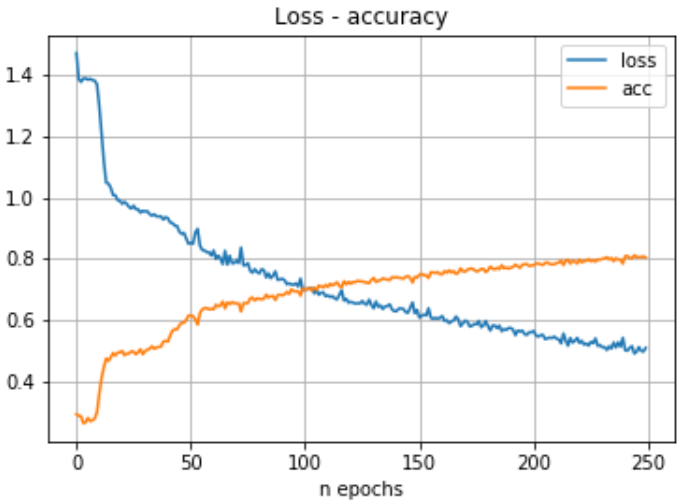

Figura 60. Red neuronal recurrente. Ensayo 8

5.3.1.2.5 Muestras de entrenamiento: 7500, muestras de validación: 2500

- Tiempo por epoch: 10s
- Pesos: 4612.
- Resultados de entrenamiento:
  - Pérdida: 0.4617
  - Precisión: 0.8241
- Resultados de validación:
  - Pérdida: 0.5105
  - Precisión: 0.8004

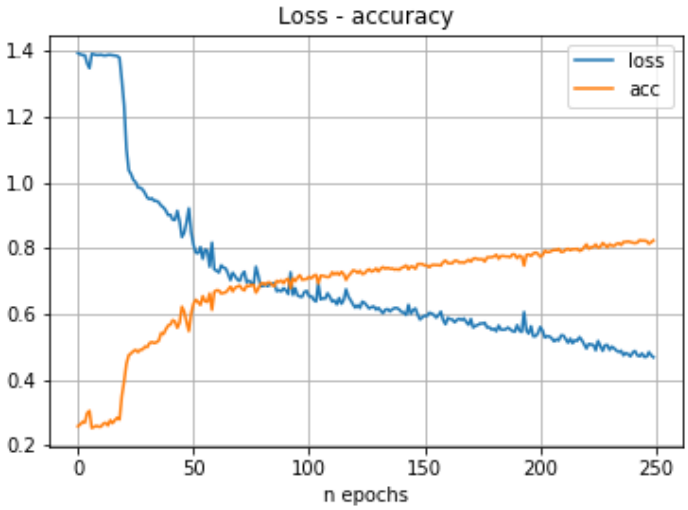

Figura 61. Red neuronal recurrente. Ensayo 9

5.3.1.2.6 Muestras de entrenamiento: 9000, muestras de validación: 3000

- Tiempo por epoch: 14s
- Pesos: 4612.
- Resultados de entrenamiento:
  - Pérdida: 0.4698
  - Precisión: 0.8234
- Resultados de validación:
  - Pérdida: 0.5138
  - Precisión: 0.8092

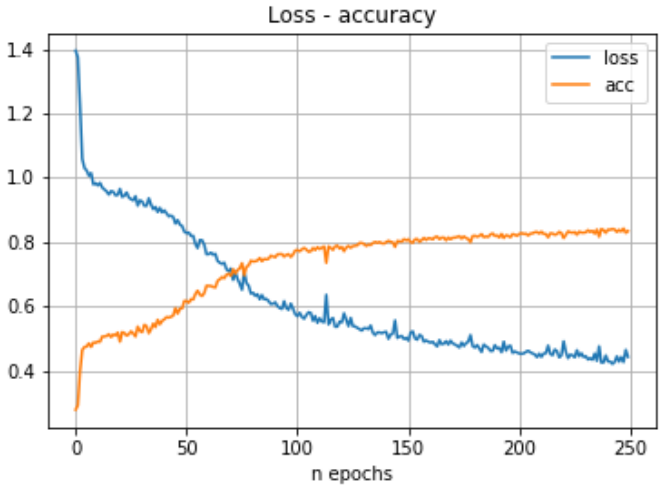

Figura 62. Red neuronal recurrente. Ensayo 10

5.3.1.2.7 Muestras de entrenamiento: 10500, muestras de validación: 3500

- Tiempo por epoch: 16s
- Pesos: 4612.
- Resultados de entrenamiento:
  - Pérdida: 0.4520
  - Precisión: 0.8305
- Resultados de validación:
  - Pérdida: 0.4917
  - Precisión: 0.8157

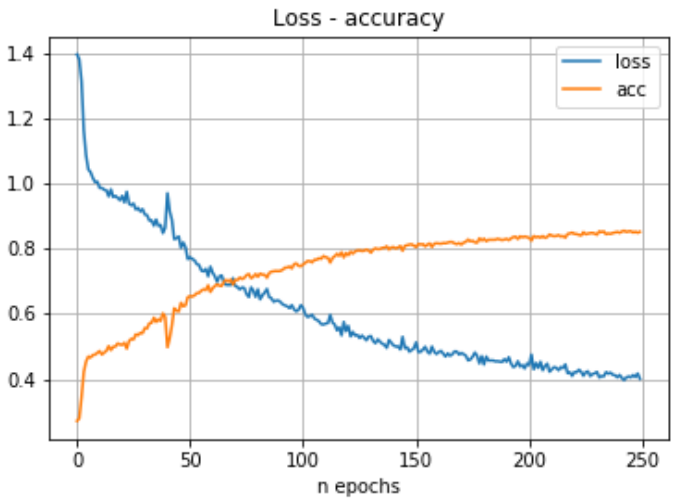

Figura 63. Red neuronal recurrente. Ensayo 11

5.3.1.2.8 Muestras de entrenamiento: 12000, muestras de validación: 4000

- Tiempo por epoch: 18s
- Pesos: 4612.
- Resultados de entrenamiento:
  - Pérdida: 0.3768
  - Precisión: 0.8628
- Resultados de validación:
  - Pérdida: 0.4009
  - Precisión: 0.8528

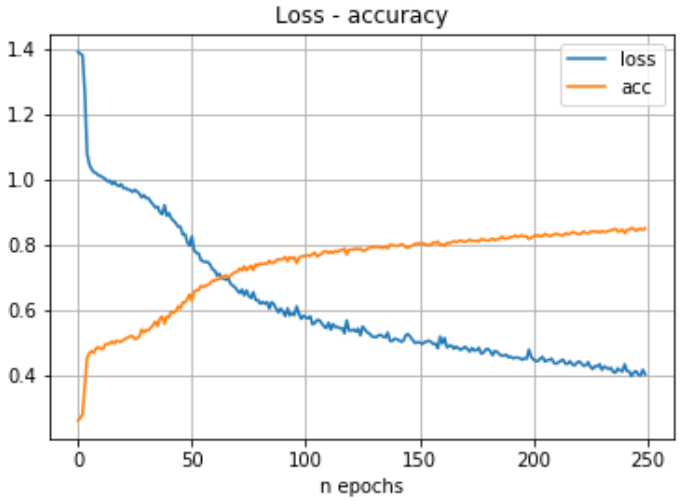

Figura 64. Red neuronal recurrente. Ensayo 12

5.3.2 Ensayos variando la morfología de la red

Observando de la Figura 65Figura 33 a la Figura 69, podemos comprobar que para el caso de las redes Recurrentes, los mejores resultados se obtienen para una red neuronal de una capa de entrada LSTM de 128 neurnas, dos capas ocultas LSTM, de 64 y 32 neuronas respectivamente y una capa de salida densa softmax de 4.

