Universidad de Alcalá
Escuela Politécnica Superior

MÁSTER EN INGENIERÍA INDUSTRIAL

Trabajo Fin de Máster

Detección de actividades anómalas en espacios públicos mediante redes neuronales profundas

Autor: Pedro López Miguel

Tutora: Marta Marrón Romera

2019

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Detección de actividades anómalas en espacios públicos mediante redes neuronales profundas

AUTOR: PEDRO LÓPEZ MIGUEL

TUTORA: MARTA MARRÓN ROMERA

TRIBUNAL:

PRESIDENTE: DANIEL PIZARRO PÉREZ

VOCAL 1º: DAVID FERNÁNDEZ BARRERO

VOCAL 2º: MARTA MARRÓN ROMERA

FECHA: .....

Resumen

Este Trabajo Fin de Máster se centra principalmente en diseñar e implementar un sistema de videovigilancia que permite, mediante el análisis de la información procedente de una secuencia de video, determinar comportamientos no habituales en contextos en los cuales la detección de una anomalía es de interés.

Se utiliza un aprendizaje semi-supervisado, estructurado en tres niveles. Un primer nivel, el cual utiliza un detector de personas basado en una red neuronal convolucional *(Faster-RCNN*). Un segundo nivel, el cual se centra en desarrollar una arquitectura recurrente para el reconocimiento de actividades mediante la utilización de redes de memoria de corto a largo plazo *(LSTM).* Y finalmente, se plantea un tercer nivel de detección de anomalías, el cual clasificaría la acción como anómala a partir de la obtención de las diferentes probabilidades de ocurrencia.

Palabras claves: Videovigilancia, detección de anomalías, red neuronal, *LSTM*, *Faster-RCNN*.

Abstract

The aim of this Master's Thesis will be mainly focused on designing and implementing a video-surveillance system that allows, through the analysis of information from a video sequence, to determine unusual behaviors under situations where detection of an anomaly would be interesting.

Semi-supervised learning will be used and structured in three different levels. A first one level, which uses a person detector based on a convolutional neuronal network (*Faster-RCNN*). A second one level, which focuses on developing a recurrent architecture for the recognition of activities through the use of *Long Short-Term Memory* networks (*LSTM*). Finally, a third level of anomalies detection is introduced, which classifies the action as anomalous from the obtaining of the different probabilities of occurrence.

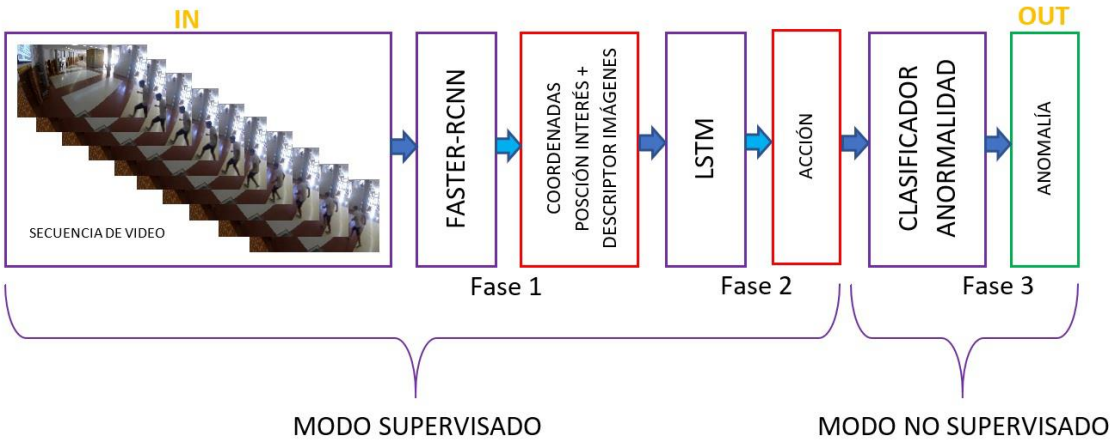
Keywords: Video surveillance, anomalies detection, neural network, *LSTM*, *Faster-RCNN*.

Resumen extendido

El sistema de vídeo vigilancia implementado permite, de forma automática, comprender lo que está sucediendo en una determinada escena de interés, observando las actividades habituales o no habituales que tienen lugar para poder determinar comportamientos o situaciones anómalas, mediante el análisis de la información procedente principalmente de una secuencia de video.

Las anomalías en vídeos de vigilancia se podrían describir como eventos u objetos con baja probabilidad de ocurrencia, pues se trata de actos no habituales en los escenarios de interés que intentan comprometer la seguridad de las personas en ellos.

En este trabajo se utiliza un aprendizaje semi-supervisado, estructurado en tres niveles, tal como se puede visualizar en la figura siguiente.



Esquema general del sistema de detección de anomalías

Un primer nivel, el cual utiliza un detector de personas basado en una red neuronal convolucional *(Faster-RCNN*). Un segundo nivel, el cual se centra en desarrollar una arquitectura recurrente para el reconocimiento de actividades mediante la utilización de redes de memoria de corto a largo plazo *(LSTM).* Y finalmente, un tercer nivel de detección de anomalías, el cual realiza la clasificación de la acción como anómala mediante la obtención de las diferentes probabilidades de ocurrencia.

El módulo de detección de personas es el primer elemento del sistema de vídeo vigilancia que se ha desarrollado. Su objetivo ha sido detectar en que zonas de la imagen es posible que se encuentre una persona u objeto, para posteriormente determinar la acción desarrollada por la misma.

El detector de personas implementado se basa en la utilización de redes convolucionales. Este tipo de redes, han demostrado ser muy eficientes en tareas de *Image Understanding*, tanto en la clasificación de imágenes, como en la detección de objetos y personas. En concreto, para la detección de personas se utiliza una variante denominada *Faster-RCNN*, la cual se divide principalmente en tres etapas:

- En la primera etapa, se utiliza una red entrenada para obtener las regiones de interés, *Region Porposal Network (RPN).*
- La segunda, consiste en una gran red neuronal convolucional que extrae un vector de características de longitud fija para cada región.
- Y, por último, la tercera etapa suele consistir en un clasificador por regresión lineal que permita validar las detecciones.

En las pruebas realizadas con este detector se han obtenido unas métricas bastante aceptables (mAP=45%), aunque a nivel de prueba con situaciones reales su funcionamiento muestra un comportamiento bastante mejor que el indicado por los valores de las métricas resultantes.

Para el segundo nivel, se ha utilizado un módulo de predicción basado en redes neuronales recurrentes (RNN), que permiten la identificación de una determinada acción realizada por una persona a lo largo de una secuencia de video. En concreto se han utilizado para este trabajo, las redes de memoria a corto y largo plazo o *Long-Short Term Memory* (*LSTM*), que es una versión mejorada de la típica *RNN* y se han comprobado que son la solución más efectiva para resolver los problemas referentes a la predicción con secuencias. Las redes *LSTM* tienen una ventaja sobre las redes neuronales clásicas y las redes *RNN*, y es que tienen la propiedad de recordar selectivamente los patrones por largos períodos de tiempo. Esto es una ventaja, ya que parte de muchas acciones, para ser identificadas dependen de las características ocurridas en un instante de tiempo anterior. De hecho, la información proporcionada por los *frames* anteriores de una secuencia de video, permiten identificar si la acción se trata de una persona que está corriendo o simplemente andando, o se ha caído y se está levantando o simplemente se está levantando de un asiento, es decir, conocer el pasado es uno de los principales factores decisivos para las predicciones en secuencias temporales. Uno de los problemas de las redes neuronales tradicionales es que predicen sin tener en cuenta lo ocurrido anteriormente, ya que todos los casos de prueba se consideran independientes unos de otros, el modelo se ajusta para un instante de tiempo particular, sin considerar lo ocurrido en instantes de tiempo anteriores, es decir, carecen de memoria, por lo que no son muy útiles si se quiere analizar una secuencia temporal de imágenes. Esta dependencia del tiempo se logra a través de las redes *LSTM*.

Las evaluaciones realizadas con este tipo de redes *LSTM* utilizando como secuencias de pruebas videos correspondientes a los *dataset KTH* o *Weizmann* han dado resultados muy satisfactorios llegando incluso a obtener precisión del 100% con determinadas secuencias.

Finalmente, en el tercer nivel se plantea una técnica de detección de anomalías utilizando un método estadístico simple, no supervisado, para determinar si un suceso es ánomalo. Se analizan las acciones predichas previamente, que transcurren en una determinada secuencia, y dependiendo del contexto o escenario en el cual transcurren, se determina si se trata de un suceso anómalo.

Contenido

Índice de figuras

Índice de tablas

Glosario

ACF

AggreGate Channel Features, 7, 8

CNN

Convolutional Neural Network, I, III, 2, 8, 12, 15, 19, 20, 21, 22

Faster-RCNN

Faster Region-based Convolutional Neural Network, I, III, IV, 2, 7, 8, 23

Fast-RCNN

Fast Region-based Convolutional Neural Network, 2, 8, 21, 22, 23

LRCN

Long-term Recurrent Convolutional Networks, 62, 63, 65, 66, 88

LSTM

Long-Short Term Memory, I, III, IV, 3, 4, 26, 28, 29, 30, 32, 35, 36

PCA

Principal Component Analysis, 5

R-CNN

Region-based Convolutional Neural Network, 2, 7, 8, 18, 22

ReLU

Rectified Linear Unit, 14, 16, 17

RNN

Recurrent Neural Network, IV, 3, 26, 27, 28, 29, 36

RPN

Region Proposal Network, IV, 2, 23

SDG

Stochastic Gradient Descent, 28

SVM

Support Vector Machines, 5, 21

Capítulo 1

Introducción

1.1 *Motivación*

El Trabajo Fin de Máster (TFM) que se propone se encuadra dentro del proyecto HEIMDAL [1], del grupo GEINTRA [2] de la Universidad de Alcalá, y se centra principalmente en realizar el análisis, diseño e implementación de sistemas de vídeo vigilancia que permitan, de forma automática, comprender lo que está sucediendo en una determinada escena de interés, observando las actividades habituales o no habituales que tienen lugar para poder determinar comportamientos o situaciones anómalas, mediante el análisis de la información procedente principalmente de una secuencia de video.

Estas escenas de interés abarcan tanto los espacios interiores como exteriores, entre los que se pueden encontrar puntos calientes de ciudades, entradas o alrededores de estadios o instalaciones deportivas y de ocio y recreo, trazados ferroviarios (vías, pasos a nivel y entradas y salidas de túneles), accesos a medios de transporte (tren, avión, barco), edificios públicos, y en general cualquier lugar en el que la detección de una anomalía sea de interés para la seguridad.

Este trabajo pretende contribuir al contexto descrito con una propuesta en la que se apliquen técnicas de aprendizaje mediante de redes neuronales profundas, siendo este un campo que ha cobrado notable fuerza en el mundo de la visión por computador en los últimos años, debido principalmente al avance que han experimentado los sistemas de computación, que permiten procesar grandes cantidades de información en espacios cortos de tiempo.

Este TFM trata, por tanto, de un tema de investigación práctico y en el que actualmente la comunidad científica y las empresas de ingeniería están dedicando importantes esfuerzos económicos.

1.2 *Objetivos*

Las anomalías en vídeos de vigilancia se podrían describir como eventos u objetos con baja probabilidad de ocurrencia, pues se trata de actos no habituales en los escenarios de interés que intentan comprometer la seguridad de las personas en ellos.

Varios son los trabajos científicos que intentan abordar este problema mediante la utilización de redes neuronales profundas o *Deep Learning*, ya sea mediante aprendizaje supervisado como no supervisado [3] [4] [5].

En este trabajo, se pretende utilizar un aprendizaje semi-supervisado, estructurado en tres niveles, tal como se puede visualizar en la Figura 1. La metodología en tres niveles se basa en tres fases cada una con un objetivo diferente.

Una primera fase, que se engloba dentro de la parte supervisada, en la cual se utiliza un detector de personas u objetos basado en una red neuronal convolucional o *Convolutional Neural Network (CNN)*, como es el detector *Faster Region-based Convolutional Neural Network (Faster-RCNN*) [6]. Se han elegido estos modelos basados en redes neuronales convolucionales por ser los más utilizados actualmente para tareas de interpretación de imágenes, siendo los detectores de tipo *Region-based Convolutional Neural Network (R-CNN)* los más efectivos para tareas de detección de personas u objetos.

El detector *Faster-RCNN* presenta una arquitectura, que se detallará en el capítulo 3 de esta memoria, que tiene como principal característica que fusiona las redes *Region Proposal Network (RPN)* y *Fast-RCNN* en una única red, compartiendo sus características convolucionales y reduciendo el tiempo de ejecución. La *RPN* está entrenada para generar propuestas de regiones de interés, que son utilizadas por la red *Fast-RCNN* para la detección de objetos. Este detector es completamente convolucional y predice las coordenadas que delimitan las personas u objetos de interés en la imagen, asignando una probabilidad de acierto por cada objeto detectado. Estos límites serán útiles para determinar la zona de la imagen de la cual se quiere identificar la acción para posteriormente evaluar si se trata de una anomalía.

En el capítulo 5, se realizará una descripción detallada del proceso de entrenamiento de este tipo de detector. Debido a la gran cantidad de datos que capturan los sistemas de video vigilancia, solo una pequeña parte de ellos será utilizada durante el entrenamiento, ya que la totalidad de la misma implicaría una gran cantidad de tiempo para su procesamiento. Como resultado de esta merma de información, se obtendrá a una falta de precisión en el entrenamiento de la red, por ello, es necesario realizar un reentrenamiento, también denominado *Fine Tunning* de una red neuronal con modelo pre-entrenado de redes neuronales profundas conocido entre la comunidad científica y entrenado para el reconocimiento de objetos como es *Alexnet* [7], del que se modificará parte de su arquitectura para adaptarla a los requerimientos de la aplicación y facilitar el entrenamiento requerido para obtener una fiabilidad adecuada.

Una segunda fase, la cual se centra en desarrollar una arquitectura recurrente adecuada para el aprendizaje visual con el objetivo de ser aplicada a tareas de reconocimiento de

actividades en video vigilancia, pudiendo obtener una descripción del contexto, dentro de lo que la comunidad científica ha dado en llamar *Image Understanding*, que permita determinar la existencia de una anomalía [8]. El objetivo es determinar el comportamiento del objeto de interés mediante la utilización de redes de memoria de corto a largo plazo o *Long-Short Term Memory (LSTM)*, este tipo de redes neuronales recurrentes o *Recurrent Neural Network (RNN)* son muy utilizadas en tareas de predicción de secuencia [9]. Para su entrenamiento se usarán diferentes bases de datos de interés en el contexto del reconocimiento de acciones, tanto públicas (KTH [10], *Weizzman* [11]) como propias (GBA [12]). La red neuronal recurrente será entrenada a partir de descriptores obtenidos utilizando diferentes que técnicas permitirán obtener las características de determinados patrones de comportamiento (acción de correr, saltar, etc.) para posteriormente incluir un clasificador estadístico que permita detectar la anomalía. En el capítulo 5, se describen estas técnicas, que requieren la selección de descriptores adecuados, imágenes de alta resolución y grandes volúmenes de datos.

Finalmente, una tercera fase de detección de anomalías, que se engloba dentro de la parte no supervisada del sistema propuesto, en la cual se propone realizar la clasificación del comportamiento o acción como anómalo, sabiendo que los eventos anómalos ocurren con baja frecuencia respecto a un modelo de normalidad. Por lo tanto, se puede establecer cuál es la normalidad asociada a un escenario concreto mediante la obtención de las diferentes probabilidades de ocurrencia de las acciones. Por este motivo, en un determinado escenario concreto, como pueden ser centros comerciales, aeropuertos, etc., es decir, espacios de pública concurrencia, se puede obtener un patrón de comportamiento que determine si una acción puede ser considerada una anomalía, siendo este el objetivo a desarrollar en la última fase del sistema propuesto.

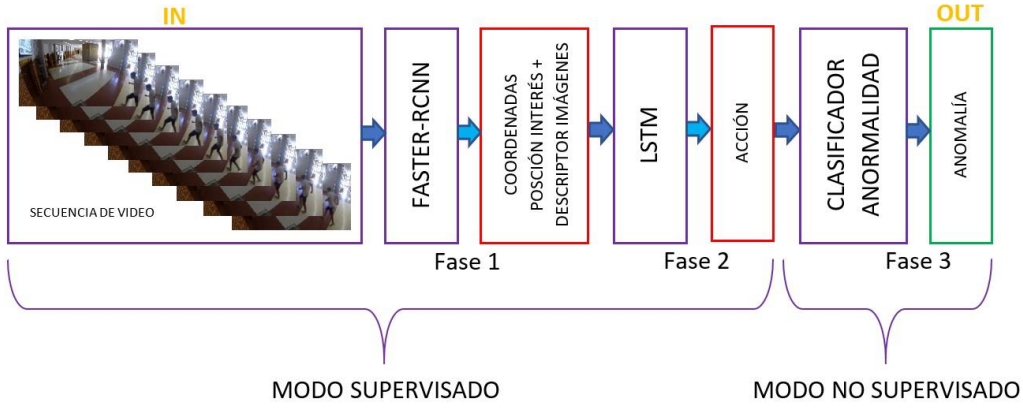


Figura 1 Esquema general del sistema de detección de anomalías

1.3 *Organización de la Memoria*

Esta memoria se ha estructurado de tal modo que permita al lector llegar a entender el proceso global del modelo desarrollado en este Trabajo Fin de Máster, incluyendo todas y cada una de las partes necesarias para poder llegar a un completo entendimiento de este y

poder en un futuro implementar otros modelos basados en el mismo concepto o en segmentos de la arquitectura planteada.

En el primer capítulo, se ha descrito brevemente la finalidad del trabajo realizado, mencionando a modo de introducción los trabajos previos que han servido como base en el desarrollo de este, y se ha presentado la estructura completa del sistema.

En el segundo capítulo, se realiza una breve descripción de las técnicas empleadas para el desarrollo del sistema que se plantea en este trabajo.

En el tercer capítulo**,** se realiza una descripción de los conceptos teóricos que involucran el desarrollo de las diferentes fases propuestas para la realización del sistema de detección de anomalías. Primeramente, se describen las redes neuronales convolucionales como preámbulo de las redes neuronales profundas, con el objetivo de llegar a entender el funcionamiento de los detectores de personas y objetos, basados en este tipo de redes, que son empleados en este trabajo. Se continúa con una descripción detallada del comportamiento de las redes neuronales recurrentes, particularizando para el caso de las redes de memoria de corto a largo plazo (*LSTM*), que serán utilizadas para la predicción de las acciones ocurridas en las secuencias de video a analizar. Y finalmente, se realiza una descripción del método estadístico simple empleado para interpretar si las acciones predecidas pueden ser consideras como una anomália en función del contexto o escenario en el cual transcurren.

En el cuarto capítulo, se realiza una breve descripción de las bases de datos de videos e imágenes utilizadas para la realización de los entrenamientos. También se describen las métricas que se emplerarán para realizar el cálculo de la precisión de los resultados que se obtendrán en las diferentes pruebas realizadas.

En el quinto capítulo, se describe el desarrollo completo de las arquitecturas de los modelos planteados.

En el capítulo sexto, se muestran los resultados obtenidos en la evaluación de las diferentes arquitecturas planteadas indicando los parámetros empleados en la realización de los entrenamientos.

Finalmente, en el séptimo capítulo, se exponen las conclusiones obtenidas en base a los resultados y se proponen nuevas ideas a realizar en trabajos futuros.

Capítulo 2

Contexto

2.1 *Introducción*

Muchos de los avances en el reconocimiento de voz, el procesado de lenguaje natural o la visión por computador han sido posibles gracias a una técnica denominada aprendizaje profundo o *Deep Learning*, siendo esta técnica parte de una de las áreas de la inteligencia artificial conocida como aprendizaje automático o *Machine Learning*.

El *Machine Learning* permite crear sistemas que utilizan algoritmos para analizar los datos, y predecir comportamientos futuros de manera automática. Es decir, tienen la capacidad de aprender automáticamente con el fin de encontrar patrones que permitan construir un modelo para predecir y clasificar los elementos de entrada.

Los algoritmos utilizados se agrupan, en general, en tres grandes categorías: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo o *Reinforcement Learning*.

El aprendizaje es supervisado cuando los datos utilizados para el entrenamiento incluyen la solución deseada, llamada etiqueta o *label*. Algunos de los algoritmos más populares de *Machine Learning* en esta categoría son la regresión lineal, la regresión logística, las *Support Vector Machines (SVM)*, los árboles de decisión y las redes neuronales.

Sin embargo, en el aprendizaje no supervisado, los datos de entrenamiento no incluyen las etiquetas, y será el algoritmo el que intentará clasificar la información por sí mismo. Algunos de los algoritmos más conocidos de esta categoría son *clustering (K-means)* o *Principal Component Analysis (PCA)*.

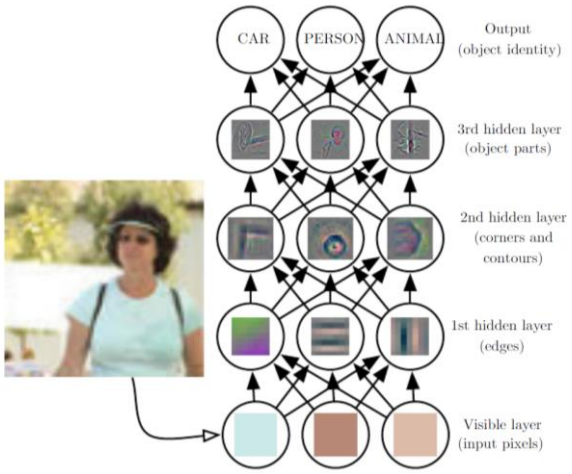
El algoritmo de aprendizaje por refuerzo es aquel en el que el algoritmo de aprendizaje recibe algún tipo de valoración acerca de la exactitud de la respuesta dada. Cuando la respuesta es correcta, el aprendizaje por refuerzo es similar al aprendizaje supervisado, pues en ambos casos, se recibe información acerca de lo precisa que ha sido la respuesta.

Sin embargo, cuando la respuesta es errónea el aprendizaje supervisado proporciona información sobre lo que debería haber respondido, mientras que el aprendizaje por refuerzo solo informa acerca de que el comportamiento no es el adecuado y cuánto error se ha cometido. Esta aproximación, la del aprendizaje por refuerzo, está inspirada en la propia naturaleza, en la forma en la que personas y animales aprenden.

2.2 *Redes neuronales artificiales y Deep Learning*

Un caso especial de algoritmos de *Machine Learning* son las redes neuronales artificiales. Su funcionalidad se puede considerar como el de las neuronas humanas y su capacidad para la obtención de resultados. En el caso concreto de *Deep Learning,* los modelos están compuestos de múltiples capas de neuronas utilizadas para aprender representaciones de datos, con múltiples niveles de abstracción que realizan una serie de transformaciones lineales y no lineales, y que, a partir de los datos de entrada, generan una salida próxima a la esperada. El aprendizaje supervisado, en este caso, consiste en obtener los parámetros de esas transformaciones (los pesos W_i y el sesgo b de la red), y consigue que esas transformaciones sean óptimas, es decir, que la salida producida y la esperada difieran muy poco.

El *Deep Learning* maneja redes neuronales artificiales con múltiples capas, que están apiladas una encima de la otra, de ahí el término profundo o *deep*, donde cada una de ellas, está a su vez compuesta por una gran cantidad de neuronas. Cada neurona se define con sus parámetros (pesos W_i y el sesgo b) que, a su vez, realiza una transformación simple de los datos que recibe de la salida de la capa anterior para pasarlos a las de la capa posterior. La unión de todas las capas permite obtener el patrón que representa al modelo buscado.



Fuente: Deep Learning Book (Goodfellow, 2016)

Figura 2 Ejemplo de modelo de aprendizaje profundo

En la Figura 2, se muestra un ejemplo de un modelo de aprendizaje profundo, en el que el modelo divide el conjunto de valores de pixeles que representan la imagen de entrada, en una serie de mapas más simples de forma anidada, cada uno generado por una capa

diferente del modelo. La entrada se corresponde con una capa visible, la cual contiene las variables que son observadas. A continuación, una serie de capas ocultas se encargan de extraer características cada vez más abstractas de la imagen. Estas capas se llaman ocultas porque sus valores de entrada no se dan en los datos proporcionados; en su lugar, el modelo debe determinar las relaciones existentes en los datos observados. Por ejemplo, en la primera capa oculta se puede identificar fácilmente los bordes que contienen los objetos de la imagen, comparando el brillo de los píxeles vecinos. Dada la descripción de los bordes de la primera capa oculta, la segunda capa oculta puede buscar fácilmente esquinas y contornos que dan información de las siluetas. A partir de estos datos, la tercera capa oculta puede detectar partes enteras de objetos específicos. Y finalmente, con estas partes del objeto se puede reconocer los objetos completos presentes en la imagen.

Uno de los problemas principales de *Deep learning*, como se verá más adelante en este trabajo, es la dificultad de definir la configuración del modelo que se pretenden entrenar. Al existir diferentes capas, que sirven para diferentes propósitos, y con una gran variedad de parámetros a definir, los cuales tienen una alta relevancia en el resultado final, se requieren muchas horas de aprendizaje y práctica para obtener unos resultados óptimos.

2.3 *Detectores de objetos y personas*

La detección de personas es el primer paso del sistema de vídeo vigilancia que se pretende desarrollar. La dificultad en la detección de personas, las cuales se pueden englobar como si fueran un determinado objeto, se encuentra, principalmente, en determinar un modelo genérico de estas, debido a la variabilidad mostrada en las imágenes a procesar, ya que en ellas se pueden llegar a mostrar diferentes puntos de vista de la cámara, de posturas de los actores, movimientos solapados, además del propio comportamiento producido por interacción entre las personas y objetos.

El objetivo en este TFM es detectar en que zonas de la imagen es posible que se encuentre una persona u objeto, para posteriormente, como se describirá más adelante, determinar la acción desarrollada por la misma.

Existen diferentes algoritmos de detección de personas. Durante el desarrollo de este TFM se ha trabajado utilizando principalmente dos, el detector *AggreGate Channel Features (ACF*), y el detector *Faster Region-based Convolutional Neural Network (Faster-RCNN).*

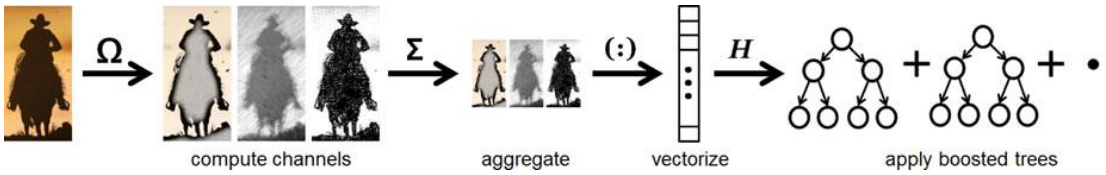
Cada uno de estos detectores emplea diferentes tipos de modelos que fijan las características que deben tener las regiones candidatas para poder considerarlas una persona u objeto. Establecen dos fases, una de detección de las posibles regiones de interés y otra del propio modelado de personas u objetos.

Estos dos detectores obtienen las posibles regiones de interés mediante búsqueda exhaustiva. Este tipo de búsqueda suelen dar buenos resultados y consiste en, mediante una ventana con un tamaño determinado, ir recorriendo la imagen de entrada y comparar la zona correspondiente a la ventana con una imagen de persona delimitada por las coordenadas correspondientes a sus cuatro vértices (*bounding box*). Estas coordenadas son

proporcionadas por el etiquetado de la imagen, denominado *Ground Truth*. De este modo, se obtiene una gran cantidad de regiones de interés con determinados grados de parecido.

El algoritmo de detección *AggreGate Channel Features (ACF)* [13], es un detector que realiza una búsqueda exhaustiva en las imágenes. Dada una imagen de entrada *I*, se calcularán diferentes canales, *C = Ω(I)*. Posteriormente, todos los bloques de píxeles en C se sumarán y se suavizarán realizando un sub-muestreado. Las características vienen dadas entonces por la búsqueda de los píxeles en los diferentes canales agregados. Finalmente, se realiza el entrenamiento y se utilizan árboles de decisión sobre estas características para distinguir el objeto del fondo utilizando ventanas multi-escala deslizantes [14].

En la Figura 3 se muestra un esquema de los diferentes pasos que realiza el detector *ACF*.



Fuente: Fast Feature Pyramids for Object Detection (Dollár, Appel, Belongie, & Perona, 2014)

Figura 3 Esquema de funcionamiento del detector *ACF*

Aunque el detector *ACF*, obtiene una buena precisión en la detección de personas, el objetivo principal de este trabajo es utilizar un enfoque diferente, por lo que se centrará en la utilización de algoritmos basados en redes convolucionales o *CNN*. Los sistemas o detectores de personas basados en redes convolucionales hacen uso del aprendizaje profundo o *Deep Learning* para la extracción y selección de las características discriminantes de las personas y su modelado. Un ejemplo de estos detectores son *Region-based Convolutional Neural Network (R-CNN), Fast Region-based Convolutional Neural Network (Fast-RCNN)* y *Faster Region-based Convolutional Neural Network (Faster-RCNN)*, siendo estos métodos de detección mucho mejores y con un rendimiento significativamente mayor que los basados en en detectores de tipo *ACF* [15], como se verá más adelante en las pruebas realizadas de evaluación de los detectores (Figura 68)

2.4 *Detección de anomalías*

La detección de anomalías utiliza técnicas que se usan para identificar patrones inusuales que no se ajustan al comportamiento esperado.

Las anomalías se pueden clasificar como [16]:

- **Anomalías puntuales:** un único suceso o acción puede ser considerado anómalo si su comportamiento difiere demasiado del resto. En una secuencia de video donde los sucesos que ocurren muestran comportamientos típicos, pero de repente aparece un suceso o acción (p.e., la caída de una persona) que difiere mucho de las acciones que ocurren normalmente en un determinado entorno, esto se considerará como una anomalía puntual.

- **Anomalías contextuales**: la anormalidad es específica del contexto. Este tipo de anomalía es común en los datos de series temporales. Un ejemplo que puede reflejar este tipo de anomalías es el de un grupo de personas corriendo. Si sucede en un entorno en el cual este tipo de acción es común, como puede ser un parque, se puede considerar como un evento normal, pero si este hecho ocurre en un recinto cerrado como un aeropuerto se podría considerar una anomalía
- **Anomalías colectivas:** Si una colección de instancias de datos relacionados es anómala con respecto a todo el conjunto de datos, se denomina como una anomalía colectiva. Las instancias de datos individuales en una anomalía colectiva pueden no ser anomalías por sí mismos, pero su aparición juntas como una colección es anómala. Por ejemplo, si una persona está corriendo en un aeropuerto, podría ser un considerado un suceso normal, ya que la persona simplemente corre porque puede perder un vuelo, pero cuando es un grupo de personas corriendo la situación cambia y se podría considera como una anomalía colectiva.

La detección de anomalías puede ser abordada de muchas maneras dependiendo de la naturaleza de los datos y de las circunstancias. A continuación, se resume una clasificación de algunas de esas técnicas.

- **Método basado en reglas estáticas**

La idea es identificar una lista de anomalías conocidas y luego escribir reglas para detectar esas anomalías. La identificación de las reglas se realiza en función del conocimiento previo de los sucesos en función del contexto en el que ocurren.

Las reglas estáticas se utilizan con la hipótesis de que las anomalías siguen la regla 80/20, donde la mayoría de las ocurrencias no son anómalas.

Los métodos basados en reglas estáticas no son óptimos y la identificación de esas reglas es a menudo una tarea compleja y subjetiva. Por lo tanto, el enfoque estadístico o basado en el *Machine learning* o aprendizaje automático, que aprende automáticamente las reglas generales, es preferible a las reglas estáticas.

- **Métodos estadísticos simples**

El enfoque más simple para identificar irregularidades en un conjunto de sucesos es marcar los sucesos que se desvían de las propiedades estadísticas comunes de una distribución probabilística.

Se podría definir que un suceso es considerado anómalo si se desvía en relación con la desviación estándar de la media del conjunto total de sucesos.

- **Enfoques basados en *Machine Learning***

A continuación, se presenta una breve descripción de las técnicas más utilizadas, basadas en el *machine learning,* para la detección de anomalías.

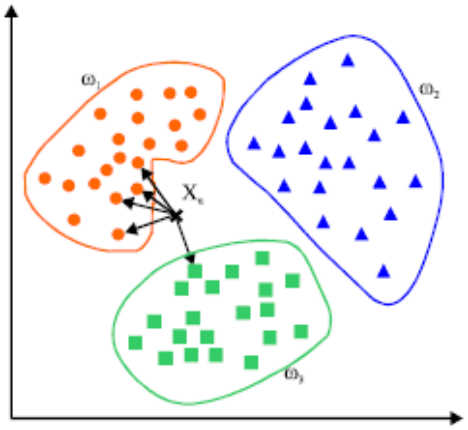
  - **Algoritmo *k-Neighbors***

El algoritmo clasifica cada suceso en el grupo que corresponda, según tenga *k* vecinos más cerca de un grupo o de otro (Figura 4). Es decir, calcula la distancia del elemento nuevo

a cada uno de los existentes, y ordena dichas distancias de menor a mayor para ir seleccionando el grupo al que pertenece. Por tanto, el grupo de mayor frecuencia será el que tenga menores distancias.

K-Nearest-Neighbor es un algoritmo de tipo supervisado basado en instancia, esto quiere decir que el algoritmo no aprende explícitamente un modelo sino que memoriza las instancias de entrenamiento que son usadas para la fase de predicción.

Puede usarse para clasificar nuevas muestras (valores discretos) o para predecir (regresión, valores continuos). Se utiliza para clasificar valores buscando los puntos de datos más similares (por cercanía, mediante la distancia Eucledia o una medida similar dependiente del tipo de los datos) aprendidos en la etapa de entrenamiento y haciendo conjeturas de nuevos puntos basado en esa clasificación. Los puntos de datos normales ocurren alrededor de un vecindario denso y las anomalías están lejos.



Fuente: Mathworks

Figura 4 Ejemplo K-Neighbors basado en distancia euclidea

- ***Clustering***

La agrupación de detección de anomalías basada en *clustering* es uno de los conceptos más populares en el aprendizaje sin supervisión. Los puntos de datos similares tienden a pertenecer a grupos o *clusters* similares, esta clasificación viene determinada por la distancia de sus centroides.

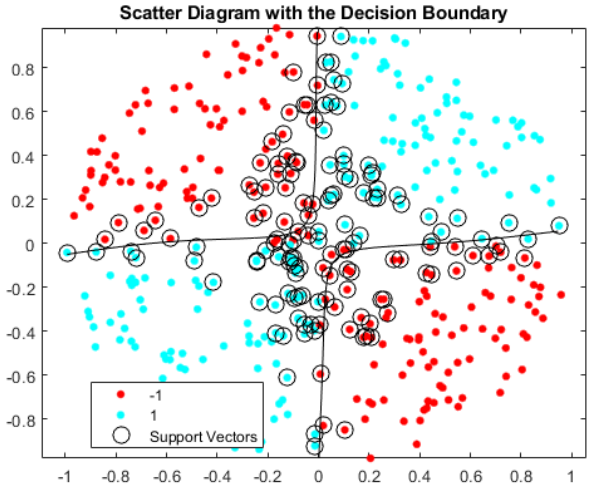
K-means es un algoritmo de *clustering* ampliamente utilizado. Crea *clusters* similares de k de puntos de datos. Las instancias de datos que caen fuera de estos grupos podrían ser potencialmente marcadas como anomalías.

- **SVM**

Una máquina de vectores de soporte o *Support Vector Machines* (SVM) es un algoritmo de aprendizaje supervisado que se puede emplear para clasificación binaria o regresión. Las máquinas de vectores de soporte son muy populares en aplicaciones como el

procesamiento del lenguaje natural, el habla, el reconocimiento de imágenes y la visión artificial.

El objetivo de este tipo de algoritmos de aprendizaje supervisado es dado un conjunto de datos de entrenamiento con sus etiquetas de clase, entrenar para construir un modelo que prediga la clase de una nueva muestra o de un conjunto de test. Consiste en construir un hiperplano en un espacio de dimensionalidad muy alta (o incluso infinita) que separe las clases existentes. Una buena separación entre las clases permite una clasificación correcta de la nueva muestra, es decir, se necesita encontrar la máxima separación a los puntos más cercanos a este hiperplano (Figura 5).



Fuente: Mathworks

Figura 5 Ejemplo del hiperplano que separa la muestra en dos clases

Capítulo 3

Marco Teórico

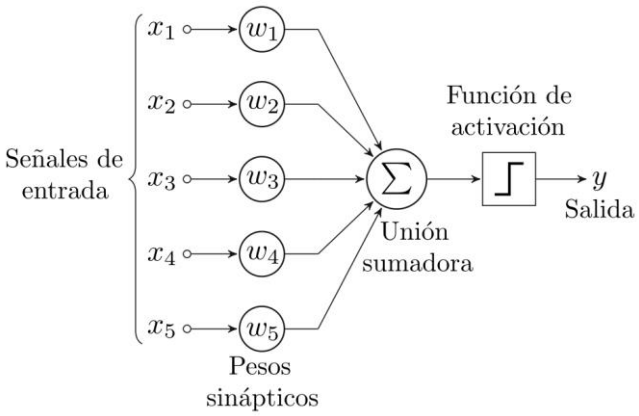
3.1 *Introducción a las redes neuronales convolucionales*

Una red neuronal convolucional o *Convolutional Neural Network (CNN)* es un caso concreto de redes neuronales profundas, que fueron ya usadas a finales de los 90, pero que en estos últimos años se han popularizado enormemente al ser muy utilizadas en tareas de visión por ordenador. Estas redes están demostrando que para el reconocimiento de objetos en imágenes, son las más robustas y exitosas que hay por ahora, como se demostró en el 2012, cuando una CNN ganó el desafío de reconocimiento de objetos de *Imagenet Large Scale Visual Recognition Challenge* [17], momento a partir del cual solo se han logrado mejoras [7].

3.1.1 Arquitectura básica

Una red neuronal convolucional o *CNN*, es muy similar a una red neuronal ordinaria y funciona de forma análoga a como lo haría el sistema visual primario en el ser humano.

Un rasgo diferencial de las *CNN* respecto a las redes neuronales ordinarias es que sus entradas son imágenes, esto permite codificar ciertas propiedades en la arquitectura para reconocer elementos concretos en las imágenes. Se puede decir que es un tipo de red neuronal para el procesado de datos que vienen en forma de múltiples matrices, como por ejemplo los datos de una imagen que conforman una matriz formada por pixeles.



Fuente: https://es.wikipedia.org/wiki/Perceptr%C3%B3n

Figura 6 Diagrama de una red neuronal básica. Perceptrón

Igual que ocurre con las redes neuronales ordinarias (Figura 6), una red de tipo *CNN* se compone de neuronas que tienen pesos y sesgos que pueden aprender. Cada neurona recibe algunas entradas, realiza un producto escalar y luego aplica una función de activación. Al ser explícitamente las entradas imágenes, permiten codificar ciertas propiedades en la arquitectura; permitiendo ganar en eficiencia y reducir la cantidad de parámetros en la red.

Las redes neuronales convolucionales vienen a solucionar el problema de que las redes neuronales ordinarias no trabajan bien para imágenes de mucha definición; las redes neuronales convolucionales trabajan modelando de forma consecutiva pequeñas piezas de información para luego combinar esta información en las capas más profundas de la red.

Una manera de entenderlas es que la primera capa intentará detectar los bordes y establecer patrones de detección de bordes. Luego, las capas posteriores trataran de combinarlos en formas más simples y, finalmente, en patrones de las diferentes posiciones de los objetos, iluminación, escalas, etc. Las capas finales intentarán hacer coincidir una imagen de entrada con todos los patrones y obtener una predicción final como una suma ponderada de todos ellos. De esta forma las redes neuronales convolucionales van aprendiendo diferentes niveles de abstracción, lo que permite que utilizando redes con muchas capas se puedan conseguir identificar estructuras más complejas en los datos de entrada dando predicciones bastantes precisas.

Al igual que en las redes neuronales, no existe un número fijo de capas que se deben utilizar, esto es una elección que se considera de acuerdo con el problema a resolver. Se debe tener en cuenta que una red excesivamente profunda y con pocos datos puede no llegar a aprender nada y de la misma forma cuando se tiene gran cantidad de datos suele ser conveniente crear una mayor profundidad (mayor número de filtros, capas, etc.).

En general, las redes neuronales convolucionales van a estar construidas con una estructura que contendrá cuatro tipos distintos de capas:

1. Operación de convolución o *Convolution*
2. Capa rectificadora o *ReLU*
3. Capa de submuestreo o *Pooling*
4. Capa totalmente conectada o *Fully connected*

a. Operación de convolución

Lo que distingue a las redes neuronales convolucionales de cualquier otra red neuronal es que utilizan una operación de máscara llamada convolución en alguna de sus capas, en lugar de utilizar la multiplicación de matrices que se aplica generalmente. La máscara de convolución recibe como entrada una imagen y luego aplica sobre ella un filtro o *kernel,* que devuelve un mapa de las características de la imagen original, que además reduce el tamaño de la imagen de entrada conservando la relación espacial entre los pixeles.

El propósito principal de una capa convolucional es detectar características o rasgos visuales en las imágenes como aristas, líneas, gotas de color, etc. Esta es una propiedad muy interesante, porque una vez entrenada la red para aprender una característica en un punto de la imagen se puede reconocer después en cualquier parte de esta.

Una característica importante es, por tanto, que las capas convolucionales pueden aprender jerarquías espaciales de patrones preservando las relaciones espaciales, por ejemplo, una primera capa convolucional puede aprender elementos básicos (como aristas), y una segunda capa convolucional puede aprender patrones compuestos de elementos básicos aprendidos en la capa anterior, y así sucesivamente, hasta ir aprendiendo patrones muy complejos. Esto permite que las redes neuronales convolucionales aprendan eficientemente conceptos visuales cada vez más complejos y abstractos.

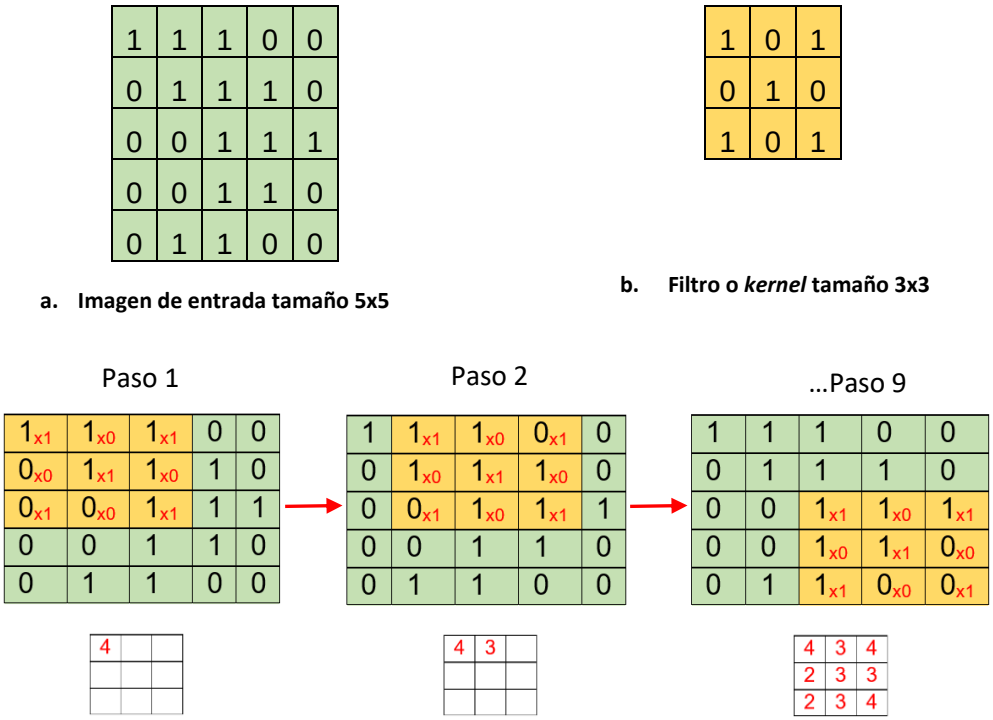
En general, las capas convoluciones operan generando vectores, llamados *feature maps*, con dos ejes espaciales de altura y anchura (*height* y *width*), además de un eje de canal (*channels*) también llamado profundidad (*depth*). Para una imagen de color RGB, la dimensión del eje *depth* es 3, pues la imagen tiene tres canales: rojo, verde y azul (*red, green* y *blue*). Para una imagen en blanco y negro la dimensión del eje *depth* es 1 (nivel de gris).

En el ejemplo siguiente se explica cómo funciona la convolución aplicada a imágenes. Cada imagen puede considerarse como una matriz de valores de los píxeles.

Considere una imagen de 5 x 5 (Figura 7-a), cuyos valores de los píxeles son sólo 0 y 1 (en una imagen real, por ejemplo, en escala de grises, el rango de valores de píxel comprendería entre 0 y 255):

Considere también otra matriz de 3x3, la cual sería la que denominamos filtro o *kernel*, como se muestra en la Figura 7-b.

Entonces, la convolución de la imagen de entrada con la matriz 3x3 denominada filtro o *kernel* daría como resultado las matrices mostradas en la Figura 7-c.



a. Imagen de entrada tamaño 5x5

b. Filtro o *kernel* tamaño 3x3

c. Resultado proceso convolución sobre la imagen de entrada

Figura 7 Ejemplo de convolución sobre una imagen

Para comprender el funcionamiento que se realiza durante la convolución, basta con observar cómo se desplaza el filtro de 3x3 sobre la imagen original en pasos de 1 pixel (esto se denomina, como se verá más adelante, parámetro de *stride*). Para cada paso, se realiza la multiplicación de cada pixel por el valor contenido en la misma posición de la matriz filtro y posteriormente, se realiza la suma de los elementos resultantes pertenecientes a la matriz filtro. De este modo se obtiene una matriz reducida denominada mapa de descriptores o *feature map* que conserva la relación espacial entre pixeles de la imagen de entrada.

De este modo, diferentes valores contenidos en la matriz filtro producirán diferentes mapas de descriptores. De ahí que en una *CNN*, el objetivo principal del entrenamiento sea aprender los valores de la matriz filtro, para obtener unos *feature maps* de interés, aunque inicialmente haya que especificar parámetros como el tamaño del filtro. Un mayor número de filtros permitirá obtener un mayor número de descriptores lo que permitirá que la red pueda reconocer los objetos de una manera más precisa.

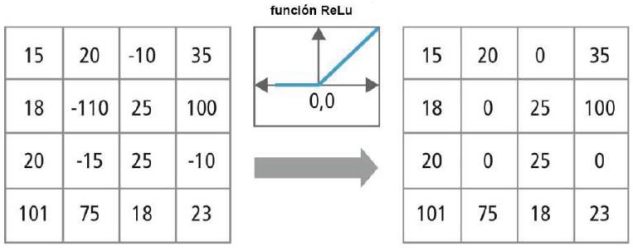
Efectivamente, el tamaño del mapa de descriptores es controlado por parámetros que son definidos antes de desarrollar el proceso de convolución. Los principales parámetros de las capas convolucionales son el tamaño del filtro (o ventana, *window*), el número de filtros (profundidad), el paso o *stride* y el relleno o *padding*.

- El tamaño de la ventana (*window_height × window_width*) especifica el tamaño de la imagen de la que se mantiene información de píxeles cercanos. Es usualmente de 3×3 o 5×5. Indica además el tamaño del mapa de descriptores o características que se quiere manejar a la salida (profundidad de salida o *output_depth*). Acostumbra a ser 32 o 64.
- Profundidad (*depth*): Corresponde al número de filtros utilizados para desarrollar la operación de convolución. Por cada filtro utilizado se obtiene como resultado un mapa de descriptores diferente. Cada mapa de descriptores es apilado como una matriz 2D, siendo su profundidad (*depth*) igual al número de filtros utilizado.
- Paso *(stride)*: Otro parámetro que se puede especificar en una capa convolucional es el parámetro paso o *stride*, que indica el número de pasos en que se mueve la ventada de los filtros, es decir, es el número de pixeles utilizados por el filtro para desplazarse por la imagen de entrada. Por ejemplo, si *stride* toma el valor 1, el filtro se desplazará de pixel en pixel a través de la imagen. Un valor grande de este parámetro genera mapas de descriptores más pequeños, y por tanto, menor tamaño de la información que pasará a la siguiente capa. Pero en realidad el parámetro *stride* es raramente utilizado para reducir los tamaños de los descriptores entre capas; para ello se utilizan las operaciones de *pooling* que se describen más adelante.
- *Zero-padding*: Normalmente, después realizar una convolución, el resultado muestra un tamaño inferior al de la entrada, pero si lo que se pretende es obtener una imagen de salida de las mismas dimensiones que la entrada, es necesario utilizar el parámetro *padding* en las capas convolucionales. Este parámetro rellena la matriz de entrada con ceros alrededor del borde antes de realizar la convolución, de modo que se pueda aplicar el filtro a los pixeles de borde evitando, que el tamaño a la salida de la capa convolucional sea menor que a la entrada. Dependiendo del valor introducido se podrá controlar el tamaño de los mapas de descriptores de salida.

b. Operación Rectified Linear Unit (RLU)

Después de aplicar la operación de convolución en una o varias capas de una red neuronal, suele incluirse otra capa que realice la operación *Rectified Linear Unit (ReLU)*.

La función de activación *ReLU* activa la salida de una neurona si la entrada está por encima de cierto umbral. El comportamiento más habitual es que mientras la entrada tenga un valor por debajo de cero, la salida será cero, y para un valor superior a cero, asigna a su salida el mismo valor de la entrada. De esto modo, la capa incorpora un comportamiento lineal en la red, que permite que su fase de aprendizaje sea más efectivo y más rápido (ver Figura 8).



Fuente: https://www.embedded-vision.com/platinum-members/cadence/embedded-vision training/documents/pages/neuralnetworksimagerecognition

Figura 8 Ejemplo de función *ReLU* aplicada a una matriz de datos

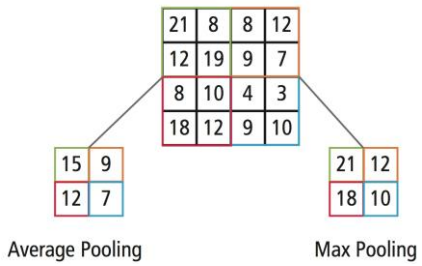
c. Capa de sub-muestreo (Pooling)

Las redes neuronales convolucionales acompañan las capas de convolución con unas capas de submuestreo o *pooling*, que se colocan generalmente entre la salida lineal de la capa *ReLU* y la función de activación posterior. Se utiliza para reducir la dimensionalidad de la imagen, lo que reduce el coste computacional de las siguientes convoluciones, el número de parámetros de la red e incluso permite controlar más fácilmente el sobre-aprendizaje de la red.

Una primera aproximación para entender su funcionamiento es ver que las capas de *pooling* hacen una simplificación de la información generada por una capa convolucional y crean una versión condensada de la información contenida en estas. Hay que indicar que con la transformación de *pooling* se mantiene la relación espacial de los pixeles.

Tal como se ha mencionado anteriormente, la capa convolucional suele albergar más de un filtro, y por tanto, como se aplica el *pooling* a cada uno de ellos separadamente, la capa de *pooling* contendrá también tantos filtros de *pooling* como de filtros convolucionales.

Se puede realizar *pooling* de varios tipos, siendo lo más común *max-pooling* y *average-pooling*. En el primer caso se elige el valor máximo entre un conjunto de píxeles y en el segundo caso se calcula el valor medio entero. La dimensionalidad de la capa de *pooling* es variable, aunque lo más común es que sea de 2x2, 3x3 o 4x4 (ver Figura 9).



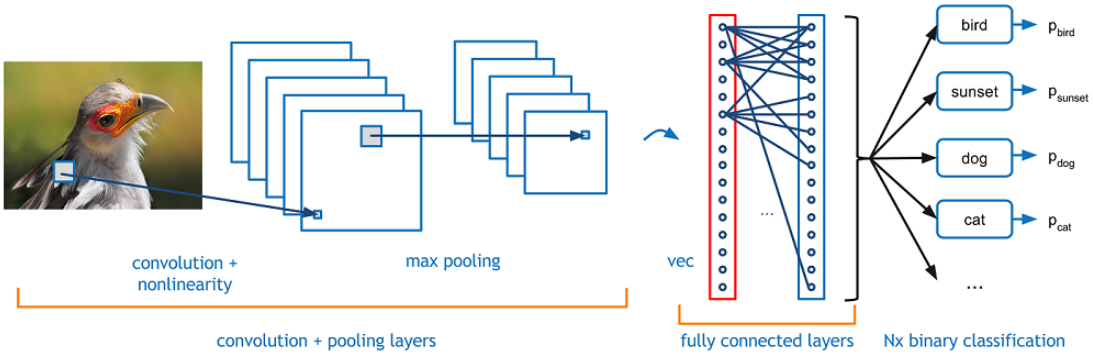
Fuente: https://www.embedded-vision.com/platinum-members/cadence/embedded-vision training/documents/pages/neuralnetworksimagerecognition

Figura 9 Ejemplo de funciones *max-pooling* y *average-pooling*

d. Capa clasificadora totalmente conectada

Al final de las capas convolucionales y de *pooling*, se utilizan generalmente capas completamente conectadas (o *fully-connected*) en las que cada pixel se clasifica considerándose como una neurona separada. Esta última capa clasificadora tendrá, por tanto, tantas neuronas como el número de clases que se debe predecir (Figura 10).

La capa de conexión entre la salida de la de *pooling* y la de clasificación (*fully connected*) se conoce como capa de reconstrucción o *reshape*, ya que convierte la matriz de datos de salida de las convoluciones en un vector plano.



Fuente: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

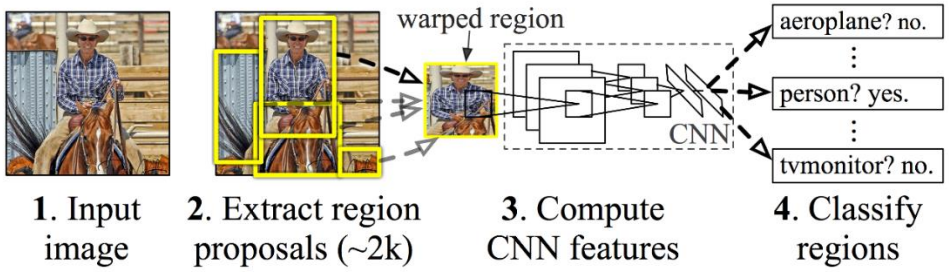
Figura 10 Ejemplo de la capa totalmente conectada o *fully connected*

3.1.2 Detección de Regiones con CNN (R-CNN)

Las redes convolucionales han demostrado ser muy eficientes en tareas de *Image Understanding*, tanto en la clasificación de imágenes, como en la detección de objetos y personas. Si se compara la clasificación con la detección, ésta última ofrece una mayor complejidad, lo que conlleva la utilización en estas tareas de detectores más complejos. Estos detectores son los llamados *R-CNN*, los cuales se dividen en tres etapas, mostradas en la Figura 11:

- En la primera etapa, se generan propuestas a regiones de interés para la detección.
- La segunda, consiste en una gran red neuronal convolucional que extrae un vector de características de longitud fija para cada región.
- Y, por último, la tercera etapa suele consistir en un clasificador por regresión lineal que permita validar las detecciones.

Sin embargo, este algoritmo de detección de personas conlleva un alto coste computacional que lo hace extremadamente lento debido a la necesidad de extraer las regiones propuestas de interés [18].



Fuente: Rich feature hierarchies for accurate object detection and semantic segmentation (Girshick, Donahue, 2013)

Figura 11 Esquema de funcionamiento del detector *R-CNN*

Entrando en más detalle los principales pasos que sigue una *R-CNN* (ver Figura 11), que pasan a describirse en los siguientes apartados, son:

1. Entrada de la imagen a procesar
2. Extracción de las regiones propuestas: Se utiliza un algoritmo de extracción de regiones que propone habitualmente más de 2000 zonas.
3. Por cada región propuesta,
   - Ajustar el tamaño de la imagen al tamaño de la entrada de la *CNN*.
   - Procesa los descriptores obtenidos de la *CNN*.
4. Clasificación del objeto identificado en cada región.

a. Búsqueda selectiva de las regiones de interés

En la búsqueda selectiva o *selective search*, el algoritmo comienza con muchas regiones iniciales diminutas para posteriormente ir incrementando el tamaño de las mismas [19].

A continuación, se localizan las dos regiones más similares y se fusionan. La similitud S entre la región a y b se define como:

$$S(a,b) = S_{texture}(a,b) + S_{size}(a,b) \quad (1)$$

donde $S_{texture}(a,b)$ mide la similitud visual y se define como la intersección de los valores del histograma obtenidos a partir de las mediciones de textura proporcionadas por el algoritmo *SHIFT-like* [20]; y S_{size} se utiliza para fusionar regiones más pequeñas para evitar que una sola región mayor abarque a todas las demás.

El algoritmo continúa fusionando las regiones hasta que finalmente se combinan en un único conjunto. En la Figura 12, se observa en la primera fila, cómo las regiones se van incrementando en tamaño, y a su vez, en la segunda fila de imágenes, se muestran los rectángulos azules que delimitan todas las posibles regiones propuestas realizadas durante el proceso de fusión. Finalmente, el rectángulo verde muestra las regiones de interés propuestas para el detector.

Fuente: Segmentation as selective search for object recognition (van de Sande, Uijlings, Gevers, & Smeulders, 2011)

Figura 12 Ejemplo de funcionamiento del algoritmo de búsqueda de la región de interés

b. Modificación del tamaño de la imagen

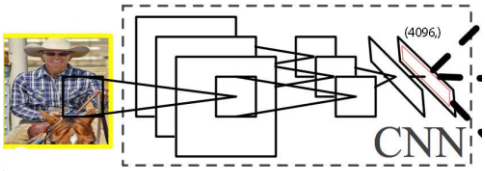
Para cada región propuesta, se utiliza una *CNN* para extraer las características. Como una *CNN* toma una imagen de tamaño fijo, es necesario modificar el tamaño de la región propuesta al tamaño de entrada de la red *CNN*, en el caso utilizado para este trabajo esta *CNN* se basa en la red *Alexnet* cuya capa de entrada únicamente acepta imágenes de tamaño 227 x 227 en RGB (ver Figura 13).

Fuente: Rich feature hierarchies for accurate object detection and semantic segmentation (Girshick, Donahue, 2013)

Figura 13 Modificación del tamaño de imagen a la dimensión de entrada de la red *CNN*

3.1.3 Extracción de descriptores a partir de una CNN

El paso siguiente es utilizar la *CNN* para obtener un descriptor cuya dimensión es de 4096, tal y como se muestra en la Figura 14.

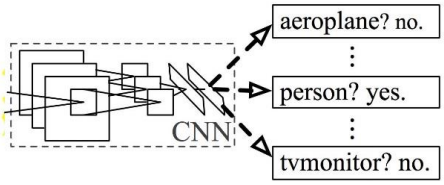


Fuente: Rich feature hierarchies for accurate object detection and semantic segmentation (Girshick, Donahue, 2013)

Figura 14 Obtención del descriptor a partir de la red *CNN*

a. Clasificación

Finalmente, un clasificador tipo *SVM* (*Support Vector Machines*) es utilizado para realizar la clasificación e identificar finalmente el objeto, como ilustra la Figura 15:



Fuente: Rich feature hierarchies for accurate object detection and semantic segmentation (Girshick, Donahue, 2013)

Figura 15 Clasificación e identificación de un objeto

b. Cálculo coordenadas objeto mediante regresión lineal

La propuesta original de la posición del recuadro que bordea al objeto requiere una mayor precisión, por este motivo, mediante regresión lineal se calcula las coordenadas definitivas de la región que contiene al objeto identificado.

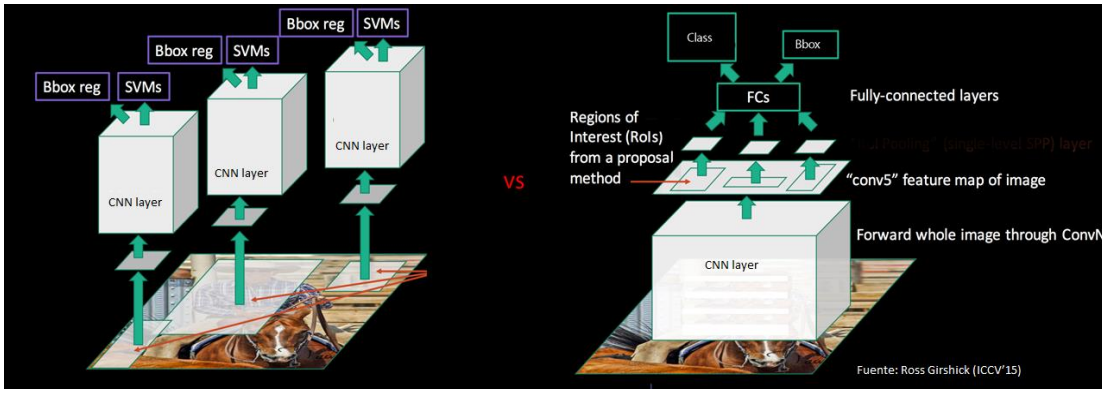
3.2 *Variantes de las RCNN*

3.2.1 Fast-RCNN

Las R-*CNN* son lentas durante su entrenamiento debido principalmente a que se tienen 2000 regiones propuestas las cuales deben ser procesadas una a una por la red *CNN* para extraer los descriptores de características.

Los mapas de características en una red *CNN*, tal como se ha comentado anteriormente, describen las características espaciales de una imagen. Por este motivo, para optimizar el proceso en lugar de procesar 2000 regiones de interés en sus respectivos mapas de características, es decir, invocar 2000 veces a la *CNN*, se invoca a la imagen completa una vez a través de la red *CNN* y a partir del mapa de características generado se obtienen las

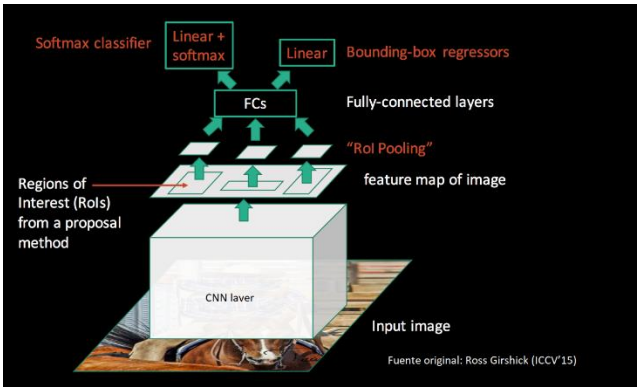
regiones de interés. Esta evolución que mejora las prestaciones de la *R-CNN* se denomina *Fast-RCNN* [21], y la Figura 16 muestra la relación.



Fuente: ICCV 2015 Tutorial on Tools for Efficient Object Detection. (Ross Girshick December 11, 2015)

Figura 16 Esquema de funcionamiento del detector *R-CNN* vs el detector *Fast-RCNN*

Los mapas de vectores de características obtenidos en la capa de *CNN* son utilizados para obtener las regiones de interés. El detector *Fast-RCNN* sin embargo, en lugar de generar una pirámide de capas, *R-CNN* combina los *Regions Of Interest (*ROI) en una sola capa utilizando la función *RoI pooling* (ver Figura 17).



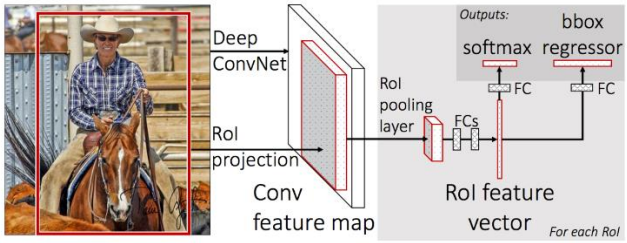
Fuente: ICCV 2015 Tutorial on Tools for Efficient Object Detection. (Ross Girshick December 11, 2015)

Figura 17 Esquema de funcionamiento del detector *Fast-RCNN*

La capa de *RoI pooling* utiliza la función *max-pooling*, cuyo propósito es convertir las regiones de interés de entrada a la función, que tienen tamaños no uniformes, para obtener a su salida, mapas de características de tamaño fijo H × W, donde H y W son parámetros ajustables.

Finalmente, la salida de vectores procedente de la capa *RoI pooling* se conecta a una capa completamente conectada o *fully connected*, la cual se divide a su vez en dos capas iguales. Una de ellas produce estimaciones de probabilidades *softmax* sobre n clases de objetos más una clase de tipo *background*, La otra de las capas genera, mediante regresión lineal, cuatro valores reales para cada una de las n clases de objetos. Cada conjunto de cuatro valores indica las coordenadas de las posiciones del rectángulo delimitador para

cada una de las clases detectadas. La Figura 18 muestra el funcionamiento de esta etapa de salida.



Fuente: *Fast-RCNN* (Girshick R. , 2015)

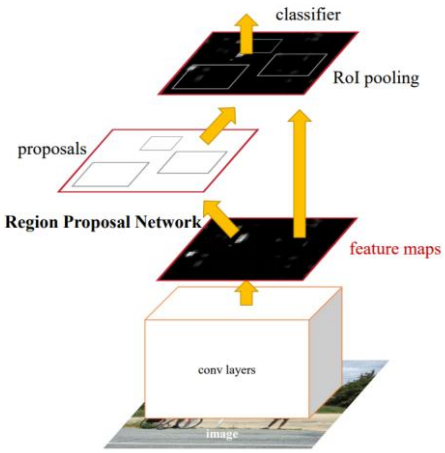
Figura 18 Esquema de funcionamiento de la capa *RoI pooling* en el detector *Fast-RCNN*

3.2.2 Faster-RCNN

La diferencia entre *Fast-RCNN* y *Faster-RCNN* es que no se utiliza un método especial para determinar las regiones de interés. En su lugar, se utiliza una red que será entrenada para obtener las regiones de interés o *Region Porposal Network (RPN)* [6]. Esta red *RPN* toma los mapas de vectores de características como entrada y genera a su salida una serie de regiones de interés. Estas propuestas de regiones se envían luego a la capa de agrupación *RoI pooling* del detector *Fast-RCNN*. En resumen, se trata de una evolución del detector *Fast-RCNN* al cual se le ha incorporado una red *RPN* consiguiendo que está sea más rápida de funcionar (ver Figura 19).

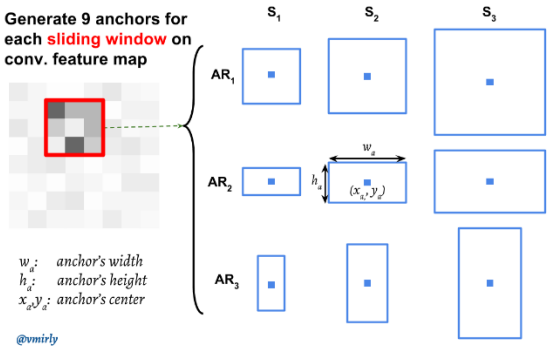
La red *RPN* es una red compuesta por una capa de convolución y utiliza una ventana de tamaño 3x3 que se desliza sobre los mapas de vectores de características con profundidad K.

En *Faster-RCNN*, se genera una propuesta de región de interés por cada ventana deslizante pero realmente la red *RPN* es capaz de escalar o cambiar la relación de aspecto de esta ventana para generar un mayor número de regiones de interés. Se generan para cada ventana deslizante, un conjunto de 9 propuestas de región definidas como *anchor boxes* que tienen el mismo centro (x_a, y_a) pero con 3 relaciones de aspecto diferentes y 3 escalas diferentes, como se muestra en la Figura 20. Hay que tener en cuenta que todas estas coordenadas se calculan con respecto a la original.



Fuente: *Faster-RCNN: Towards Real-Time Object Detection with Region Proposal Networks* (Ren, He, Girshick, & Sun, 2015)

Figura 19 Esquema de funcionamiento del detector *Faster-RCNN*



Fuente: Vahid Mirjalili https://twitter.com/vmirly

Figura 20 Esquema de funcionamiento de las propuestas de interés del detector *Faster-RCNN*

Además, para cada *anchor box*, se calcula un valor p^* que indica cuánto se superponen estos con los recuadros de delimitación o *bounding box* cuyas coordenadas se definen en los *ground truth* asociados al etiquetado de cada imagen.

$$p^* = \begin{cases} 1 \; si \; IoU > PositiveOverlapRange \\ -1 \quad si \; IoU < NegativeOverlapRange, \\ 0 \quad\quad el \; resto \end{cases} \tag{2}$$

donde IoU es la intersección sobre la unión y se define como:

$$IoU = \frac{Anchor\ box \cap Ground\ \ truth}{Anchor\ box \cup Ground\ \ truth} \tag{3}$$

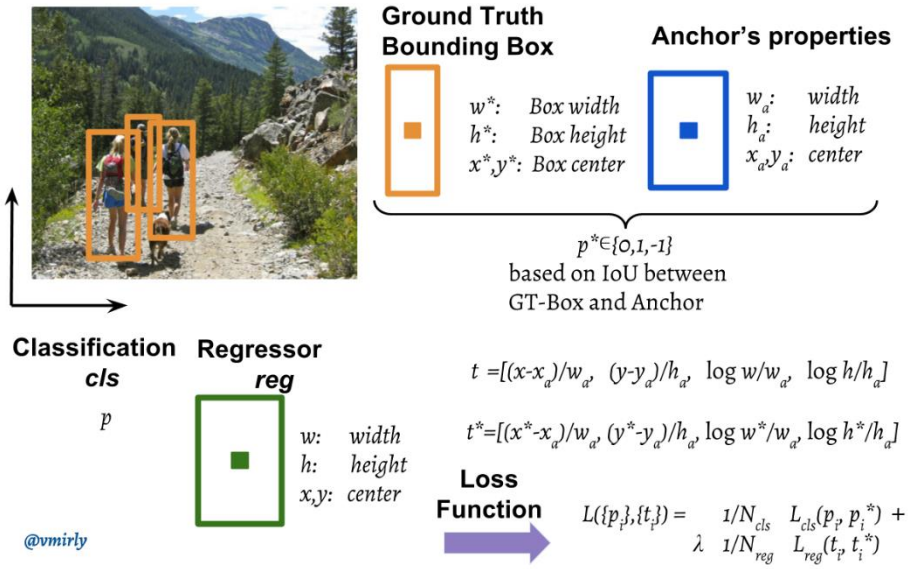
y '*PositiveOverlapRange*' y '*NegativeOverlapRange*' son parámetros que pueden ser fijados durante el entrenamiento de la red.

Finalmente, para cada ventana deslizante, se obtiene a la salida un vector con 256 características que se envía a una red más pequeña que tiene dos tareas: clasificación (*cls*) y regresión (*reg*). La salida de la regresión lineal proporciona un vector con coordenadas que indican el recuadro que delimita al objeto detectado (*x, y, w, h*).

La salida de la subred de clasificación proporciona unas estimaciones de la probabilidad que indica si el cuadro delimitador pronosticado contiene un objeto (1) o es de fondo (0 para ningún objeto).

La función de pérdidas se define sobre la salida de ambas subredes, con 2 términos y un factor de equilibrio λ.

La Figura 21 muestra un diagrama de funcionamiento de esta etapa de clasificación que aparece al final de la red.



Fuente: Vahid Mirjalili https://twitter.com/vmirly

Figura 21 Esquema de funcionamiento de la etapa de clasificación del detector *Faster-RCNN*

3.3 *Redes Neuronales Recurrentes (RNN)*

3.3.1 Introducción y tipos

Los problemas de predicción de secuencias se consideran como uno de los problemas más difíciles de resolver. Estos incluyen una amplia variedad de problemas como el reconocimiento de voz, modelado de lenguaje, traducción, subtítulos de imágenes, predecir la

próxima palabra en el teclado de un móvil, incluso, como el caso que se pretende resolver en este TFM, descibir la acción que transcurre en una determinada secuencia de video.

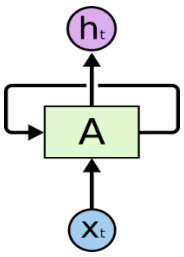
Con los avances recientes que han surgido, las redes de memoria a corto y largo plazo o *Long-Short Term Memory (LSTM)*, se tratan de una versión mejorada de la típica *RNN* y, que será utilizada en este trabajo para determinar el reconocimiento de actividades [8], se ha comprobado que son la solución más efectiva para resolver los problemas referentes a la predicción con secuencias.

Las redes *LSTM* tienen una ventaja sobre las redes neuronales clásicas y las redes *RNN*, y es que tienen la propiedad de recordar selectivamente los patrones por largos períodos de tiempo.

3.3.2 Funcionamiento de la RNN

Un modelo de aprendizaje automático simple o una red neuronal artificial puede aprender a predecir los datos basándose en una serie de características propias contenidas en los mismos. Pero puede ocurrir que aparte de depender de estas características, también en gran parte dependan de las características ocurridas en un instante de tiempo anterior, es decir, estos datos que deben ser aprendidos representan una secuencia temporal, como puede ser una secuencia de video. De hecho, en el caso que se pretende implementar en este trabajo, la información proporcionadas por los *frames* anteriores de una secuencia de video, permiten identificar si la acción se trata de una persona que está corriendo o simplemente andando, o se ha caído y se está levantando o simplemente se está levantando de un asiento, es decir, conocer el pasado es uno de los principales factores decisivos para las predicciones en secuencias temporales. Uno de los problemas de las redes neuronales tradicionales es que predicen sin tener en cuenta lo ocurrido anteriormente, ya que todos los casos de prueba se consideran independientes unos de otros. El modelo se ajusta para un instante de tiempo particular, sin considerar lo ocurrido en instantes de tiempo anteriores, es decir, carecen de memoria, por lo que no son muy útiles si se quiere analizar una secuencia temporal de imágenes. Esta dependencia del tiempo se logra a través de las Redes Neuronales Recurrentes (*RNNs*).

Las redes neuronales recurrentes (RNNs) han mostrado recientemente un gran comportamiento al abordar diversas tareas de modelado de secuencias en el aprendizaje automático, como el reconocimiento automático de voz, la traducción de idiomas y la generación de reconocimiento de actividades para imágenes y constituyen una herramienta muy apropiada para modelar series temporales. Se trata de un tipo de redes con una arquitectura que implementa una cierta memoria y, por lo tanto, un sentido temporal. Los valores de las variables de salida (predicciones) se determinan a partir de valores de salida obtenidos en instantes anteriores [22] [23].

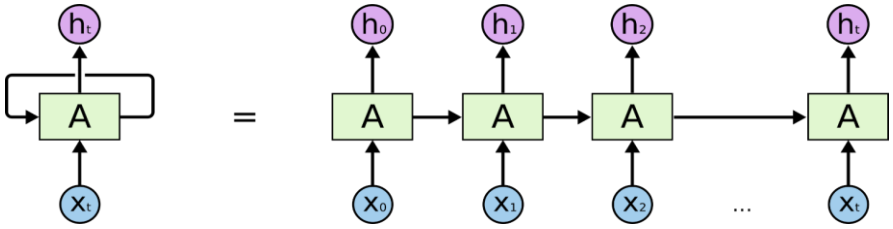


Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 22 Esquema de una red *RNN* típica

En la Figura 22, se muestra un trozo de red neural recursiva que se denomina A, x_t sería la entrada a la red y h_t sería la salida de la red. Un bucle permite pasar la información de un determinado paso de ejecución de la red al siguiente paso a ejecutar.

Una red neuronal recurrente sería como una secuencia de copias múltiples de la misma red, cada uno pasando un mensaje a su sucesor. En la Figura 23 se muestra la red desplegada.

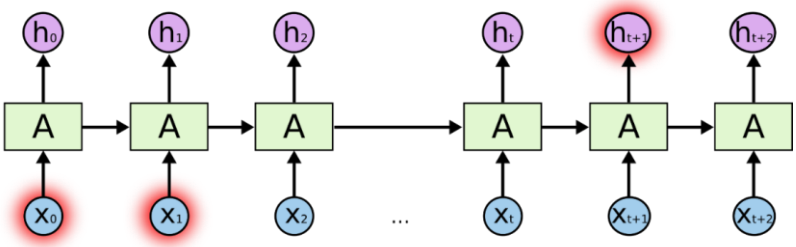


Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 23 Esquema de una red *RNN* desplegada

Cada predicción en el instante de tiempo t (h_t) depende de todas las predicciones previas y de la información que se aprende de ellas. Al observar a la red desplegada se puede determinar que presenta una arquitectura muy útil para utilizar con secuencias temporales de datos.

Las *RNNs* pueden manejar secuencias en gran medida, pero no del todo. Las *RNNs* son útiles cuando se trata de contextos cortos, es decir, la información pasada sea reciente. A medida que crece esa brecha, existe una dependencia a largo plazo de la información y las *RNNs* son incapaces de aprender correctamente. Para poder identificar una secuencia temporal y memorizarla, es necesario que los modelos sean capaces de comprender y recordar el contexto contenido en las secuencias y esto no es posible con una simple *RNN*, siendo este es el principal problema de este tipo de redes (ver Figura 24).



Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 24 Problema de la dependencia a largo plazo de una red *RNN*

La razón de este problema es que las *RNN* simples son difíciles de entrenar utilizando el algoritmo por descenso de gradientes estocástico o *stochastic gradient descent (SDG)* [24] y se ha comprobado que muestran problemas para aprender secuencias con dependencias relativamente largas debido al *vanishing gradient* y/o fenómenos de *exploding gradient*. Para una red neuronal convencional, la actualización de ponderación que se aplica en una capa particular es un múltiplo de la tasa de aprendizaje, el término de error de la capa anterior y la entrada a esa capa. Por lo tanto, el término de error para una capa particular es en algún lugar un producto de todos los errores de las capas anteriores. Cuando se trata de funciones de activación como la función sigmoidea, los pequeños valores de sus derivadas se multiplican varias veces a medida que avanzamos hacia las capas de inicio. Como resultado de esto, el gradiente casi desaparece a medida que avanzamos hacia las capas de inicio, y se vuelve difícil entrenar estas capas. Un caso similar se observa en Redes Neuronales Recurrentes. *RNN* recuerda cosas solo por períodos de tiempo pequeños, es decir, si necesitamos la información después de un tiempo pequeño, puede ser reproducible, pero una vez que se introducen más datos, esta información se pierde en alguna parte. Para abordar esta limitación, los investigadores han desarrollado una serie de técnicas en arquitecturas de red y algoritmos de optimización, entre los cuales las aplicaciones más exitosas son las unidades de memoria de corto a largo plazo (*LSTM*) en *RNN*.

3.4 *Redes LSTM*

Las redes de memoria de corto a largo plazo o *Long Short-Term Memory*, usualmente denominada como *LSTM*, son un tipo especial de *RNN*, capaz de aprender dependencias a largo plazo. Fueron introducidos por Hochreiter y Schmidhuber [25], y trabajan bastante bien en una gran variedad de problemas, siendo ampliamente utilizadas. Las redes *LSTM* se diseñan explícitamente para evitar el problema de la dependencia a largo plazo. Recordar información durante largos períodos de tiempo es prácticamente su comportamiento por defecto.

Una unidad *LSTM* utiliza una celda de memoria que puede mantener su valor de estado durante un tiempo prolongado, y un mecanismo de compuertas que contiene tres compuertas no lineales, una entrada, una salida y una compuerta de olvido. La función prevista de las puertas es regular el flujo de señales dentro y fuera de la celda, para ser eficaz en la regulación de dependencias de largo alcance y lograr un entrenamiento exitoso.

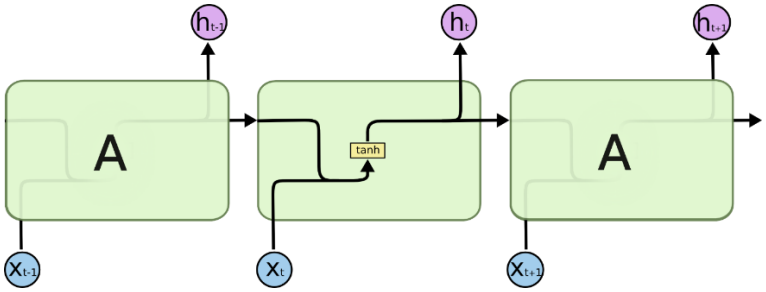
Una capa *LSTM* es una capa de red neuronal recurrente (*RNN*) que permite la compatibilidad con series de tiempo y datos de secuencia en una red. La capa realiza interacciones aditivas y multiplicativas, que pueden ayudar a mejorar el flujo de gradiente en secuencias largas durante el entrenamiento. De esta forma, las *LSTM* como no manipulan toda la información, sino que la modifican un poco, pueden olvidar y recordar cosas selectivamente. Las capas *LSTM* son las más adecuadas para aprender dependencias a largo plazo (dependencias de pasos de tiempo distantes).

La información en un estado de celda particular tiene tres dependencias diferentes:

1. El estado de celda anterior, es decir, la información que estaba presente en la memoria después del paso de tiempo anterior.
2. El estado oculto anterior, es decir, este es el mismo que el resultado de la celda anterior.
3. La entrada en el paso de tiempo actual, es decir, la nueva información que se está introduciendo en ese momento.

3.4.1 Arquitectura de las redes de tipo LSTM

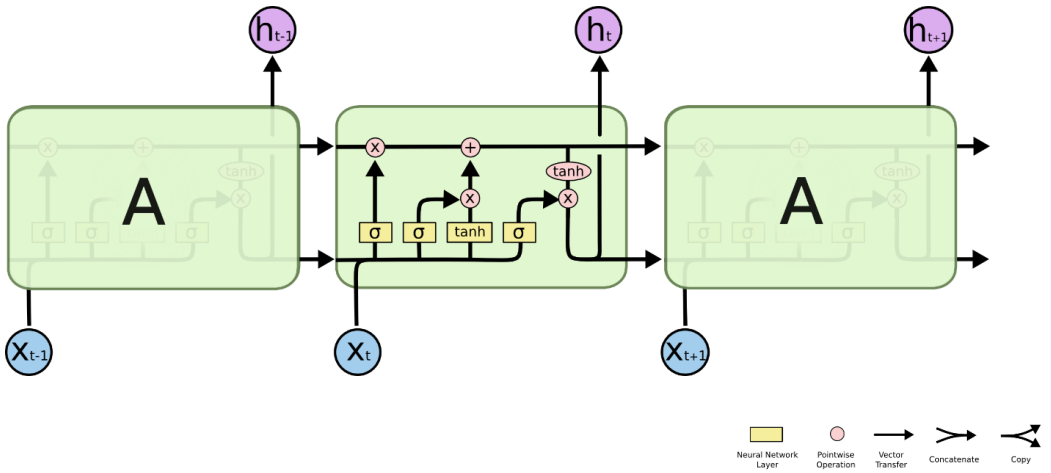
Todas las redes neuronales recurrentes tienen la forma de una secuencia de módulos neuronales que se repiten. En una *RNN* estándar, este módulo de repetición tiene una estructura muy simple, tal como se observa en la Figura 25, sólo contienen una capa con una función de activación tipo tangente hiperbólica, *tanh* (ver apartado 3.5.2 c).



Fuente: http://colah.github.io/posts/2015-08-Understanding-*LSTMs* (Olah, 2015)

Figura 25 Estructura interna de una red *RNN* típica

Las redes de tipo *LSTM* también tienen esta estructura, pero el módulo neuronal que se repite tiene una estructura diferente, ya que tiene cuatro capas por cada red neuronal en lugar de una como ocurre en las *RNNs* (ver Figura 26).