5.3.2.1 Red neuronal de una capa recurrente de 64 neuronas.

La red neuronal está compuesta por una capa de entrada LSTM de 64 neuronas y una capa de salida densa con función de activación softmax de 4 neuronas.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 2 s.
- Pesos: 17924.
- Resultados de entrenamiento:
    - Pérdida: 0.3400
    - Precisión: 0.8778
- Resultados de validación:
    - Pérdida: 0.3590
    - Precisión: 0.8732

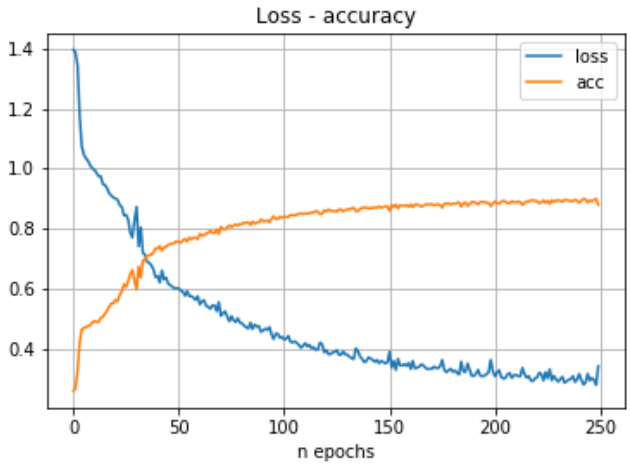

Figura 65. Red neuronal recurrente. Ensayo 13

5.3.2.2 Red neuronal de dos capas recurrentes: 64 + 32 neuronas.

La red neuronal está compuesta por una capa de entrada de 64 neuronas, una capa oculta de 32 neuronas y una capa de salida densa con función de activación softmax de 4 neuronas. Las capas de entrada y ocultas son LSTM.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 4 s.
- Pesos: 30468.
- Resultados de entrenamiento:
    - Pérdida: 0.3225
    - Precisión: 0.8836
- Resultados de validación:
    - Pérdida: 0.5295
    - Precisión: 0.8462

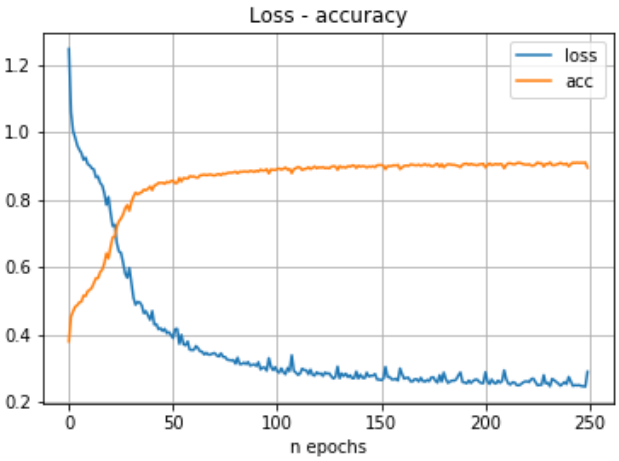

Figura 66. Red neuronal recurrente. Ensayo 14

5.3.2.3 Red neuronal de tres capas recurrentes: 128 + 64 + 32 neuronas.

La red neuronal está compuesta por una capa de entrada de 128 neuronas, dos capas ocultas de 64 y 32 neuronas y una capa de salida densa con función de activación softmax de 4 neuronas. Las capas de entrada y ocultas son LSTM.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 8 s.
- Pesos: 130820.
- Resultados de entrenamiento:
  - Pérdida: 0.2413
  - Precisión: 0.9131
- Resultados de validación:
  - Pérdida: 0.2512
  - Precisión: 0.9084

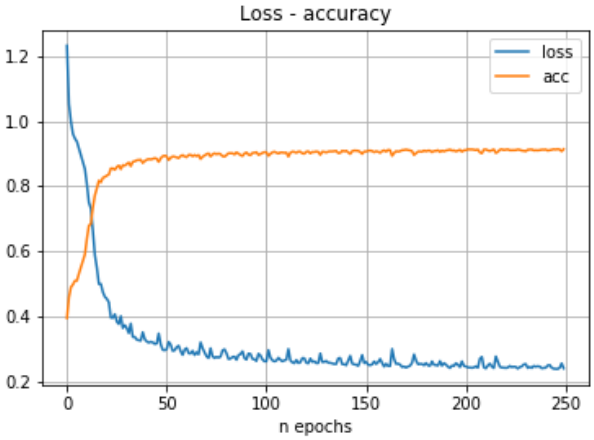

Figura 67. Red neuronal recurrente. Ensayo 15

5.3.2.4 Red neuronal de cuatro capas recurrentes: 256 + 128 + 64 + 32 neuronas.

La red neuronal está compuesta por una capa de entrada de 256 neuronas, tres capas ocultas de 128, 64 y 32 neuronas y una capa de salida densa con función de activación softmax de 4 neuronas. Las capas de entrada y ocultas son LSTM.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 14 s
- Pesos: 528132.
- Resultados de entrenamiento:
  - Pérdida: 0.3743
  - Precisión: 0.8444
- Resultados de validación:
  - Pérdida: 0.4424
  - Precisión: 0.8222

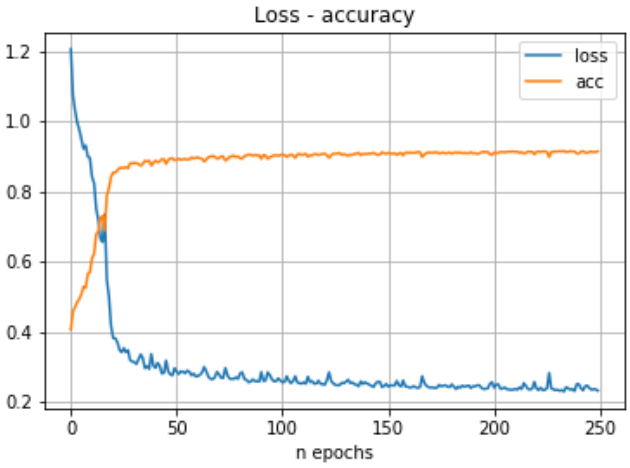

Figura 68. Red neuronal recurrente. Ensayo 16

5.3.2.5 Red neuronal de cinco capas recurrentes: 512 + 256 + 128 + 64 + 32 neuronas.

La red neuronal está compuesta por una capa de entrada de 512 neuronas, dos capas ocultas de 256, 128, 64 y 32 neuronas y una capa de salida densa con función de activación softmax de 4 neuronas. Las capas de entrada y ocultas son LSTM.