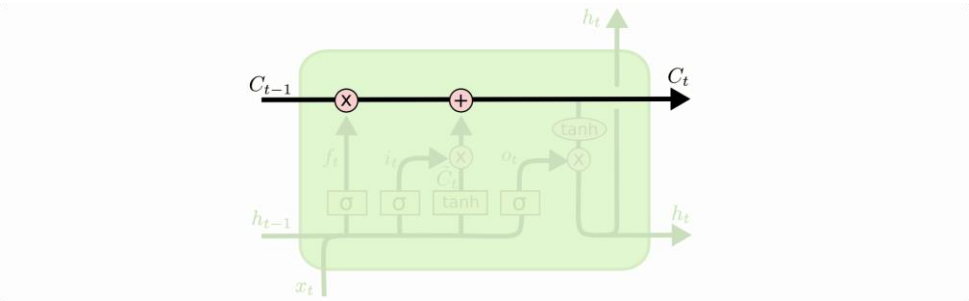
Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 26 Estructura interna de una red *LSTM*

Una red *LSTM* típica se compone de diferentes bloques de memoria llamados celdas. Hay dos estados que se transfieren a la siguiente celda; el estado de la celda y el estado oculto. Los bloques de memoria son responsables de recordar cosas y las manipulaciones de esta memoria, remover o agregar información al estado de la celda, se realizan a través de tres mecanismos principales, llamados puertas que se comentarán más adelante.

La clave principal de las redes *LSTM* es lo que se denominará estado de la celda o *cell state* (ver Figura 27), en la imagen siguiente viene reflejada como la línea horizontal que se ejecuta a través de la parte superior del diagrama.

El estado de la celda se ejecuta directamente por toda la cadena, con sólo algunas interacciones lineales menores. Es muy fácil que la información fluya a lo largo de ella sin cambios.

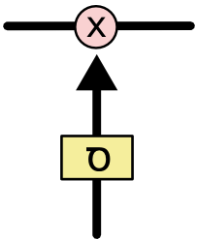


Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 27 Estructura interna de una red *LSTM*: estado de celda

Una *LSTM* tiene tres puertas, para proteger y controlar el estado de la celda.

Las puertas son una manera de permitir que la información se incorpore al estado de la celda. Se componen de una capa neuronal cuya función es de tipo sigmoide y de una operación de tipo multiplicación, tal y como muestra la Figura 28.

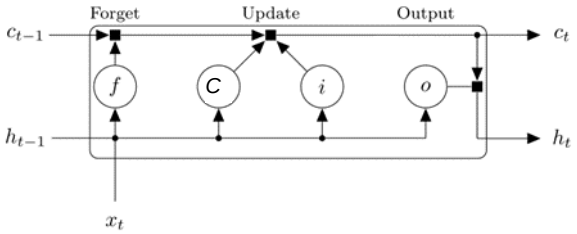


Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 28 Estructura interna de una red *LSTM*: puertas

La función sigmoidea emite números entre cero y uno, describiendo la cantidad de cada componente que se debe dejar pasar, un valor de cero significa que no pasa ningún tipo de información, mientras que un valor de uno significa que pasa toda la información.

El diagrama que se muestra en la Figura 29 ilustra el flujo de datos en el paso de tiempo t. El diagrama resalta cómo las compuertas olvidan, actualizan y generan los estados de celda y salida.



Fuente: Mathworks (Matlab 2018a)

Figura 29 Estructura interna de una red *LSTM*: flujo de datos

La Tabla 1 resume los componentes que controlan el estado de celda y el estado de salida de la capa.

Componente	Propósito
Input Gate (i)	Nivel de control de la actualización del estado de la celda
Forget Gate (f)	Nivel de control del restablecimiento del estado de la celda (olvidar)
Layer input (C)	Agregar información al estado de la celda
Output Gate (o)	Nivel de control del estado de la celda añadido al estado de salida

Tabla 1 Componentes que controlan el estado de celda y el estado de salida de una *LSTM*

Los pesos que se pueden aprender de una capa de *LSTM* son los pesos de entrada W, los pesos recurrentes R y el sesgo b. Las matrices W, R y b son concatenaciones de los pesos de entrada, los ponderadores recurrentes y el sesgo de cada componente, respectivamente. Estas matrices se concatenan de la siguiente manera:

$$W = \begin{bmatrix} W_i \\ W_f \\ W_C \\ W_o \end{bmatrix}; R = \begin{bmatrix} R_i \\ R_f \\ R_C \\ R_o \end{bmatrix}; b = \begin{bmatrix} b_i \\ b_f \\ b_C \\ b_o \end{bmatrix}, \quad (4)$$

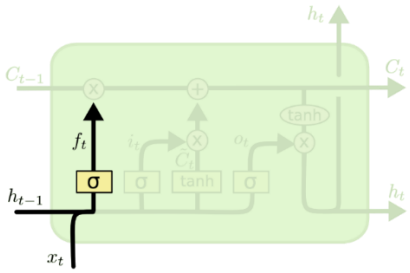
donde *i*, *f*, *C* y *o* denotan la puerta de entrada, la puerta de olvido, la entrada a la capa y la puerta de salida, respectivamente.

A continuación, se explicará con más detalle cada una de las puertas, indicando las ecuaciones dinámicas para los bloques de memoria *LSTM* correspondientes a una arquitectura *LSTM* estándar para cada componente en el paso de tiempo t.

a. Forget Gate

Una puerta de olvido o *Forget Gate* es responsable de eliminar la información del estado de la celda. La información que ya no se necesita para que el *LSTM* entienda las cosas o la información que es de menor importancia se elimina mediante la multiplicación de un filtro. Esto es necesario para optimizar el rendimiento de la red *LSTM*.

Esta puerta tiene dos entradas, h_{t-1} es el estado oculto de la celda anterior o la salida de la celda anterior y x_t es la entrada en ese paso de tiempo particular. Las entradas dadas se multiplican por las matrices de peso y se agrega un sesgo. A continuación, la función sigmoidea se aplica a este valor. La función sigmoide produce un vector, con valores que van de 0 a 1, correspondientes a cada número en el estado de la celda. Básicamente, la función sigmoidea es responsable de decidir qué valores conservar y cuáles descartar. Si se emite un '0' para un valor particular en el estado de celda, significa que la puerta de olvidar quiere que el estado de la celda olvide por completo esa información. Del mismo modo, un '1' significa que la puerta de olvidar quiere recordar toda esa información. Este resultado de vector de la función sigmoidea se multiplica al estado de celda, como se muestra en la Figura 30.



$$f_t = \sigma\left(R_f h_{t-1} + W_f x_t + b_f\right) \quad (5)$$

σ denota la función sigmoidea dada por $\sigma(x) = (1 + e^{-x})^{-1}$

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

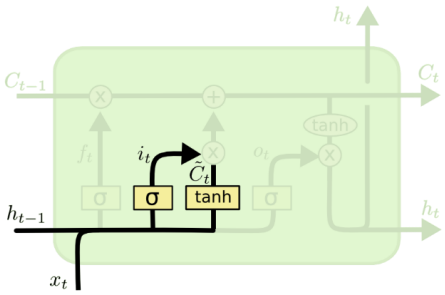
Figura 30 Estructura interna de una red *LSTM*: *Forget Gate*

b. Input Gate

La puerta de entrada o *Input Gate* es responsable de agregar información al estado de la celda. Esta adición de información es básicamente un proceso de tres pasos como se indica a continuación.

1. Regulación de los valores que se deben agregar al estado de la célula mediante una función sigmoidea. Esto es básicamente muy similar a *Forget Gate* y actúa como un filtro para toda la información de h_{t-1} y x_t.
2. Crear un vector que contenga todos los valores posibles que se pueden agregar (como se percibe desde h_{t-1} y x_t) al estado de la celda. Esto se hace usando la función *tanh*, que genera valores de -1 a +1.
3. Multiplicar el valor del filtro regulador (la puerta sigmoidea) por el vector creado (la función *tanh*) y luego agregar esta información útil al estado de la celda a través de la operación de suma (Figura 31).

Una vez que se ha completado este proceso de tres pasos, sólo la información que es importante y no es redundante es agregada al estado de la celda (ver Figura 32).

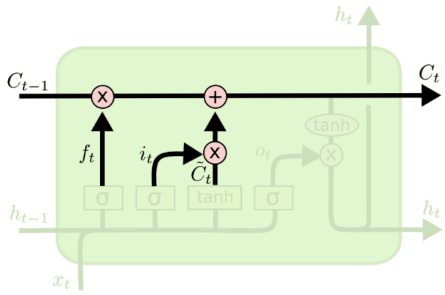


$$i_t = \sigma(R_i h_{t-1} + W_i x_t + b_i) \quad (6)$$

$$\tilde{C}_t = tanh(R_C h_{t-1} + W_C x_t + b_C) \quad (7)$$

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 31 Estructura interna de una red *LSTM*: *Input Gate paso 1*



Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 32 Estructura interna de una red *LSTM*: *Input Gate paso 2*

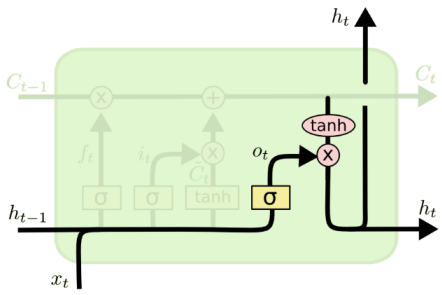
El estado de la celda en el paso de tiempo t viene dado por:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \quad (8)$$

donde $\odot$ denota el producto Hadamard (multiplicación de vectores con elementos).

c. Output Gate

No toda la información que se ejecuta a lo largo del estado de la celda es válida para propagarse a la salida. Este trabajo de seleccionar información útil del estado actual de la celda y mostrarlo como salida se realiza a través de la puerta de salida u *Output Gate* (ver Figura 33).



$$o_t = \sigma(R_o h_{t-1} + W_o x_t + b_o) \qquad (9)$$

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs (Olah, 2015)

Figura 33 Estructura interna de una red *LSTM*: Output *Gate*

Finalmente, se necesita conocer el estado de salida oculto en el paso de tiempo t que viene dado por:

$$h_t = o_t \odot tanh\, C_t \qquad (10)$$

Esta salida se basará en el estado de la celda, pero será una versión filtrada.

El funcionamiento de una puerta de salida se puede dividir de nuevo en tres pasos:

1. Creación de un vector después de aplicar la función *tanh* al estado de la celda, escalando así los valores al rango -1 a +1.
2. Hacer un filtro usando los valores de h_{t-1} y x_t, de modo que pueda regular los valores que deben salir del vector creado anteriormente. Este filtro nuevamente emplea una función sigmoidea.
3. Multiplicar el valor de este filtro regulador por el vector creado en el paso 1 y enviarlo como salida y también al estado oculto de la siguiente celda.

El filtro debe basarse en los valores de entrada y estado oculto y aplicarse en el vector de estado de la celda.

Finalmente, la salida de la red *LSTM*, se corresponderá con un vector que contienen todas las salidas y viene representado por $\mathbf{Y}_t = [h_{t-n}, \ldots, h_{t-1}]$.

Los *LSTM* son una solución muy prometedora para los problemas relacionados con la secuencia y la serie temporal. Sin embargo, una desventaja que se encuentra es la dificultad de entrenarlos, ya que se requiere una gran cantidad de tiempo y recursos del sistema.

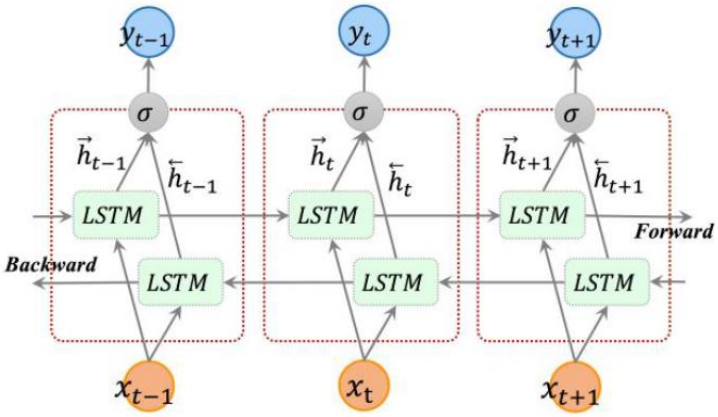
3.4.2 Variantes de la LSTM: Bidireccional LSTM

Lo que se ha descrito hasta ahora es una *LSTM* clásica. Pero no todas las redes *LSTM* son las mismas que las anteriores. De hecho, parece que casi todos los trabajos realizados sobre el

tema, que involucran a redes *LSTM*, usan una versión ligeramente diferente. En este caso también se utilizará una variación de estas denominada bidireccional *LSTM*.

La idea de la red *LSTM* bidireccional proviene de la *RNN* bidireccional [26], que procesa secuencias de datos en ambas direcciones, es decir, hacia adelante y hacia atrás, utilizando dos redes recurrentes separadas, ambas conectadas a la misma capa de salida, en algunos casos, se usa una tercera red en lugar de la capa de salida.

Se ha demostrado que las redes bidireccionales son sustancialmente mejores que las unidireccionales en muchos campos relacionados con el aprendizaje de secuencias como, por ejemplo, el reconocimiento de voz.



Fuente: Deep Bidirectional and Unidirectional *LSTM* Recurrent Neural Network for Network-wide Traffic Speed Prediction. (Cui, Ke, & Wang, 2018)

Figura 34 Arquitectura desplegada de una red *LSTM* bidireccional con tres pasos consecutivos

La estructura de una red *LSTM* bidireccional desplegada [27] , que contiene una capa *LSTM* hacia adelante y una capa *LSTM* hacia atrás, se muestra en la Figura 34. La secuencia de salida de la red *LSTM* hacia adelante, $\vec{h}$, se calcula iterativamente usando entradas en una secuencia positiva de tiempo t-n a tiempo t-1, mientras que la secuencia de salida de la red *LSTM* hacia atrás, $\overleftarrow{h}$, se calcula utilizando las entradas inversas desde el tiempo t-n hasta el t-1. Tanto las salidas de capa hacia adelante como hacia atrás se calculan utilizando las ecuaciones de actualización *LSTM* estándar presentadas en los apartados anteriores (3.4.1). La red *LSTM* bidireccional genera un vector de salida, $\mathbf{Y}_t$ en el cual cada elemento se calcula usando la siguiente ecuación:

$$y_t = \sigma \cdot \left(\overrightarrow{h_t}, \overleftarrow{h_t}\right) \qquad (11)$$

,

donde la función σ se usa para combinar las dos secuencias de salida. Esta función puede ser de concatenación, una función de suma, una función promedio o una función de multiplicación. De forma similar a la red *LSTM*, la salida final de una red *LSTM* bidireccional puede representarse mediante un vector, $\mathbf{Y}_t = [y_{t-n}, \ldots, y_{t-1}]$.

3.5 *Entrenamiento de una red neuronal*

En este apartado se va a presentar una visión intuitiva de los componentes principales del proceso de aprendizaje de una red neuronal. Además, se verán algunos de los parámetros más relevantes en *Deep Learning*.

En la fase de entrenamiento de un modelo se aprenden los valores ideales para los parámetros del modelo (los pesos Wi y el sesgo b). En el aprendizaje supervisado, la manera de conseguirlo es aplicar un algoritmo de aprendizaje automático que obtenga el valor de estos parámetros analizando una gran cantidad de ejemplos etiquetados e intentar determinar unos valores para estos parámetros del modelo que minimicen el error, que indica como de mala ha sido una predicción en un ejemplo concreto. Si la predicción del modelo es perfecta, el error es cero.

3.5.1 Proceso de aprendizaje de una red neuronal

Una red neuronal está formada de neuronas conectadas entre ellas, y a su vez, cada conexión de la red neuronal está asociada a un peso que marca la importancia que tendrá esa relación en la neurona al multiplicarse por el valor de entrada.

Cada neurona tiene una función de activación que define la salida de la neurona, como puede ser la función *sigmoid*, *tanh*, etc., que serán vistas más adelante. La principal utilidad de la función de activación es introducir la no linealidad durante el proceso de entrenamiento del modelado de la red.

Durante el entrenamiento de la red neuronal (ver Figura 35), se produce un proceso iterativo de ida y vuelta que atraviesa las capas de neuronas que componen la red, en el cual, los valores de los pesos W_{ij} y sesgos b_j se van ajustando al modelo requerido. El proceso de ida o de propagación hacia delante se denomina *forward-propagation* y el proceso de vuelta o en el cual la información se retro-propaga en la red, se denomina *back-propagation*.

La primera fase de *forward-propagation* se da cuando se pasan los datos de entrada a través de la red, de tal manera, que todas las neuronas se adapten o aprendan en función de la información que reciben de las neuronas de la capa anterior y la envíen a las neuronas de la capa siguiente. Cuando los datos han atravesado todas las capas, y todas sus neuronas han realizado sus cálculos, se llegará a la capa final con un resultado de predicción de la etiqueta (*label*) para aquellos ejemplos de entrada.

Una función de perdidas (*loss*) es utilizada para estimar el error y para comparar y cuantificar lo bueno o malo que ha sido el resultado de la predicción en relación con el resultado correcto. El objetivo, es que este valor sea cero, de ahí que a medida que se entrena el modelo los pesos de las interconexiones de las neuronas se irán ajustando de manera automática hasta obtener la mejor predicción posible.

Una vez calculado el error, este es propagado hacia atrás en la red, es decir, se retro-propaga (*back-propagation*). El error se propaga desde la capa de salida hacia todas las neuronas de la capa oculta que están interconectadas con la salida. Estas capas ocultas reciben una fracción del valor de error, en función de la contribución aportada por cada neurona a la salida. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red han recibido la fracción de error correspondiente en función de la contribución al error total.

Una vez propagado el error a través de la red, los pesos de las conexiones entre neuronas se ajustan. Para lograr que el error se aproxime a cero se utiliza un algoritmo iterativo llamado descenso por gradiente (*gradient descent*). Este algoritmo modifica el valor de los pesos en pequeños incrementos obtenidos a partir del cálculo de la derivada (o gradiente) de la función de perdidas (*loss*). Esta derivada permite ver en qué dirección desciende hacia el mínimo global de la función; este algoritmo se realiza en sucesivas iteraciones (lo que se denomina *epochs*, como se verá más adelante) utilizando lotes de datos (*batches*).

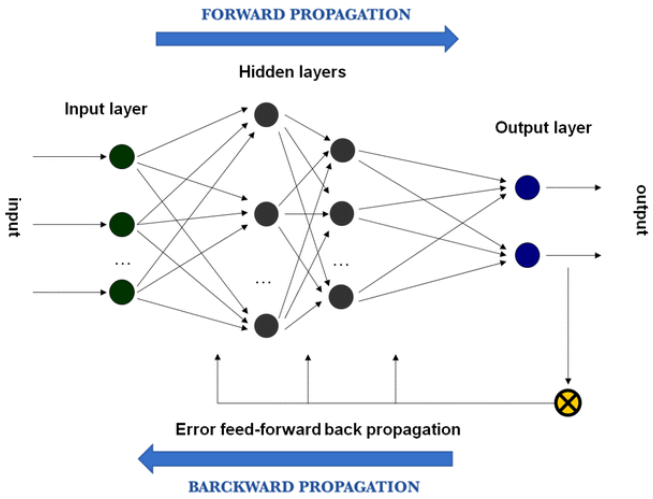


Figura 35 Fases de entrenamiento de una red neuronal

3.5.2 Funciones de activación

Las funciones de activación se utilizan para propagar hacia adelante la salida de una neurona. Esta salida la reciben las neuronas de la siguiente capa a las que está conectada esta neurona. Como se ha comentado anteriormente, la función de activación sirve para introducir la no linealidad durante el proceso de entrenamiento del modelo de la red. Las funciones de activación más utilizadas son las que se muestran a continuación.

a. Función de activación linear

La función de activación linear es la función identidad en la que la señal de salida es igual a la de entrada.

$$F_k(x) = x \qquad (12)$$

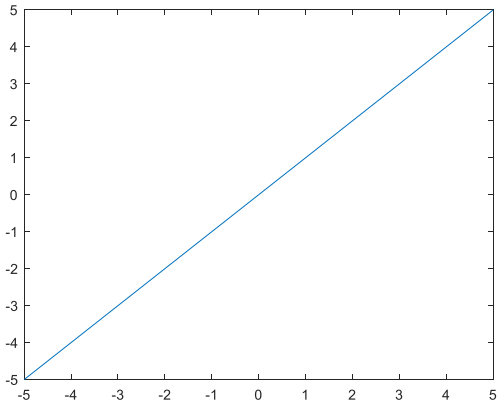


Figura 36 Función de activación *Linear*

b. Función de activación sigmoid

La función *sigmoid* permite reducir valores extremos o atípicos en datos válidos sin eliminarlos. Una función sigmoidea convierte variables independientes de rango casi infinito en probabilidades simples entre 0 y 1. Con la función sigmoidal el valor dado por la función es cercano a uno de los valores asintóticos, es decir, la mayor parte de su salida estará muy cerca de los extremos de 0 o 1.

Esto hace que en la mayoría de los casos, el valor de salida esté comprendido en la zona alta o baja del sigmoide. De hecho, cuando la pendiente es elevada, esta función tiende a la función escalón. Sin embargo, la importancia de la función sigmoidal es que su derivada siempre es positiva y cercana a cero para los valores grandes positivos o negativos; además, toma su valor máximo cuando x = 0.

Esto hace que se puedan utilizar reglas de aprendizaje definidas para las funciones escalón, con la ventaja, respecto a esta función, de que la derivada está definida en todo intervalo.

$$F_k(x) = \frac{1}{1 + e^{-x}} \qquad (13)$$

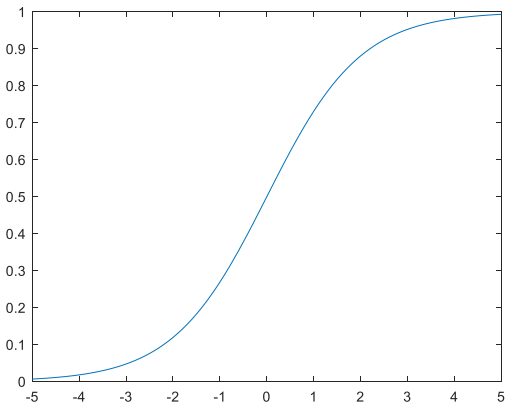


Figura 37 Función de activación *Sigmoid*

c. Función de activación tanh

La función de activación tangente hiperbólica (*tanh*) se emplea en los casos que presentan variaciones suaves de valores positivos y negativos de la señal a clasificar. Como se puede ver en su descripción es una de las funciones más empleadas en entrenamientos supervisados, como en el caso del entrenamiento de retro-propagación del error. Debe de tenerse cuidado de emplear esta función entre los umbrales positivos y negativos antes de la saturación, de otra forma la salida siempre generará valores saturados iguales a 1 y -1.

$$F_k(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad (14)$$

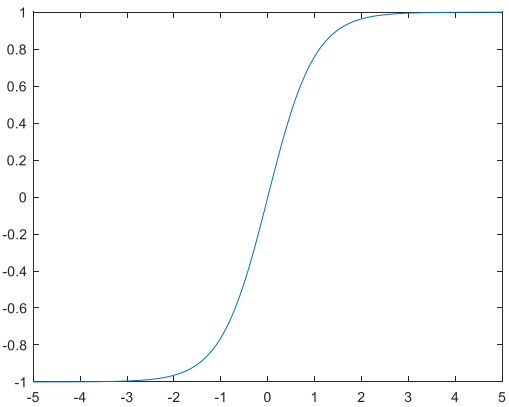


Figura 38 Función de activación *tanh*

d. Función de activación softmax

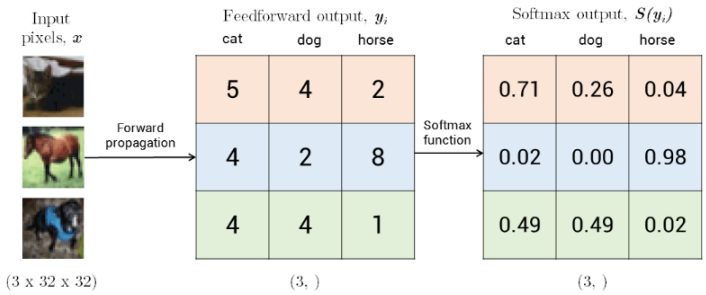
La función de activación *softmax* es utilizada como una capa de salida en la red neuronal, en la que cada neurona en esta capa *softmax* depende de las salidas de todas las otras neuronas de la capa, puesto que la suma de la salida de todas ellas debe ser 1. Permite que, para cada ejemplo de entrada, obtener como vector de salida de la red neuronal una distribución de probabilidad sobre un conjunto de etiquetas mutuamente excluyentes, es decir, normaliza un vector x de dimensión K, de valores reales arbitrarios, en un vector σ(x) de dimensión K cuyas componentes suman 1, en otras palabras, un vector de probabilidad.

Para ello *softmax* utiliza el valor exponencial de las evidencias calculadas y luego las normaliza de modo que sumen uno, formando una distribución de probabilidad. La probabilidad de pertenencia a la clase K es:

$$\sigma(\boldsymbol{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \qquad (15)$$

El efecto que se consigue con el uso de exponenciales es que una unidad más de evidencia tiene un efecto multiplicador y una unidad menos tiene el efecto inverso. Lo interesante de esta función es que una buena predicción tendrá una sola entrada en el vector con valor cercano a 1, mientras que las entradas restantes estarán cerca de 0. En una predicción débil tendrán varias etiquetas posibles, que tendrán más o menos la misma probabilidad.

La capa s*oftmax* se encontrará a menudo en la capa de salida de un clasificador.



Fuente: https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

Figura 39 Ejemplo de salida de la capa softmax sobre tres clases

3.5.3 Elementos del backpropagation

El *backpropagation* es un método utilizado para alterar los pesos y sesgos de la red neuronal en la dirección correcta. Empieza primero por calcular el término de pérdidas (*loss*), para luego, ajustar los parámetros de la red neuronal en orden inverso con un algoritmo de optimización teniendo en cuenta el error calculado.

Matlab permite utilizar tres algoritmos diferentes de optimización: SGDM (*stochastic gradient descent with momentum*), RMSProp y Adam. En el desarrollo de este trabajo, se ha utilizado el algoritmo de optimización SGDM que es el utilizado mayormente para el entrenamiento de redes neuronales.

a. Gradient descent

Todos los métodos indicados anteriormente utilizan como base para su funcionamiento el descenso por gradiente (*gradient descent*) siendo uno de los algoritmos de optimización más comunes en *Machine Learning* y *Deep Learning*.

El algoritmo de descenso de gradiente actualiza los parámetros de red (pesos y sesgos) para minimizar la función de pérdida dando pequeños pasos en la dirección del gradiente negativo de la función de pérdidas, para ello aplica la primera derivada (gradiente) a la función de pérdidas en cada iteración proporcionando la pendiente de la función en un determinado punto.

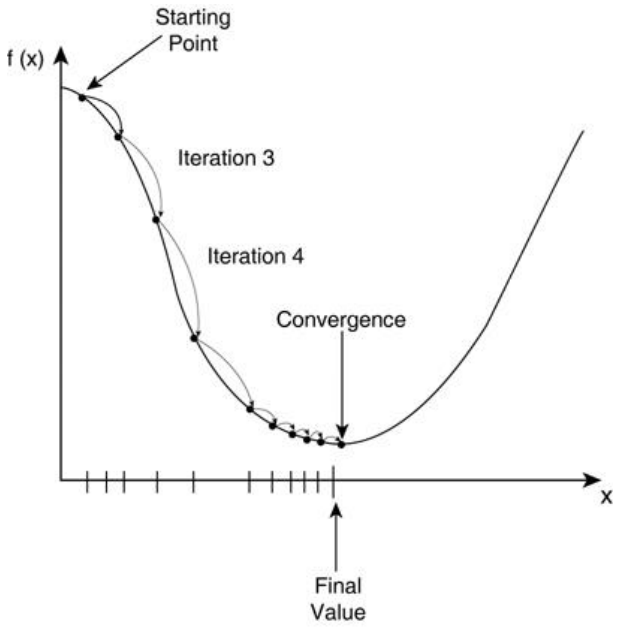
$$\theta_{n+1} = \theta_n - \alpha \cdot \nabla E(\theta), \qquad (16)$$

donde n representa el número de iteración, α> 0 es la velocidad de aprendizaje (*learning rate*) que será explicado más adelante, θ es el vector de parámetro, y E(θ) es la función de pérdidas. El gradiente de la función de pérdidas, ∇E(θ), se evalúa usando todo el conjunto de entrenamiento, y el algoritmo de descenso de gradiente estándar usa todo el conjunto de datos a la vez.

El proceso consiste en encadenar las derivadas de la función de pérdidas de cada capa oculta a partir de las derivadas de la función de pérdidas de su capa superior, incorporando su función de activación en el cálculo, de ahí que las funciones de activación deben ser derivables. En cada iteración, una vez todas las neuronas disponen del valor del gradiente de la función de pérdidas que les corresponde, se actualizan los valores de los parámetros en el sentido contrario a la que indica el gradiente. El gradiente, en realidad, siempre apunta en el sentido en el que se incrementa el valor de la función de pérdidas. Por tanto, si se utiliza el negativo del gradiente se puede conseguir el sentido en el cual la función de pérdidas tiende a reducirse.

Para determinar el siguiente valor, el algoritmo de *gradient descent* modifica el valor inicial para ir en sentido contrario al del gradiente, añadiendo una cantidad proporcional a este. La magnitud de este cambio está determinada por el valor del gradiente y por un parámetro denominado *learning rate*, que se definirá más adelante. El algoritmo *gradient*

descent repite este proceso de manera iterativa hasta llegar a un punto más allá del cual no puede disminuir la función de pérdidas [28].



Fuente: Fernando Sancho Caparrini http://www.cs.us.es/~fsancho/?e=165

Figura 40 Proceso iterativo del algoritmo Gradient Descent

b. Stochastic Gradient Descent (SGD)

El algoritmo de descenso de gradiente estocástico evalúa el gradiente y actualiza los parámetros usando un subconjunto del conjunto de entrenamiento. Este subconjunto se llama *minibatch*. Cada evaluación del gradiente usando el *minibatch* es una iteración. En cada iteración, el algoritmo da un paso hacia la minimización de la función de pérdida. El paso completo del algoritmo de entrenamiento en todo el conjunto de entrenamiento con *minibatchs* es un *epoch*.

Este método es más rápido y se requieren menos cálculos para actualizar los parámetros de la red neuronal que realizando la evaluación del gradiente sobre todo el conjunto de datos de entrenamiento. Además, el cálculo simultáneo del gradiente para muchos ejemplos de entrada puede realizarse utilizando operaciones con matrices que se implementan de forma muy eficiente con GPU.

3.5.4 Parametrización de los modelos

Hay modelos que permiten mejorar la precisión (*accuracy*), pero para esto hay que tener mucho conocimiento y práctica para manejarse bien con los parámetros que pueden ser ajustados: por ejemplo, con un simple cambio de la función de activación de la primera capa, pasando de una *sigmoid* a una *ReLU*, se puede obtener una mejora de la precisión con un tiempo de cálculo aproximadamente el mismo. También es posible aumentar el número de *epochs*, agregar más neuronas en una capa o agregar más capas. Sin embargo, en estos casos las mejoras en precisión penalizan el tiempo de ejecución del proceso de aprendizaje. En definitiva, que encontrar la mejor arquitectura con los mejores parámetros de las funciones de activación requiere cierta pericia y experiencia dadas las múltiples posibilidades.

A continuación, se va a introducir alguno de los parámetros más habituales.

a. Número de epochs

El número de veces en las que todos los datos de entrenamiento han pasado por la red neuronal durante el proceso de entrenamiento se indica mediante el parámetro *epochs*.

b. Batch size

Tal como se ha comentado antes, se pueden organizar los datos de entrenamiento en pequeños lotes (*mini-batch*) para pasarlos por la red. El *batch-size* es el parámetro que indica el tamaño que se usará de estos lotes en una iteración del entrenamiento para actualizar el gradiente. El tamaño óptimo dependerá de muchos factores, entre ellos de la capacidad de memoria de la GPU utilizada para hacer los cálculos.

c. Learning rate

El vector de gradiente tiene una dirección y una magnitud. Los algoritmos de *gradient descent* multiplican la magnitud del gradiente por un escalar conocido como *learning rate* (α).

$$\theta_{n+1} = \theta_n - \alpha \cdot \nabla E(\theta) \qquad (17)$$

El valor adecuado de este parámetro depende del modelo, pero en general, si este es demasiado grande, se está dando pasos enormes que podría ser bueno para ir rápido durante el proceso de aprendizaje, pero es posible que no se logre obtener el mínimo de la función de perdidas, dificultando así el proceso de aprendizaje y penalizando la precisión en el resultado.

Por el contrario, si el *learning rate* es pequeño, se harán avances constantes y pequeños, mejorando el acercamiento al mínimo local, pero esto provocando que el proceso de aprendizaje sea muy lento.

El mejor *learning rate* en general es aquel que disminuye a medida que el modelo se acerca a una solución. Por ejemplo, en Matlab, para conseguir este efecto, se puede indicar mediante el parámetro '*LearnRateSchedule','piecewise'*, que se usa para disminuir el

learning rate a medida que van pasando *epochs* para permitir que el aprendizaje avance más rápido al principio con *learning rates* más grandes. A medida que se avanza, se van haciendo ajustes cada vez más pequeños para facilitar la convergencia al mínimo de la función de pérdidas durante el proceso de entrenamiento.

d. Momentum

El algoritmo de descenso de gradiente estocástico puede oscilar a lo largo del descenso mientras se acerca al valor mínimo local óptimo. Si se añade un término de tipo impulso a la actualización del parámetro se puede llegar a reducir esta oscilación. El descenso de gradiente estocástico con actualización de momento se define como:

$$\theta_{n+1} = \theta_n - \alpha \cdot \nabla E(\theta) + \gamma \cdot (\theta_n - \theta_{n-1}), \quad (18)$$

donde γ determina la contribución del paso de gradiente anterior a la iteración actual y se denomina *momentum*.

e. Gradient Clipping

Si el gradiente aumenta exponencialmente, entonces el entrenamiento se vuelve inestable y diverge en unas pocas iteraciones, este efecto se denomina *gradient explosion* y se indica mediante la función de perdidas (*loss*) que tiende a '*NaN*' o '*Inf*'. Como se ha explicado anteriormente, el gradiente toma la dirección que determina el incremento más rápido en el error, mientras que la dirección opuesta, es decir, la dirección negativa, determina el decremento más rápido en el error. Por tanto, el error puede reducirse ajustando cada peso en la dirección negativa del gradiente. Dependiendo de las capas de la red, si los pesos adquieren un valor alto, ya sea al inicio o durante el entrenamiento, provocan durante la propagación del error a través de la red que el gradiente se dispare o desvanezca, ya que este gradiente es función de todos los pesos de la red. De manera resumida podríamos expresar el gradiente como $\nabla E(\theta) = W^{T-1}$, siendo W los pesos y T el número de iteraciones del error. Si, por ejemplo, W toma un valor de 1,5 y suponemos que T son 20, el gradiente tomaría un valor de 3325,3 lo que se denominaría como un *gradient explosion* y se refleja a nivel computacional como un Infinito ('*Inf*') o *Not a number* ('*NaN*'). En el otro extremo, si W=0.9, el gradiente tomaría un valor de 0.1216, lo que se denominaría vanishing gradient. Es por eso, que W tiene que ser lo más próximo a 1 para no provocar estos efectos.

El parámetro *gradient clipping* ayuda a prevenir el efecto del *gradient explosion* limitando su valor, estabilizando el entrenamiento a tasas de aprendizaje más altas y en presencia de valores atípicos. El *gradient clipping* permite entrenar a las redes más rápido y, por lo general, no afecta la precisión de la tarea aprendida.

Hay dos tipos de *gradient clipping*:

1. Re-escalando el gradiente en función de un umbral y sin cambiar la dirección de este. Los parámetros '*l2norm*' y *'global-l2norm*' del método '*GradientThresholdMethod*' permiten ajustar el *gradient clipping*.

2. El parámetro '*absolute-value*' del método '*GradientThresholdMethod*' permite limitar cualquier derivada parcial superior a un valor umbral, lo que puede ocasionar que el gradiente cambie arbitrariamente de dirección. El uso de este método puede tener un comportamiento impredecible, pero los cambios suficientemente pequeños evitan que la red diverja.

f. L2 Regularization

Agregar un término de regularización para los pesos a la función de pérdidas E(θ) es una forma de reducir el *overfitting*. El término de regularización también se denomina *weight decay*. La función de pérdidas con el término de regularización toma la forma:

$$ER(\theta) = E(\theta) + \lambda \cdot \Omega(w), \tag{19}$$

donde w es el vector de ponderación, λ es el factor de regularización (coeficiente) y la función de regularización Ω(w) es:

$$\Omega(w) = \frac{1}{2} w^T w \tag{20}$$

Hay que tener en cuenta que los sesgos no están regularizados. Se puede especificar el factor de regularización λ utilizando el argumento '*L2Regularization*'.

Capítulo 4

Metodología de experimentación

4.1 *Conjunto de datos para entrenamiento (Training Dataset)*

Para la configuración y evaluación de un modelo en *Machine Learning*, y por ende *Deep Learning*, habitualmente se dividen los datos disponibles en tres conjuntos: datos de entrenamiento (*training*), datos de validación (*validation*) y datos de prueba (*test*). Aunque en muchos casos el conjunto de validación y de prueba es el mismo, normalmente, se suele dividir el conjunto de datos a utilizar un 70% de los datos para entrenamiento, un 10% para los datos de validación y un 20% para los de pruebas.

Los datos de entrenamiento son los que se utilizan para que el algoritmo de aprendizaje obtenga los parámetros del modelo. Puede ocurrir, una vez finalizado el entrenamiento, que el modelo se ajuste de tal modo a los datos de entrada que provoque que éste no sea capaz de realizar las predicciones correctas con nuevos datos que nunca ha visto antes. En estos casos, se dice que el entrenamiento presenta *overfitting*. Un indicador que advierte de este hecho es si se observa que el valor de precisión obtenido durante el entrenamiento con los datos de validación difiere bastante del valor de precisión obtenido durante la evaluación del modelo realizado con los datos de prueba. Para solucionar este problema, se debería modificar el valor de ciertos parámetros o incluso reducir el número de neuronas de las capas ocultas, para a continuación, volver a realizar un nuevo entrenamiento y evaluar de nuevo el modelo.

Si el caso que se presenta es que, durante el entrenamiento, no se obtiene un error bajo, se dice que el entrenamiento presenta *underfitting*. En este caso, el problema estriba en la falta de datos de entrenamiento o también que el número de neuronas elegido para las capas ocultas es demasiado bajo.

Es habitual evaluar las arquitecturas de las redes neuronales propuestas empleando bases de datos de imágenes de forma que sea posible comparar de forma cuantitativa los resultados obtenidos en los diferentes entrenamientos realizados.

Existen una multitud de bases de datos que son ampliamente utilizadas para el análisis de comportamientos. A continuación, se presentan las que serán utilizadas en este trabajo tanto para realizar los respectivos entrenamientos.

4.1.1 Standford 40

El *dataset Stanford40* [29] contiene imágenes de personas que realizan 40 acciones diferentes (ver Figura 41), como por ejemplo, aplaudir, escalar, cocinar, beber, pescar y pasear al perro. Se proporciona por cada imagen, la información (*ground truth*) de las coordenadas correspondientes a un cuadro delimitador o *bounding box* de la persona que está realizando la acción, indicando también en esta anotación, el nombre del archivo de la imagen. Hay 9532 imágenes de tamaños que van desde los 300x400 pixeles hasta los 1000x 600 pixeles, en total con 180-300 imágenes por clase de acción.

Este *dataset* será utilizado para realizar el entrenamiento de la red *Faster-RCNN*.

Fuente: Human action recognition by learning bases of action attributes and parts.,» de *ICCV*, 2011.

Figura 41 Ejemplo de imágenes de las 40 acciones del *dataset Stanford40*

4.1.2 Weizmann Dataset

La base de datos *Weizmann* fue creada por la *Universidad Machon Weizmann* (Rehovot, Israel) para la verificación de un nuevo algoritmo de detección de actividad humana [11].

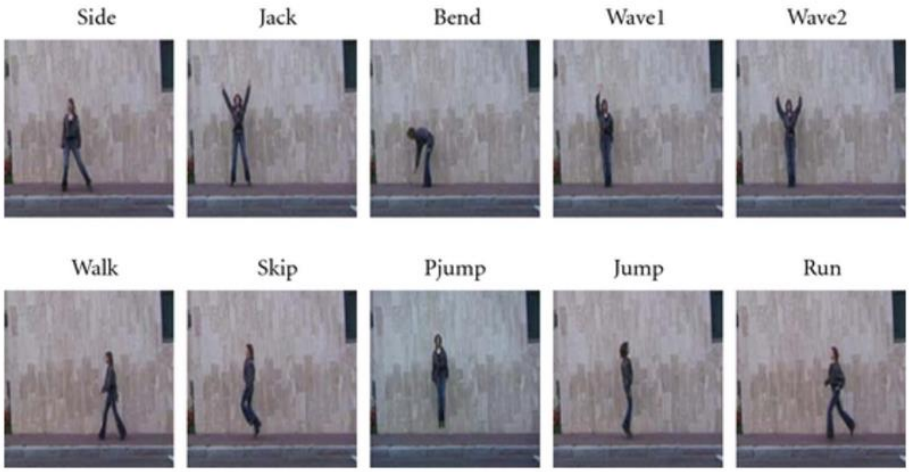
La base de datos *Weizmann* está formada por 10 clases de acciones: Andar o *Walk*, Correr o *Run*, Saltar o *Jump*, Galopar o *Gallop Sideways*, Agacharse o *Bend*, Saludar con Una Mano o *One-hand Wave*, Saludar con Dos Manos o *Two-hands Wave*, Saltar en el Sitio o *Jump in Place*, Saltar a la Comba o *Jumping Jack* y Saltar con una Pierna o *Skip*. Estas acciones están realizadas por 9 actores diferentes. En total la base de datos está formada

por 93 vídeos en formato .avi, ocupando un tamaño total de unos 454MB. Las imágenes hacen uso de una cámara estática, con una resolución de 180x144 pixeles.

Esta base de datos se desarrolla en dos experimentos diferenciados, de los cuales para este trabajo se utilizarán los de la primera fase.

La primera fase, o los primeros experimentos, son grabados en baja resolución a través de una cámara estática de 180x144 píxeles a 25fps colocada frente al actor de las acciones.

El escenario de grabación es un fondo homogéneo, en exteriores, realizando la acción de manera individual. En este tipo de grabaciones no hay distintos enfoques ni tampoco posibilidad de interacción o aparición de nuevas personas en la imagen.



Fuente: "Actions as Space-Time Shapes". Instituto de ciencias de Weizmann, 2007

Figura 42 Ejemplo de imágenes de las 10 acciones del *dataset Weizzman*

4.1.3 KTH Dataset

Esta base de datos creada por el Real Instituto de Tecnología de Estocolmo, Suecia [10], está compuesta por 600 vídeos estudiando seis tipos de acciones diferentes realizadas por 25 actores Andar o *Walk*, Correr Despacio o *Jogging*, Correr o *Run*, Boxear o *Boxing*, Sacudir las Manos o *Hand Waving* y Aplaudir o *Hand Clapping*. En la Figura 43, se muestran algunos ejemplos de imágenes de esta base de datos.

Es considerada la base de secuencias de acciones pública de mayor tamaño por lo que es la base de datos estándar en el análisis del comportamiento humano. La base de datos está desarrollada en cuatro escenarios diferentes:

- Al aire libre
- Al aire libre con variación de escala

- Al aire libre con cambio de vestuario del actor
- En el interior de una habitación con el fondo blanco

Los vídeos están grabados con una resolución de 160x120 pixels Todas las secuencias (en total 2391 secuencias), en blanco y negro, son tomadas sobre fondos homogéneos a través de una cámara estática con una resolución de 160x120 píxeles, a una velocidad de 25fps y con una técnica de grabación en RGB.

Fuente: http://www.nada.kth.se/cvap/actions/

Figura 43 Ejemplo de imágenes de las 6 acciones del *dataset* KTH

4.1.4 GBA Dataset

GBA es la base de datos desarrollada por el grupo GEINTRA de la UAH [12]. Consta de una colección de videos grabados a una resolución de 720x1280 y 50fps. Para la realización del entrenamiento de la red *Faster-RCNN*, se han utilizado fotogramas correspondientes a los vídeos 17, 19, 21, 22 correspondientes al año 2016, los cuales están etiquetados con su respectivo *ground truth* [30]. Estos videos constan de 20954 imágenes y se ha procedido a utilizar 15671 *bounding box* que se corresponden con las siguientes acciones:

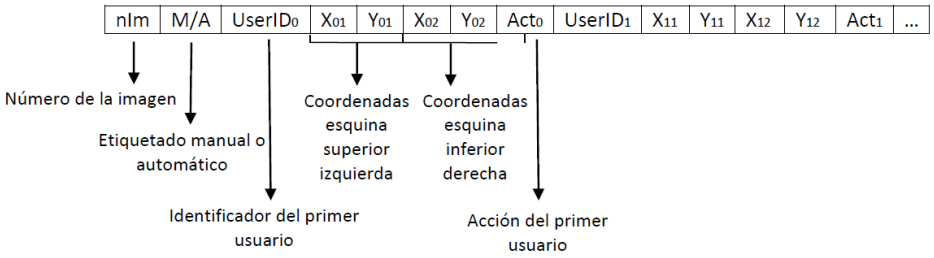
Acción	Nº Bounding Boxes
Andar (*walking*)	10239
Caerse (*falling*)	1477
Correr (*running*)	2384

Pararse (*standing up*)	6220
Sentarse (*sitting down*)	1549

Tabla 2 Número de secuencias por acción correspondientes al dataset GBA

En estos archivos *ground truth*, como se observa en la Figura 44, se almacenan una línea por cada imagen etiquetada, la cual contiene la siguiente información:

- El número de la imagen dentro de la secuencia.
- Una variable que indica si el etiquetado es manual (1) o automático (0).
- Por cada usuario etiquetado en la imagen aparecerán 6 valores:
  - El identificador del usuario.
  - Las cuatro coordenadas "x" e "y" de la esquina superior izquierda y la esquina inferior derecha.
  - El número asociado a la acción realizada (0:andar; 1:correr; 2:sentarse; 3:caerse; 4:parado).



Fuente: Boggian Arévalo 2016

Figura 44 Formato del archivo Ground Truth de la base de datos GBA

A continuación, se muestra un mosaico con imágenes correspondientes al *dataset* GBA en las cuales se recuadran las cinco acciones mencionadas utilizando las coordenadas proporcionadas por la información contenida en los *ground truth.*

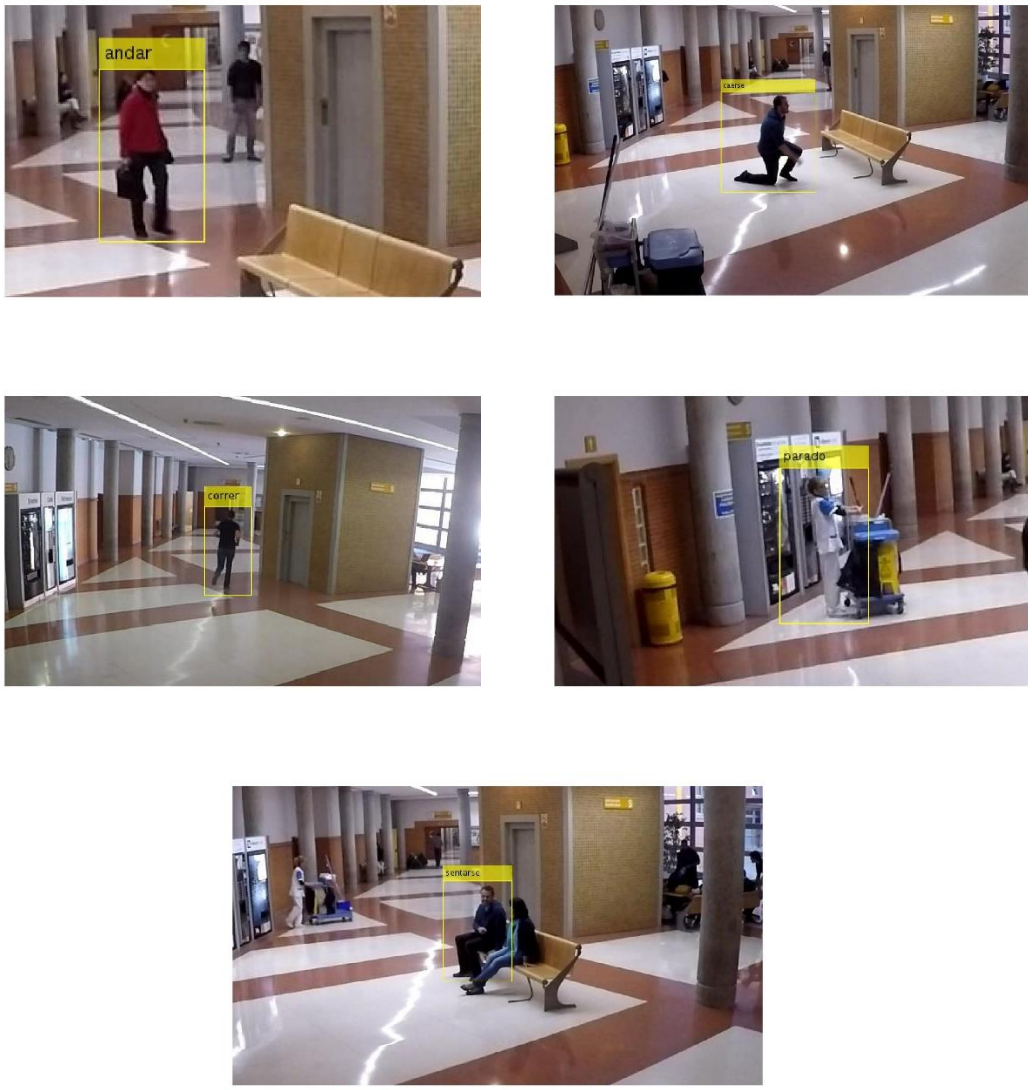


Figura 45 Ejemplo de imágenes de las 5 acciones del dataset GBA

Para la realización del entrenamiento con la red *LSTM* se han utilizado videos recortados a 50fps, de tamaño 180x320 pixeles y de duración estimada entre 1 y 3 segundos aproximadamente correspondientes a este *dataset*. Se ha realizado una modificación o data *augmentation* de las imágenes correspondientes a las secuencias utilizadas, la cual se explica en el apartado 4.3.

4.2 *Métricas de evaluación*

4.2.1 Métrica utilizada en la evaluación del detector de personas

mAP es un tipo de métrica utilizada para evaluar la precisión de un detector de objetos como el utilizado en este trabajo. Este tipo de métrica es utilizado en competiciones como Pascal VOC [31] o COCO [32].

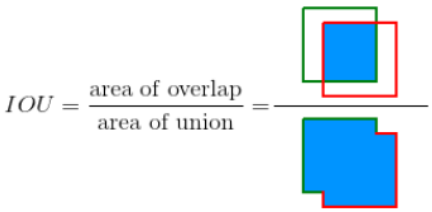
Para obtener este parámetro primero hay que aclarar brevemente determinados conceptos.

Como ya se ha comentado anteriormente, IoU (Intersección sobre la unión) es una medida que evalúa la superposición entre dos 2 regiones delimitadas. Esto mide como de buena es la predicción del detector de objetos comparada con la información proporcionada por el *Ground Truth* (límites reales del objeto).

Partiendo de una región proporcionada por el *ground truth* que denominaremos A y de una región obtenida con el detector denominada B, el valor de *IoU* indicará si la detección es válida, Verdadero positivo, o no, es decir, un falso positivo.

IOU viene dado por el área de superposición entre la región proporcionada por el detector y la región proporcionada por el *ground truth* dividido por el área de unión entre ellos (ver ecuación (3))

La Figura 46 muestra el IoU entre una región proporcionada por el *ground truth* (verde) y una región detectada (rojo)



Fuente: https://people.cs.pitt.edu/~kovashka/cs1699/hw4.html

Figura 46 Ejemplo de cálculo de IoU

- **Verdadero Positivo o *True Positive* (TP)**: Se trata de una detección correcta. Es una detección con un valor de *IoU* >= valor de umbral o *threshold.*
- **Falso Positivo o *False Positive* (FP)**: Se trata de una detección errónea. Es una detección con un valor de *IoU* < valor de umbral o *threshold*
- **Falso negativo o *False Negative* (FN)**: Se trata de una región definida por un *ground truth* que no ha sido detectada.
- **Verdadero Negativo o *True Negative* (TN)**: No se utiliza para el cálculo de esta métrica. Representa una detección errónea. En la detección de objetos hay muchas posibles regiones delimitadas que no deberían detectarse dentro de una imagen. Por lo tanto, TN sería todas las regiones posibles que no se detectaron correctamente. Es por eso que no es utilizado por las métricas.
- **Umbral o *threshold***: según la métrica, generalmente se establece en 50%, 75% o 95%. Pascal VOC utiliza un valor de 50%, mientras que COCO utiliza diferentes valores desde 50 hasta 95%.

La precisión mide qué tan precisas son las predicciones, es decir, cual es el porcentaje de las predicciones positivas son correctas.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{total\ de\ detecciones} \quad (21)$$

La sensibilidad o exhaustividad o *Recall* es la capacidad de un modelo para encontrar todos los casos relevantes (todas las regiones proporcionadas por los *ground truth*). Es el porcentaje de verdaderos positivos detectado entre todas los *ground truth* y viene dado por:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{total\ de\ \text{ground truths}} \quad (22)$$

La curva de precisión x exhaustividad es una buena forma de evaluar el rendimiento de un detector de objetos a medida que cambia el nivel de confianza. Hay una curva para cada clase de objeto. Un detector de objetos de una clase particular se considera bueno si su predicción se mantiene alta a medida que aumenta la exhaustividad, lo que significa que, si varía el umbral de confianza, la precisión y la exhaustividad seguirán siendo altas.

Un detector de objetos malo necesita aumentar el número de objetos detectados para recuperar todos los *ground truth*. Es por eso por lo que la curva de precisión x exhaustividad generalmente comienza con valores de alta precisión, disminuyendo a medida que aumenta la exhaustividad. Este tipo de curva es utilizada por el desafío PASCAL VOC 2012.

La precisión media es otra forma de comparar el rendimiento de los detectores de objetos es calcular el área bajo la curva de precisión x exhaustividad. Como las curvas AP a menudo son curvas en zigzag (ver Figura 47) que frecuentemente suben y bajan, la comparación de curvas diferentes (detectores diferentes) generalmente no es una tarea fácil, porque las curvas tienden a cruzarse entre sí con mucha frecuencia. Es por eso que *Average Precision* (AP), es una métrica numérica que permite comparar diferentes detectores. En la práctica, AP es la precisión promediada en todos los valores de exhaustividad o *recall* entre 0 y 1.

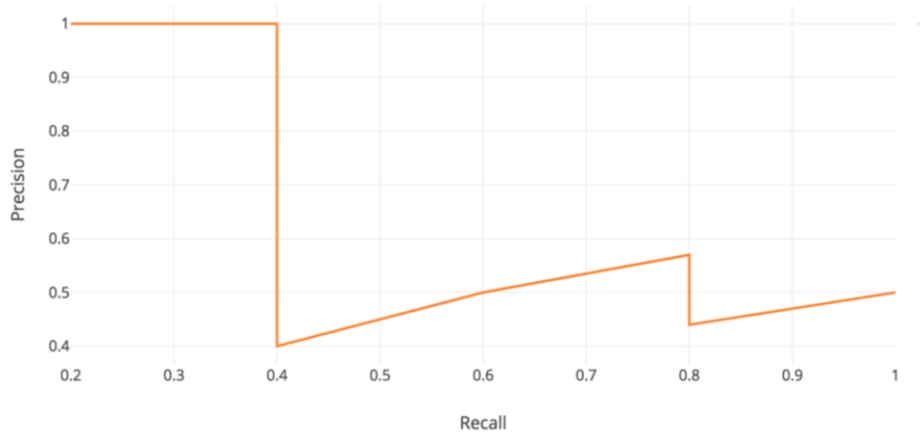


Figura 47 Ejemplo de curva de precisión x exhaustividad en zigzag

4.2.2 Métrica de evaluación del predictor de acción

Las bases de datos de secuencias de video utilizadas para el entrenamiento y su posterior evaluación se basan en los *dataset* KTH, GBA y *Weizzman*, descritas anteriormente.

La métrica utilizada para cuantificar la precisión del detector de acción se obtiene simplemente aplicando la siguiente ecuación:

$$Precisión = \frac{\sum(Acciones\ predichas == Acciones\ etiquetadas)}{\sum Acciones\ etiquetadas} \quad (23)$$

Donde *Acciones predichas* son los nombres de las acciones proporcionadas por la red de detección de acciones basada en *LSTM* y *Acciones etiquetadas* son los nombres de las acciones que han sido asignadas a un conjunto de secuencias de videos cuyas acciones son conocidas.

4.3 *Transformación de imágenes (Data Augmentation)*

Los conjuntos de datos más populares tienen del orden de decenas de miles de imágenes o más. Tener un gran conjunto de datos es crucial para un buen rendimiento. Pero aun teniendo pocas imágenes la red neuronal puede funcionar bien. Como ya se ha explicado anteriormente, cuando se entrena un modelo de aprendizaje automático, lo que realmente se está haciendo es ajustar sus parámetros de manera que pueda asignar una entrada particular (por ejemplo, una imagen) a alguna salida (una etiqueta). Si el modelo tiene muchos parámetros, se tendrá que mostrar al modelo de aprendizaje automático una cantidad proporcional de ejemplos para obtener un buen rendimiento. Además, la cantidad de parámetros que se necesita es proporcional a la complejidad de la tarea que el modelo debe realizar.

Para obtener más datos, solo se necesita hacer modificaciones menores al conjunto de datos existente. Cambios menores tales como volteos, translaciones o rotaciones. La red neuronal interpretará que estas son imágenes distintas.

Una red neuronal convolucional que puede clasificar robustamente objetos incluso si se coloca en orientaciones diferentes se dice que tiene la propiedad llamada invarianza. Más específicamente, una CNN puede ser invariante para la translación, el punto de vista, el tamaño, la iluminación o una combinación de los anteriores.

Esta es esencialmente la premisa de *data augmentation*. En la práctica, se tiene un conjunto de datos de imágenes obtenidas en un contexto limitado de condiciones. Sin embargo, la aplicación a desarrollar puede funcionar en una variedad de condiciones diferentes, como variación de la orientación, de la ubicación, escalas diferentes, niveles de

brillo, etc. Para ello, se generan estas situaciones entrenando la red neuronal con datos adicionales modificados.

Existen dos opciones para realizar la transformación de las imágenes. Una opción es realizar todas las transformaciones necesarias de antemano, esencialmente aumentando el tamaño de su conjunto de datos. La otra opción es realizar estas transformaciones justo antes de introducir el conjunto de imágenes al modelo de aprendizaje.

La primera opción se conoce como transformación *offline* y es la realizada en este trabajo. Este método es adecuado para conjuntos de datos relativamente más pequeños, ya que se consigue aumentar el tamaño del conjunto de datos por un factor igual al número de transformaciones que se realiza, por ejemplo, al voltear todas las imágenes, se aumentaría el tamaño del conjunto de datos un factor de 2.

La segunda opción se conoce como transformación *online*. Este método es adecuado para conjuntos de datos más grandes, ya que, al no poder realizar la transformación previa de un tamaño tan grande de datos, en su lugar, se realiza las transformaciones en cada *minibatch* de datos de entrada al modelo de aprendizaje.

Algunas de las técnicas de aumento básicas, pero más potentes que se utilizan popularmente son:

1. Volteo o *Flip*: se realiza un volteo de las imágenes de manera horizontal y vertical.

2. Rotación o *Rotate*: Se realiza un giro de las imágenes. Se debe tener en cuenta que las dimensiones de la imagen pueden no conservarse después de la rotación.

3. Escalar o *Resize*: La imagen se puede escalar hacia afuera o hacia adentro. Al escalar hacia afuera, el tamaño de la imagen final será mayor que el tamaño de la imagen original. Al escalar hacia adentro se reduce el tamaño de la imagen.

4. Recortar o *Crop*: Se realiza un recorte de una determinada zona de la imagen.

5. Translacion o *Translation*: La translación solo implica mover la imagen a lo largo de la dirección X, Y o ambas. Este método de aumento es muy útil ya que la mayoría de los objetos se pueden ubicar en casi cualquier lugar de la imagen. Esto obliga a la red neuronal convolucional a buscar en todas partes.

6. Ruido Gaussiano o Noise: El overfitting generalmente ocurre cuando la red neuronal intenta aprender características de alta frecuencia, patrones con alta ocurrencia, que pueden no ser útiles. El ruido gaussiano, que tiene una media cero, tiene puntos de datos en todas las frecuencias, distorsionando de manera efectiva las características de alta frecuencia. Agregar la cantidad justa de ruido puede mejorar la capacidad de aprendizaje.

7. Brillo y Contraste o brightness & contrast: Se incrementa o decrementa el nivel de brillo y contraste de las imágenes.

A veces no todas las técnicas de data augmentation tienen sentido para un conjunto de datos. En este trabajo, dado que se va a aplicar a secuencias de imágenes ya recortadas no tiene sentido aplicar la opción de *Crop*. Tampoco se aplicará la opción de rotación o *Rotate* ya que tampoco tiene sentido en una secuencia temporal de imágenes que estás estén giradas en diferentes ángulos. Se realiza un *data augmentation* aplicando el volteo

horizontal o *flip*, aumento de brillo y contraste, la combinación de ambas y finalmente se añade también ruido a las imágenes.

En las pruebas realizadas en este trabajo se ha observado que no se consigue mejoría entrenado el *dataset* con todas las opciones de *data augmentation* descritas, respecto a aplicar únicamente la opción de volteo horizontal o *flip*, es por esto que en la mayoría de las evaluaciones realizadas únicamente se aplica este tipo de efecto (ver Figura 48).

Secuencia de video original **Secuencia de video volteada (flip)**

Figura 48 Ejemplo del resultado de aplicar *data augmentation* a una secuencia de video correspondiente al *dataset* GBA

Capítulo 5

Arquitectura del sistema

5.1 *Detección de personas*

Como paso inicial para construir un modelo basado en el detector *Faster-RCNN*, se procedió a intentar crear una arquitectura que se adaptase al tipo de objetos y escenarios que se pretenden tratar en este trabajo. Para ello, se implementó una arquitectura con una serie de parámetros que permitieran que el proceso de entrenamiento de la red neuronal profunda fuera el más eficiente.

Primeramente, se define una capa de entrada '*ImageInputLayer*'. En esta capa se definen tres parámetros fundamentales como son la altura y anchura de la imagen, el número de canales y el método de normalización de los datos.

Los valores de altura y anchura de la imagen dependen del mínimo tamaño esperado del objeto a detectar, en el caso que se plantea en este trabajo interesa que puedan ser detectadas las personas que se encuentran al fondo de una determinada imagen, por ello se selecciona un valor de 32x32. Respecto al número de canales, se utilizarán los 3 canales de datos correspondientes a los colores Rojo, Verde, Azul (RGB) estando los niveles de píxeles en el rango [0,255].

La normalización de datos es un paso importante que garantiza que cada parámetro de entrada (píxel, en este caso) tenga una distribución de datos similar. Esto hace que la convergencia del gradiente sea más rápida al entrenar la red. La normalización de datos se hace restando la media de cada píxel, y luego dividiendo el resultado entre la desviación estándar. La distribución de tales datos se asemejaría a una curva gaussiana centrada en cero.

$$\frac{x - x_{media}}{x_{std}} \qquad (24)$$

La siguiente capa, de tipo convolucional, aplica filtros que se van deslizando a través de los datos de entrada, tal como se explicó anteriormente (ver apartado 3.1.1 a). La capa convoluciona la entrada moviendo los filtros a lo largo de la entrada de modo vertical y horizontal y calcula el producto escalar de los pesos con la entrada, y añadiendo posteriormente un término de sesgo.

Los parámetros de configuración de esta capa convolucional serían de 32 filtros de tamaño 3x3, un valor de *stride* de 1, tanto en vertical como en horizontal, y un valor de *zero-padding* de tamaño 1 a lo largo de todos los bordes de la entrada de la imagen.

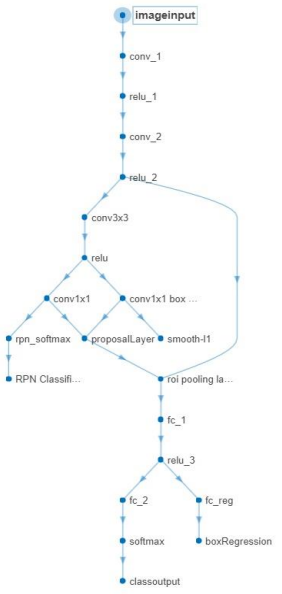
A continuación, se incluye una capa *ReLU* que realiza una operación en la cual cada elemento de entrada de valor inferior a cero se establece a cero. Y de nuevo, se repite otra capa de convolución con los mismos parámetros establecidos en la capa 2, a la cual le sigue otra capa *ReLU*.

Se aplica una capa *MaxPooling* para reducir la dimensión, el tamaño de la ventana es de 3x3, con un desplazamiento de 2 para el eje vertical y de 2 para el eje horizontal. Se añade una capa de conexión (‘*fullyConnectedLayer*’) con la cual se multiplica la entrada por los pesos correspondientes y se añade el sesgo correspondiente. Se configura con 64 neuronas lo que nos proporciona un vector de salida de tamaño 64.

De nuevo otra capa *ReLU* para rectificar los datos de la salida anterior se vuelve a insertar otra capa de conexión (’*fullyonnectedLayer*’) cuya salida contiene el número de clases más la clase *background*.

Finalmente, una capa de clasificación (‘*classificationLayer*’) en la cual proporciona la clase en función de la probabilidad de salida asignada en la capa *softmax* anterior.

En la Tabla 3, se muestra de manera resumida la arquitectura descrita anteriormente:



Nº LAYER	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	data 32x32x3 images with 'zero' normalization	Image Input	32x32x3	
2	conv1 32 3x3x3 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	32x32x32	Weights 3x3x3x32 Bias 1x1x32
3	relu1 ReLU	ReLU	32x32x32	
4	conv2 32 3x3x32 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	32x32x32	Weights 3x3x32x32 Bias 1x1x32
5	relu2 ReLU	ReLU	32x32x32	
6	conv3x3 32 3x3x32 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	32x32x32	Weights 3x3x32x32 Bias 1x1x32
7	relu3 ReLU	ReLU	32x32x32	
8	conv1x1 26 1x1x32 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	32x32x26	Weights 1x1x32x26 Bias 1x1x26
9	rpn_softmax rpn softmax	RPN Softmax		
10	RPN Classification Output cross-entropy loss with 'object' and 'background' classes	RPN Classification Output		
11	conv1x1 box regression (RPN) 52 1x1x32 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	32x32x52	Weights 1x1x32x52 Bias 1x1x52
12	smooth-l1 smooth-l1 loss	Box Regression Output		
13	proposalLayer region proposal with 13 anchor boxes	Region Proposal	1x4	
14	roi pooling layer ROI Max Pooling with pooled output size [15 15]	ROI Max Pooling	15x15x32	
15	fc1 64 fully connected layer	Fully Connected	1x1x64	Weights 64x7200 Bias 64x1
16	relu3 ReLU	ReLU	1x1x64	
17	fc2 2 fully connected layer	Fully Connected	1x1x2	Weights 2x64 Bias 2x1
18	softmax softmax	Softmax	1x1x2	
19	classoutput crossentropyex with classes 'object' and 'background'	Classification Output		
20	fcreg 4 fully connected layer	Fully Connected	1x1x4	Weights 4x64 Bias 4x1
21	boxRegression smooth-l1 loss	Box Regression Output		

Tabla 3 Arquitectura de red Faster-RCNN

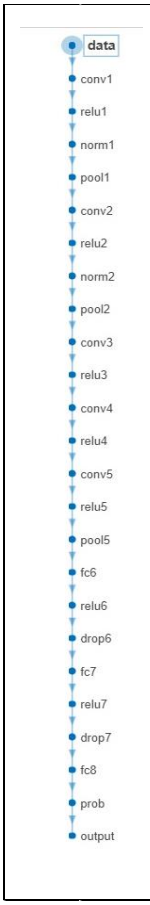
Este modelo de arquitectura propuesto es bastante complicado de entrenar ya que consume muchos recursos hardware. En el apartado 6.1, en el cual se muestran los resultados de las evaluaciones de los diferentes modelos planteados se indica como no se pudo entrenar la arquitectura debido a la falta de potencia de computación. Por este motivo se decidide partir de una *CNN* existente y realizar una transferencia del aprendizaje o *retraining*.

5.1.1 Transferencia de aprendizaje del modelo Alexnet

Este proceso de transferencia implica el ajuste de parámetros de un modelo previamente entrenado, en este caso *Alexnet* [7] y proporcionar datos nuevos que contienen clases previamente desconocidas por la red, en este caso las clases a utilizar serán las proporcionadas por el *dataset* GBA. La ventaja de la transferencia es que se necesitan muchos menos datos de entrada de forma que el tiempo de cálculo se reduce considerablemente.

Alexnet se desarrolló para ser ejecutadas sobre GPUs ya que son idóneas para paralelizar tareas de entrenamiento de las redes neuronales. En concreto se utilizó dos de GPUs (GTX 580) para entrenar una red convolucional con 650.000 neuronas, en base a 1.200.000 imágenes de alta definición pertenecientes a la base de datos de imágenes *Imagenet* [33]. Esta red, entró en la Competición *Imagenet* de 2012 [17], y ganó el premio en la prueba de reconocimiento Error de Test Top-5, alcanzando un error del 15.3% en la clasificación de 150.000 fotografías,de Flickr.

La arquitectura que presenta Alex-Net (ver Tabla 4) se basa en un modelo de red neuronal convolucional profunda con múltiples capas ocultas, incluye una capa de entrada, cinco capas convolucionales, tres capas de *pooling*, tres capas totalmente conectadas y una capa de salida con 1000 etiquetas de clases diferentes. En todo el modelo, la función de aprendizaje se lleva a cabo en la capa convolucional con dos canales que se procesan por separado, y se mezclan solo en la tercera capa de extracción de características. La primera capa totalmente conectada es la mezcla cruzada de las características provenientes de dos grupos, y posteriormente se repiten dos capas totalmente conectadas que se combinan para obtener un vector de características de dimensión 4096.



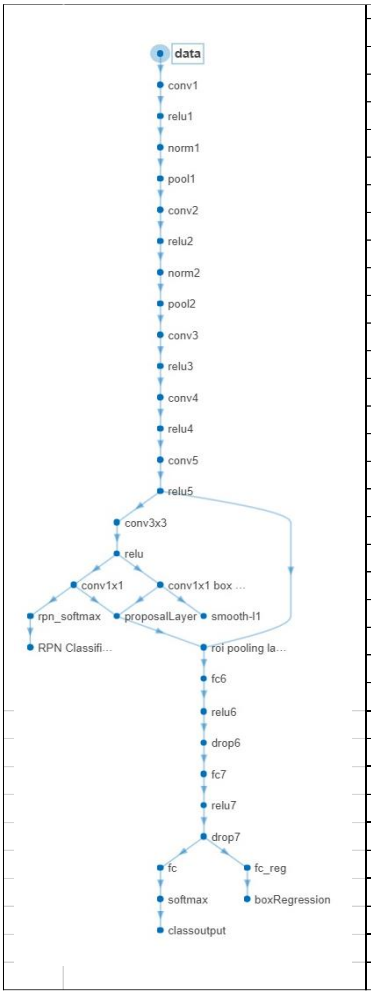
Nº LAYER	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	data 227x227x3 images with 'zero' normalization	Image Input	227x227x3	
2	conv1 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]	Convolution	55x55x96	Weights 11x11x3x96 Bias 1x1x96
3	relu1 ReLU	ReLU	55x55x96	
4	norm1 cross channel normalization with 5 channels per elemnet	Cross Channel Normalization	55x55x96	
5	pool1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	27x27x96	
6	conv2 256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	27x27x256	Weights 5x5x48x256 Bias 1x1x256
7	relu2 ReLU	ReLU	27x27x256	
8	norm2 cross channel normalization with 5 channels per elemnet	Cross Channel Normalization	27x27x256	
9	pool2 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	13x13x256	
10	conv3 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x384	Weights 3x3x256x384 Bias 1x1x384
11	relu3 ReLU	ReLU	13x13x384	
12	conv4 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x384	Weights 3x3x192x384 Bias 1x1x384
13	relu4 ReLU	ReLU	13x13x384	
14	conv5 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x256	Weights 3x3x192x256 Bias 1x1x256
15	relu5 ReLU	ReLU	13x13x256	
16	pool5 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	6x6x256	
17	fc6 4096 fully connected layer	Fully Connected	1x1x4096	Weights 4096x9216 Bias 4096x1
18	relu6 ReLU	ReLU	1x1x4096	
19	drop6 50% droput	Dropout	1x1x4096	
20	fc7 4096 fully connected layer	Fully Connected	1x1x4096	Weights 4096x4096 Bias 4096x1
21	relu7 ReLU	ReLU	1x1x4096	
22	drop7 50% droput	Dropout	1x1x4096	
23	fc8 1000 fully connected layer	Fully Connected	1x1x1000	Weights 1000x4096 Bias 1000x1
24	prob softmax	Softmax	1x1x1000	
25	output crossentropyex with 'tench' and 999 other classes	Classification Output		

Tabla 4 Arquitectura del modelo *Alexnet*

Para realizar la transferencia de aprendizaje, se extraen las últimas capas a la arquitectura de la *CNN Alexnet* (ver Tabla 4), en este caso, la capa 23, 24 y 25, de este modo se mantienen los pesos ya ponderados del resto de capas, y se agrega una nueva capa completamente conectada para el conjunto de datos nuevos que se van a entrenar y que contiene 6 clases (5 acciones correspondientes al *dataset* GBA más una denominada de fondo o *background*), además de una capa *softmax* y de una capa de salida de clasificación.

En la nueva capa *'fullyConnectedLayer'* es donde se asignan los parámetros de configuración para la transferencia de aprendizaje, de tal modo que se indica al algoritmo de entrenamiento, que se debe acelerar el proceso de aprendizaje respecto del resto de capas, ya que como se ha comentado no interesa que los pesos del resto de capas se modifiquen al mismo ritmo, para ello, se aumentaran los factores de aprendizaje. El parámetro '*WeightLearnRateFactor*' y '*BiasLearnRateFactor*' se fijan a 20, esto quiere decir, que el algoritmo utilizará para esta capa un valor de *learn rate*, tanto para los pesos como para el sesgo, de valor 20 veces el valor global de *learn rate* fijado para entrenar todo el modelo.

En la Tabla 5, se muestra como quedaría configurado el modelo de la arquitectura aplicando la transferencia de aprendizaje.



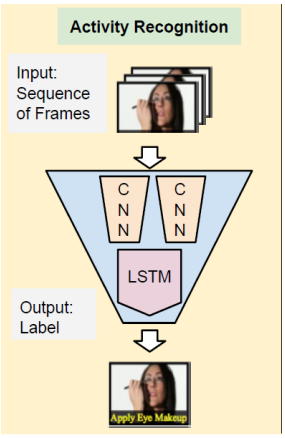
Nº LAYER	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	data 227x227x3 images with 'zero' normalization	Image Input	227x227x3	
2	conv1 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]	Convolution	55x55x96	Weights 11x11x3x96 Bias 1x1x96
3	relu1 ReLU	ReLU	55x55x96	
4	norm1 cross channel normalization with 5 channels per elemnet	Cross Channel Normalization	55x55x96	
5	pool1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	27x27x96	
6	conv2 256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	27x27x256	Weights 5x5x48x256 Bias 1x1x256
7	relu2 ReLU	ReLU	27x27x256	
8	norm2 cross channel normalization with 5 channels per elemnet	Cross Channel Normalization	27x27x256	
9	pool2 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	13x13x256	
10	conv3 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x384	Weights 3x3x256x384 Bias 1x1x384
11	relu3 ReLU	ReLU	13x13x384	
12	conv4 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x384	Weights 3x3x192x384 Bias 1x1x384
13	relu4 ReLU	ReLU	13x13x384	
14	conv5 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x256	Weights 3x3x192x256 Bias 1x1x256
15	relu5 ReLU	ReLU	13x13x256	
16	conv3x3 256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x256	Weights 3x3x256x256 Bias 1x1x256
17	relu ReLU	ReLU	13x13x256	
18	conv1x1 32 1x1x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	13x13x32	Weights 1x1x256x32 Bias 1x1x32
19	rpn_softmax rpn softmax	RPN Softmax	169x16x2	
20	RPN Classification Output cross-entropy loss with 'object' and 'background' classes	RPN Classification Output		
21	conv1x1 box regression (RPN) 64 1x1x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	13x13x64	Weights 1x1x256x64 Bias 1x1x64
22	smooth-l1 smooth-l1 loss	Box Regression Output		
23	proposalLayer region proposal with 16 anchor boxes	Region Proposal	1x4	
24	roi pooling layer ROI Max Pooling with pooled output size [6 6]	ROI Max Pooling	6x6x256	
25	fc6 4096 fully connected layer	Fully Connected	1x1x4096	Weights 4096x9216 Bias 4096x1
26	relu6 ReLU	ReLU	1x1x4096	
27	drop6 50% droput	Dropout	1x1x4096	
28	fc7 4096 fully connected layer	Fully Connected	1x1x4096	Weights 4096x4096 Bias 4096x1
29	relu7 ReLU	ReLU	1x1x4096	
30	drop7 50% droput	Dropout	1x1x4096	
31	fc 2 fully connected layer	Fully Connected	1x1x2	Weights 2x4096 Bias 2x1
32	softmax softmax	Softmax	1x1x2	
33	classoutput crossentropyex with classes 'persona' and 'background'	Classification Output		
34	fcreg 4 fully connected layer	Fully Connected	1x1x4	Weights 4x4096 Bias 4x1
35	boxRegression smooth-l1 loss	Box Regression Output		

Tabla 5 Arquitectura red Faster RCNN con transferencia de aprendizaje de *Alexnet*

5.2 *Predicción de acciones*

En el trabajo desarrollado por la División de Computación de la Universidad de California [8] se propone un modelo de Red Convolucional recurrente a largo plazo o *Long-term Recurrent Convolutional Networks (LRCN)*.

LRCN combina un extractor de características visuales basado en una *CNN* con un modelo basado en una red *LSTM*, que como se ha detallado en el capítulo 3, puede aprender a reconocer y sintetizar dinámicas temporales en tareas que implican datos secuenciales, en este caso, de tipo visual. Este modelo *LRCN* funciona pasando cada entrada visual o *frame,* a través de una extracción de características, para producir una representación vectorial de longitud fija que luego es utilizada, por una red de tipo recursivo de tipo *LSTM*, para el reconocimiento de la actividad (ver Figura 49).



Fuente: (Long-term recurrent convolutional networks for visual recognition and description., 2015)

Figura 49 Modelo LRCN de reconocimiento de actividad

5.2.1 Extracción de características

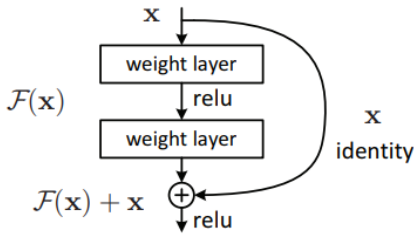
La extracción de características se considera como la activación de una determinada capa de la red, y en este caso, se realiza utilizando una de las capas totalmente conectada correspondiente a un modelo pre-entrenado como poder ser *Alexnet* o *Resnet50*. Este tipo de capas, tal como se ha comentado anteriormente en este trabajo, se sitúan tras la última de las capas de agrupación para convertir los mapas de características bidimensionales en un vector unidimensional. Las capas totalmente conectadas trabajan como una red neuronal tradicional y contienen la mayoría de los parámetros de la red. El vector unidimensional de salida de la red suele ser de longitud predefinida, por ejemplo, si se utiliza la capa totalmente conectada denominada como '*fc7*' de *Alexnet*, la salida proporcionada será un vector unidimensional de 4096 valores, mientras que la activación de la capa '*fc1000*' de *Resnet50* devuelve un vector unidimensional de tamaño 1000.

ResNet50 [34] es una red neuronal convolucional, compuestas por 177 capas, creada por el equipo de Microsoft [34] que ganó la competición ILSRVC2015 [35], superando el rendimiento de un humano al aplicarlo sobre el conjunto de datos *ImageNet* [17] [33].

ResNet50 es la abreviatura para definir Red Residual o *Residual Network*. Como indica el propio nombre de la red, la ventaja que introduce esta red es la del aprendizaje residual. El aprendizaje residual intenta resolver los problemas derivados de la dificultad de entrenar redes neuronales profundas como son la saturación de la precisión y su rápida degradación.

Tal como se ha explicado a lo largo de este trabajo, en una red neuronal convolucional profunda, se apilan varias capas y se entrena para la tarea en cuestión. La red aprende varias características de bajo, medio y alto nivel al final de sus capas. El aprendizaje residual (ver Figura 50), en vez de tratar de aprender algunas características, se trata de aprender con los residuos que se denominan F(x). Un residuo se puede entender como la resta de la característica aprendida proporcionada por la capa de entrada n, que se denominará identidad (x), al mapa de características que se desea obtener en la capa n+x, denominado H(x). *Resnet50* realiza el entrenamiento de este residuo que es más fácil, proporcionando en su salida el mapa de características deseado H(x) = F(x) + x. El entrenamiento de esta

forma de redes es más fácil de entrenar que las redes neuronales convolucionales profundas simples y también se resuelve el problema de degradación de la precisión.



Fuente: Deep Residual Learning for Image Recognition.2015

Figura 50 Bloque de aprendizaje residual

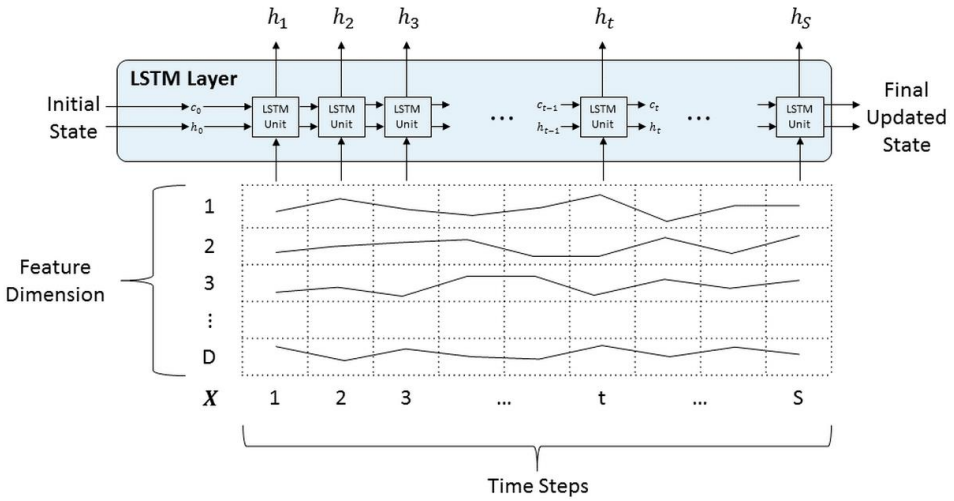
La capa '*fc1000*', igual que la capa '*fc7*' de *Alexnet*, es una capa totalmente conectada que se ubica en el final de la red *Resnet*50 (ver Figura 51). Este tipo de capa, tal como se comentó en el capítulo 3, aprenden combinaciones de las características aprendidas de las capas anteriores.

171	'bn5c_branch2c'	Batch Normalization	Batch normalization with 2048 channels
172	'add_16'	Addition	Element-wise addition of 2 inputs
173	'activation_49_relu'	ReLU	ReLU
174	'avg_pool'	Average Pooling	7x7 average pooling with stride [7 7] and padding [0 0 0 0]
175	'fc1000'	Fully Connected	1000 fully connected layer
176	'fc1000_softmax'	Softmax	softmax
177	'ClassificationLayer_fc1000'	Classification Output	crossentropyex with 'tench' and 999 other classes

Figura 51 Ubicación de la capa 'fc1000' dentro del modelo Resnet50

5.2.2 Arquitectura del modelo predictor de acciones

La Figura 52, muestra cómo se estructuran los datos que contienen los vectores de características de cada *frame,* de una determinada secuencia de video, para que la red *LSTM* pueda entrenarlos correctamente. Por ejemplo, una determinada secuencia de video cuya duración es de 1 segundo y está grabado a 25fps, constaría de un array de datos con un tamaño DxS, siendo D la dimensión del vector de características (4096 valores si se utiliza la capa totalmente conectada '*fc7*' del modelo *Alexnet* ó 1000 si se trata de la capa *'fc1000'* del modelo *Resnet50*) y S el número de *frames* de la secuencia, en este caso, 25. De tal modo, que en el instante t_0 se introduce en la red *LSTM* el *frame* 1, en el t_1 el *frame 2*, así sucesivamente hasta llegar al último instante de tiempo t_S que contendrá el último *frame* de la secuencia de video.