El optimizador utilizado es Adam.

Función de coste entropía categórica cruzada.

- Tiempo por epoch: 34 s.
- Pesos: 2109188.
- Resultados de entrenamiento:
    - Pérdida: 0.2547
    - Precisión: 0.9044
- Resultados de validación:
    - Pérdida: 1.6022
    - Precisión: 0.6466

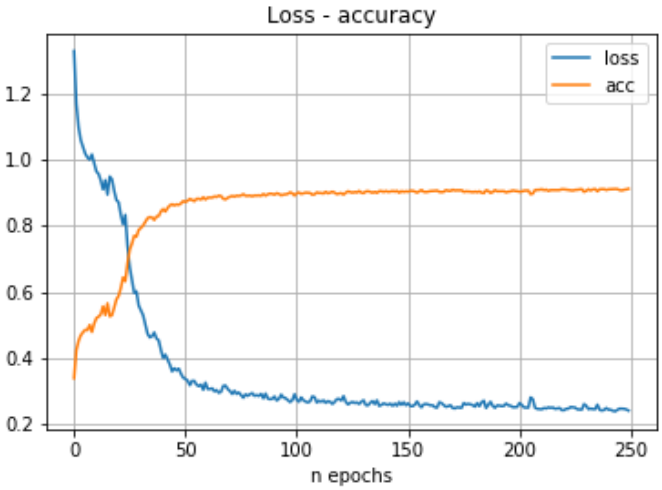

Figura 69. Red neuronal recurrente. Ensayo 17

5.4 Comparaciones

En este apartado se comparan los resultados obtenidos para las tres redes neuronales utilizadas: la red neuronal *Fully connected*, la Convolucional y la Recurrente.

5.4.1 Resultados frente a tiempo de simulación de los datos de entrada

Se analiza el impacto que tiene la duración de los movimientos en la precisión con la que los clasifica.

En las Figura 70 y Figura 71 podemos ver que dependiendo de la duración del movimiento, unas redes u otras aportan mejores resultados. Para movimientos de corta duración, las redes neuronales recurrentes aportatán mejores resultados que las convolucionales. En esta comparativa, la morfología de la red neuronal recurrente no está optimizada, por lo que la precisión de la red convolucional en todos los casos es mayor que la de la red neuronal recurrente.

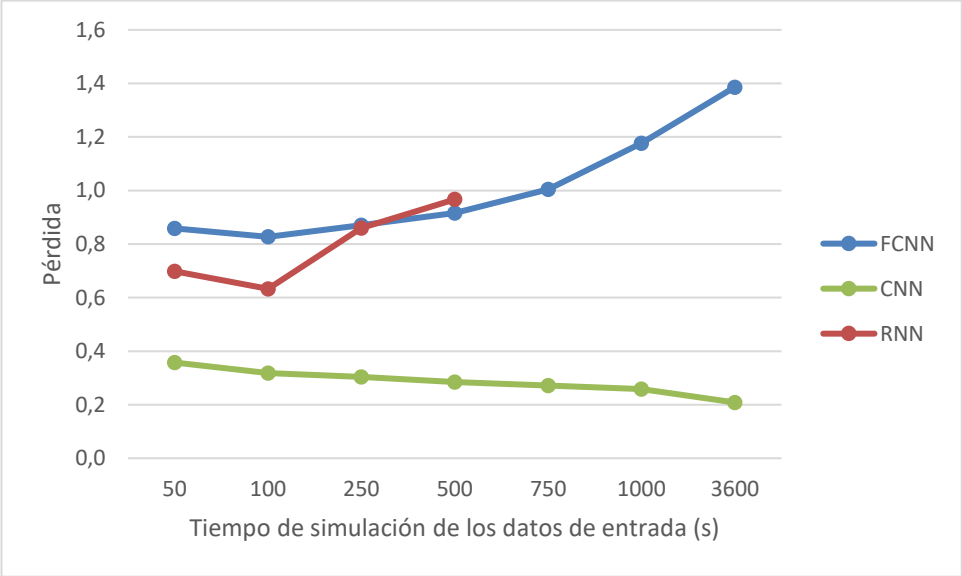

Figura 70. Pérdida frente al tiempo de simulación de los datos de entrada

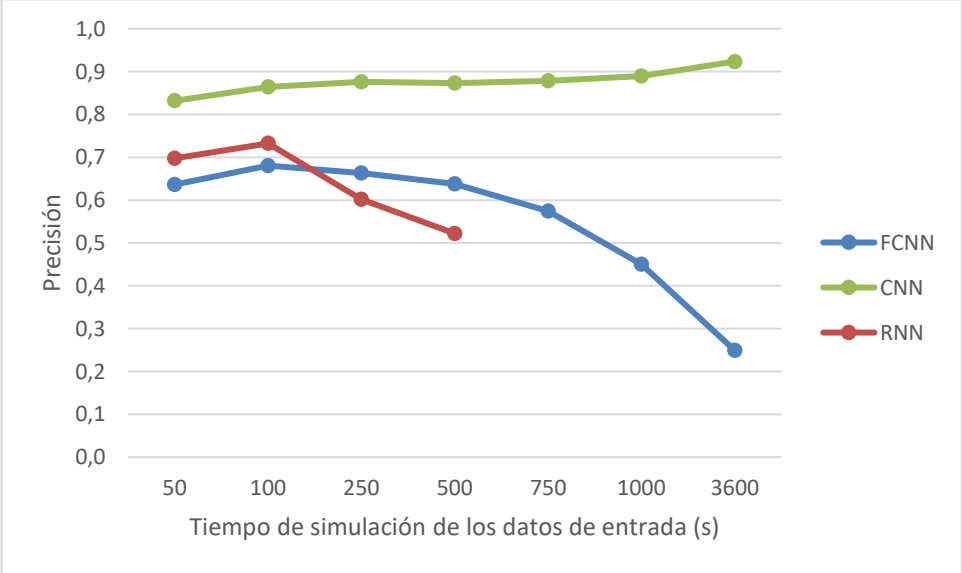

Figura 71. Precisión frente al tiempo de simulación de los datos de entrada

5.4.2 Resultados frente a número de set de datos de entrada

Se analiza el impacto que tiene el número se sets de entrada en el resultado.

En las Figura 72 y Figura 73 podemos ver que las tres redes tienen una variación de los resultados similar al aumentar el número de sets de datos. Como estaba previsto, los resultados mejoran (aumenta la precisión y disminuye la pérdida) al aumentar el número de sets de datos.