Fuente: Mathworks (Matlab 2018a)

Figura 52 Secuencia de aprendizaje de la red LSTM

Una vez descrito cómo se estructura el modelo *LRCN*, se pasa describir la arquitectura base de entrenamiento de la misma, a partir de la cual se van a realizar diferentes evaluaciones del modelo variando diferentes parámetros del algoritmo de entrenamiento y en algunos casos modificando la estructura propia de la arquitectura del modelo empleado.

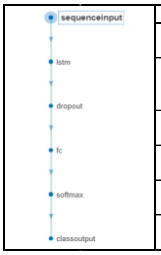
Un primer paso es especificar el tamaño de entrada de la red, el cual depende principalmente de la dimensión del vector unidimensional de los descriptores de características, que como se ha explicado anteriormente, utiliza un modelo pre-entrenado como es *Alexnet* o *Resnet50*. Dependiendo del modelo utilizado en la extracción de características, es necesario especificar en Matlab como parámetro de entrada a la red, la dimensión del vector obtenido, para ello, se utilizará la capa denominada '*SequenceInputLayer'*.

El siguiente paso es especificar el tipo de capa recursiva, en los diferentes entrenamientos se han utilizado capas de tipo *LSTM* y capas de tipo *LSTM* bidireccional, incluso la combinación de ambas de forma apilada. Dependiendo de la profundidad que se desee que tenga la red recursiva se deberán definir las unidades ocultas o el número de neuronas ocultas utilizadas por la red recursiva. En las arquitecturas empleadas en este trabajo se ha fijado un valor de 256 neuronas ocultas, ya que se ha observado que utilizar valores inferiores como 100 o valores superiores de hasta 1000 producen un resultado similar, incluso en algunos casos el establecer un valor alto ocasiona problemas de computación que pueden provocar errores durante el entrenamiento del modelo planteado. Otro parámetro que se debe indicar en la red *LSTM*, es el formato de salida de datos. Matlab permite seleccionar el modo '*sequence*', que asigna una etiqueta o un valor por cada dato de la secuencia, en el caso que se está tratando, este dato sería un vector de características que describe cada *fram*e; o seleccionar el modo *'last'*, que asigna una única etiqueta o valor que se correspondería con el resultado de predecir toda la secuencia

completa de *frames*. En algunos casos, como se verá más adelante, se utilizarán los dos modos cuando se conectan dos redes recursivas en cascada.

El siguiente paso es introducir una capa de tipo *dropout*, la cual se configura con un valor muy alto de 0.9, ya que se ha demostrado en otros trabajos [8], y en las pruebas realizadas, que para este tipo de entrenamientos de secuencias, valores altos de *dropout* pueden evitar problemas de *overfitting*. A esta capa *dropout,* le sigue una capa totalmente conectada de tamaño igual al número de clases o acciones a predecir, una capa de tipo *softmax* y finalmente, una capa de clasificación que proporciona el valor de la acción correspondiente a la secuencia de entrada.

En la Tabla 6, se muestra de manera resumida la arquitectura simple descrita anteriormente:



Nº LAYER	NAME	TYPE	ACTIVATIONS	LEARNABLES	STATES
1	sequenceinput Sequence input with 4096 dimension	Sequence Input	4096		
2	lstm LSTM with 256 hidden units	LSTM	256	InputWeights 1024x4096 RecurrentWeights 1024x256 Bias 1024x1	HiddenState 256x1 CellState 256x1
3	dropout 90% dropout	Dropout	256		
4	fc 5 fully connected layer	Fully Connected	5	Weights 5x256 Bias 5x1	
5	softmax softmax	Softmax	5		
6	classoutput crossentropyex with 'falling' and 4 other classes	Classification Output			

Tabla 6 Arquitectura simple de LRCN

5.3 *Detección de anomalías*

En este TFM se ha planteado que la técnica de detección de la anomalía se realice de manera independiente de la predicción de la acción, proponiendo la utilización de un método estadístico no supervisado para determinar si un suceso es ánomalo. A diferencia de otros trabajos, en los cuales se predice la anomália en base a un método supervisado, en el cual no se identifica la acción que transcurre en una determinada secuencia de video sino que se identifica directamente si la escena contiene un suceso anómalo [36], en este trabajo, se ha considerado que es más funcional predecir las acciones que transcurren en una determinada secuencia, y analizar a posteri, dependiendo del contexto en el cual transcurre si se trata de un suceso anómalo. Este mecanismo permite que determinadas acciones se pueden considerar anómalas o no, dependiendo del escenario o contexto en el cual se desarrollan.

El algoritmo que se plantea se divide en dos pasos:

1. Calcular la distribución de probabilidad de las diferentes acciones predecidas por la red *LRCN* en tiempo de ejecución, mediante la función *softmax*.
2. Mediante el Teorema de Bayes y conociendo las probabilidades de ocurrencia de las acciones dependiendo del contexto, se determina la probabilidad condicionada de anomalía de este.

El Teorema de Bayes es un procedimiento que permite expresar la probabilidad condicional de un evento aleatorio A dado B, en términos de la distribución de probabilidad del evento B, dado A, y la distribución de probabilidad de solo A.

La fórmula para calcular la probabilidad en este tipo de sucesos se define matemáticamente como:

$$P[A_n/B] = \frac{P[B/A_n] \cdot P[A_n]}{\sum P[B/A_i] \cdot P[A_i]} \quad (25)$$

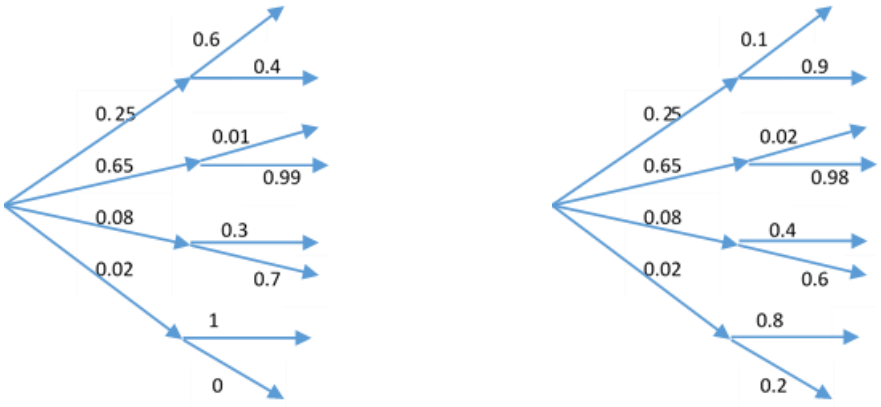
Donde $P[B/A_n]$ es la probabilidad de anomalía de un determinado suceso o acción en un determinado escenario o contexto y sobre el que se tiene información previa; y $P[A_n]$ es la probabilidad de los distintos sucesos condicionados o probabilidades de la ocurrencia de cada suceso obtenidas en tiempo de ejecución, mediante la función *softmax*.

Para comprobar que este tipo de técnica es útil a la vez que simple, se propone el siguiente ejemplo:

Las probabilidades de que una determinada acción se considere anomalía en dos escenarios diferentes, como son un parque o un aeropuerto, son las indicadas en la tabla siguiente:

Aeropuerto	**Parque**	**Probabilidades generadas por función *softmax***
Probabilidades conocidas del contexto	**Probabilidades conocidas del contexto**	
P[Anomalia/Correr]= 60%	P[Anomalia/Correr]= 10%	P[Correr]= 25%
P[Anomalia/Andar] =1%	P[Anomalia/Andar] =2%	P[Andar]= 65%
P[Anomalia/Caer]= 30%	P[Anomalia/Caer]= 40%	P[Caer]= 8%
P[Anomalia/Pelear]=100%	P[Anomalia/Pelear]=80%	P[Pelear]= 2%

Tabla 7 Probabilidad de anomalía en dos escenarios diferentes



a **B**

Figura 53 Diagramas de árbol asociado a la probabilidad condicionada en los dos escenarios. (a) aeropuerto (b) parque

Aplicando la ecuación (25), se puede calcular la probabilidad condicionada de que una determinada acción se considere anómala dependiendo de cada escenario, siendo estas para cada escenario las mostradas en la Tabla 8:

Acción	Aeropuerto	Parque
Correr	74.8%	29%
Andar	3.2%	15.1%
Caer	12%	37.2%
Pelear	10%	18.6%

Tabla 8 Probabilidad condicionada en diferentes escenarios para una misma acción

En los resultados de la Tabla 8 se observa que para una misma salida proporcionada por la función *softmax* se obtienen diferentes resultados dependiendo del escenario o contexto. Es este ejemplo, se observa como la acción correr en un aeropuerto da un valor de 74.8% de probabilidad de ser considerada como anomalía. Si se establece como umbral de anomalía un valor del 50%, para esta determinada secuencia, en la cual se acumulan varios sucesos correspondientes a la acción “correr” en un aeropuerto, se considerarían como una anomalía. Sin embargo, en el otro escenario, esta situación, cuyo porcentaje baja al 29%, no superaría el umbral establecido considerándose una situación normal.

Este método estadístico simple se demuestra que es bastante útil para determinar la existencia de anomalías y también es útil para filtrar falsos positivos. En los resultados obtenidos se observa que la salida de la función *softmax* ha dado una probabilidad de un 2% para la acción denominada “pelear”. Se podría establecer que este tipo de acción siempre debe ser considerado anomalía independientemente del escenario o contexto en el que ocurra, pero se puede dar el caso de que ese 2% sea obtenido por un falso positivo durante la predicción de las acciones, por lo que hasta que no se concatenen más sucesos de este tipo no debería ser considerado como anomalía. Con un incremento a un 15% de ocurrencia del suceso o acción “pelear”, en la función de sálida softmax, ya se podría asegurar que no es debido a un falso positivo sino a una reiteración de la acción, siendo ahora la probabilidad condicionada del 53.2%, sobrepasando el umbral establecido y, por lo tanto, ya se consideraría una anomalía.

Capítulo 6

Resultados

Una vez descrita la arquitectura del modelo de detección de personas que se pretende entrenar, es necesario primeramente identificar el hardware utilizado para los diferentes entrenamientos realizados a lo largo de este trabajo, el cual se basa en un ordenador portátil HP Omen con procesador Core i7-7700HQ @ 2.80GHz con 16Gb RAM y una GPU Nvidia GeForce GTX 1050 con 4Gb. El software utilizado para compilar el código es Matlab en su versión 2017b/2018a con Neural Network y Parallel Computing Toolboxes, además de utilizar driver CUDA 9.2.

6.1 *Resultados experimentales y evaluación del detector de personas*

Hay dos elementos comunes para todos los entrenamientos del modelo de detección de personas realizados, el conjunto de imágenes compuesto por las de entrenamiento (*train data*) junto con las de validación y test (*test data*), los cuales se basan en los *dataset* de *Stanford40 Actions* y el *dataset* GBA antes mencionados; y las anotaciones (*ground truth*), que contienen las posiciones de los diferentes elementos que aparecen en las imágenes incluidas en el dataset [30].

Para la realización de los entrenamientos utilizando el *dataset* GBA, es necesario adaptar el formato del archivo que contiene los datos de *ground truth* [30], a un formato compatible con las funciones de Matlab (ver el script en el apartado 8.1). Con los datos proporcionados por el *ground truth* se genera el archivo compatible tal como se muestra en la Figura 54.

En un primer intento se procedió a entrenar el modelo de red *Faster-RCNN* descrito en la Tabla 3 utilizando los parámetros de entrenamiento mostrados en la Tabla 9 y utilizando datos del *dataset* GBA. De las 15671 imágenes que componen este *dataset*, 12729 imágenes se utilizan para el entrenamiento y 2942 imágenes para la validación y test.

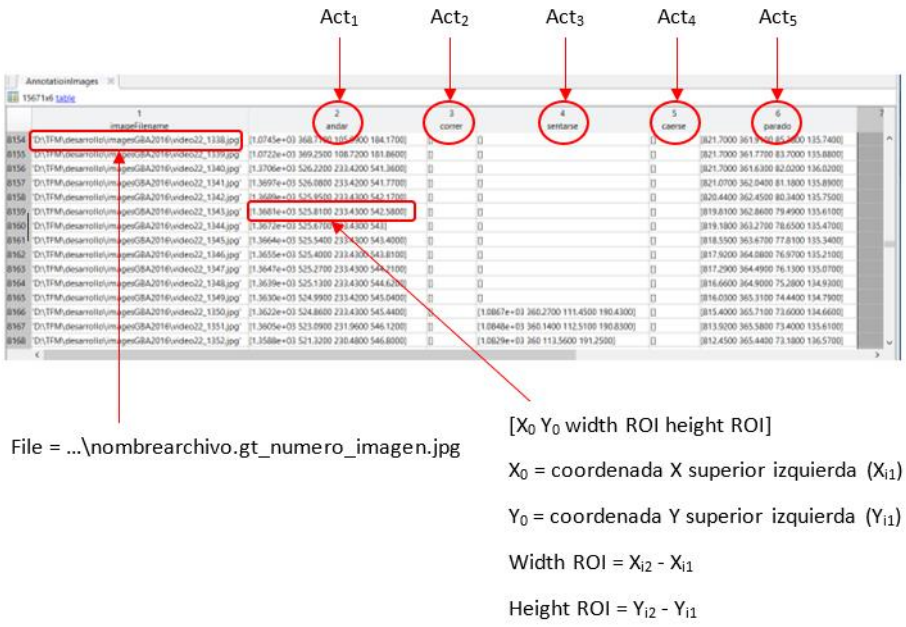


Figura 54 Ejemplo del formato de datos requerido por Matlab para entrenar un detector *Faster-RCNN*

Después de varios intentos fue imposible realizar el entrenamiento debido a una falta de capacidad computacional, por este motivo se decidió partir de una red ya entrenada como es *Alexnet* y realizar una transferencia de aprendizaje, tal como se ha explicado en el apartado 5.1.1.

Faster RCNN				
	Solver	MiniBatch Size	Max Epochs	Initial Learn Rate
Step 1	Sgdm	30	10	1e-5
Step 2	Sgdm	30	10	1e-5
Step 3	Sgdm	30	10	1e-3
Step 4	Sgdm	30	10	1e-3
Opciones Generales	Smallest Image Dimension		Box Pyramid Scale	
	400		1.2	

Tabla 9 Opciones de entrenamiento red *Faster-RCNN*

La nueva arquitectura planteada en la Tabla 5, parte de unos pesos y sesgos ya entrenados de la red *Alexnet.* Para realizar este nuevo entrenamiento se utiliza el 80% del *dataset* como imágenes de entrenamiento y el 20% restante como imágenes de test.

Después de numerosas pruebas se consiguió que el detector finalizara correctamente el entrenamiento configurando las opciones de entrenamiento de las cuatro etapas del detector con 10 *epoch* y un *minibatch* de 30. Para los valores de *learn rate* se establecen

valores bajos para que los pesos y sesgos ya entrenados no sean modificados drásticamente, con un valor de 1e-5 para las etapas 1 y 2 y de 1e-6 para las etapas 3 y 4.

Se fijan como parámetros de IoU para el '*NegativeOverlapRange*' un valor de 0 a 0.3 y para el '*PositiveOverlapRange*' de 0.6 a 1 y con un valor de '*Strongest Regions*' de 2000. Las imágenes son re-escaladas en su lado menor a un valor de 400 pixeles para reducir el coste computacional y evitar problemas de memoria. Y finalmente se establece un valor de '*BoxPyramidScale*' de 1.2.

La Tabla 10 muestra, de manera resumida, los parámetros de entrenamiento empleados.

Faster RCNN				
	Solver	**MiniBatch Size**	**Max Epochs**	**Initial Learn Rate**
Step 1	Sgdm	30	10	1e-5
Step 2	Sgdm	30	10	1e-5
Step 3	Sgdm	30	10	1e-6
Step 4	Sgdm	30	10	1e-6
Opciones Generales	**NegativeOverlapRange**		**PositiveOverlapRange**	
	0 a 0.3		0.6 a 1	
	Strongest Regions			
	2000			
	Smallest Image Dimension		**Box Pyramid Scale**	
	400		1.2	

Tabla 10 Opciones del 1er entrenamiento red *Faster-RCNN* con transferencia de aprendizaje

Finalizado el entrenamiento se ejecuta el script de test sobre el 20% de imágenes reservadas del *dataset* GBA. Los resultados obtenidos no son aceptables al ejecutar el detector sobre uno de los videos incluidos en el *dataset* GBA, tal como se puede observar en la Figura 55 y en la Tabla 11, con unos valores para cada clase de mAP:

Acción	Andar	Correr	Sentarse	Caerse	Parado
mAP@0.5	1.02 %	0.98%	0%	0%	0%

Tabla 11 Resultados del 1er entrenamiento realizado con el detector de personas

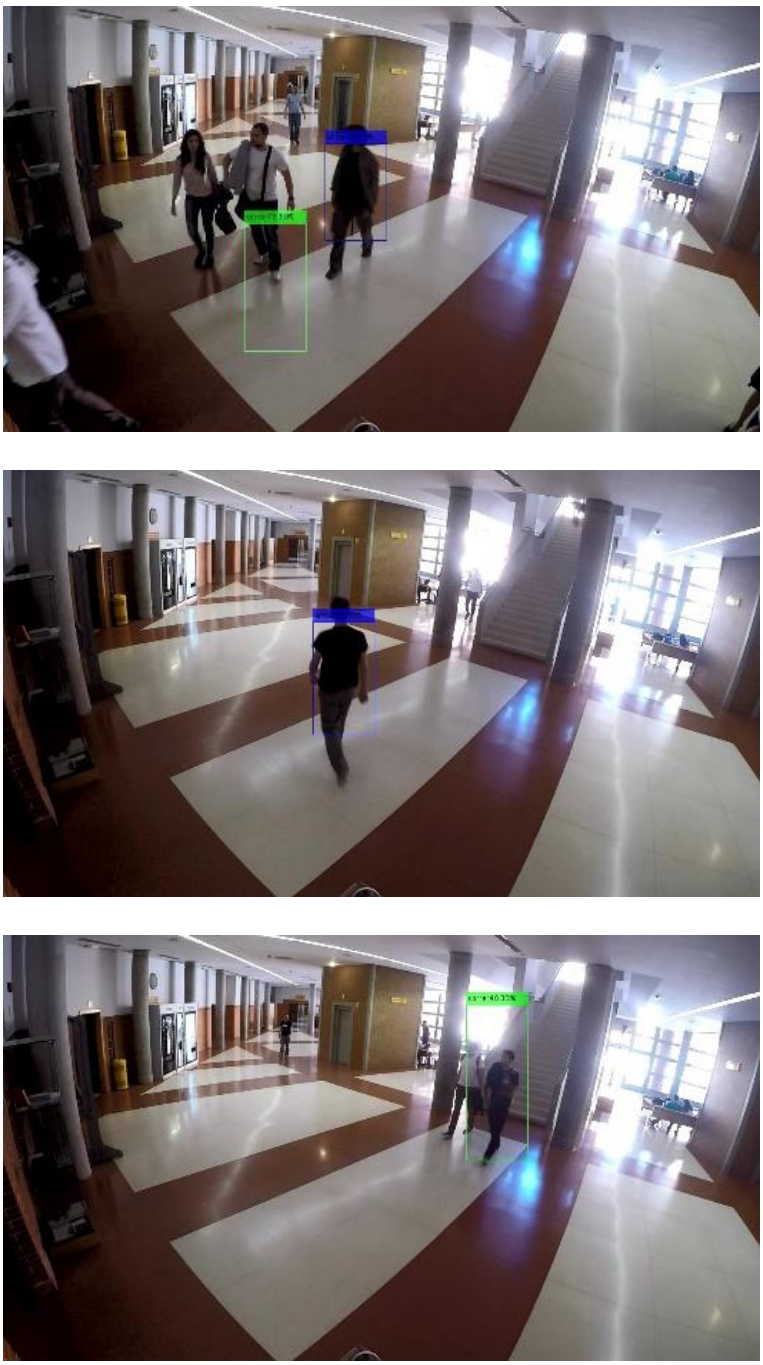


Figura 55 Resultado de ejecutar el detector de personas del 1er entrenamiento sobre un video de prueba

Los resultados obtenidos al realizar la evaluación anterior podrían ser mejorados si se incrementa la base de datos de imágenes, distribuyendo el número de datos entre el de acciones de una manera más equitativa, ya que la base de datos actual contiene un número de imágenes, asociados a una determinada acción, casi diez veces superior al número de imágenes que representan otras accciones, como en el caso de la acción andar respecto a las acciones caer o estar sentado. Otro factor que modificar es el número de *epoch*, mejora que se ha intentado pero que no se ha conseguido entrenar, ya que fallaba continuamente por errores de memoria de GPU.

A la vista de estos resultados, se procedió a unificar todas las acciones en una sola, para la detección, de tal modo que el detector tenga como finalidad única la detección de la persona, y se proceda posteriormente a identificar la acción realizada mediante otro tipo de red como son las redes recursivas tipo *LSTM* comentadas en el capítulo 3.

Se modifica nuevamente el archivo de *ground truth* con el formato adecuado para que Matlab lo reconozca en cada imagen, aunando en una única acción denominada "persona" las cinco acciones anteriores.

Para reducir el coste computacional y el tiempo de entrenamiento se reduce el número de imágenes de entrenamiento a 3918 que se corresponden con el 20% del *dataset* y se utilizaran 2000 imágenes para realizar el test.

Se modifican las opciones de entrenamiento de las cuatro etapas del detector disminuyendo de 10 a 5 *epoch*, con un *minibatch* de 8. Se modifica el parámetro de re-escalado de 400 a 360 pixeles para reducir aún más el coste computacional respecto al caso anterior y evitar problemas de memoria. El resto de los parámetros se mantienen con los mismos valores establecidos anteriormente.

La Tabla 12 muestra, de manera resumida, los nuevos parámetros de entrenamiento empleados en la prueba.

<table>
<tr><th colspan="5">Faster RCNN</th></tr>
<tr><th></th><th>Solver</th><th>MiniBatch Size</th><th>Max Epochs</th><th>Initial Learn Rate</th></tr>
<tr><td>Step 1</td><td>Sgdm</td><td>8</td><td>5</td><td>1e-5</td></tr>
<tr><td>Step 2</td><td>Sgdm</td><td>8</td><td>5</td><td>1e-5</td></tr>
<tr><td>Step 3</td><td>Sgdm</td><td>8</td><td>5</td><td>1e-6</td></tr>
<tr><td>Step 4</td><td>Sgdm</td><td>8</td><td>5</td><td>1e-6</td></tr>
<tr><td rowspan="6">Opciones Generales</td><td colspan="2">NegativeOverlapRange</td><td colspan="2">PositiveOverlapRange</td></tr>
<tr><td colspan="2">0 a 0.3</td><td colspan="2">0.6 a 1</td></tr>
<tr><td colspan="4">Strongest Regions</td></tr>
<tr><td colspan="4">2000</td></tr>
<tr><td colspan="2">Smallest Image Dimension</td><td colspan="2">Box Pyramid Scale</td></tr>
<tr><td colspan="2">360</td><td colspan="2">1.2</td></tr>
</table>

Tabla 12 Opciones del 2º entrenamiento de la red *Faster-RCNN*

Los resultados obtenidos mejoran considerablemente, tal como se puede observar en la Figura 56, resultado de ejecutar el detector sobre uno de los videos incluidos en el *dataset* GBA, con un valor de mAP 47.2%.

El valor de precisión obtenido es bastante mejor que anterior y su funcionamiento es bastante aceptable como se observa en la Figura 57, aunque se podría mejorar la precisión.

mAP@0.5 47.2%

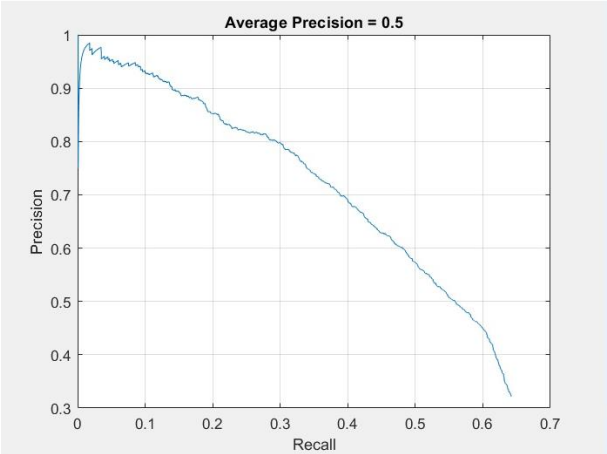


Figura 56 Resultado mAP del detector de personas del 2º entrenamiento

Figura 57 Resultado de ejecutar el detector de personas del 2º entrenamiento sobre un video de prueba

Se realiza un nuevo entrenamiento manteniendo los parámetros establecidos en el entrenamiento anterior, pero incrementando el número de *epoch* de 5 a 10.

Contrario a lo esperado la precisión obtenida es inferior a la del caso anterior, tal y como se puede observar en la Figura 58. Este hecho puede deberse al modo de trabajar de Matlab cuando reinicia un entrenamiento al haber fallado el anterior: si durante el entrenamiento surgen errores de desbordamiento de memoria o de acceso a la GPU que provocan que el entrenamiento se detenga bruscamente, Matlab guarda, cada cierto número de iteraciones (*CheckPoints*), los pesos y sesgos obtenidos hasta ese instante, de tal modo que cuando se reinicia el entrenamiento después de un fallo, se parte de unos valores de un entrenamiento parcial. Hay veces que estos valores pueden ser aceptables para una determinada etapa del detector, por lo que, aunque se reduzca el valor de '*InitalLearnRate*' para que no se modifiquen sustancialmente, estos varían ligeramente, lo que puede provocar, una vez finalizado el entrenamiento del detector, que este no tenga la precisión esperada. Esto también provoca la falta de repetibilidad del algoritmo al reentrenar con un mismo *dataset* y los mismos parámetros, lo que difícilta la comparación de resultados cuando se estudia el efecto de la modificación de los parámetros del modelo.

mAP@0.5 43.1%

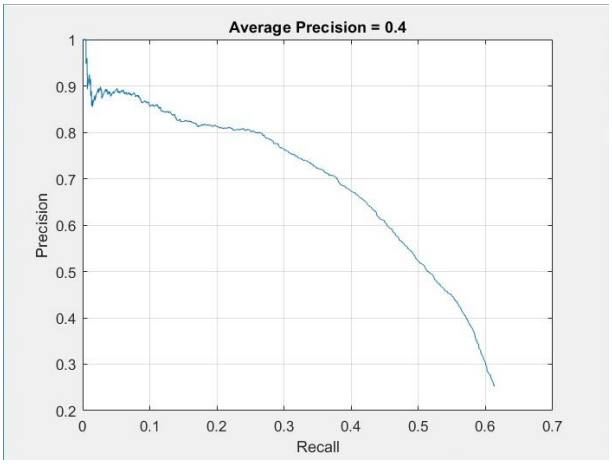


Figura 58 Resultado mAP del detector de personas del 3er entrenamiento

Se prueba a reentrenar de nuevo el detector, manteniendo el número de *epoch* a 5, pero incrementando el número de imágenes de entrenamiento al 80% del *dataset*.

Tal como se observa en la Figura 59, se obtiene un resultado de mAP de 48.1%, por lo que se mejora el resultado en un 0.9% respecto al mejor entrenamiento evaluado hasta el momento.

Llegados a este punto se concluye que independientemente del número de *epoch* del entrenamiento, con el tamaño actual del *dataset*, la mejora de la precisión no va a aumentar claramente. Incluso comparando el funcionamiento del detector sobre el mismo video mostrado anteriormente (Figura 57) el resultado es similar. Posiblemente incrementando el *dataset* y realizando un entrenamiento sin errores, es posible que la precisión se incremente.

mAP@0.5 48.1%

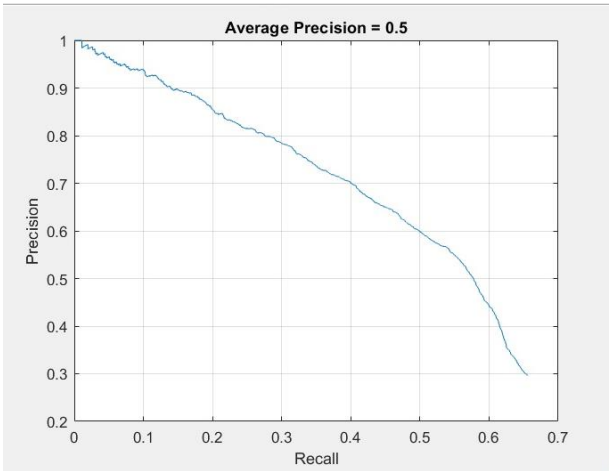


Figura 59 Resultado mAP del detector de personas del 4º entrenamiento

Aplicando los mismos conceptos aprendidos al entrenar el detector con el *dataset* GBA, se entrena otro detector con una base de datos diferente como es *Stanford40*, explicada en el apartado 4.1.1 de este trabajo, con objeto de comprobar que los parámetros utilizados permiten obtener un detector más preciso que el obtenido con el *dataset* GBA.

En este caso también es necesario adaptar el archivo de *ground truth* que proporciona el *dataset Stanford*40 para que Matlab lo interprete correctamente.

Se configuran las opciones de entrenamiento utilizando los mismos parámetros que se usados en el 1er entrenamiento (ver la Tabla 10). Se utiliza un número de imágenes de entrenamiento correspondiendo al 80% del *dataset*.

Los datos obtenidos no son aceptables (ver la Tabla 13), del mismo modo que ocurre con el *dataset* GBA cuando se intenta entrenar las cinco acciones que lo componen.

Acción	Jumping	Playing guitar	Riding a bike	Resto de acciones
mAP@0.5	1.2%	2.3%	1.1%	0%

Tabla 13 Resultados del 5º entrenamiento del detector de personas

En la Figura 60, se muestra el resultado de ejecutar este detector sobre uno de los videos del *dataset* GBA. Como allí se observa el resultado es poco eficiente.

Figura 60 Resultado de ejecutar el detector de personas del entrenamiento 5ª sobre un video de prueba

El resultado de aplicar este detector sobre la propia base de datos *Stanford40*, tampoco da buenos resultados, tal como se muestra en la Figura 61. En la mayor parte de los casos el detector no es capaz predecir correctamente la etiqueta, en este caso acción, pero sí que hay un alto acierto en recuadrar al objeto identificado, en este caso una persona.

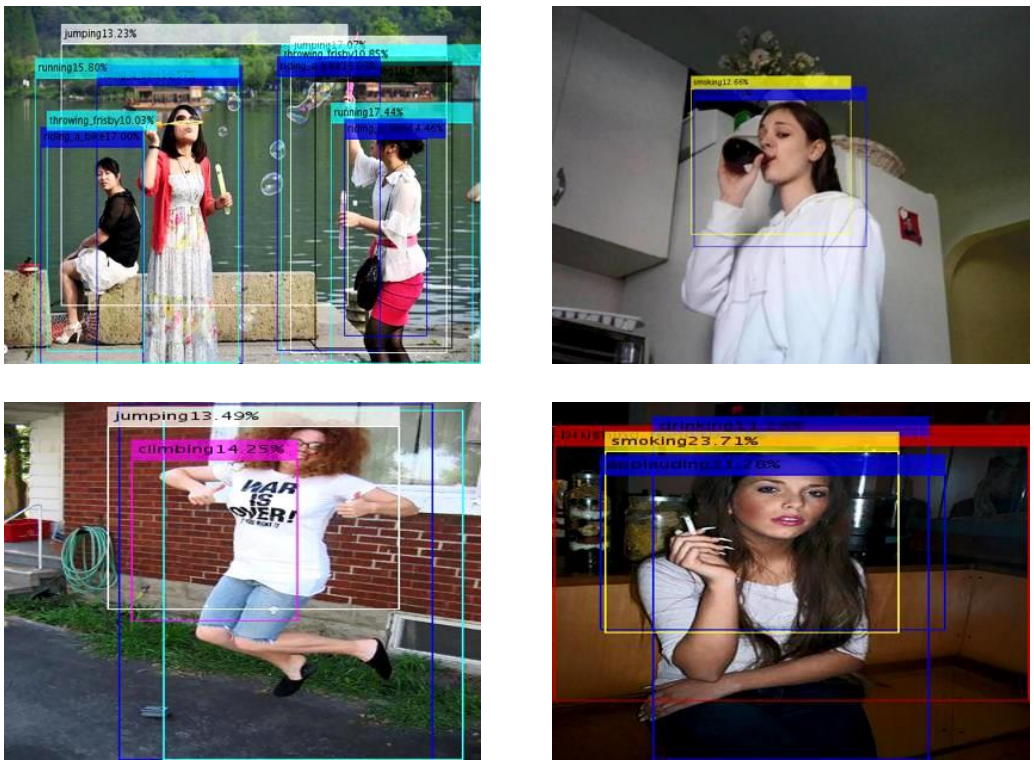


Figura 61 Resultado de ejecutar el detector de personas del 5º entrenamiento sobre imágenes de *Stanford40*

De nuevo, y como se observa que el detector de personas funciona, se aunan todas las acciones en una única, igual que se hizo para la base de datos GBA.

Para ello, se configuran las opciones de entrenamiento de las cuatro etapas del detector con 10 *epoch* y un *minibatch* de 5. Los valores de learn rate se establecen en 1e-4 para las etapas 1 y 2, incrementándose respecto a entrenamientos anteriores, y de 1e-6 para las etapas 3 y 4.

Se fijan como parámetros de *IoU* para '*NegativeOverlapRange*' unos valores de 0 a 0.3 y para '*PositiveOverlapRange*' unos de 0.6 a 1 y con un valor de '*Strongest Regions*' de 1000 que permita realizar el entrenamiento más rápido. De nuevo, las imágenes son re-escaladas en su lado menor a 400 pixeles y se mantiene el valor de '*BoxPyramidScale*' a 1.2.

La Tabla 14 muestra, de manera resumida, los parámetros de entrenamiento empleados. El número de imágenes de entrenamiento es de 3000 y de 1000 para el test.

En la Figura 62, se observa que al ejecutar el detector entrenado sobre el *dataset Stanford*40 el acierto es bastante aceptable dando un valor de mAP de 69.07%, pero cuando se ejecuta sobre el *dataset* GBA, los resultados empeoran bastante (mAP=0.5%).

Faster RCNN				
	Solver	MiniBatch Size	Max Epochs	Initial Learn Rate
Step 1	Sgdm	5	10	1e-4
Step 2	Sgdm	5	10	1e-4
Step 3	Sgdm	5	10	1e-6
Step 4	Sgdm	5	10	1e-6
Opciones Generales	NegativeOverlapRange		PositiveOverlapRange	
	0 a 0.3		0.6 a 1	
	Strongest Regions			
	1000			
	Smallest Image Dimension		Box Pyramid Scale	
	400		1.2	

Tabla 14 Opciones del 6º entrenamiento de la red *Faster-RCNN*

mAP@0.5 69.07%

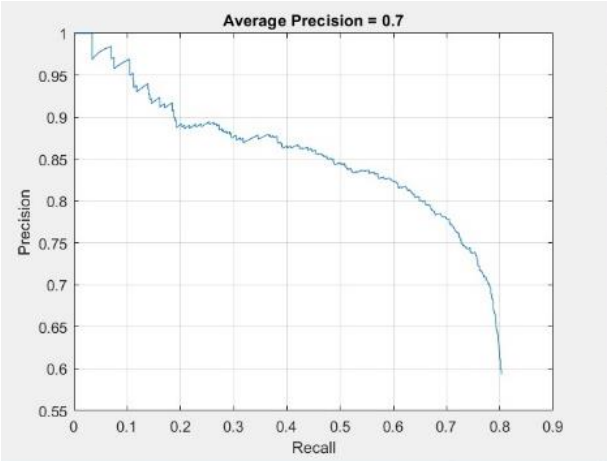
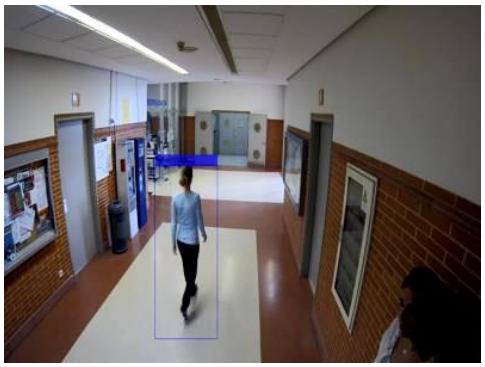


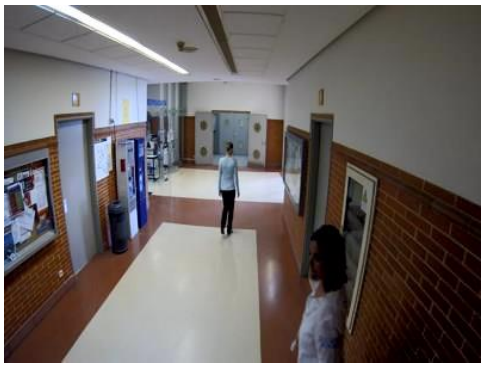
Figura 62 Resultado mAP del detector de personas del 6º entrenamiento

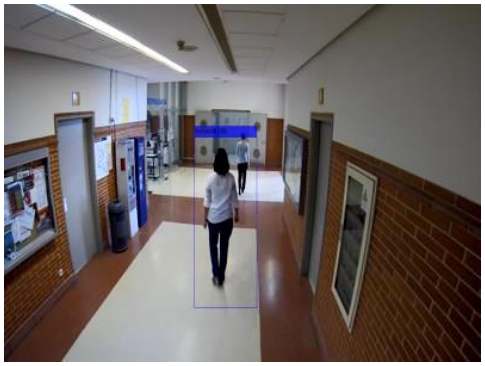
Ya que en evaluaciones anteriores se utilizó para test un 20% de las imágenes del *dataset*, y con objeto de comparar estos resultados con los obtenidos anteriormente, se ejecuta el detector sobre este 20% de imágenes del *dataset,* en lugar de las 1000 inicialmente propuestos. Se observa que el mAP disminuye del 69.07% al 65.42%. Aun disminuyendo el valor de mAP, se puede aceptar el detector como bueno.

Se observa en la Figura 63, que el detector entrenado con el *dataset Stanford40* es incapaz de detectar personas alejadas y por tanto de tamaño pequeño, solo consiguiendo detectarlas cuando están en primeros planos cercanos. Esto es debido principalmente a que el *dataset Stanford40* está compuesto de imágenes con personas en un plano muy cercano, ocupando el *bounding box* de detección un área muy grande en relación con el tamaño de la imagen (aprox. una relación superior a 1/10). En el *dataset* GBA, ocurre

exactamente lo contrario, ya que el área del *bounding box* de detección es bastante más pequeña en relación con el tamaño de la imagen (aprox. una relación superior a 1/180), lo que exige un detector de personas cuya posición está más alejada. Es decir, el entrenamiento con GBA permite una mayor sensibilidad en la detección, para detectar objetos más pequeños.

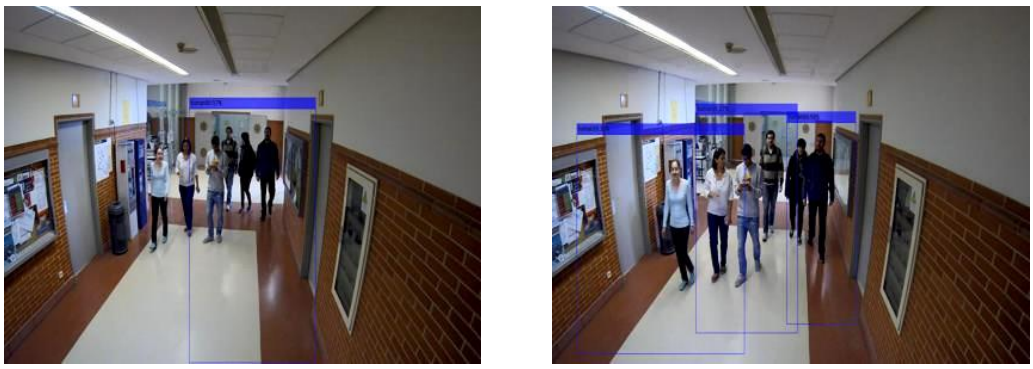






En los planos cercanos, la detección proporcionada por el detector es aceptable.

Cuando la figura a detectar se aleja, el detector es incapaz de realizar la detección



Otro problema observado, es que el *bounding box* es demasiado ancho en la detección, no se ajusta bien a la figura detectada.

Figura 63 Resultado de ejecutar el detector de personas del 6º entrenamiento sobre un video de prueba

A continuación, en la Figura 64 se muestra como el detector entrenado al ejecutarlo sobre el *dataset Stanford40* muestra resultados bastante aceptables, incluso cuando aparecen varias personas en escena.

Figura 64 Resultado de ejecutar el detector de personas del 6º entrenamiento sobre imágenes de *Stanford40*

Se repite a continuación el entrenamiento con la misma configuración, pero incrementando el número de imágenes de 3000 al 80% del total del *dataset* para realizar el entrenamiento y el 20% restante para realizar el test. Curiosamente el resultado es bastante peor que el obtenido con un número de imágenes de entrenamiento inferior, con un mAP de 47.97%. Este resultado puede ser debido a que, al aumentar el conjunto de imágenes de entrenamiento, también se requiere un numero de *epoch* mayor, que no se implementa por problemas de computación.