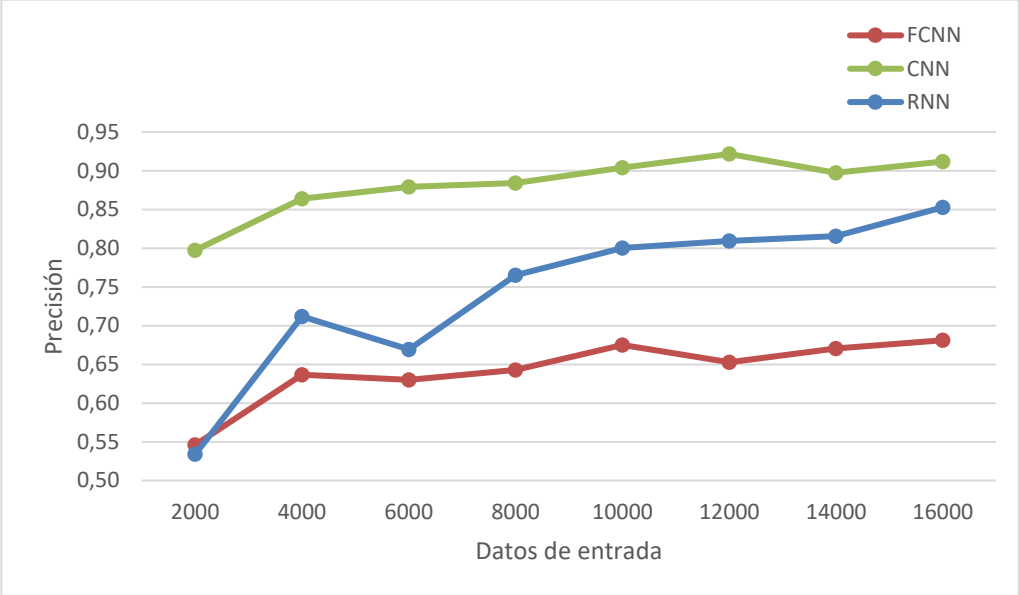

Figura 72. Precisión frente a número de sets de datos de entrada

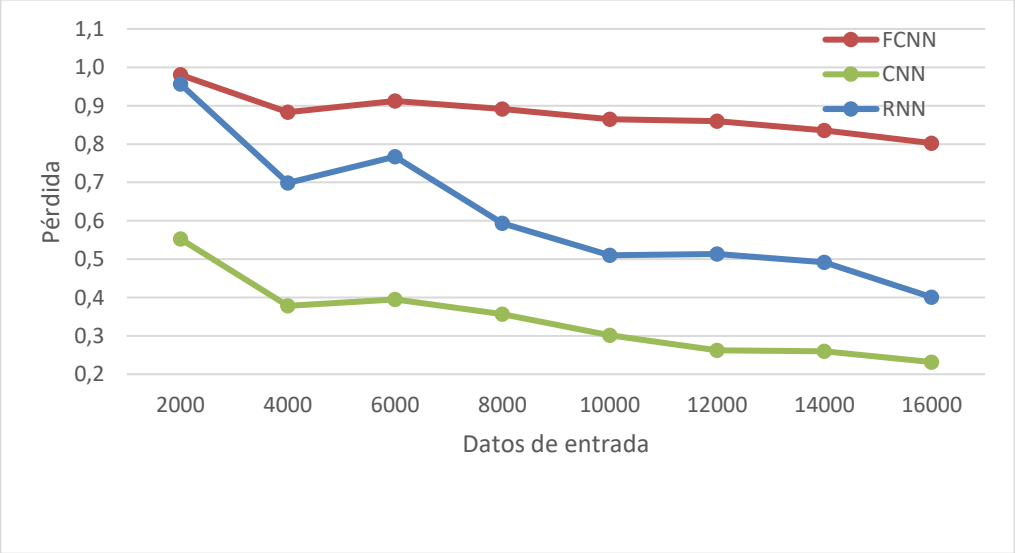

Figura 73. Precisión frente a número de sets de datos de entrada

5.4.3 Resultados frente a tiempo de ejecución

Se analiza el tiempo que tarda la red neuronal en realizar un ciclo de entrenamiento. Esto es interesante para analizar el coste computacional de cada tipo de red neuronal.

En las Figura 74 y Figura 75 podemos ver que las redes neuronales *Fully connected* requieren menor coste computacional, aunque no se obtienen tan buenos resultados como con las otras dos.

Las oscilaciones que se observan en las gráficas se deben a que se han incluido todos los ensayos realizados, con

diferentes números y tamaños de los datos de entrada y diferentes morfologías de las distintas redes.

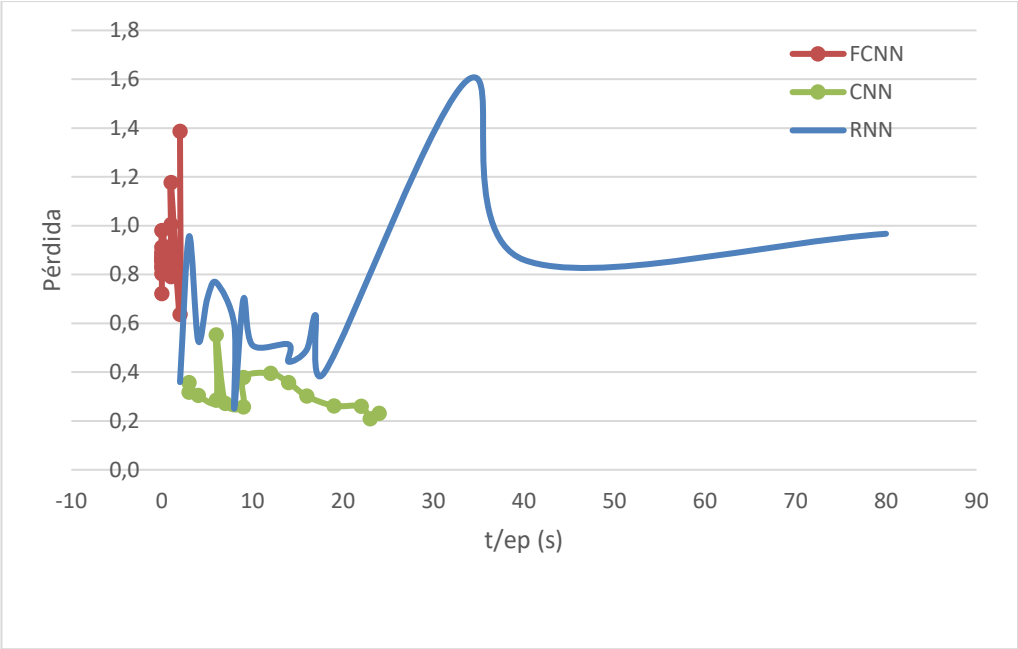

Figura 74. Pérdida frente a tiempo por ciclo de entrenamiento.

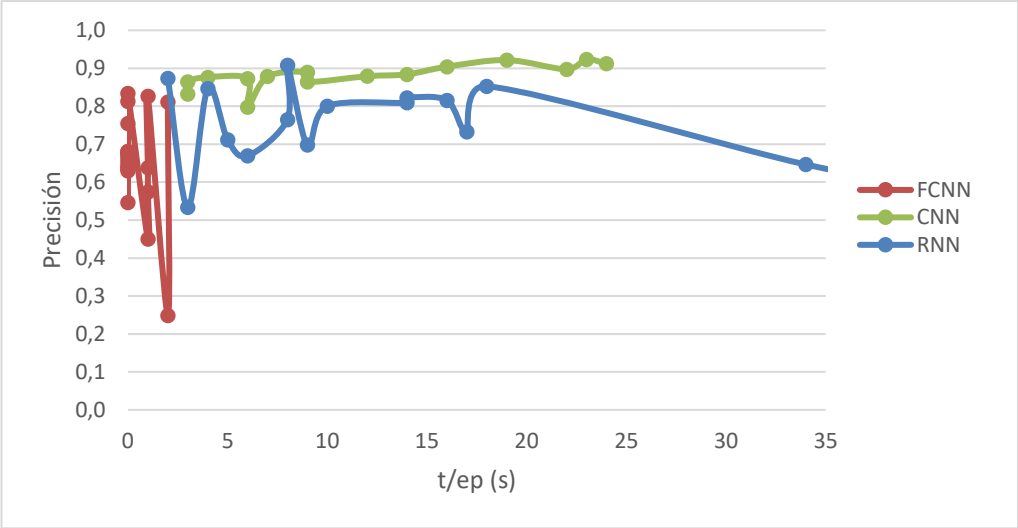

Figura 75. Precisión frente a tiempo por ciclo de entrenamiento.

6 CONCLUSIONES

Con la herramienta Bonn Motion se han generado 16000 simulaciones de movimiento de 3600 segundos, 4000 de cada tipo de movimiento que se ha clasificado en este proyecto, Manhattan, Gauss Markov, Random Waypoint y Truncated Levy Walk.

Se han realizado diferentes clasificadores de movimiento basados en redes neuronales *fully connected*, convolucionales y recurrentes.

Las redes neuronales *fully connected* han presentado peores resultados que las convolucionales y las recurrentes.

Para clasificar modelos de movimiento de corta duración, las redes neuronales recurrentes han presentado mejores resultados que las convolucionales. Las convolucionales han mejorado su resultado cuando la duración del patrón de movimiento se ha aumentado.

Para cualquier red neuronal, los resultados han mejorado cuando se ha aumentado el número de muestras del set de datos.

Las redes neuronales *fully connected* han sido las que menor coste computacional han requerido, seguidas de las redes convolucionales y por último las recurrentes. Para el entrenamiento de redes neuronales recurrentes profundas (con capas ocultas) ha sido necesario ejecutar el código mediante la GPU, ya que mediante la CPU, el tiempo que se ha requerido para un ciclo de entrenamiento ha rondado los diez minutos, imposibilitando la ejecución de los 250 ciclos que se han establecido en este proyecto. Usando la GPU, el tiempo que se ha requerido se ha reducido a una vigésima parte.

Los mejores resultados han sido:

- Para una red neuronal convolucional de 16 filtros de tamaño 8, con un set de datos de entrada de 12000 muestras de entrenamiento, 4000 de validación y ambas de duración 3600 segundos, la precisión de validación ha sido de 0.9120 y la pérdida de 0.2318.
- Para una red neuronal recurrente con tres capas recurrentes LSTM, de una capa de 128, 64 y 32 neuronas respectivamente, con un set de datos de entrada de 12000 muestras de entrenamiento, 4000 de validación y ambas de duración 100 segundos, la precisión de validación ha sido de 0.9084 y la pérdida de 0.2413.

7 TRABAJOS FUTUROS

El clasificador de movimiento que se ha realizado en este proyecto aporta buenos resultados, no obstante, se podría seguir entrenando los modelos que aquí se presentan con el fin de mejorar la precisión. Para esto, las acciones que se podrían hacer son las siguientes:

- Aumentar el set de datos de entrada. Para ello, se podrían generar nuevas simulaciones con Bon Motion o con las simulaciones que ya tenemos, aplicar la técnica de *step forward* que consiste en fragmentar los patrones que ya tenemos en patrones con duraciones más cortas.
- Aumentar el número de ciclos de entrenamiento. Esto es un poco arriesgado, porque aumentar en exceso este valor podría generar *overfitting*.
- En el caso de las redes neuronales convolucionales, se podría estudiar con movimientos más largos (en tiempo).

Por otro lado, se podría diseñar una red neuronal mixta que incluya capas recurrentes y convolucionales, con el objetivo de hacer más versátil el clasificador, con la mejor precisión para cualquier duración de los patrones de movimiento.

Por último, podría someterse el clasificador a patrones de movimiento reales.

BIBLIOGRAFÍA

[1] C.Santana [DotCSV] *¿Qué es el Machine Learning? ¿Y Deep Learning? Un mapa conceptual* [Archivo de video]. Recuperado de: https://www.youtube.com/watch?v=KytW151dpqU

[2] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, *Learning representations by back propagating errors* , 1986.

[3] Karpathy, A. (21.05.2015), *The Unreasonable Effectiveness of Recurrent Neural Networks* [Mensaje en un blog]. Andrej Karpathy blog. Recuperado de: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

[4] Colyer, A. (23.03.2017), *Recurrent Neural Network models* [Mensaje en un blog]. The Morning Paper. Recuperado de: https://blog.acolyer.org/2017/03/23/recurrent-neural-network-models/

[5] Torres, J. *Deep Learning. Introducción práctica con Keras. Primera Parte,* 2018.

[6] Torres, J. *Deep Learning. Introducción práctica con Keras. Segunda Parte,* 2019.

[7] Calvo, D. (13.07.2017), *Clasificación de redes neuronales artificiales* [Mensaje en un blog]. Diego Calvo. Recuperado de: http://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[8] Karn, U. (11.08.2016), *An Intuitive Explanation of Convolutional Neural Networks* [Mensaje en un blog]. The data science blog. Recuperado de: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[9] G. Walunjkar and A. Koteswara Rao, *Simulation and Evaluation of Different Mobility Models in Disaster Scenarios,* 2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, Bangalore, India, 2019, pp. 464-469.*

[10] Nagel, B. (10.10.2019), *Deep Learning Shifting from TensorFlow to Pytorch* [Mensaje en un blog]. Pure AI. Recuperado de: https://pureai.com/articles/2019/10/10/machine-learning-framework-popularity.aspx

[11] B. Liang and Z. J. Haas, "Predictive distance-based mobility management for PCS networks", *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, New York, NY, USA, 1999, pp. 1377-1384 vol.3, doi: 10.1109/INFCOM.1999.752157.