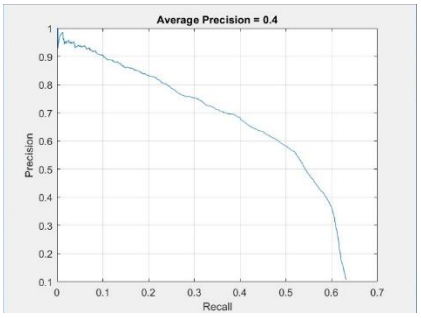
En otras pruebas realizadas, se modifica el *learn rate* de la etapa 1 y 2 de la red de 1e-4 a 1e-5, y se incrementa el número de '*StrongestRegions*' de 1000 a 2000, manteniendo el resto de los parámetros igual. Se observa una disminución de mAP a un valor de 58.55%. Finalmente, para comprobar hasta que punto afecta el valor de *learn rate, se* modifica el *learn rate* a 1e-3, manteniendo los valores del resto de parámetros. El detector no entrena correctamente dando un valor de mAP=0, por lo que se vuelve al valor optimo para *learn rate* de 1e-4.

Finalmente, y a la vista de que para planos cercanos el detector entrenado con el *dataset Stanford40* trabaja bien, se intenta reentrenar el detector así entrenado con el *dataset* GBA utilizando nuevamente una transferencia de aprendizaje, con los pesos y sesgos obtenidos del detector basado en el *dataset Stanford40* (utilizando el 6º entrenamiento, con precisión mAP de 69.07%, Figura 62), para conseguir que el detector aúne una buena detección tanto en planos cercanos como lejanos.

Se mantiene en este caso la misma configuración de los parámetros del 6º entrenamiento (Tabla 14) y se realiza el entrenamiento con 3918 imágenes y el test con 3000 de GBA, consiguiendo un mAP de 44.94% (ver Figura 65 7º entrenamiento).

Se realiza de nuevo otro entrenamiento, incrementando el número de imágenes de entrenamiento al 80% de las imágenes totales del *dataset* GBA y manteniendo el resto de los parámetros de configuración. El resultado es prácticamente el mismo, siendo el valor de mAP 45.34% (ver Figura 65, 8º entrenamiento).

mAP@0.5 44.94%



mAP@0.5 45.34%

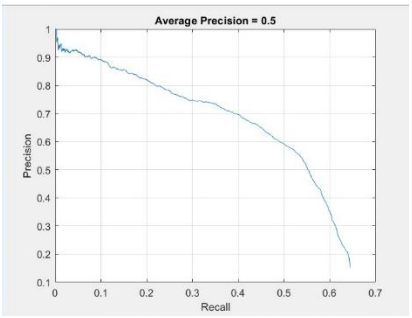
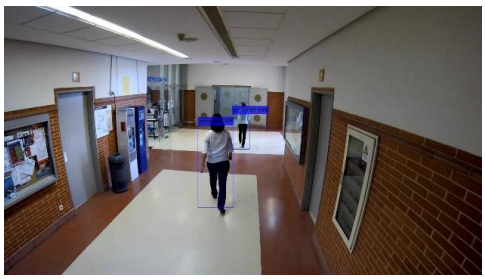


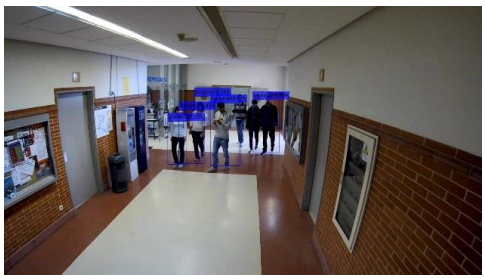
Figura 65 Resultado mAP del detector de personas del 7º y 8º entrenamiento

A continuación, se muestra una comparativa del funcionamiento del detector obtenido en el 8º entrenamiento utilizando el *dataset* GBA inicializado con los pesos del detector *Stanford40* y con un mAP de 45.34% (Figura 66), con el detector obtenido en el 4º entrenamiento utilizando el *dataset* GBA y que obtuvo un valor de mAP de 48.1% (Figura 67). Se pretende comprobar el comportamiento de los detectores en la detección en planos cercanos y lejanos.

1



2



3

4

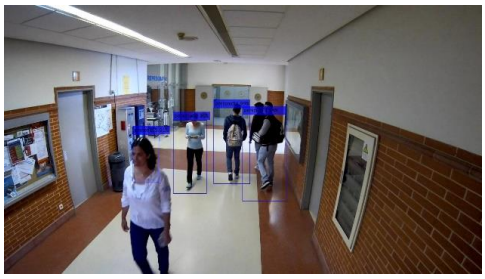
5

6

7

8

9



10

11

13

15

Figura 66 Imágenes utilizando detector GBA entrenado a partir de los pesos del detector Stanford40 mAP=45.34% (8º entrenamiento)

12

14

16

Figura 67 Imágenes utilizando detector GBA con un mAP=48.1% (4º entrenamiento)

Aunque la precisión del detector entrenado directamente a partir del *dataset* GBA es superior en un 3% (Figura 67), al ver resultados cualitativos se observa que el comportamiento de detección en figuras que aparecen en planos medios y lejanos (Imágenes 1 a 8) es similar, pero cuando la persona a detectar aparece en un plano cercano (Imágenes 9 a 16) funciona mejor el detector inicializado con los pesos del detector *Stanford40*. Esto es debido, a como se ha comentado anteriormente, a que la totalidad de imágenes de *Stanford40* son personas que aparecen en plano cercano a la cámara.

Este hecho evidencia lo importante que es en el proceso de aprendizaje de la red que los *dataset* utilizados para el entrenamiento contengan una variedad amplia de objetos o figuras desde diferentes ángulos y distancias respecto de la cámara.

Dados estos resultados se utilizará este detector entrenado a partir del *dataset Stanford40* y del *dataset* GBA en el sistema global de detección de anomalías que se pretende desarrollar en este trabajo.

Para finalizar este apartado dedicado al módulo de detección de personas, sería conveniente justificar a nivel experimental porque se ha descartado otros detectores como los basados en *ACF* descritos en el capítulo 3.

Para esta evaluación se utiliza un detector *ACF* entrenado con *INRIA person dataset* [37].

Se ejecuta este detector sobre un conjunto de 3000 imágenes correspondientes al *dataset* GBA dando como resultado un valor de mAP de 39.72%. Este valor es superior a lo que se obtuvo con el detector entrenado a partir del *dataset Stanford40* cuando se ejecutaba sobre el *dataset* GBA, en ese caso era mAP de 0.5%, pero inferior al obtenido entrenando el detector GBA, inicializado con el detector *Stanford40*, que obtiene un valor de mAP de 45.34%.

En la Figura 68, se muestra un ejemplo al ejecutar el detector *ACF* sobre un video correspondiente al *dataset* GBA.

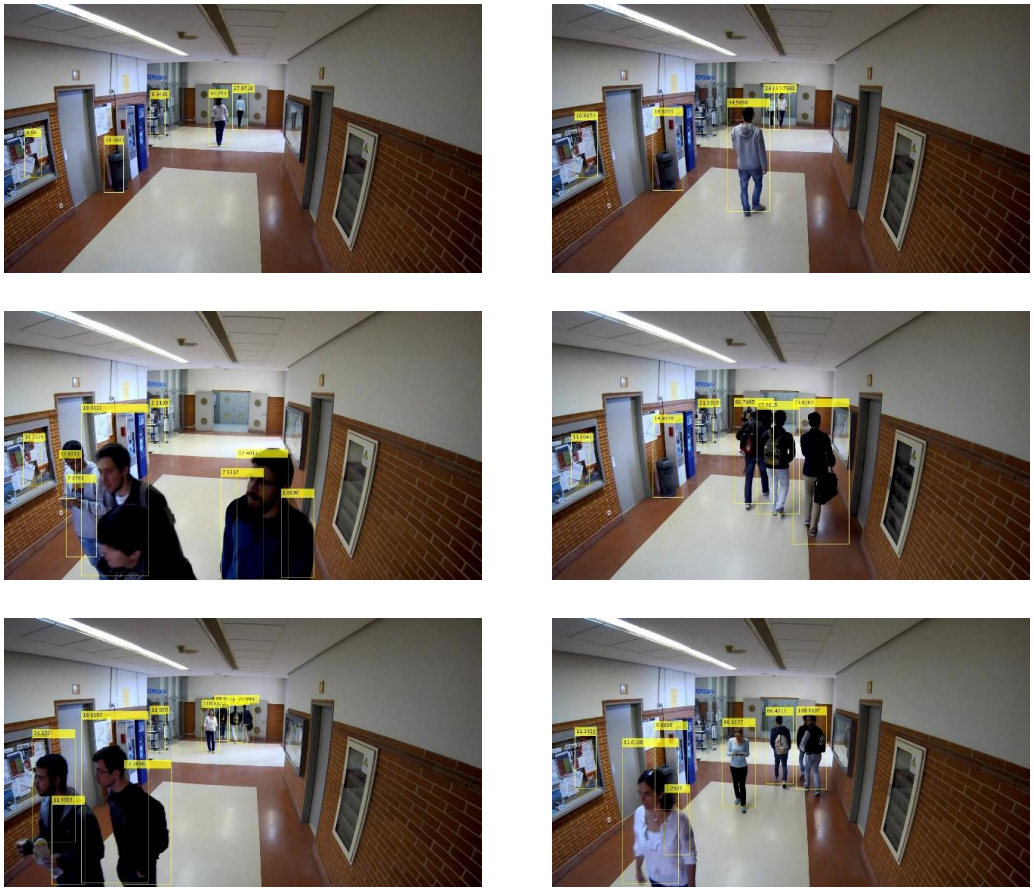


Figura 68 Resultado de ejecutar detector de personas basado en ACF sobre un video de prueba de GBA

6.2 *Resultados experimentales y evaluación del predictor de acción*

Se incluyen en este apartado las 2 evaluaciones realizadas del algoritmo de reconocimietno de acción.

6.2.1 Evaluación del modelo basado en KTH

Las secuencias de video utilizadas para la evaluación de la arquitectura del modelo planteado para el predictor de acciones a evaluar están extraídas del *dataset KTH*.

Se ha procedido a dividir los videos del *dataset KTH*, los cuales están grabados a 25fps, en secuencias de 1 segundo, es decir, cada secuencia contendrá 25 *frames*. Se utiliza para extraer los vectores de características de cada *frame*, la capa '*fc7*' de *Alexnet*.

Se adapta el conjunto de datos obtenido para que el array de datos de entrenamiento contenga el mismo número de secuencias por acción, siendo este valor de 263.

Matlab no permite asignar un conjunto de datos de validación durante el entrenamiento de redes que contengan capas de tipo *LSTM*, por lo que, únicamente se reservará del conjunto de datos de entrenamiento un 10% para realizar el test una vez finalizado el entrenamiento.

Dado que la duración de las secuencias es la misma para todo el *dataset*, los posibles problemas del relleno de secuencias o *padding* no se van a dar. En la siguiente figura, se muestra la distribución ordenada de las secuencias utilizadas en el entrenamiento, correspondiente al *dataset* utilizado, siendo el mismo tamaño fijo para todas las secuencias.

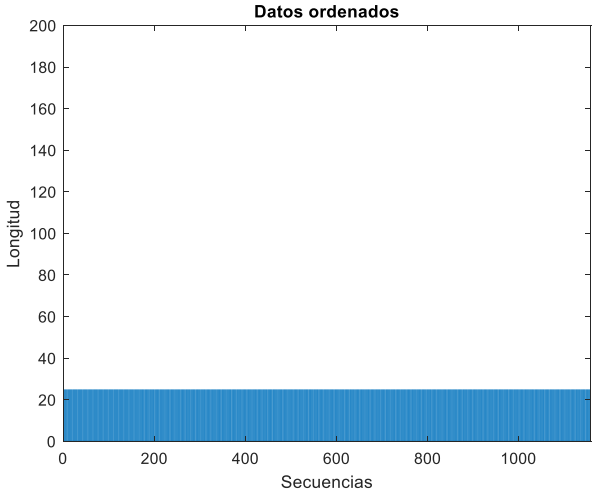


Figura 69 Distribución de las secuencias utilizadas del *dataset* KTH

Durante el entrenamiento, de forma predeterminada, Matlab puede dividir los datos de entrenamiento en *minibatchs* facilitando el entrenamiento del modelo. Dado que los datos utilizados para el entrenamiento se corresponden con el 90% de los datos totales, se puede determinar que un tamaño óptimo de *minibatch* podría ser de 16, valor seleccionado por ser múltiplo del total de secuencias utilizadas para el entrenamiento, de tal modo que los datos de entrenamiento quedan divididos de manera uniforme.

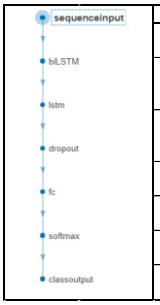
Se indica el tipo de optimizador, en el caso utilizado en este trabajo es 'sdgm'. El umbral de gradiente o 'GradientThreshold' se fija a 6. Cabe destacar que, durante los entrenamientos realizados, el hecho de tener un valor bajo de gradiente ha provocado fallos durante el entrenamiento que se evitan si este parámetro se fija a *'Inf'*. Cierto es que

indicar valores bajos evitan que el descenso no oscile tan bruscamente, realmente este parámetro se comporta como un filtro, mejorando sustancialmente los resultados.

Se establece un número de *epoch* igual a 600 y se configura para que el *learn rate* vaya disminuyendo en un factor de 0.1 cada 150 *epoch*, de este modo se consigue que el entrenamiento comience modificando los pesos de una manera más rápida al principio de este, para finalmente, estos varíen de una manera más lenta al final, consiguiendo un entrenamiento óptimo.

Se establece un valor de *dropout* de 0.9 y se modifica el parámetro de formato de salida de la capa bidireccional *LSTM* de 'last' a '*sequence'*, este cambio se realiza para adaptar el formato de salida al formato de entrada de la capa *LSTM* que se añade en cascada.

En la Tabla 15, se muestra de forma resumida la arquitectura del modelo utilizado en la evaluación descrita.



Nº LAYER	NAME	TYPE	ACTIVATIONS	LEARNABLES	STATES
1	sequenceinput Sequence input with 4096 dimension	Sequence Input	4096		
2	BIlstm bllstm with 256 hidden units	BiLSTM	512	InputWeights 2048x4096 RecurrentWeights 2048x256 Bias 2048x1	HiddenState 512x1 CellState 512x1
3	lstm LSTM with 256 hidden units	LSTM	256	InputWeights 1024x512 RecurrentWeights 1024x256 Bias 1024x1	HiddenState 256x1 CellState 256x1
4	dropout 90% dropout	Dropout	256		
5	fc 5 fully connected layer	Fully Connected	5	Weights 5x256 Bias 5x1	
6	softmax softmax	Softmax	5		
7	classoutput crossentropyex with 'boxing' and 4 other classes	Classification Output			

Tabla 15 Arquitectura del modelo *LSTM* basado en el *dataset* KTH

En la Tabla 16 se muestran de forma resumida los parámetros de entrenamiento mencionados en los párrafos antriores:

Parámetro	valor
Solver	Sgdm
Max Epochs	600
MiniBatch Size	16
Initial Learn Rate	0.1
shuffle	never
L2Regularization	0.0005
Momentum	0.95
GradientThresholdMethod	Global-l2norm
GradientThreshold	6
LearnRateSchedule	piecewise
LearnRateDropFactor	0.1
LearnRateDropPeriod	150

Tabla 16 Parámetros de entrenamiento del modelo *LSTM* basado en el *dataset* KTH

La precisión obtenida, evaluando la red entrenada obtenida con el 10% de las secuencias reservadas para tal fin, es de un 99.37%. En la Figura 70, se muestra el gráfico de evolución del entrenamiento realizado.

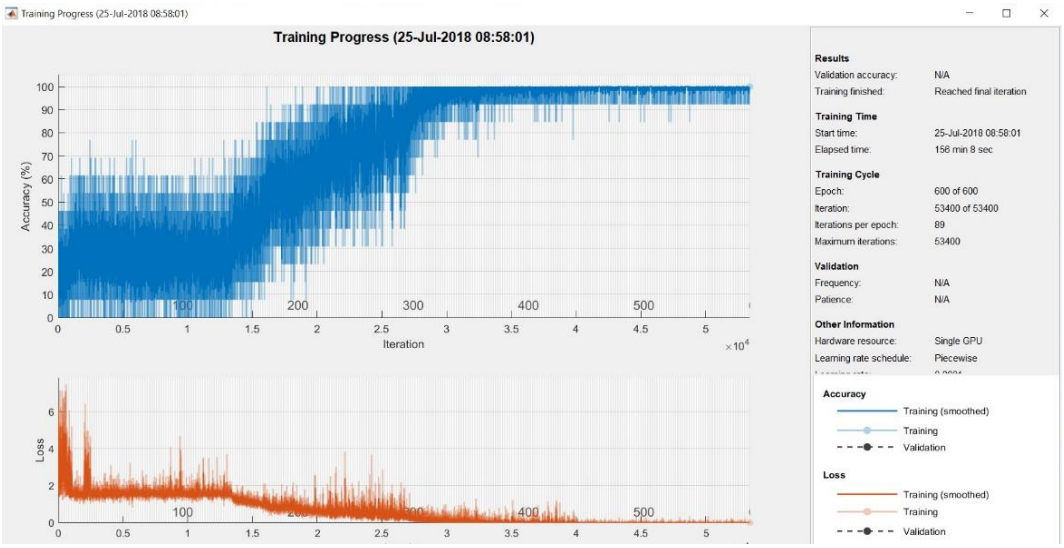


Figura 70 Evolución del entrenamiento del modelo basado en el *dataset* KTH con resultado 99.37%

6.2.2 Evaluación del modelo basado en Weizmann

A nivel comparativo, se realizan varias pruebas entrenado el modelo anteriormente descrito utilizando como secuencias de entrenamiento el *dataset Weizmann*. Se generan los datos de entrenamiento utilizando también los vectores de características extraídos con la capa '*fc7*' de la red *Alexnet*.

Este array contiene las siguientes acciones y números de secuencias.

- *Bend*: 9 secuencias
- *GalopSideways*: 9 secuencias
- *Jump*: 9 secuencias
- *JumpingJack*: 9 secuencias
- *JumpInPlace*: 9 secuencias
- *OneHandWave*: 9 secuencias
- *Running*: 10 secuencias
- *TwoHandsWave*: 9 secuencias
- *Walk*: 10 secuencias

En total 93 secuencias, de las cuales se utilizan para entrenamiento 82 y 11 para test. En la Figura 71 se muestra la distribución de secuencias en función de su longitud (número de *frames*) que van de los 42 *frames* a los 146.

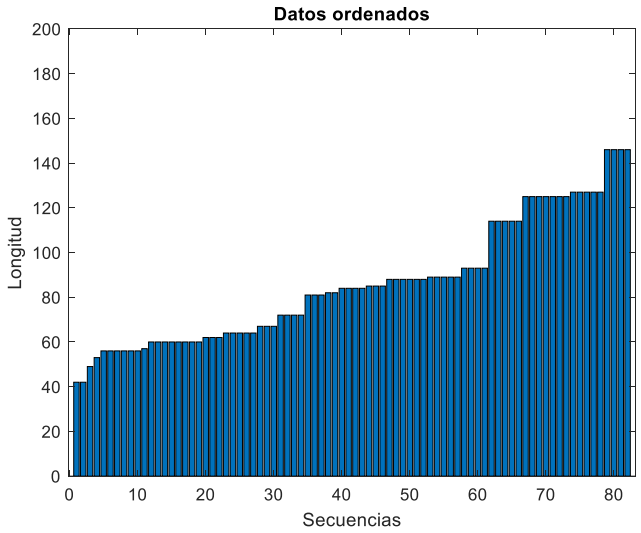
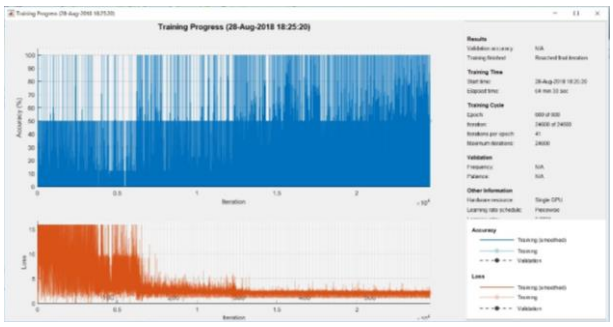


Figura 71 Distribución de las secuencias utilizadas del *dataset Weizzman*

La arquitectura de la red es idéntica a la utilizada en el proceso de entrenamiento del *dataset* KTH (Tabla 15) que se ha realizado anteriormente y que tan buenos resultados a demostrado. En la primera prueba se utilizan exactamente los mismos parámetros de entrenamiento (Tabla 16), siendo el resultado obtenido de un 0% de precisión. En un segundo intento, se incrementa el número de *epoch* de 600 a 2000, reduciendo el factor *learn rate* en un factor de 0.1 cada 750 *epoch*, Se modifica el parámetro '*GradientThreshold*' de 2 a '*Inf*' para evitar errores durante el entrenamiento, al tener este una duración mayor que el anterior. Y finalmente, se reduce el parámetro *dropout* de 0.9 a 0.7, ya que el *dataset* de entrada, al contener muy pocas secuencias, no se quiere forzar tan drásticamente el proceso de eliminación aleatoria de neuronas. Igualmente, el resultado obtenido es 0% de precisión. Finalmente, se realiza otra prueba modificando el número de *epoch* a 1000. Se disminuye el factor *learn tate* de 0.1 a 0.01, reduciendo el mismo cada 300 *epoch* en un factor de 0.1, de este modo se reduce la velocidad de aprendizaje para que en teoría la precisión aumente paulatinamente. Igual que en los dos casos anteriores, la precisión obtenida es de 0% (ver Figura 72).

Como era de esperar este *dataset* que contiene diez acciones, con muy pocas secuencias por acción, no es el más indicado para utilizarlo en el entrenamiento de la red *LSTM*.



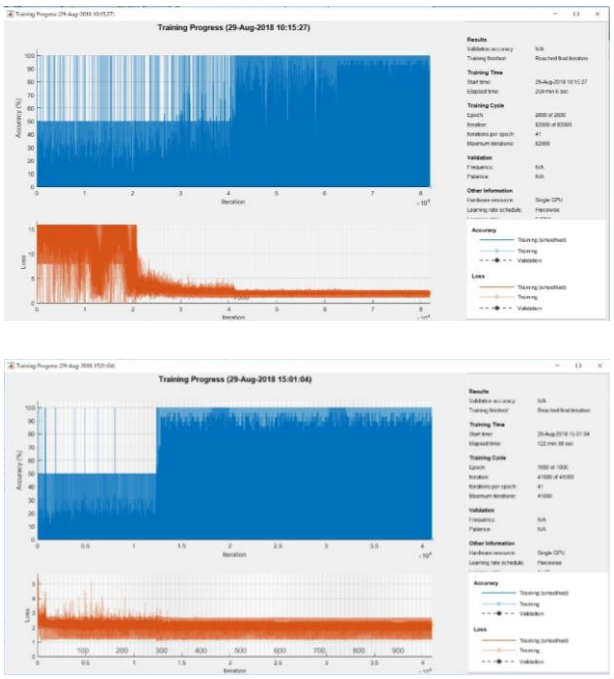


Figura 72 Diferentes evoluciones de los entrenamientos realizados con el *dataset Weizmann*

6.2.3 Evaluación del modelo basado en GBA

Del mismo modo que se ha realizado una evaluación del modelo utilizando los *dataset* KTH y *Weizmann*, se intenta entrenar el modelo utilizando como base de secuencias de entrenamiento los videos recortados correspondientes al *dataset* GBA. Como ya se ha comentado anteriormente (apartado 4.1.4), este *dataset* consta de secuencias grabadas a 50fps de duración entre 1 y 3 segundos y recortadas a un tamaño de 320x180 pixeles. Para incrementar el número de videos se ha utilizado la opción *flip* que genera una imagen especular de cada imagen, tal como se ha descrito en el apartado 4.3.

En total la cantidad de secuencias por acción utilizada son 56 *falling*, 104 *running*, 71 *sitting down*, 132 *standing* up, 178 *walking*, de las cuales el 90% se utilizarán para realizar el entrenamiento y el resto para realizar el test.

Durante el entrenamiento, de forma predeterminada, el software divide los datos de entrenamiento en *minibatchs* y realiza un *padding que re*llena las secuencias para que tengan la misma longitud. Aplicar un *padding* elevado, es decir, rellenar en exceso puede tener un impacto negativo en el rendimiento de la red. Para evitar que el proceso de entrenamiento agregue un *padding elevado*, es conveniente ordenar los datos de entrenamiento en función de la duración o cantidad de *frames* de la misma y elegir un tamaño de *minibatch* óptimo para que las secuencias tengan una longitud similar en ese determinado lote de secuencias.

En la siguiente figura, se muestra la distribución ordenada de las secuencias utilizadas en el entrenamiento, correspondiente al *dataset* utilizado.

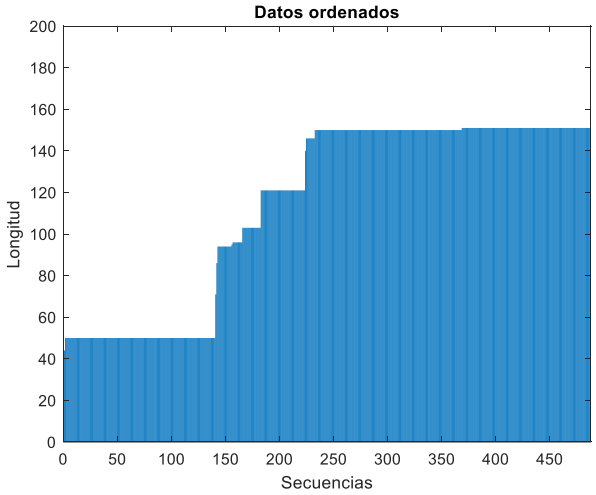


Figura 73 Distribución de las secuencias ordenadas del *dataset* GBA

Observando la figura anterior se puede determinar que un tamaño óptimo de *minibatch* podría ser 17 para dividir los datos de entrenamiento de manera uniforme y reducir el efecto provocado por el *padding* en los *minibatchs*.

Se especifica que la duración de la secuencia sea la más larga o '*longest'*, de tal manera que los datos tengan la misma longitud que las secuencias de mayor duración. También se debe indicar que los datos permanezcan ordenados por longitud de secuencia, por lo que especificará que nunca se mezclen los datos ('*never shuffle'*).

Se mantiene el optimizador, '*sdgm*', y el umbral de gradiente o '*GradientThreshold'* se fija a '*Inf*', para evitar que valores inferiores provoquen fallos de entrenamiento.

El número máximo de *epoch* se fija a 1000 y el *learn rate* utilizado es 0.1. Y se configura el *learn rate* para que disminuya progresivamente en un factor de 0.1 cada 300 *epoch*.

En resumen, la arquitectura del modelo empleado será la misma que la utilizada para la evaluación con el *dataset* KTH (Tabla 15). Los parámetros de entrenamiento utilizados en este caso son los mostrados en la tabla siguiente.

Parámetro	valor
Solver	Sgdm
Max Epochs	1000
MiniBatch Size	17
Initial Learn Rate	0.1
shuffle	never
L2Regularization	0.0005
Momentum	0.95
GradientThresholdMethod	Global-l2norm
GradientThreshold	Inf
LearnRateSchedule	piecewise
LearnRateDropFactor	0.1

LearnRateDropPeriod	300

Tabla 17 Parámetros de entrenamiento del modelo *LSTM* basado en el *dataset* GBA

El entrenamiento realizado, aunque no ha presentado ningún error, no muestra buenos resultados siendo el resultado de la precisión obtenida de un 32.3%.

Se realiza un nuevo entrenamiento, manteniendo la misma arquitectura, pero modificando el número de *epoch*, que se incrementa a 3000. El parámetro *learn rate* se reduce cada 1000 *epoch* en un factor 0.1 y se reduce el umbral de gradiente de '*Inf*' a 6.

En este caso se ha obtenido una precisión del 41.5%. Como era de esperar el incremento de *epoch* ha mejorado la precisión del predictor, pero aún sigue siendo insuficiente.

Se mantiene la misma arquitectura y se reduce el valor de '*dropout*' de 0.9 a 0.2 para comprobar si el entrenamiento tiene un comportamiento peor, tal como se indica en diversos trabajos realizados sobre el reconocimiento de actividades [8]. La precisión obtenida con este modelo es del 21.54%, los resultados han empeorado bastante respecto a los casos evaluados antes, lo que verifica que es mejor mantener un *dropout* elevado.

Como se observa que se ha obtenido un resultado bastante bueno en el entrenamiento del modelo basado en el *dataset* KTH (Figura 70), se vuelve a realizar un nuevo entrenamiento utilizando los mismos parámetros de entrenamiento (Tabla 16). En esta ocasión, se obtiene un valor de precisión de un 33.85%.

En la siguiente evaluación se va a realizar una transferencia de aprendizaje (apartado 5.1.1), inicializando la red con los pesos obtenidos del modelo de red entrenada con el *dataset* KTH que proporcionó una precisión del 99,37% (Figura 70). Como base de datos de secuencias de entrenamiento se utilizará el *dataset* GBA que se ha estado utilizando antes.

De igual modo que ya se realizó para el entrenamiento del detector de personas, se sustituyen las tres últimas capas del modelo pre-entrenado con KTH encargadas de la clasificación, por las nuevas capas de clasificación adaptadas a las acciones del *dataset* GBA.

Se ajustan los factores '*WeightLearnRateFactor*' y '*BiasLearnRateFactor*', de la capa completamente conectada, a un valor de 100, de este modo, al fijar un valor bajo de *learn rate* global de 0.001 se mantienen los pesos de las primeras capas de la red pre-entrenada invariantes, pero los encargados de la clasificación se reentrenan completamente para adaptarse a las etiquetas de las correspondientes acciones contenidas en el *dataset* GBA.

La precisión obtenida mejora, pero sigue mostrando un valor aún bajo de 41.54% (tal y como se muestra en la Figura 74).

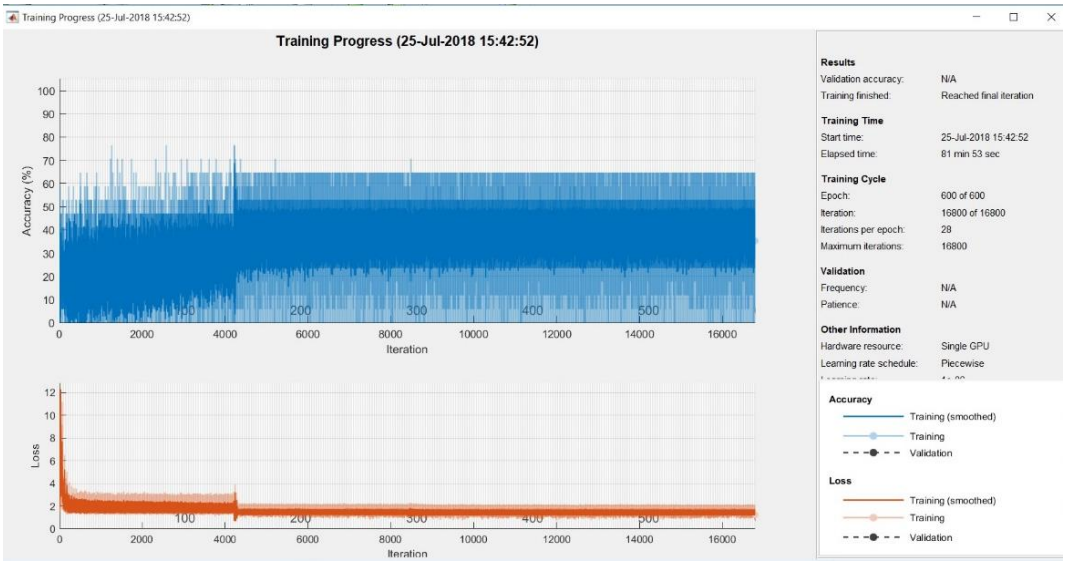


Figura 74 Evolución del entrenamiento de la red *LRCN* con GBA y transferencia de aprendizaje de KTH (precisión 41.54%)

Se repite de nuevo el entrenamiento anterior reduciendo los factores '*WeightLearnRateFactor*' y '*BiasLearnRateFactor*', de la capa completamente conectada de 100 a 10 y se decrementa aún más el *learn rate* global a un valor de 0.0001.

La precisión obtenida ha disminuido a un valor de 27.69%. Se ha forzado demasiado a que los pesos no se modifiquen lo suficiente para ajustarse correctamente a las secuencias del *dataset* GBA. En otras pruebas realizadas, manteniendo el mismo *learn rate* global, pero modificando los factores '*WeightLearnRateFactor*' y '*BiasLearnRateFactor*' a 100, el resultado es exactamente igual.

Se repite nuevamente la evaluación utilizando la arquitectura (Tabla 15) y los parámetros de entrenamiento del modelo utilizado para entrenar el *dataset* KTH (Tabla 16) pero la base de datos utilizada esta vez se basa en el *dataset* GBA, pero esta vez modificada respecto a la utilizada en evaluaciones anteriores. Se realiza una modificación de los videos o *data augmentation*, variando el brillo, el contraste, y a la vez, aplicando volteo horizontal o *flip*. Con estas modificaciones de las secuencias de video se consigue incrementar el número de videos para realizar el entrenamiento. Se obtiene una base de datos ecualizada de 451 secuencias, en este caso de 25 *frames* fijos por cada secuencia.

En esta ocasión, se detiene el entrenamiento a la mitad, ya que se observa que la función de pérdidas durante el entrenamiento no va a converger a cero.

Se realizan numerosos entrenamientos más, modificando ligeramente diferentes parámetros sin obtener mejorías significativas. Igualmente se realizan diversas pruebas utilizando otras capas de la red *Alexnet* para realizar la extracción de descriptores, como son la capa '*fc6*' y '*fc8*' obteniendo resultados similares que con la capa '*fc7*'.

En vista de que no se consigue una precisión aceptable, se utiliza para la extracción de características de las secuencias de video del *dataset* GBA, la capa '*fc1000*' de *Resnet50* (apartado 5.2.1) manteniendo la misma distribución de secuencias de la Figura 73.

La arquitectura del modelo es similar a la utilizada en la Tabla 15, excepto que el parámetro de entrada de la capa '*sequenceInputLayer*' se fija a un valor de 1000.

Se configura un *minibatch* de tamaño 16, con número de *epoch* de 600. El *learn rate* se establece en 0.1, el cual va disminuyendo cada 150 *epoch* por un factor de 0.1. se disminuye aún más el umbral de gradiente a un valor de 2, permaneciendo el resto de los parámetros igual que en los casos anteriores.

En la Tabla 18, se muestra un resumen de los parámetros de entrenamiento utilizados.

Parámetro	valor
Solver	Sgdm
Max Epochs	600
MiniBatch Size	16
Initial Learn Rate	0.1
shuffle	never
L2Regularization	0.0005
Momentum	0.95
GradientThresholdMethod	Global-l2norm
GradientThreshold	2
LearnRateSchedule	piecewise
LearnRateDropFactor	0.1
LearnRateDropPeriod	150

Tabla 18 Parámetros del 1er entrenamiento del modelo *LSTM* basado en el *dataset* GBA (*Resnet50*)

La precisión obtenida es de 79.63% (Figura 75), siendo una mejora sustancial utilizar *Resnet50* como herramienta para obtener los descriptores de características, ya que incrementa la precisión en casi el doble respecto a *Alexnet*, pero el tiempo de ejecución aumenta considerablemente, ya que la extracción de características utilizando la capa '*fc1000*', consume un tiempo entre 160ms y 510ms, dependiendo del tamaño de la imagen, mientras que la capa '*fc7*' del modelo *Alexnet*, consume un tiempo entre 7ms y 12ms para el mismo tamaño.

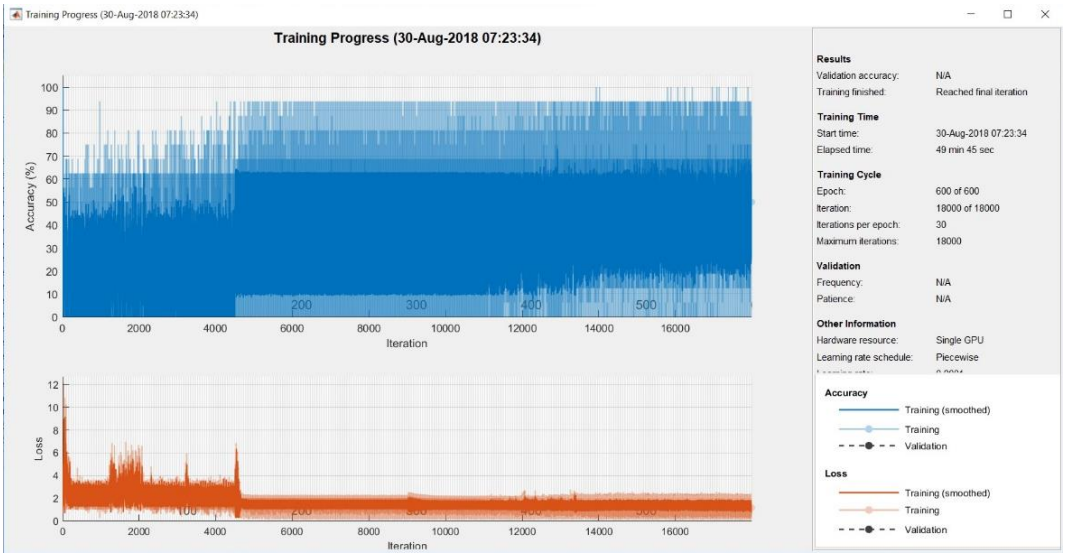


Figura 75 Evolución del 1er entrenamiento del modelo *LRCN* con *Resnet50* (precisión 79.63%)

Se modifica la arquitectura anterior eliminando la capa bidireccional *LSTM*, con el objetivo de comprobar la mejoría que produce el hecho de utilizar este tipo de capa recursiva bidireccional.

La precisión obtenida disminuye respecto al entrenamiento anterior, dando un valor de 62.96%. Utilizar una capa recursiva bidireccional mejora la precisión, por este motivo se vuelve a insertar en la arquitectura de la red, tal como se estaba utilizando en entrenamientos anteriores.

La siguiente evaluación realizada se basa principalmente en modificar el tamaño del *minibatch.* En este caso se va a incrementar el número de *epoch* a 5000, utilizando un *minibatch* de tamaño 487 que es igual al total de secuencias de entrenamiento utilizadas, es decir, en cada iteración se entrena todo el conjunto de datos.

El '*learn rate'* se reduce de 0.1 a 0.01, y se reduce en un factor de 0.1 cada 4000 *epoch*. El resto de los parámetros y el *dataset* permanece como en el último entrenamiento realizado.

En la Tabla 19, se muestra un resumen de los parámetros de entrenamiento utilizados.

Parámetro	valor
Solver	Sgdm
Max Epochs	5000
MiniBatch Size	487
Initial Learn Rate	0.01
shuffle	never
L2Regularization	0.0005
Momentum	0.95
GradientThresholdMethod	Global-l2norm
GradientThreshold	2

LearnRateSchedule	piecewise
LearnRateDropFactor	0.1
LearnRateDropPeriod	4000

Tabla 19 Parámetros del 2º entrenamiento del modelo *LRCN* basado en el *dataset* GBA (*Resnet50*)

La precisión obtenida da un resultado del 100% (ver Figura 76).

Aunque el resultado obtenido muestra una precisión elevada, realmente el modelo únicamente predice correctamente las acciones de las secuencias utilizadas durante el entrenamiento. Al utilizar una secuencia diferente a la del *dataset* de entrenamiento, la red no es capaz de predecir correctamente. Esto ocurre porque el *dataset* de entrenamiento esta formado por secuencias de diferente longitud (Figura 73), y se ha configurado el tamaño del *minibatch* a un valor igual a la suma de todas las secuencias de entrenamiento, en este caso 487. Matlab durante el entrenamiento de la red, aplica un *padding* sobre las secuencias de entrada, para que éstas tengan la misma longitud, siendo este valor igual a la longitud de la secuencia más larga. Este *padding* provoca que se complete las secuencias del *dataset* de entrenamiento con un número excesivo de datos de valor 0. Esto facilita el entrenamiento, pero a su vez, provoca que el modelo entrenado se adapte perfectamente al *dataset* de imágenes utilizado durante el entrenamiento, siendo incapaz de predecir correctamente cuando se le muestra una secuencia diferente. Se produce, por tanto en este caso, un efecto claro de *overfitting*.

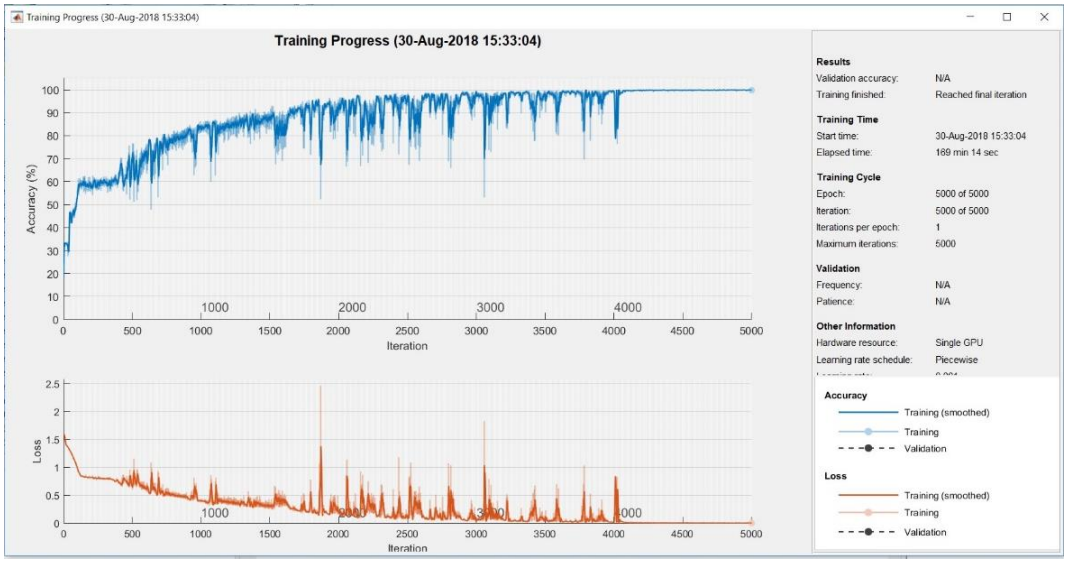


Figura 76 Evolución del 2º entrenamiento del modelo *LRCN* con *Resnet50* (precisión 100%)

Hay que indicar que se realizaron una nueva serie de pruebas repitiendo las arquitecturas y parámetros de configuración descritos anteriormente, pero modificando el *dataset* GBA incorporando nuevas secuencias de video grabadas a 50fps de duración entre 1 y 3 segundos y recortadas a un tamaño de 540x300 pixeles. Siendo el total de secuencias por acción: 33 *falling*, 95 *running*, 45 *sitting down*, 32 *getting up*, 780 *walking*. Los resultados obtenidos no mejoran los ya logrados anteriormente.