[12] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim and S. Chong, "On the Levy-Walk Nature of Human Mobility," in *IEEE/ACM Transactions on Networking, vol. 19, no. 3, pp. 630-643, June 2011, doi: 10.1109/TNET.2011.2120618*.

[13] D. Gutiérrez and S. Toral, "Deep Neuronal Based Classifiers for Wireless Multi-hop Network Mobility Models," 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), Boca Raton, FL, USA, 2019, pp. 602-607, doi: 10.1109/ICMLA.2019.00111.

[14] M. D. Gupta, S. Rajaram, N. Petrovic and T. S. Huang, "Classifiers for Motion," 18th International Conference on Pattern Recognition (ICPR'06), Hong Kong, 2006, pp. 593-596, doi:

10.1109/ICPR.2006.374.

[15] A. Ahmadi and I. Patras, "Unsupervised convolutional neural networks for motion estimation," 2016 *IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, 2016, pp. 1629-1633, doi: 10.1109/ICIP.2016.7532634.*

ANEXO I. TABLAS DE RESULADOS

Tabla de resultados de Red Neuronal Fully Conneced

	Estructura de la red			Datos				Entrenamiento			Resultados							
	Capa de entrada	Capas ocultas	Capa de salida	Pesos	Tiempo	Entr	Val	Epochs	Batch size	t/ep	Entrenamiento				Validación			
											Pérdida		Precisión		Pérdida		Precisión	
1	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	3364	50	12000	4000	250	50	0	0.8494	0.8510	0.6403	0.6375	0.8637	0.8589	0.6262	0.6366
											0.8682		0.6291		0.8841		0.6154	
											0.8485		0.6359		0.8484		0.6424	
											0.8379		0.6445		0.8393		0.6622	
2	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	12000	4000	250	50	0	0.7879	0.7970	0.6895	0.6858	0.8201	0.8270	0.6874	0.6806
											0.7913		0.6856		0.8193		0.6899	
											0.8046		0.6903		0.8479		0.6759	
											0.8043		0.6776		0.8207		0.6692	
3	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	16164	250	12000	4000	250	50	0	0.7997	0.8148	0.6792	0.6776	0.8225	0.8706	0.6924	0.6629
											0.8092		0.6779		0.8968		0.6154	
											0.8176		0.6802		0.8885		0.6849	
											0.8325		0.6731		0.8746		0.6587	
4	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	32164	500	12000	4000	250	50	1	0.8663	0.8550	0.6492	0.6553	0.9223	0.9161	0.6254	0.6380
											0.8291		0.6739		0.9184		0.6839	
											0.8094		0.6902		0.8620		0.6729	
											0.9153		0.6078		0.9616		0.5696	
5	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	48164	750	12000	4000	250	50	1	1.0568	0.9697	0.5300	0.5785	1.0920	1.0048	0.5299	0.5740
											0.9882		0.5665		1.0103		0.5434	
											0.8627		0.6468		0.9112		0.6629	
											0.9712		0.5707		1.0055		0.5599	
6	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	64164	1000	12000	4000	250	50	1	1.3223	1.1399	0.3336	0.4546	1.3282	1.1771	0.3316	0.4500
											1.3152		0.3356		1.3373		0.3378	
											0.9669		0.5641		1.0134		0.5854	
											0.9553		0.5852		1.0294		0.5451	
7	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	230564	3600	12000	4000	250	50	2	1.3864	1.3864	0.2402	0.2435	1.3863	1.3863	0.2516	0.2490
											1.3864		0.2384		1.3863		0.2486	

											1.3864		0.2457		1.3863		0.2486	
											1.3864		0.2498		1.3863		0.2473	
8	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	1500	500	250	50	0	0.9290	0.9194	0.5587	0.5937	0.9884	0.9803	0.4980	0.5460
											0.9303		0.5847		0.9533		0.5760	
											0.9154		0.6167		0.9588		0.5880	
											0.9027		0.6147		1.0207		0.5220	
9	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	3000	1000	250	50	0	0.8771	0.8784	0.6303	0.6403	0.9013	0.8828	0.5860	0.6365
											0.8766		0.6487		0.8837		0.6230	
											0.8743		0.6273		0.8798		0.6570	
											0.8856		0.6547		0.8662		0.6800	
10	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	4500	1500	250	50	0	0.8535	0.8423	0.6344	0.6514	0.9415	0.9118	0.6220	0.6302
											0.8442		0.6582		0.9120		0.6200	
											0.8338		0.6482		0.8930		0.6253	
											0.8378		0.6647		0.9007		0.6533	
11	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	6000	2000	250	50	0	0.8598	0.8437	0.6565	0.6528	0.9188	0.8912	0.6350	0.6430
											0.8375		0.6523		0.8735		0.6400	
											0.8432		0.6428		0.8940		0.6545	
											0.8341		0.6595		0.8785		0.6425	
12	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	7500	2500	250	50	0	0.8138	0.8185	0.6613	0.6687	0.8708	0.8649	0.6600	0.6752
											0.8364		0.6687		0.8808		0.6756	
											0.8193		0.6691		0.8525		0.6876	
											0.8044		0.6755		0.8553		0.6776	
13	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	9000	3000	250	50	0	0.8332	0.8226	0.6432	0.6598	0.8734	0.8597	0.6450	0.6529
											0.8177		0.6628		0.8438		0.6653	
											0.8284		0.6683		0.8905		0.6403	
											0.8112		0.6647		0.8310		0.6610	
14	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	10500	3500	250	50	0	0.8475	0.8111	0.6570	0.6727	0.8426	0.8355	0.6840	0.6707
											0.8117		0.6735		0.8502		0.6777	
											0.7934		0.6777		0.8265		0.6554	
											0.7919		0.6826		0.8228		0.6657	
15	Flatten + Dense - f.a relu - 32 n	-	Dense - f.a softmax	6564	100	12000	4000	250	50	0	0.7838	0.7817	0.6731	0.6804	0.8019	0.8024	0.6734	0.6812
											0.7876		0.6858		0.8012		0.6989	
											0.7735		0.6881		0.8056		0.6689	
											0.7820		0.6747		0.8007		0.6834	
16		Dense - f.a relu - 32 n	Dense - f.a softmax	15076	100	12000	4000	250	50	0	0.5132	0.5426	0.7939	0.7816	0.7310	0.7218	0.7657	0.7543

	Flatten + Dense - f.a relu - 64 n										0.5750		0.7662		0.7229		0.7469	
											0.5405		0.7875		0.7507		0.7339	
											0.5417		0.7786		0.6827		0.7707	
17	Flatten + Dense - f.a relu - 128 n	Dense - f.a relu - 64 n	Dense - f.a softmax	36196	100	12000	4000	250	50	0	0.3699	0.3753	0.8542	0.8506	0.9338	0.8605	0.8100	0.8132
											0.3487		0.8607		0.8259		0.8232	
		Dense - f.a relu - 32 n									0.3988		0.8391		0.8387		0.8005	
											0.3837		0.8482		0.8436		0.8190	
18	Flatten + Dense - f.a relu - 512 n	Dense - f.a relu - 128 n	Dense - f.a softmax	94820	100	12000	4000	250	50	0	0.2694	0.2923	0.8865	0.8799	0.8868	0.8536	0.8340	0.8342
		Dense - f.a relu - 64 n									0.2938		0.8797		0.7477		0.8335	
											0.3055		0.8748		0.9195		0.8207	
		Dense - f.a relu - 32 n									0.3004		0.8785		0.8605		0.8485	
19	Flatten + Dense - f.a relu - 512 n	Dense - f.a relu - 256 n	Dense - f.a softmax	277604	100	12000	4000	250	50	1	0.2248	0.2443	0.9055	0.8979	0.8498	0.7918	0.8242	0.8260
		Dense - f.a relu - 128 n									0.2668		0.8878		0.7962		0.8280	
		Dense - f.a relu - 64 n									0.2478		0.8971		0.7025		0.8285	
		Dense - f.a relu - 32 n									0.2377		0.9012		0.8185		0.8232	
20	Flatten + Dense - f.a relu - 256 n	Dropout - rate 0.2	Dense - f.a softmax	94820	100	12000	4000	250	50	2	0.3145	0.2976	0.8732	0.8804	0.6332	0.6363	0.8107	0.8110
		LSTM - 128 n																
		Dropout - rate 0.2									0.2806		0.8876		0.6394		0.8112	
		LSTM - 64 n																
		Dropout - rate 0.2									0.2919		0.8830		0.6292		0.8130	
		LSTM - 32 n																
		Dropout - rate 0.2									0.2912		0.8809		0.6073		0.8407	

Tabla 1. Resultados FCNN

Tabla de resultados de Red Neuronal Convolucional

	Estructura de la red			Datos				Entrenamiento			Resultados							
											Entrenamiento				Validación			
	número de filtros	tamaño del filtro	Pesos	Features	Tiempo	Entr	Val	Epochs	Batch size	t/ep	Pérdida		Precisión		Pérdida		Precisión	
1	16	8	22452	Posición X	50	12000	4000	250	50	3	0.5322	0.5313	0.8368	0.8481	0.3644	0.3576	0.8163	0.8318
				Posición Y							0.5318		0.8507		0.3484		0.8405	
											0.5391		0.8710		0.3298		0.8676	
											0.5221		0.8339		0.3878		0.8029	
2	16	8	48052	Posición X	100	12000	4000	250	50	3	0.5141	0.5139	0.8450	0.8726	0.3320	0.3180	0.8352	0.8642
				Posición Y							0.5079		0.8799		0.3086		0.8721	
											0.5246		0.8750		0.3211		0.8692	
											0.5087		0.8907		0.3101		0.8803	
3	16	8	124852	Posición X	250	12000	4000	250	50	4	0.5205	0.5153	0.8893	0.8935	0.3002	0.3045	0.8710	0.8764
				Posición Y							0.5055		0.8909		0.3141		0.8706	
											0.5164		0.8971		0.2941		0.8835	
											0.5187		0.8966		0.3093		0.8807	
4	16	8	252852	Posición X	500	12000	4000	250	50	6	0.5358	0.5096	0.8922	0.8920	0.2755	0.2846	0.8787	0.8729
				Posición Y							0.5228		0.8785		0.2855		0.8732	
											0.4821		0.8955		0.2865		0.8780	
											0.4975		0.9019		0.2910		0.8618	
5	16	8	380852	Posición X	750	12000	4000	250	50	7	0.5137	0.4994	0.8879	0.8919	0.2617	0.2721	0.8845	0.8782
				Posición Y							0.5185		0.8890		0.2777		0.8730	
											0.4771		0.9002		0.2753		0.8780	
											0.4881		0.8905		0.2736		0.8774	
6	16	8	508852	Posición X	1000	12000	4000	250	50	9	0.4688	0.4844	0.8934	0.8987	0.2475	0.2579	0.9020	0.8896
				Posición Y							0.5660		0.9056		0.2771		0.8700	
											0.4557		0.9019		0.2368		0.9035	
											0.4471		0.8939		0.2700		0.8830	
7	16	8	1840052	Posición X	3600	12000	4000	250	50	23	0.5935	0.5644	0.9742	0.9449	0.1334	0.2086	0.9610	0.9235
				Posición Y							0.5655		0.9272		0.2169		0.9054	
											0.5521		0.9159		0.2946		0.8933	
											0.5465		0.9624		0.1893		0.9343	
8	16	8	1840052	Posición X	3600	1500	500	250	50	6	0.8557	0.8406	0.9153	0.8660	0.5527	0.5526	0.8141	0.7973

										1.0823		0.9050		0.5052		0.8246	
				Posición Y						0.7379		0.7500		0.5632		0.7500	
										0.6865		0.8937		0.5893		0.8006	
9	16	8	1840052	Posición X	3600	3000	1000	250	50	9	0.8055	0.6654	0.9023	0.9022	0.3794	0.3780	0.8594	0.8639
				Posición Y							0.5690		0.9198		0.4096		0.8694	
											0.5813		0.9089		0.3772		0.8624	
											0.7058		0.8779		0.3459		0.8644	
10	16	8	1840052	Posición X	3600	4500	1500	250	50	12	0.5870	0.6271	0.9172	0.9264	0.4285	0.3952	0.8699	0.8790
				Posición Y							0.6576		0.9282		0.3576		0.8839	
											0.6175		0.9213		0.2897		0.8824	
											0.6463		0.9388		0.5051		0.8799	
11	16	8	1840052	Posición X	3600	6000	2000	250	50	14	0.6120	0.5775	0.9187	0.9190	0.3753	0.3570	0.8858	0.8840
				Posición Y							0.6290		0.8980		0.3309		0.8738	
											0.5664		0.9370		0.3288		0.8904	
											0.5028		0.9224		0.3931		0.8859	
12	16	8	1840052	Posición X	3600	7500	2500	250	50	16	0.5843	0.5709	0.9327	0.9349	0.3636	0.3018	0.8913	0.9042
				Posición Y							0.5037		0.9246		0.2968		0.8983	
											0.5875		0.9131		0.2593		0.8922	
											0.6080		0.9691		0.2874		0.9350	
16	16	8	1840052	Posición X	3600	9000	3000	250	50	19	0.5968	0.5697	0.9425	0.9425	0.2756	0.2620	0.9191	0.9216
				Posición Y							0.5865		0.9299		0.2525		0.9160	
											0.5718		0.9740		0.1990		0.9532	
											0.5235		0.9238		0.3208		0.8980	
17	16	8	1840052	Posición X	3600	10500	3500	250	50	22	0.4669	0.5974	0.9331	0.9243	0.2730	0.2601	0.9053	0.8974
				Posición Y							0.5751		0.9235		0.2683		0.8848	
											0.9226		0.9058		0.2594		0.8900	
											0.4249		0.9347		0.2398		0.9095	
18	16	8	1840052	Posición X	3600	12000	4000	250	50	24	0.5475	0.5462	0.9315	0.9328	0.2186	0.2318	0.9172	0.9120
				Posición Y							0.6009		0.9428		0.2434		0.9187	
											0.4645		0.9343		0.2251		0.9122	
											0.5720		0.9226		0.2402		0.8997	