En resumen, se han obtenido dos modelos entrenados basados en el *dataset* GBA, utilizando *Alexnet* y *Resnet* para obtener los vectores de características, dando como resultados unos valores de precisión de 41.54% con *Alexnet* (Figura 74) y de 79.63% con *Resnet50* (Figura 75).

6.3 *Evaluación conjunta de los modelos de detección de personas y predicción de acción*

Para realizar esta evaluación conjunta se ha implementado un script en Matlab (apartado 8.4) que conjuga el detector de personas basado en *Faster-RCNN* (apartado 5) con el predictor de acciones basado en el modelo *LRCN* (apartado 5.2.2). En la Figura 77, se muestra un esquema de funcionamiento del sistema conjunto.

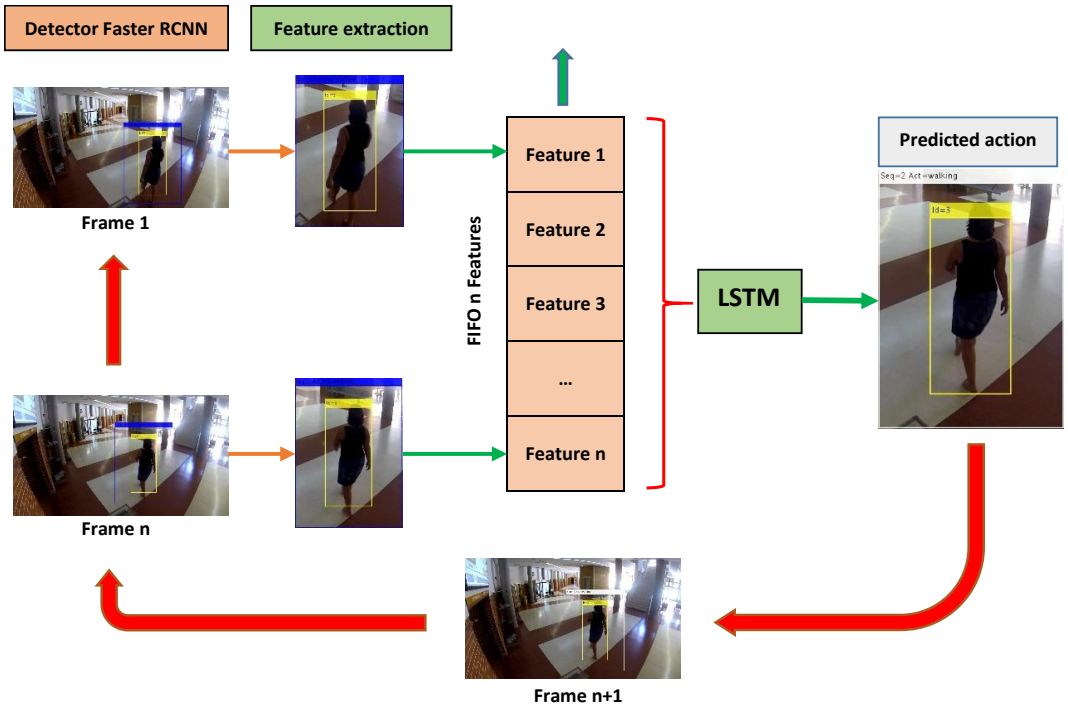


Figura 77 Esquema de funcionamiento del sistema conjunto de detección y predicción

Como primer paso, se ejecuta el detector de personas con el objetivo de localizar a la persona que aparece en la secuencia de video. En la Figura 78, se muestra como el detector basado en *Faster-RCNN* identifica la persona. El detector proporciona la información de las coordenadas del recuadro o *bounding box* que contienen la imagen de la persona detectada (recuadro amarillo). Para garantizar que el recuadro cubre toda el área de la persona detectada se realiza una ampliación del tamaño de la misma (recuadro azul), Con la información contenida en esta ventana se obtiene el vector de características, mediante las capas de activación de A*lexnet* o *Resnet50,* tal como se puede observar en la Figura 77.

La información de las coordenadas del recuadro de detección es además utilizada con otra finalidad: realizar el seguimiento o tracking de la persona a lo largo de la secuencia de video. Para ello, se utiliza un algoritmo de seguimiento o *tracking* basado en un filtro de tipo Kalman [38].

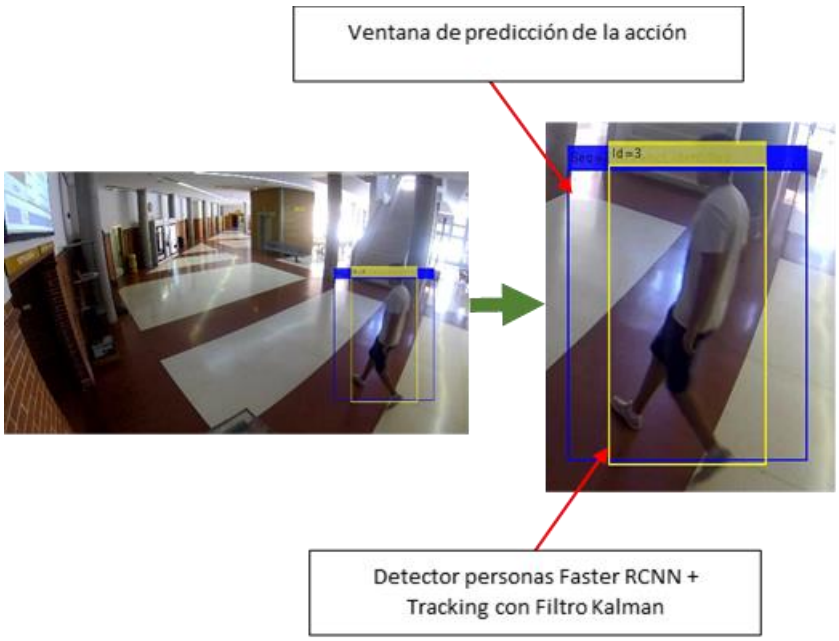


Figura 78 Detección de la persona y obtención de *bounding box*

Los vectores de características, obtenidos de las sucesivos *frames* analizados a lo largo de la secuencia de video, son almacenados en una pila tipo FIFO. La capacidad de esta pila dependerá de la cantidad de *frames* utilizados previamente como tamaño de secuencia para el entrenamiento de la red recurrente (p.e., para entrenar el modelo basado en *dataset* KTH se ha utilizado un tamaño de longitud fija de 25 *frames* por secuencia). Cuando está pila se completa, la información contenida es introducida a la red recurrente *LSTM*, la cual proporciona a su salida la predicción de la acción correspondiente al conjunto de *frames* almacenados en la pila. Una vez realizada la predicción, se realiza la detección y extracción del vector de características del siguiente *frame*. Este nuevo vector se introduce al final de la pila, desplazando el resto de los vectores almacenados, y expulsando de la pila, el primer vector correspondiente al primer *frame*. Este nuevo conjunto de vectores almacenados en la pila es nuevamente introducido a la red recurrente *LSTM* para realizar una nueva predicción. Este ciclo se repite hasta que el detector de personas deja de proporcionar información de las coordenadas de la persona detectada.

En los siguientes apartados se muestra la evaluación conjunta del detector y predictor utilizando los modelos previamente entrenados. Hay que indicar que únicamente se muestran los resultados obtenidos utilizando el modelo de predicción (*LRCN*) basado en KTH, ya que las pruebas realizadas con los modelos de predicción (*LRCN*) basados en GBA no han dado resultados satisfactorios en las evaluaciones realizadas. Se recomienda para trabajos futuros desarrollar un nuevo *dataset* de secuencias GBA más completo que se

pueda entrenar con las arquitecturas de los modelos planteados con el objetivo de conseguir, al menos, los resultados obtenidos con el modelo *LRCN* basado en KTH.

6.3.1 Evaluación mediante modelo basado en KTH

A continuación, se procede a evaluar la red recurrente *LRCN* entrenada a partir del *dataset* KTH utilizando un extractor de características basado en A*lexnet* (apartado 6.2.1).

En una primera evaluación, se utiliza para realizar la prueba varias secuencias correspondientes al *dataset* KTH.

Para esta primera evaluación se seleccionan un conjunto de videos correspondientes a las diferentes acciones (*handwaving, hadnclapping, boxing, jogging, running* y *walking)* correspondientes al *dataset* KTH. Hay que indicar que el modelo entrenado a partir de las secuencias de video del *dataset* KTH utilizaba un tamaño de *frame* 160x120 pixeles. En esta evaluación no se realiza la predicción sobre toda el *frame* sino sobre la ventana (recuadro azul) obtenida a partir del detector (recuadro amarillo) siendo esta de tamaño inferior al *frame* original. Con esto se pretende aclarar que, aunque se trate de los mismos videos de entrenamiento, realmente se está evaluando el modelo utilizando recortes de los mismos.

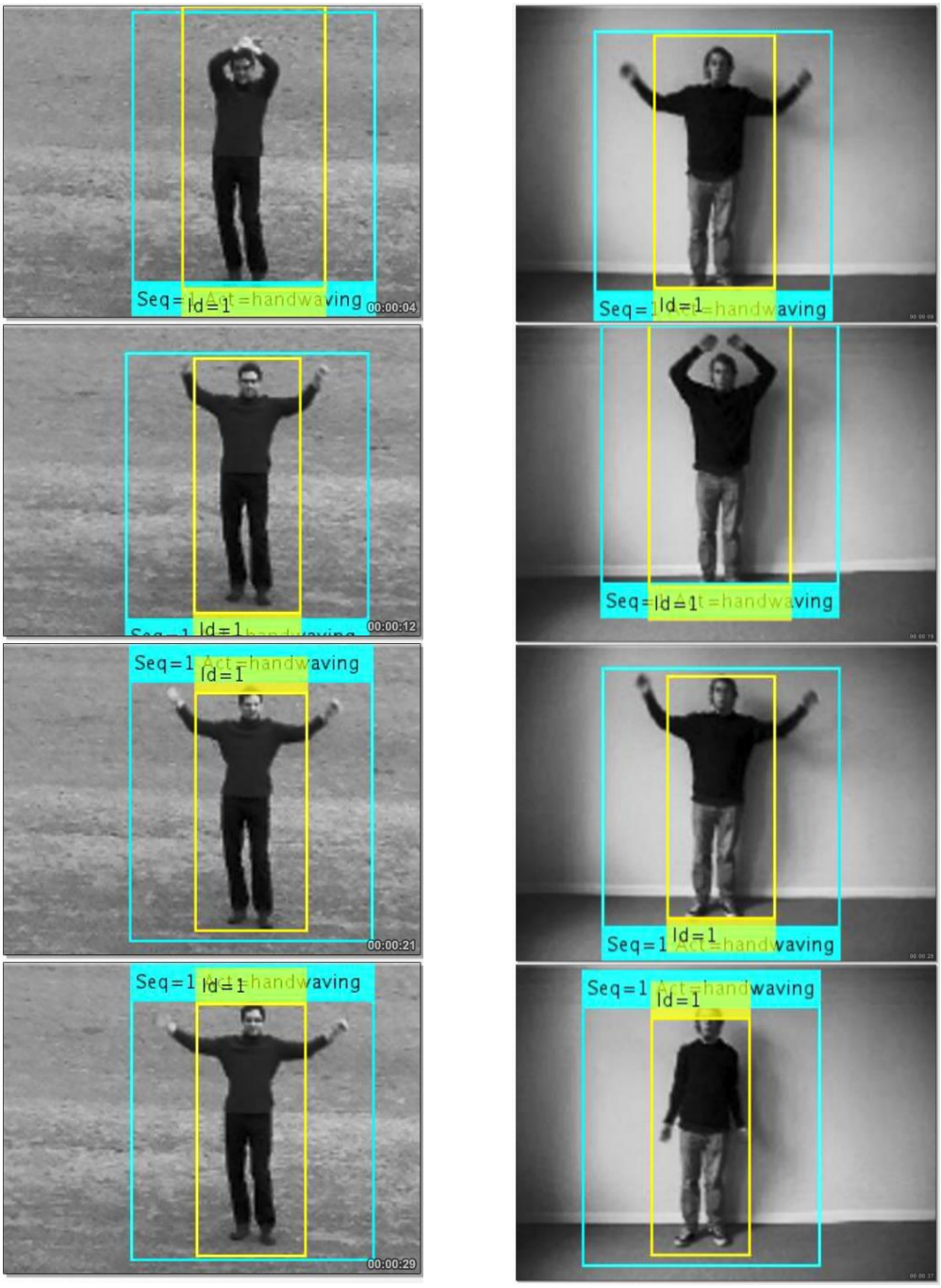


Figura 79 Evaluación del modelo sobre video *handwaving dataset* KTH

Video	Precisión	Comentarios
person01_handwaving_d1_uncomp	100%	Cámara estática
person03_handwaving_d4_uncomp	100%	Cámara estática
person09_handwaving_d2_uncomp	29.29%	Cámara en movimiento. La acción se confunde con *handclapping*
person19_handwaving_d2_uncomp	100%	Cámara en movimiento.
person25_handwaving_d1_uncomp	100%	Cámara estática

Tabla 20 Resultados de la evaluación de las secuencias KTH correspondientes a la acción *handwaving*

La evaluación realizada sobre las secuencias que contienen la acción *handwaving* ha obtenido una precisión del 100% en 4 de los 5 videos analizados. En el único video en el cual la precisión baja al 29,29% la acción predicha se confunde únicamente con la acción *handclapping*, siendo esta acción visualmente muy parecida a la acción *handwaving,* y más aún cuando en el transcurso de la secuencia, hay un efecto de cambio de zoom de la imagen, acercando y alejando a la persona, tal como se puede comprobar een la secuencia de video analizada (person09_handwaving_d2_uncomp.avi).

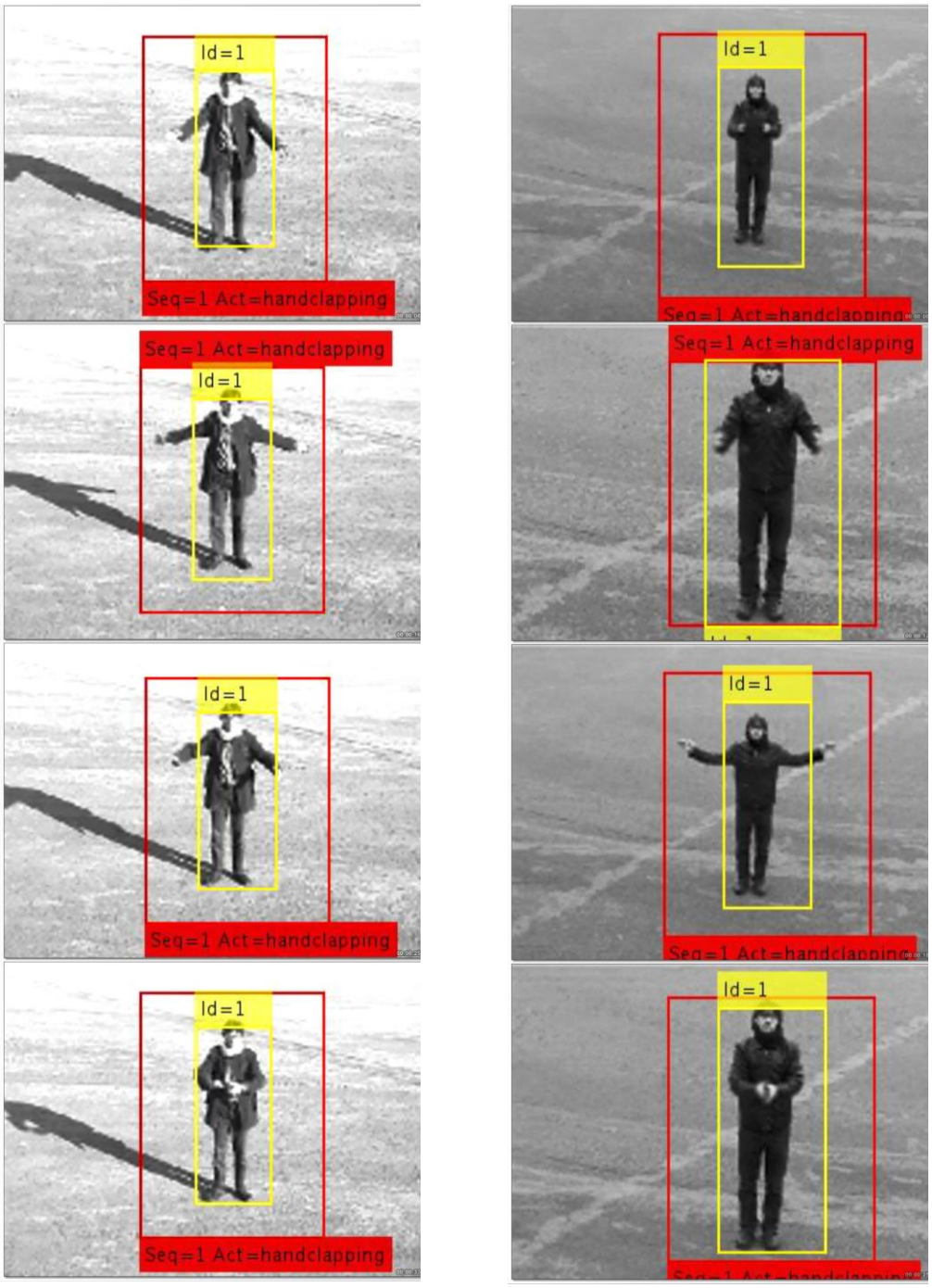


Figura 80 Evaluación del modelo sobre video *handclapping* del *dataset* KTH

En la Figura 80, se muestran dos ejemplos pertenecientes a los seis videos analizados y que se corresponden con la acción *handclapping.*

Video	Precisión	Comentarios
person01_handclapping_d1_uncomp	100%	Cámara estática
person02_handclapping_d2_uncomp	100%	Cámara estática
person10_handclapping_d4_uncomp	100%	Cámara estática
person14_handclapping_d3_uncomp	100%	Cámara estática
person25_handclapping_d2_uncomp	42.27%	La acción se confunde con *handwaving*.
person24_handclapping_d2_uncomp	79.11%	Cámara en movimiento.

Tabla 21 Resultados de la evaluación de las secuencias KTH correspondientes a la acción *handclapping*

La evaluación realizada sobre las secuencias que contienen la acción *handclapping* ha obtenido una precisión del 100% en 4 de los 6 videos analizados. En unos de los videos en el cual la precisión cae hasta el 42,27% la acción predicha es confundida únicamente con la acción *handwaving*, efecto inverso a lo que ocurría al analizar las secuencias de videos correspondientes a la acción *handwaving.* En el otro video analizado en el cual la precisión cae al 79,11% se observa que los errores en la predicción pueden ser debidos al movimiento de cámara de la secuencia de video analizada (person24_handclapping_d2_uncomp).

Figura 81 Evaluación del modelo sobre video boxing dataset KTH

Video	Precisión	Comentarios
person02_boxing_d2_uncomp	72.65%	Cámara estática
person10_boxing_d2_uncomp	81.02%	Cámara estática
person25_boxing_d2_uncomp	80.35%	Cámara estática
person17_boxing_d4_uncomp	93.22%	Cámara estática
person14_boxing_d3_uncomp	65.41%	Cámara estática

Tabla 22 Resultados de la evaluación de las secuencias KTH correspondientes a la acción boxing

Al evaluar los videos en los cuales transcurre la acción *boxing* se ha detectado que la predicción muestra unos valores aceptables (ver Tabla 22) cuando el sujeto que realiza la acción mantiene quieta la posición y únicamente realiza el movimiento de los brazos. Todos los videos analizados en los cuales el sujeto no mantenía la posición los resultados obtenidos daban unos valores de precisión muy bajos.

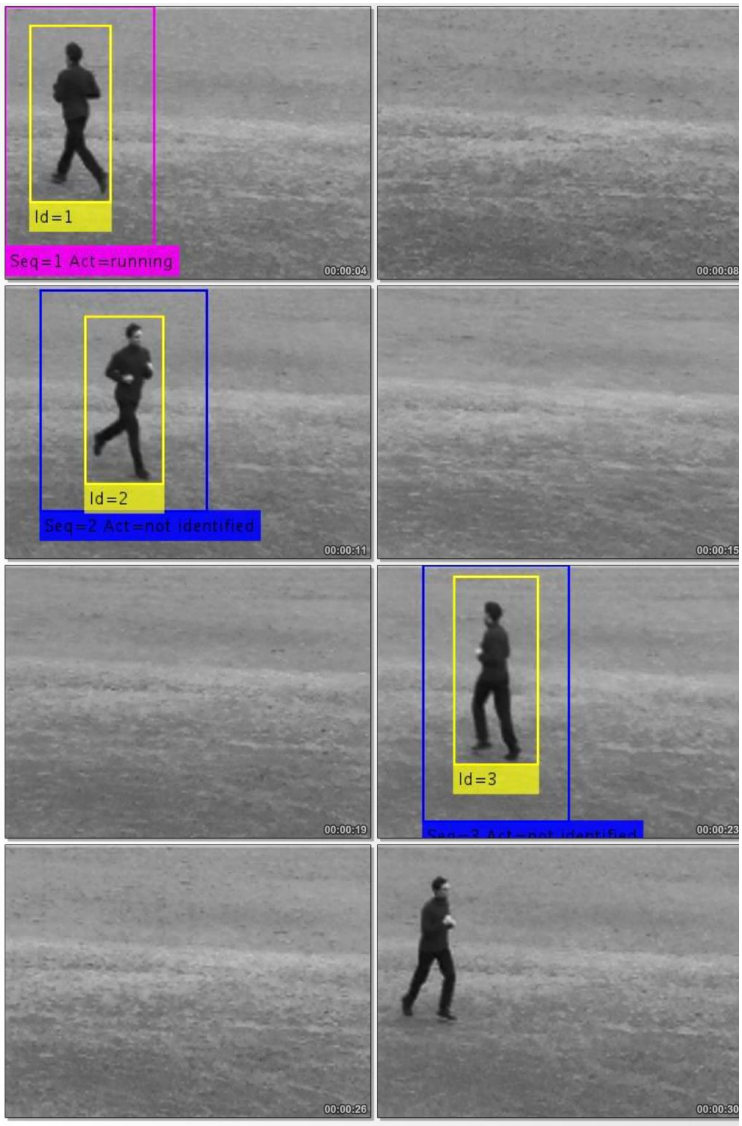


Figura 82 Evaluación del modelo sobre video jogging dataset KTH

Video	Precisión	Comentarios
person01_jogging_d2_uncomp	0%	La acción se confunde *running*
person08_jogging_d2_uncomp	0%	La acción se confunde *running*
person14_jogging_d2_uncomp	0%	La acción se confunde *running*

Tabla 23 Resultados de la evaluación de las secuencias KTH correspondientes a la acción *jogging*

En todos los videos evaluados en los cuales transcurre la acción *jogging* la acción predicha siempre ha sido confundida con la acción *running*. Hay que indicar que se si se visualizan estos videos resulta difícil diferenciar las acciones *jogging* de *running.*

Figura 83 Evaluación del modelo sobre video running dataset KTH (1/2)

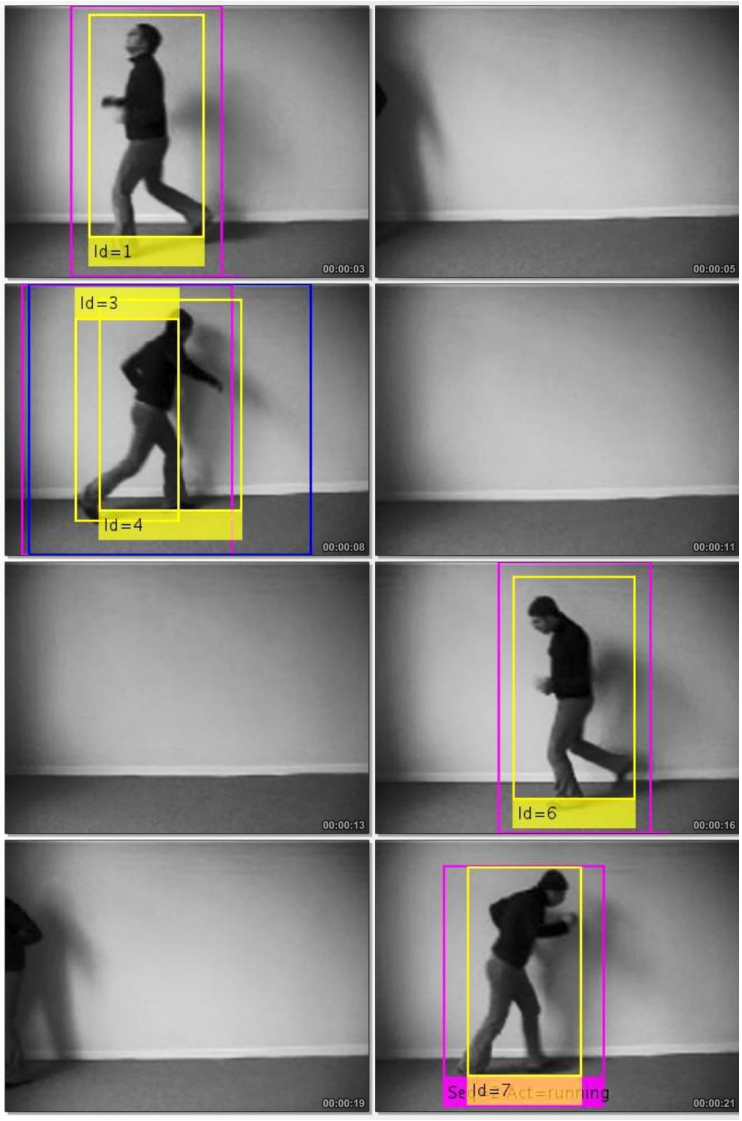


Figura 84 Evaluación del modelo sobre video *running dataset* KTH (2/2)

Video	Precisión	Comentarios
person06_running_d2_uncomp	89.28%	La acción se confunde *walking*
person09_running_d2_uncomp	100%	
person19_running_d2_uncomp	100%	
person02_running_d4_uncomp	93.50%	Reducido el predictor para que clasifique con 10 *frames*. La acción se confunde a veces con *walking*
person05_running_d1_uncomp	86.70%	Reducido el predictor para que clasifique con 10 *frames*. La acción se confunde a veces con *walking*

Tabla 24 Resultados de la evaluación de las secuencias KTH correspondientes a la acción *running*

La evaluación de las secuencias de videos en las que transcurre la acción *running* ha obtenido unos valores de precisión superiores al 85%. En todos ellos, los errores de predicción han sido provocados por confundir la acción *running* por la acción *walking*.

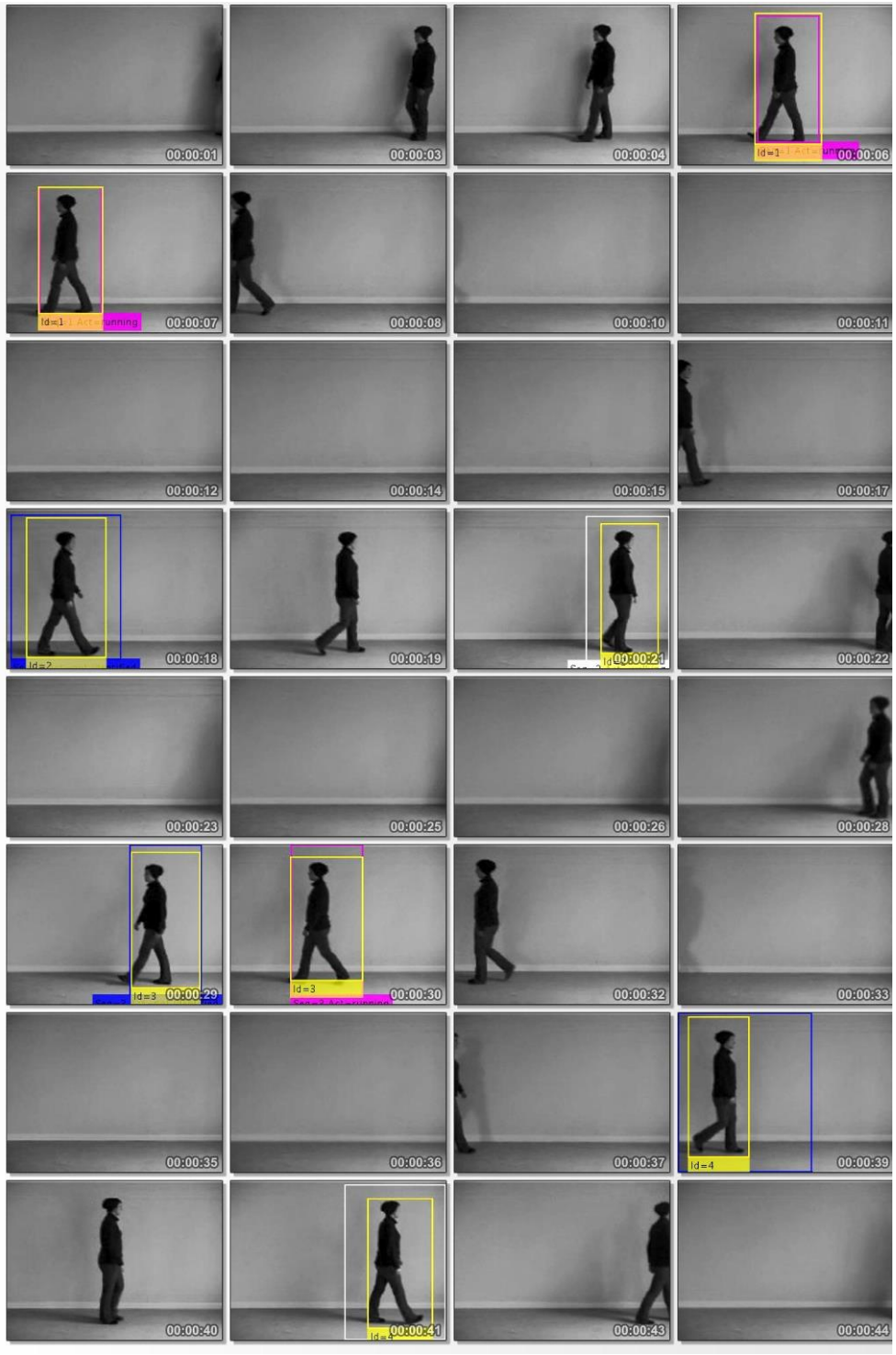


Figura 85 Evaluación del modelo sobre video *walking dataset* KTH

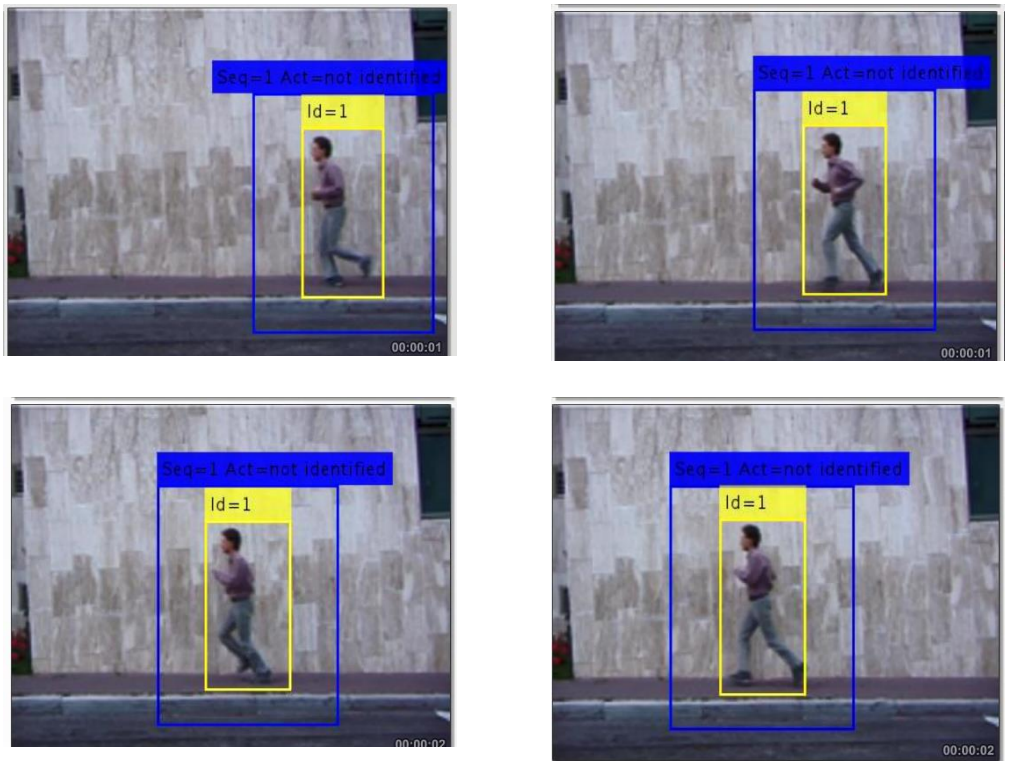
Video	Precisión	Comentarios
person01_walking_d1_uncomp	45.45%	La acción se confunde *running*.
person09_walking_d2_uncomp	0%	La acción se confunde con *running*
person10_walking_d4_uncomp	0%	La acción se confunde con *running*
person19_walking_d4_uncomp	41.17%	La acción se confunde *running*
person24_walking_d3_uncomp	30.36%	La acción se confunde *running*.

Tabla 25 Resultados de la evaluación de las secuencias KTH correspondientes a la acción *walking*

Al contrario de lo que ocurría con las secuencias de video que contienen acciones del tipo *running*, y estas eran confundidas por la acción *walking*, para el caso de las secuencias de video correspondientes a la acción *walking*, la precisión obtenida no es aceptable del todo, ya que las predicciones se confundan con la acción *running.*

Indicar, como curiosidad, que en los videos analizados siempre que el sujeto camina de derecha a izquierda la acción era predicha como *running,* pero si camina de izquierda a derecha la acción predicha es identificada como *walking.*

Una vez realizada la evaluación con los videos correspondientes al *dataset* KTH, se va a realizar una nueva evaluación utilizando como datos de test varias secuencias correspondientes al *dataset Weizmann*. Esta base de datos es ideal para evaluar el modelo basado en KTH ya que las secuencias transcurren con planos secuencia muy similares y con fondos estáticos.



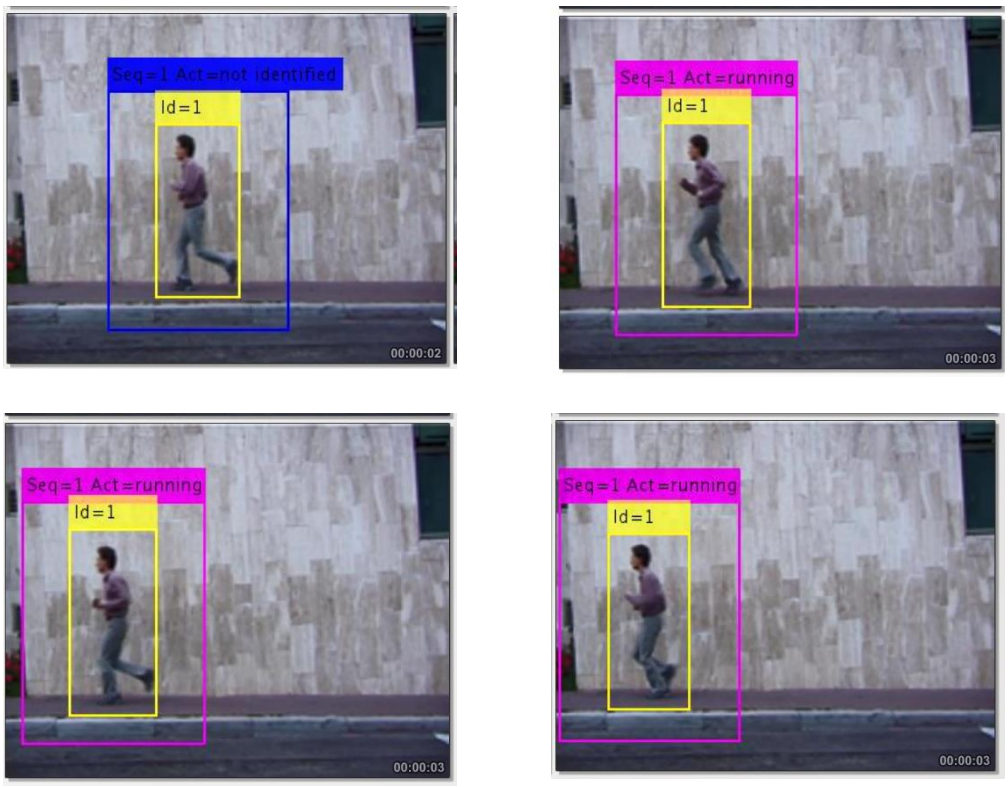


Figura 86 Evaluación del modelo sobre video running dataset Weizmann

Video	Precisión
weizmann_daria_run	100%
weizmann_denis_run	100%
weizmann_eli_run	100%
weizmann_ido_run	100%
weizmann_ira_run	100%
weizmann_lena_run1	100%[1]
weizmann_lena_run2	100%[1]
weizmann_lyova_run	100%
weizmann_moshe_run	100%
weizzman_shahar_run	100%

[1]El video no contiene suficientes *frames* por eso se reduce el predictor a 10 *frames*

Tabla 26 Resultados de la evaluación de las secuencias *Weizmann* correspondientes a la acción *running*

Al evaluar todas las secuencias correspondientes a la acción *running* del *dataset Weizmann*, la precisión obtenida en todos los videos ha sido del 100%.

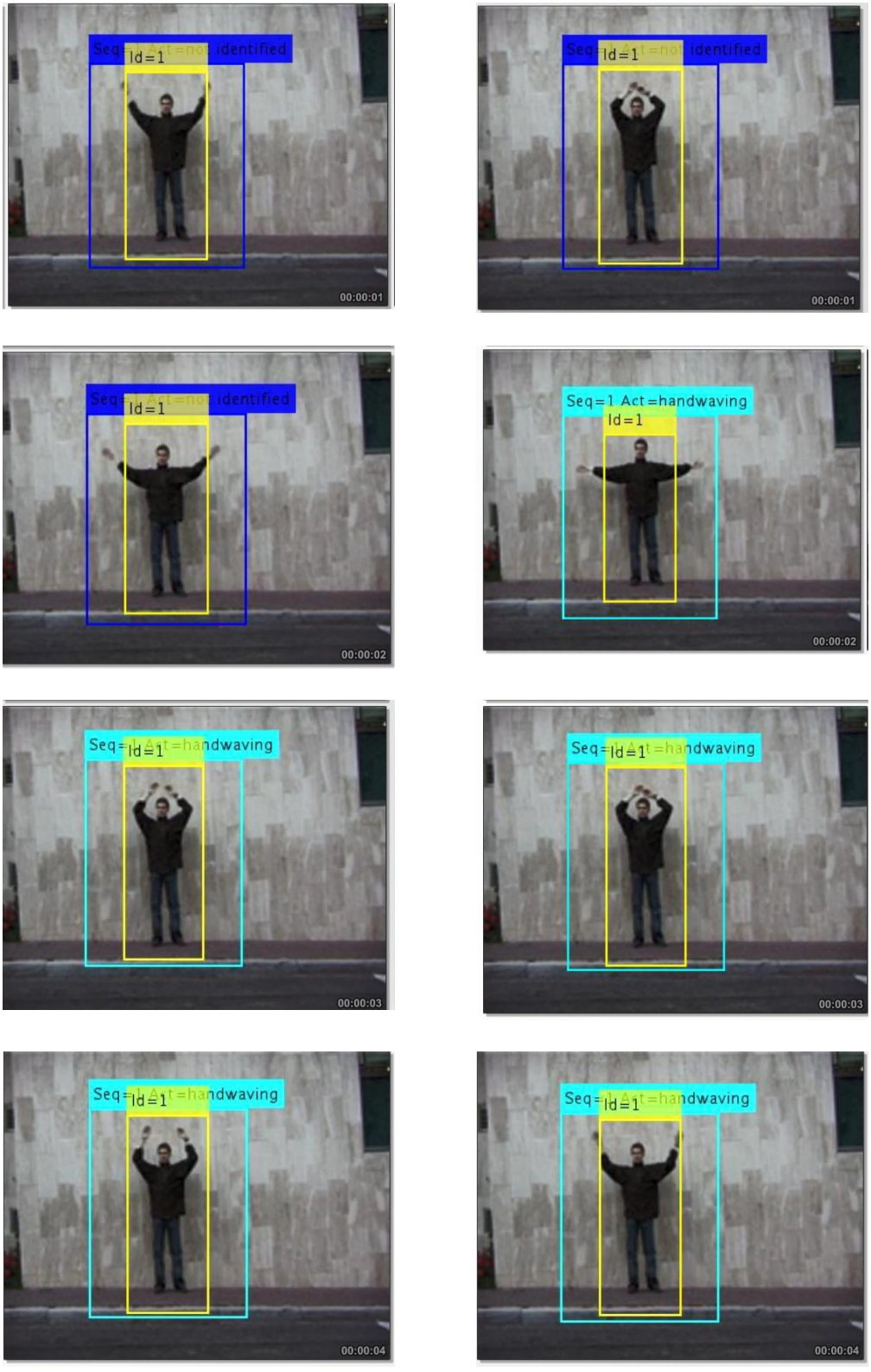


Figura 87 Evaluación del modelo sobre video *handwaving* del *dataset Weizmann*

Video	Precisión
weizmann_daria_wave2	0%[1]
weizmann_denis_wave2	0%[1]
weizmann_eli_wave2	100%
weizmann_ido_wave2	100%
weizmann_ira_wave2	0%[1]
weizmann_lena_wave2	100%
weizzman_lyova_wave2	100%
weizzman_moshe_wave2	100%
weizmann_shahar_wave2	100%

[1]La acción es confundida por *handclapping*

Tabla 27 Resultados de la evaluación de las secuencias *Weizmann* correspondientes a la acción *handwaving*

Al evaluar todas las secuencias correspondientes a la acción *handwaving* del *dataset Weizmann*, la precisión obtenida en la mayoría de los videos analizados ha sido del 100%. Hay que indicar que los videos en los cuales la precisión es del 0%, la acción predicha se corresponde con *handclapping* igual que sucedía al evaluar con los del *dataset* KTH.