Tabla 2. Resultados CNN

Tabla de resultados de Red Neuronal Recurrente

	Estructura de la red			Datos				Entrenamiento			Resultados							
	Capa de entrada	Capas ocultas	Capa de salida	Pesos	Tiempo	Entr	Val	Ep	BS	t/ ep	Entrenamiento				Validación			
											Pérdida		Precisión		Pérdida		Precisión	
1	LSTM - 32 n	-	Dense - f.a softmax	4612	50	12000	4000	250	50	9	0.8396 0.9522 0.8991 0.7964	0.8718	0.6209 0.5452 0.5731 0.6483	0.5969	0.6461 0.8331 0.6396 0.6738	0.6982	0.7252 0.6227 0.7304 0.7129	0.6978
2	LSTM - 32 n	-	Dense - f.a softmax	4612	100	12000	4000	250	50	17	1.0641 0.9945 0.9501 0.9415	0.9876	0.4623 0.5006 0.515 0.5307	0.5022	0.5403 0.7518 0.6500 0.5892	0.6328	0.7817 0.6749 0.7199 0.7534	0.7325
3	LSTM - 32 n	-	Dense - f.a softmax	4612	250	12000	4000	250	50	40	1.3234 1.3739 1.2208 1.3885	1.3267	0.331 0.3015 0.4218 0.2506	0.3262	0.7655 1.0012 0.7823 0.8909	0.8600	0.6597 0.5206 0.6607 0.5661	0.6018
4	LSTM - 32 n	-	Dense - f.a softmax	4612	500	12000	4000	250	50	80	1.2127 1.3869 1.213 1.3817	1.2986	0.4072 0.251 0.3965 0.257	0.3279	0.9135 0.9352 1.0012 1.0174	0.9668	0.5319 0.5234 0.5304 0.5029	0.5222
5	LSTM - 32 n	-	Dense - f.a softmax	4612	100	1500	500	250	50	3	0.7672 0.8726 0.8297 0.8682	0.8344	0.6547 0.5607 0.602 0.574	0.5979	0.8256 1.0258 0.9747 0.9988	0.9562	0.5980 0.5040 0.5260 0.5080	0.5340
6	LSTM - 32 n	-	Dense - f.a softmax	4612	100	3000	1000	250	50	5	0.6755 0.6562 0.6698 0.6903	0.6730	0.721 0.7333 0.7307 0.7087	0.7234	0.7565 0.6288 0.7475 0.6630	0.6990	0.6960 0.7440 0.6790 0.7290	0.7120
7	LSTM - 32 n	-	Dense - f.a softmax	4612	100	4500	1500	250	50	6	0.5469 0.6555 0.6156 0.7358	0.6385	0.7824 0.7358 0.75 0.6736	0.7355	0.5600 1.0463 0.6225 0.8408	0.7674	0.7713 0.5113 0.7307 0.664	0.6693

8	LSTM - 32 n	-	Dense - f.a softmax	4612	100	6000	2000	250	50	8	0.4754 0.5819 0.5085 0.6492	0.5538	0.8188 0.7675 0.8042 0.7278	0.7796	0.4893 0.6324 0.5856 0.6669	0.5936	0.8065 0.761 0.773 0.7195	0.7650
9	LSTM - 32 n	-	Dense - f.a softmax	4612	100	7500	2500	250	50	10	0.4684 0.5272 0.4034 0.4476	0.4617	0.8237 0.7921 0.8523 0.8281	0.8241	0.5407 0.5848 0.4335 0.4830	0.5105	0.7896 0.7624 0.8408 0.8088	0.8004
10	LSTM - 32 n	-	Dense - f.a softmax	4612	100	9000	3000	250	50	14	0.4431 0.5266 0.451 0.4585	0.4698	0.8343 0.7928 0.8367 0.8298	0.8234	0.4534 0.5410 0.6047 0.4559	0.5138	0.836 0.7853 0.7813 0.8343	0.8092
11	LSTM - 32 n	-	Dense - f.a softmax	4612	100	10500	3500	250	50	16	0.4351 0.4824 0.4007 0.4898	0.4520	0.8408 0.8153 0.8516 0.8141	0.8305	0.4486 0.5025 0.5121 0.5037	0.4917	0.8311 0.804 0.8203 0.8074	0.8157
12	LSTM - 32 n	-	Dense - f.a softmax	4612	100	12000	4000	250	50	18	0.3713 0.3866 0.3441 0.4052	0.3768	0.8665 0.8576 0.8738 0.8531	0.8628	0.3566 0.4508 0.3472 0.4489	0.4009	0.8677 0.8312 0.8775 0.8347	0.8528
16	LSTM - 64 n	-	Dense - f.a softmax	17924	750	12000	4000	250	50	2	0.3126 0.2865 0.3414 0.4195	0.3400	0.8862 0.8979 0.8792 0.848	0.8778	0.3215 0.3439 0.3304 0.4403	0.3590	0.8895 0.8852 0.8862 0.8317	0.8732
17	LSTM - 64 n	LSTM - 32	Dense - f.a softmax	30468	750	12000	4000	250	50	4	0.2903 0.2913 0.4565 0.2517	0.3225	0.8964 0.896 0.8331 0.9088	0.8836	1.0887 0.2879 0.4692 0.2722	0.5295	0.7632 0.8917 0.8275 0.9025	0.8462
18	LSTM - 128 n	LSTM - 32 n LSTM - 64 n	Dense - f.a softmax	130820	750	12000	4000	250	50	8	0.2435 0.244 0.2393 0.2382	0.2413	0.912 0.9122 0.9136 0.9145	0.9131	0.2533 0.2518 0.2443 0.2552	0.2512	0.907 0.9062 0.9105 0.9097	0.9084
19	LSTM - 256 n	LSTM - 128 n LSTM - 64 n	Dense - f.a softmax	528132	750	12000	4000	250	50	14	0.786 0.2317	0.3743	0.6392 0.9153	0.8444	0.9220 0.2614	0.4424	0.5894 0.9047	0.8222

		LSTM - 32 n									0.2332		0.9131		0.2809		0.903	
											0.2464		0.9101		0.3053		0.8915	
20	LSTM - 512 n	LSTM - 256 n	Dense - f.a softmax	2109188	750	12000	4000	250	50	34	0.2481	0.2547	0.9039	0.9044	1.6817	1.6022	0.5786	0.6466
		LSTM - 128 n									0.2408		0.9119		0.8365		0.801	
		LSTM - 64 n									0.2567		0.9046		0.2808		0.8965	
		LSTM - 32 n									0.2733		0.8973		3.6097		0.3101	

Tabla 3. Resultados RNN