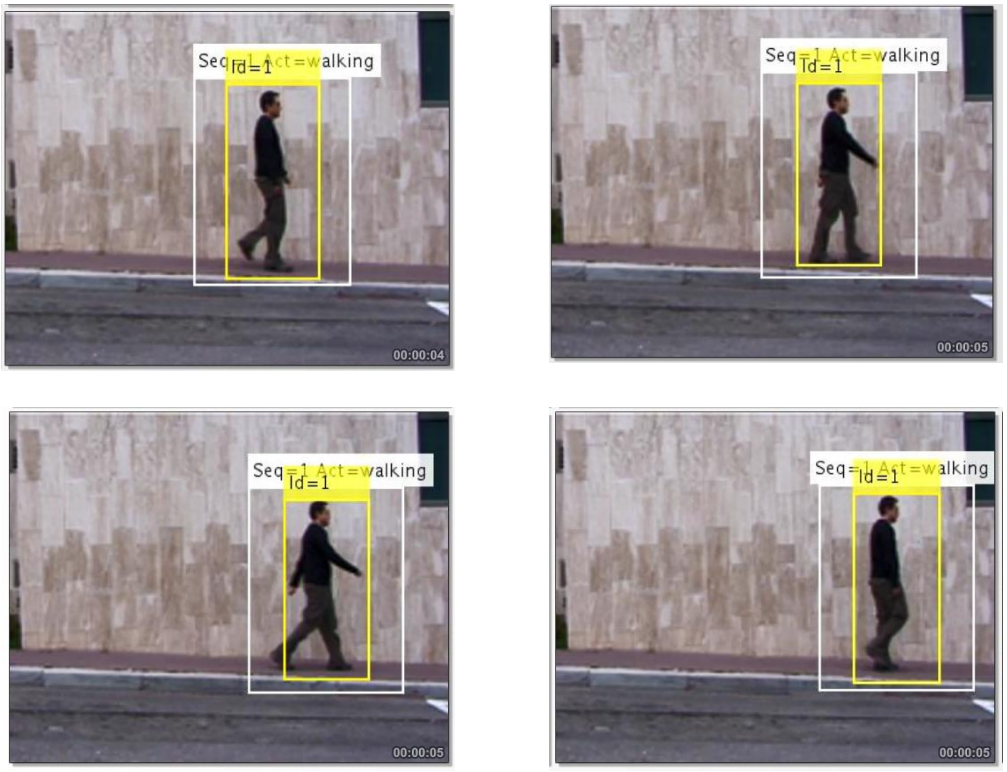


Figura 88 Evaluación del modelo sobre video *walking* del *dataset Weizmann*

Video	Precisión
weizmann_daria_walk	71.80%
weizmann_denis_walk	0%[1]
weizmann_eli_walk	90.62%
weizmann_ido_walk	100%
weizmann_ira_walk	64.44%
weizmann_lena_walk1	96.67%
weizmann_lena_walk2	29.03%
weizmann_lyova_walk	100%
weizmann_moshe_walk	0%[1]
weizmann_shahar_walk	100%

[1]La acción se confunde con *running*

Tabla 28 Resultados de la evaluación de las secuencias *Weizmann* correspondientes a la acción *walking*

Al evaluar todas las secuencias correspondientes a la acción *walking* del *dataset Weizmann*, la precisión obtenida en la mayoría de los videos analizados es bastante aceptable. E igual que sucedía al evaluar con las secuencias del *dataset* KTH, la acción *walking* es confundida en gran medida con la acción *running*

Finalmente, se intenta realizar una evaluación del modelo utilizando como video de test las secuencias del *dataset* GBA. En los casos evaluados la precisión obtenida es demasiado baja lo que indica que este modelo no es válido para este tipo de videos. Principalmente,

esto es debido a que los planos escena de estas secuencias difieren mucho de las secuencias utilizadas para el entrenamiento del modelo basado en KTH.

Figura 89 Ejemplo de predicción de acciones utilizando modelo basado en KTH sobre el *dataset* GBA

Capítulo 7

Conclusiones y trabajos futuros

7.1 *Conclusiones*

Con la realización de este trabajo se ha comprobado que el aprendizaje profundo o *Deep Learning* es una técnica óptima para el desarrollo de sistemas automáticos de detección de personas y objetos debido a su fiabilidad, que se basa en el uso de algoritmos con redes convolucionales o CNN para la extracción y selección de las características discriminantes de las personas y su modelado en múltiples situciaciones. En particular, el detector *Faster Region-based Convolutional Neural Network (Faster-RCNN)*, ha demostrado ser un detector con un rendimiento significativamente bueno, obteniendo en las pruebas realizadas unas métricas bastante aceptables (mAP=45%).

Se han evaluado modelos *LRCN* basados en redes neuronales recurrentes (*RNNs*), concretamente redes de memoria de corto a largo plazo o *Long Short-Term Memory,* usualmente denominadas *LSTM*, que han demostrado un buen comportamiento al abordar tareas de modelado, en este caso, de secuencias en el aprendizaje automático, como el reconocimiento automático de actividades en imágenes. Estos modelos han sido entrenados utilizando tanto bases de datos conocidas como KTH, como propias como GBA, obteniendo buenos resultados con el modelo basado en KTH (de un 99.37% de precisión), pero no tan buenos con el modelo basado en GBA: con valores de precisión del 41.54% utilizando como extractor de características *Alexnet* y del 79.63% con las de *Resnet*50.

Se ha comprobado que la extracción de características mediante la activación de una determinada capa de la red, utilizando una de las capas totalmente conectadas correspondiente a un modelo pre-entrenado como es *Alexnet* o *Resnet50*, es un método adecuado para transformar la información proveniente de una imagen en un vector unidimensional de datos. En concreto, se ha comprobado que *Resnet50* incrementa la precisión del detector de acciones (modelo *LRCN*) en más de un 300% respecto a utilizar *Alexnet*, pero su tiempo de computo aumenta considerablemente, ya que realizar la

extracción de características utilizando la capa *'fc1000'*, consume un tiempo entre 160ms y 510ms por cada imagen a procesar. Este tiempo depende del tamaño de la imagen recortada. Por otro lado, la capa *'fc7'* del modelo *Alexnet*, consume un tiempo bastante inferior que la capa *'fc1000'*, siendo éste entre 7ms y 12ms para un mismo tamaño de imagen.

En este trabajo, se ha utilizado un aprendizaje semi-supervisado, estructurado en tres niveles (1- detector de personas *Faster-RCNN*, 2- reconocimiento de actividades *LSTM*, 3- detección de anomalías). Se ha comprobado que la metodología en tres niveles es muy funcional ya que tiene más utilidad predecir las acciones que transcurren en una determinada secuencia, y analizar a posteriori, si se trata de un suceso anómalo permitiendo de este modo, que determinadas acciones se pueden considerar anómalas o no, dependiendo del escenario o contexto en el cual se desarrollan.

Dado que no se ha conseguido con éxito un predictor de acciones que trabaje correctamente con el *dataset* GBA*,* no ha podido ser evaluada la última fase de identificación de anomalía basada en análisis probabilístico.

Además, a lo largo del desarrollo de este trabajo se ha comprobado que Matlab puede no ser el lenguaje más adecuado para programar este tipo de algoritmos, principalmente por varios motivos. En primer lugar, la librería que ha incluido Matlab recientemente para trabajar con redes neuronales está limitadas en funcionalidades, si bien es cierto que, a medida que publican actualizaciones, esta librería va mejorando considerablemente en funcionalidad. Otros de los aspectos negativos de Matlab es la falta de transparencia del código de las funciones propias de la librería, el cual no es fácilmente accesible, lo que provoca que realmente se desconozca los que está realizando una determinada función.

Existen otras librerías basadas en el lenguaje Python como Keras [39] o entornos de programación como TensorFlow [40], Theano [41] o CNTK [42] que tienen mucha más potencia y son más versátiles, y lo que es más importante, tienen una comunidad de usuarios más amplia que realiza más aportaciones, principalmente porque se trabaja con librerías Open Source no supeditadas a una licencia como sí ocurre con Matlab.

7.2 *Trabajos futuros*

A lo largo de este trabajo, se han detectado varias cuestiones que mejorarían los resultados aquí alcanzados y abrirían otros de interés para la comunidad científica, y el grupo de investigación GEINTRA [2] en el que se enmarca:

- Desarrollar la base de datos GBA de forma más amplia, incluyendo más tipos de acciones diferentes eintentando que el número de secuencias por acción sea equilibrado para todas.
- Desarrollar la grabación de las secuencias manteniendo el plano escena desde varios enfoques diferentes (similar al de las secuencias de KTH).
- Evaluar la etapa de identificación de anomalía, aplicando la técnica que se ha propuesto en este trabajo, sobre una secuencia en la cual transcurran un conjunto variado de acciones diferentes realizadas por diferentes sujetos.

- Realizar el entrenamiento con un hardware más potente, utilizando nuevas tarjetas gráficas dedicadas que permitan realizar los entrenamientos con más volumen de datos y en menor tiempo.
- Utilizar otros lenguajes de programación más extendidos para este tipo de aplicaciones como Python utilizando librerías como Keras [39].

Planos

En este capítulo se incluyen los códigos resumidos y comentado de los scripts de Matlab principales para las tareas de detección de personas y reconocimiento de actividades.

8.1 *Script de entrenamiento de la Faster-RCNN*

Este script se ha desarrollado para realizar los entrenamientos del detector *Faster-RCNN* a partir de un conjunto de imágenes utilizado como datos de entrenamiento.

```
%Se carga el Dataset de imagenes con el formato requerido para entrenamiento RCNN
Dataset=arrayImagenes;

% distribución de los datos de entrenamiento y test de manera aleatoria
numdata=numel(Dataset.imageFilename);
idx=randperm(numdata);
indice=numdata*0.20;
idx1=idx(1:indice);
idx2=idx(indice:end)
testData=table(Dataset.imageFilename(idx1), Dataset.persona(idx1))
trainingData=table(Dataset.imageFilename(idx2), Dataset.persona(idx2))
testData.Properties.VariableNames{1}='imageFilename';
testData.Properties.VariableNames{2}='persona';
trainingData.Properties.VariableNames{1}='imageFilename';
trainingData.Properties.VariableNames{2}='persona';

%se utiliza red pre-entrenada alexnet. Se pude utilizar vgg16 o vgg19 pero el tiempo de computo es mayor
net=alexnet;
%net=detector.Network.Layers;   %se habilita en caso de tener que reiniciar el entrenamiento por que este se
ha detenido por un error

%se mantienen los pesos iniciales de alexnet. Se reentrena las tres ultimas capas con la clasificación deseada
layersTransfer = net.Layers(1:end-3);

layers = [...
   layersTransfer
   fullyConnectedLayer(2, 'WeightLearnRateFactor', 20, 'BiasLearnRateFactor', 20)
   softmaxLayer
   classificationLayer];
```

```
%Configurar opciones de entrenamiento para cada uno de los cuatro pasos del detector (explicacion parametros en apartado 5)

% Step 1. Entrenamiento del detector Faster RCNN para las clases definidas

optionsStage1 = trainingOptions('sgdm', 'MiniBatchSize',30, 'MaxEpochs', 10, 'InitialLearnRate', 1e-3); % se ajusta un valor que permita un entrenamiento más rápido pero sin penalizar la precisión

% Step 2. Entrenamiento de detector Fast-RCNN utilizando la RPN entrenada en el step 1

optionsStage2 = trainingOptions('sgdm', 'MiniBatchSize',30, 'MaxEpochs', 10, 'InitialLearnRate', 1e-3);

% Step 3. Re-entrenamiento de la red RPN utilizando los pesos obtenidos del entrenamiento realizado con el Fast-RCNN en el step 2

optionsStage3 = trainingOptions('sgdm', 'MiniBatchSize',30, 'MaxEpochs', 10, 'InitialLearnRate', 1e-5);   %se disminuye el learning rate para que el entrenamiento sea más lento y no se modifiquen totalmente los pesos obtenidos anteriormente

% Step 4. Re-entrenamiento del Fast-RCNN utilizando la red RPN actualizada en el step 3

optionsStage4 = trainingOptions('sgdm', 'MiniBatchSize',30, 'MaxEpochs', 10, 'InitialLearnRate', 1e-5); % se disminuye el learning rate para que el entrenamiento sea más lento y no se modifiquen totalmente los pesos obtenidos anteriormente

options = [optionsStage1 optionsStage2 optionsStage3 optionsStage4];

rng(0);    % Establecer la semilla del generador aleatorio a 0

% Se ejecuta el detector

detector = trainFasterRCNNObjectDetector(trainingData, layers, options, 'SmallestImageDimension',400,...

        'BoxPyramidScale', 1.2);
```

8.2 *Script de evaluación de la precisición del detector de personas*

Este script se ha desarrollado con el objetivo de evaluar los resultados obtenidos con el detector *Faster-RCNN* utilizando una métrica (apartado 4.2) ampliamente utilizada en los trabajos y competiciones relacionados con las redes neuronales aplicadas a visión.

```
resultsStruct = struct([]);
lenghtTestData=size(testData);
for  i = 1:lenghtTestData(1)
        % Lectura de cada imagen del array de imágenes de test
        I = imread(testData.imageFilename{i});
        %I=imresize(I,2); %solo para ajustar tamaño imagen a la entrada de alexnet si esta fuera superior a 227x227 pixeles
        % Se ejecuta el detector
        [bboxes, scores, labels] = detect(detector, I,'NumStrongestRegions', 2000,'Threshold', 0.5);


        % se almacenan datos en estructura
        resultsStruct(i).Boxes = bboxes;
        resultsStruct(i).Scores = scores;
        resultsStruct(i).Labels = labels;
end

% Se convierte la estructura a una tabla
results = struct2table(resultsStruct);
```

```
% Se extraen del array de test la información de las coordenadas bounding box de cada imagen
expectedResults = testData(:, 2:end);

% Evaluar la precision del detector Average Precision metric (mAP) (ver explicación en apartado 4.2)
[ap, recall, precision] = evaluateDetectionPrecision(results,expectedResults);

% Se representa la curva de precision/recall
figure
plot(recall, precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.1f', ap))
```

8.3 *Script de entrenamiento de la red LSTM*

Este script se ha desarrollado para realizar los entrenamientos de los modelos *LRCN* a partir de un conjunto de secuencias de video utilizados como datos de entrenamiento.

```
Z=[falling gettingup running sittingdown walking]; %declaracion de las clases (acciones)

%distribución de los datos de entrenamiento y test de manera aleatoria
idx = randperm(length(Z));
a=round(0.90*length(idx));
idxTrain=idx(1:a);
idxTest=idx(a+1:end);
ImagesTrain=Z(idxTrain);
ImagesTest=Z(idxTest);

%creación del array de etiquetas
for i=1:size(Z,2)
   if i<87
      Y(i,1)={'falling'};
   elseif (i>=111) && (i<221)
      Y(i,1)={'getting up'};
   elseif (i>=221) && (i<331)
      Y(i,1)={'running'};
   elseif (i>=331) && (i<441)
      Y(i,1)={'sitting down'};
   else
      Y(i,1)={'walking'};
   end
end
YTest = Y(idxTest);
Y=Y(idxTrain);
Y=categorical(Y);

%obtener la longitud de cada secuencia
numObservations = numel(ImagesTrain);
for i=1:numObservations
   sequence = ImagesTrain{i};
   sequenceLengths(i) = size(sequence,2);
end
```

```
%ordenar las secuencias de entrenamiento en función de la longitud
[sequenceLengths,idx] = sort(sequenceLengths);
ImagesTrain = ImagesTrain(idx);
Y = Y(idx);

%representación gráfica de las secuencias ordenadas
figure
bar(sequenceLengths)
ylim([0 200])
xlabel("Secuencias")
ylabel("Longitud")
title("Datos ordenados")

%Realiza retraining a partir de los pesos iniciales y los recurrentes de otra red ya entrenada
%layersTransfer = net.Layers(1:end-3);

numClasses = 5;
layers = [...
   %layersTransfer   %habilitar para realizer retraining
   sequenceInputLayer(4096)   %alexnet 4096; resnet50 1000
   bilstmLayer(256,'OutputMode','sequence')
   lstmLayer(256,'OutputMode','last')
   dropoutLayer(0.9)
   %fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',20)    %habilitar en caso
de retraining
   fullyConnectedLayer(numClasses)
   softmaxLayer
   classificationLayer];


%shuffle = 'every-epoch';
shuffle = 'never';    %siempre que la longitud de las secuencias sea variable se recomienda no mezclarlas
maxEpochs =5000;
miniBatchSize = 495;
InitialLearnRate=0.01;

%Parametros de configuración del entrenamiento (ver apartado 3.5.4)
options = trainingOptions('sgdm', ...
   'MaxEpochs',maxEpochs, ...
   'MiniBatchSize',miniBatchSize, ...
   'Shuffle', shuffle, ...
   'InitialLearnRate',InitialLearnRate, ...
   'L2Regularization',0.0005, ...
   'Momentum',0.95, ...
   'GradientThresholdMethod','global-l2norm', ...
   'GradientThreshold',2, ...
   'Plots', 'training-progress', ...
   'LearnRateSchedule', 'piecewise', ...
   'LearnRateDropFactor', 0.1, ...
   'LearnRateDropPeriod', 4000);
rng(0);   %inincializa el valor de la semilla del generador de número aleatorios

netLstm = trainNetwork(ImagesTrain,Y,layers,options);   %Inicia el entrenamientos con las opciones
configuradas
```

```
%Evaluación utilizando las secuencias de test para comprobar la precisión obtenida
YPred = classify(netLstm,ImagesTest);
%[netLstm,YPred,scores] = classifyAndUpdateState(netLstm,ImagesTest);    %además de clasificar actualiza el
estado de la celda de la unidad de memoria LSTM
YTest=categorical(YTest);
acc = sum(YPred == YTest)./numel(YTest)
```

8.3.1 Script para extraer frames y obtener descriptores

Este script se ha desarrollado con el objetivo de extraer los *frames* de una secuencia de video y convertir cada uno, a un vector de características mediante la utilización de una de las capas de activación de *Alexnet o Resnet50* (apartado 5.2.1). Los vectores de características son ordenados en único array con el formato admitido por las capas de tipo *LSTM* (Figura 52).

```
%recopilacion videos avi o mp4
fullname = fullfile('..\ruta en disco a directorio que contiene los videos','*.mp4');

archivo=dir(fullname);
% Se carga la red pre-entrenada AlexNet o Resnet50
net = alexnet;
layer = 'fc7';
%net=resnet50;
%layer='fc1000';

k=1;
t=[];
for i=1:length(archivo)
        arch=archivo(i).name;
        name=archivo.folder;
        v = vision.VideoFileReader(strcat(name,'\',arch));   %se crea objeto de lectura de videos
        %se habilita si se desea crear video avi con las modificaciones de la imagenes
        %filename = [sprintf('nombre_video_%06d',i) '.avi'];
        %videowrite=vision.VideoFileWriter(strcat(name,'\',filename), 'FrameRate',50);
        videoPlayer = vision.VideoPlayer;
        j=1;
        t=[];
        flag=~isDone(v);
        while flag
                video = step(v); %se adquiere un frame
                step(videoPlayer,video); %se visualiza el frame
        if ismatrix(video)
                video = cat(3,video,video,video); %se crean tres dimensiones si nos la tuviera
        end

        %video=imresize(video, [224 224]); %224x224 para resnet 50; 227x227 para alexnet
        %funciones para modificar aspecto videos Data Augmentation
        %video=fliplr(video); %crea la imagen especular del original. Volteo horizontal
        %video=imadjust(video,[0.2, 0.6],[]); %ajusta el contraste
        %video=imnoise(video); %introduce ruido en la imagen
        %video=video+0.5; %incrementa brillo
        %video=video-0.2; %decrementa brillo oscureciendo la imagen
        %step(videowrite,video);  %se graba el frame en el video avi. Se habilita si previamente se ha definido
        el objeto
```

```
        Features = activations(net,video,layer);    %se obtiene el vector de características del frame
        t(:,j)=Features;   %se acumula en un array los descriptores. En el formato admitido para el
        entenamiento Lstm (ver apartado 5.2)

        flag=~isDone(v);
        j=j+1;

        if (~flag&&j==51 ||j==51)    %si llega a 50 frames se genera una nueva secuencia
                arrayfeatures{k}=t;  %arrayfeatures contiene todas las secuencias convertidas a vectores de
                características. Tamaño 4096x50 si se ha obtenido con alexnet o 1000x50 si se ha obtenido
                con resnet50
                k=k+1;
                j=1;
                t=[];
        end
  end

  release(videoPlayer);
  %release(videowrite);    %se habilita si previamente se ha creado el objeto
  release(v);

end
```

8.4 *Script de la aplicación completa*

Este script ha sido desarrollado con el objetivo de englobar el detector de personas *Faster-RCNN* y el predictor de acciones *LRCN.* Se ha incorporado un filtro de tipo *Kalman* para facilitar el seguimiento o tracking de la persona detectada a lo largo de una secuencia de video. Parte del código utilizado en este script para modelar el filtro *Kalman* está extraído directamente del ejemplo de la documentación de Matlab (ejemplo *"Track Pedestrians from a Moving Car"*).

```
%se crea objeto de video para almacenar el video original con las detecciones y predicciones
p1 = VideoWriter('..\ruta en disco almacenar video salida\Videos_test.avi');
p1.FrameRate = 50;
open(p1);

% Carga el detector pre-entrenado Faster RCNN
detector=detectorFasterRCNN;
rng(0);

numMaxSequences=4; % Número máximo de detección de acciones simultaneas en una escena
position=zeros(1,numMaxSequences);
% Se inicializa estructura
featureStruct=struct([]);
featurePosition=[];
% Se carga la red encargada de obtener los descriptores
%netFeature=resnet50;
netFeature=alexnet;
% Se carga la red preentrenada LSTM
%netLstm=netGBA;
netLstm=netKTH;
```

```
% Inicializa un detector multiobjeto y establece parámetros de filtro
[tracker, positionSelector] = setupTracker();
% En cada instante de tiempo, se ejecuta el detector, se actualiza el seguimiento
% con los resultados de detección y muestra los resultados.
% Se configura el reproductor de video
% se indica la ruta en la cual se localiza el video a analizar
videoFile   = '..\ruta en disco video a analizar\video.mp4';
videoReader = VideoReader(videoFile);
videoPlayer = vision.DeployableVideoPlayer();
videoPlayer.Location=[0 0];
currentStep = 0;

action_count=1;
action_list={[]};

cont = hasFrame(videoReader);
while cont
   % se actualiza el contador de frames
   currentStep = currentStep + 1;

   % Lectura del siguiente frame
   frame = readFrame(videoReader);
   %frame=imresize(frame,2); % Utilizar si el video es de tamaño inferior a 227x227 o 224x224 dependiendo de
la red utilizada
   writeVideo(p1,frame);
   frame2=frame;
   % Se ejecuta el detector. La funcion retorna los resultados en un objeto requerido por multiObjectTracker.
   detections = detectObjects(detector, frame, currentStep);

   % Utilizando un listado de objectos de detección, retorna los 'tracks'
   % actualizados para cada intstante de tiempo (currentStep).
   if currentStep == 1
      costMatrix = zeros(0, numel(detections));
      [confirmedTracks,~,allTracks] = updateTracks(tracker, detections, currentStep, costMatrix);
   else
      costMatrix = detectionToTrackCost(allTracks, detections, positionSelector, tracker.AssignmentThreshold);
      [confirmedTracks,~,allTracks] = updateTracks(tracker, detections, currentStep, costMatrix);
   end

   % Elimina bounding box que sobrepasen el tamaño de la imagen
   ImageSize=[720 1280]; % Tamaño imagen secuencias GBA
   %ImageSize=[240 320]; % Tamaño imagen secuencias KTH
   %ImageSize=[288 360]; % Tamaño imagen secuencias Weizzman
   %ImageSize=[360 640]; % Tamaño imagen secuencias AVENUE
   confirmedTracks = removeNoisyTracks(confirmedTracks, positionSelector, ImageSize);

   if ~isempty(confirmedTracks)
      % Se crea array de etiquetas
      labels = cell(numel(confirmedTracks), 1);
      % Se crea array de colores de los recuadros
      colours={'blue', 'green', 'red', 'cyan', 'magenta', 'white','yellow'};d
      % Se crea array de clases de la red LSTM
      classes=[{'not identified'};netLstm.Layers(7).ClassNames];
      cadCol=cell(numel(confirmedTracks), 1);
```

```
        % Obtiene a partir de los 'tracks' confirmados, los 'bounding box' correctos
        bboxes = getTrackPositions(confirmedTracks, positionSelector);
        cadCol=[];
        for i = 1:numel(confirmedTracks)
            box = bboxes(i, :);
            idxTrackId=confirmedTracks(i).TrackID;
            idxPosition=find(position==idxTrackId);
            % Se llama a la función encargada de realizar la detección de la acción
            [position, featureStruct,featurePosition,rectangle] = detectedAction(frame2,netFeature, netLstm, ...
                confirmedTracks, idxTrackId, box, numMaxSequences, position, featureStruct,featurePosition);

            if idxPosition <= numMaxSequences
                % Se insertan los bounding box y etiquetas obtenidos sobre la imagen
                label1 = sprintf('Seq=%d Act=',idxPosition);
                label1 = strcat(label1,featureStruct(idxPosition).action);
                action_list(action_count)=featureStruct(idxPosition).action;
                action_count=action_count + 1;
                ind=contains(classes,featureStruct(idxPosition).action);
                cadCol=colours{ind};
                frame = insertObjectAnnotation(frame, 'rectangle', rectangle,label1, 'Color', cadCol, ...
                'FontSize', 15, 'TextBoxOpacity', .8, 'LineWidth', 2);
            end

            labels{i} = sprintf('Id=%d',idxTrackId);
        end

        frame = insertObjectAnnotation(frame, 'rectangle', bboxes, labels, 'Color', 'yellow', ...
        'FontSize', 15, 'TextBoxOpacity', .6, 'LineWidth', 2);
    end

    % Visualiza cada frame etiquetado
    videoPlayer(frame);
    %se graba el frame tres veces para qpoder visualizar posteriormente el video de manera más lenta y observar
los resultados
    writeVideo(p1,frame);
    writeVideo(p1,frame);
    writeVideo(p1,frame);

    % Sale de la aplicación si el video ha finalizado o el usuario ha cerrado la figura.
    cont = hasFrame(videoReader) && isOpen(videoPlayer);
end
close(p1);

%%
% La función setupTracker, es una función implementada por MATLAB y crea un 'multiObjectTracker' que
% analiza múltiples objetos utilizando filtros de Kalman. Al crear un multiObjectTracker considera lo siguiente:
% FilterInitializationFcn:En este caso, se espera que los objetos tengan un movimiento de velocidad constante.
% AssignmentThreshold: indica cuanto de lejos puede haber una detección. Este ejemplo utiliza el ratio de
% superposición obtenida a partir del bounding box como función de costo, con el umbral establecido en 0.999.
% NumCoastingUpdates: Cuántas veces se actualiza la detección del bounding box sin una obtener una
% predicción del detector antes de descartarla por completo. El valor predeterminado para este parámetro es 5.
% ConfirmationParameters: Los parámetros para confirmar un tracking. Se inicializa un nuevo tracking con cada
% detección no asignada. Algunas de estas detecciones pueden ser falsas, por lo que todos los tracks se
% inicializan como Tentativas. Para confirmar un tracking, debe detectarse al menos M veces en N
% actualizaciones. La elección de M y N depende de la visibilidad de los objetos.
```

```
% Se utiliza el valor predeterminado de 3 detecciones de 5 actualizaciones.
% Las salidas de la función son:
% tracker: el objeto multiObjectTracker que es configurado.
% positionSelector: Una matriz que especifica qué elementos del vector de estado
% son la posición: position = positionSelector * State

function [tracker, positionSelector] = setupTracker()

   % Creación del objeto tracker.
   tracker = multiObjectTracker('FilterInitializationFcn', @initBboxFilter, ...
      'AssignmentThreshold', 0.999, ...
      'NumCoastingUpdates', 5, ...
      'ConfirmationParameters', [3 5], ...
      'HasCostMatrixInput', true);

   % El vector Estado es: [x; vx; y; vy; w; vw; h; vh]
   % [x;y;w;h] = positionSelector * State
   positionSelector = [1 0 0 0 0 0 0 0; ...
                       0 0 1 0 0 0 0 0; ...
                       0 0 0 0 1 0 0 0; ...
                       0 0 0 0 0 0 1 0];
end

%%
% initBboxFilter es una función implementada por MATLAB que define un filtro
% Kalman para realizar un filtrado de los bounding box obtenidos.

function filter = initBboxFilter(Detection)

% Paso 1: Define un modelo de movimiento y Estado
%   Utiliza un modelo de velocidad constante para el bounding box.
%   El estado es [x; vx; y; vy; w; wv; h; hv]
%   El estado de la matriz de transición es:
%      [1 dt 0  0 0  0 0  0;
%       0  1 0  0 0  0 0  0;
%       0  0 1 dt 0  0 0  0;
%       0  0 0  1 0  0 0  0;
%       0  0 0  0 1 dt 0  0;
%       0  0 0  0 0  1 0  0;
%       0  0 0  0 0  0 1 dt;
%       0  0 0  0 0  0 0  1]
%   Se asume dt = 1. No se considera el modelo de transición variable en el
%   tiempo para el filtro Kalman lineal.
   dt = 1;
   cvel =[1 dt; 0 1];
   A = blkdiag(cvel, cvel, cvel, cvel);

% Paso 2: Define el proceso de ruido.
%   El ruido del proceso representa las partes del proceso que el modelo no tiene
%   en cuenta. Por ejemplo, en un modelo de velocidad constante, la aceleración
%   no se tiene en cuenta
   G1d = [dt^2/2; dt];
   Q1d = G1d*G1d';
   Q = blkdiag(Q1d, Q1d, Q1d, Q1d);
```

```
% Paso 3: Se define el modelo de medida
%   Solo la posición ([x;y;w;h]) es medida.
%   El modelo de medida es
   H = [1 0 0 0 0 0 0 0; ...
      0 0 1 0 0 0 0 0; ...
      0 0 0 0 1 0 0 0; ...
      0 0 0 0 0 0 1 0];

% Paso 4: Mapear las medidas del sensor a un vector de estado inicial. Debido a
% que no hay una medición de la velocidad, las componentes v se inicializan a 0:
   state = [Detection.Measurement(1); 0; Detection.Measurement(2); 0; ...
         Detection.Measurement(3); 0; Detection.Measurement(4); 0];

% Paso 5: Mapear el ruido de medición del sensor a una covarianza de estado. Para
% las partes del estado en que el sensor midió directamente, se usan las
% componentes de ruido de medición correspondientes. Para las partes que el
% sensor no mide, se supone una gran covarianza de estado inicial. De esa manera,
% futuras detecciones pueden ser asignadas al tracking.
   L = 100; % valor grande
   stateCov = diag([Detection.MeasurementNoise(1,1), L, ...
               Detection.MeasurementNoise(2,2), L, ...
               Detection.MeasurementNoise(3,3), L, ...
               Detection.MeasurementNoise(4,4), L]);

% Paso 6: Se crea el filtro correcto:
%   Se utiliza trackingKF como filtro para el tracking ya que se supone un modelo lineal
   filter = trackingKF(...
      'StateTransitionModel', A, ...
      'MeasurementModel', H, ...
      'State', state, ...
      'StateCovariance', stateCov, ...
      'MeasurementNoise', Detection.MeasurementNoise, ...
      'ProcessNoise', Q);
end

%%
% detectObjects es la función que detecta personas en una imagen.

function detections = detectObjects(detector, frame, frameCount)

   % Ejecuta el detector y obtinen una lista de bounding boxes: [x, y, w, h]
   bboxes = detect(detector, frame,'ExecutionEnvironment','gpu','Threshold',
0.95,'NumStrongestRegions',1000,'SelectStrongest',true);

   % Se define la medida de ruido
   L = 100;
   measurementNoise = [L 0  0  0; ...
               0 L  0  0; ...
               0 0 L/2 0; ...
               0 0  0 L/2];

   % Se agrupan las detecciones como una lista de informes de tipo ObjectDetection.
   numDetections = size(bboxes, 1);
   detections = cell(numDetections, 1);
```

```
    for i = 1:numDetections
        detections{i} = objectDetection(frameCount, bboxes(i, :), ...
            'MeasurementNoise', measurementNoise);
    end

end

%%
% removeNoisyTracks es una función que elimina las detecciones ruidosas.
% Se considera ruidosa si el bounding box detectado es demasiado pequeño.
% Típicamente, esto implica que la persona está lejos.

function tracks = removeNoisyTracks(tracks, positionSelector, imageSize)

    if isempty(tracks)
        return
    end

    % Extrae la posición de todos las detecciones
    positions = getTrackPositions(tracks, positionSelector);
    % La detección es invalida si la posición del bounding box detectado está
    % fuera de la imagen o es demasiado pequeña.
    invalid = ( positions(:, 1) < 1 | ...
            positions(:, 1) + positions(:, 3) > imageSize(2) | ...
            positions(:, 3) <= 20 | ...
            positions(:, 4) <= 20 );
    tracks(invalid) = [];

end

%%
% detectionToTrackCost es una función que calcula el costo para la asignación
% de detecciones. La asignación de detecciones de objetos al tracking actual
% se realiza minimizando el costo. El costo se calcula utilizando la función
% bboxOverlapRatio, y este costo es la relación de superposición entre el
% bounding box predicho y el detectado. Se supone que la persona se mueve
% gradualmente en cuadros consecutivos.
% Primero, se calcula el costo de asignar cada detección usando la medida
% bboxOverlapRatio. A medida que las personas se acercan o se alejan de la
% cámara, el centroide no puede describir con precisión su movimiento. El costo
% tiene en cuenta la distancia en el plano de la imagen y la escala de los
% bounding boxes. Esto evita asignar detecciones que están alejadas de la cámara
% a las que están más cerca de la cámara, incluso si los centroides predichos y
% detectados coinciden. La elección de esta función de costo facilita el cálculo
% sin tener que recurrir a un modelo dinámico más sofisticado. Los resultados se
% almacenan en una matriz M por N, donde M es el número de pistas y N es el
% número de detecciones.
% El valor del costo de no asignar una detección depende del rango de valores
% devueltos por la función de costo. Este valor debe ser sintonizado experimentalmente.
% Si se establece demasiado bajo, aumenta la posibilidad de crear un nuevo
% tracking. Si se configura demasiado alto, puede resultar en una que un mismo
% tracking se agrupen una serie de objetos en movimiento que realmente están separados.

function costMatrix = detectionToTrackCost(tracks, detections,positionSelector, threshold)
```

```
    if isempty(tracks) || isempty(detections)
        costMatrix = zeros(length(tracks), length(detections));
        return
    end

    % Calcula la relación de superposición entre los bounding box pronosticados y
    % los detectados, y calcula el costo de asignar cada detección. El costo es
    % mínimo cuando el bbox predicho está perfectamente alineado con el bbox
    % detectado (la relación de superposición es uno).

    % Recupera la posición de los bounding boxes.
    trackBboxes = getTrackPositions(tracks, positionSelector);
    % Comprueba que el ancho y el alto son valores positivos antes de calcular el
    % ratio de superposición
    trackBboxes(:, 3) = max(eps, trackBboxes(:, 3));
    trackBboxes(:, 4) = max(eps, trackBboxes(:, 4));

    % Se extraen todos los bounding box detectados.
    allDetections = [detections{:}];
    bboxes = reshape([allDetections(:).Measurement], 4, length(detections))';

    % se calculan todos los costos asignados por pares.
    costMatrix = 1 - bboxOverlapRatio(trackBboxes, bboxes);
    % Se establece el costo de la asignación no realista en Inf si hay una
    % pequeña superposición de bounding box.
    costMatrix(costMatrix(:) > threshold) = Inf;

end

%%
% detectedAction es una función que se encarga de reconocer la actividad de la
% persona que aparece en la secuencia de imagenes

function [position, FeatureStruct,FeaturePosition,rectangle] = detectedAction(Im,netFeature,netLstm,
Tracks,idx_TrackId, boundingBox, NumMaxSequences, position, FeatureStruct,FeaturePosition)

    % Se selecciona la capa utilizada para la extraccion de caracterisitcas
    layer='fc7';
    %layer='fc1000';
    % Se busca la posición en el array en la cual el identificador actual
    % coincida con el identificador del Track actual
    idx_position=find(position==idx_TrackId);
    % existe en la lista el trackID?

    if ~isempty(idx_position)
        % contador de frames de una secuencia
        FeaturePosition(idx_position)=FeaturePosition(idx_position) + 1;
        % Se llama a la funcion adjustBoundingBox que obtiene un bounding box
        % de tamaño fijo en función del bounding box detectado
        [Im,rectangle,Im2]=adjustBoundingBox(Im,boundingBox,idx_position,FeaturePosition,FeatureStruct);
        FeatureStruct(idx_position).rect=rectangle;

        % Se ejecuta la extracción de característcas
        FeatureStruct(idx_position).t(:,FeaturePosition(idx_position))=activations(netFeature,Im,layer);
```

```
      % Si ya hay en la pila x frames
     if FeaturePosition(idx_position) >= 25

        ImageTest{1}=FeatureStruct(idx_position).t;
        %se llama a la red lstm

        %sequenceLengthNet=151-FeaturePosition(idx_position);
        rng(0);
       %[FeatureStruct(idx_position).updateNet, action] =
    classifyAndUpdateState(FeatureStruct(idx_position).updateNet,ImageTest,'MiniBatchSize',1,
    'SequenceLength','shortest');
        %action=classify(netLstm, ImageTest, 'MiniBatchSize',1, 'SequenceLength','shortest');
        action=classify(netLstm, ImageTest);

        FeatureStruct(idx_position).action = cellstr(action);
        % FIFO: se elimina el primer frame para añadir el siguiente y
        % mantener una secuencia fija de x frames
        FeatureStruct(idx_position).t=FeatureStruct(idx_position).t(:,2:end);
        FeaturePosition(idx_position)=FeaturePosition(idx_position) - 1;

     else
        FeatureStruct(idx_position).action = {'not identified'};
     end

   else %nuevo ID
     idx_posFree=find(position==0); %queda alguna posición libre
     rectangle=[];

     if (~isempty(idx_posFree) && idx_posFree(1) <= NumMaxSequences) %si estan alguna o todas libres asigna
la primera
        FeaturePosition(idx_posFree(1))=1;
        position(idx_posFree(1))=idx_TrackId; %asigna en la posiicon el ID
        FeatureStruct(idx_posFree(1)).t=([]); %se inicializa pila de descriptores
        FeatureStruct(idx_posFree(1)).action = {'not identified'}; % se asigna acción inicial
        FeatureStruct(idx_posFree(1)).updateNet=netLstm; %se asigna red lstm apar la secuencia
        FeatureStruct(idx_posFree(1)).rect=[]; %se inicaliza variable bounding box
        %resetState(FeatureStruct(idx_posFree(1)).updateNet); % se resetea el estado de celda de la red lstm

        [Im, rectangle]=adjustBoundingBox(Im,boundingBox,idx_posFree(1),FeaturePosition,FeatureStruct);
        FeatureStruct(idx_posFree(1)).rect=rectangle;

        FeatureStruct(idx_posFree(1)).t(:,FeaturePosition(idx_posFree(1)))=activations(netFeature,Im,layer);

     else
        arrayTrackId=struct2cell(Tracks);
        arrayTrackId=arrayTrackId(1,:);
        arrayTrackId=cell2mat(arrayTrackId);

          for i=1:numel(position)
            pairs=find(arrayTrackId==position(i));
            if isempty(pairs)  %se comprueba que id viejos siguen estando
                position(i)=0; %la posicion se reinicia porque ya no hay ningun track con esa idx y se deja libre
            end
          end
      end
```

```
    end
end

%%
%Función adjustBoundingBox extrae del frame original el trozo de imagen en función del detector de personas
para su posterior extracción de características

function [I1,rect,I2]=adjustBoundingBox(I1,bboxes,position,featurePosition,featureStruct)

    % Se calcula la relacion de aspecto que debe mantener el recuadro obtenido en función de las imagenes
    % utilizadas en el entrenamiento
    %ratio=320/180; % Utilizar con secuencias GBA
    %ratio=540/300; % Utilizar con secuencias GBA2018
    ratio=160/120; % Utilizar con secuencias KTH
    %ratio=288/360; % Utilizar con secuencias Weizzman
    %ratio=640/360; % Utilizar con secuencias AVenue

    %bounding boxes: [x, y, w, h]
    x_centro=bboxes(1,1) + (bboxes(1,3)/2); % x_centro=xmin + widht/2;
    y_centro=bboxes(1,2) + (bboxes(1,4)/2); % y_centro=ymin + height/2;

    if featurePosition(position)==1 % Si es la primera detección se calculan todos los parametros [x, y, w, h]
        rect(1,4)=bboxes(1,4)*1.5; % height (alto imagen)=1*altura detector.
        rect(1,3)=bboxes(1,3)*2.2; % width (ancho imagen)=2*anchura detector
        rect(1,2)=y_centro - rect(1,4)/2; % ymin =y_centro - alto imagen / 2

        if rect(1,2) < 1 % si ymin es mayor que el valor de alto de la imagen original se limita valor.
            rect(1,2)=1;
        end

        if rect(1,4)+rect(1,2)>size(I1,1) % se verifica que no supere anchura de imagen original
            rect(1,4)=size(I1,1)-rect(1,2); % si supera se ajusta al máximo peromitido
        end

        % en base al ratio de la imagen y al valor del recuadro detectado (anchoxalto) se
        % obtiene la proporcion a aplicar
        proporcion=ratio/(rect(1,4)/rect(1,3)); % se calcula la proporcion que debe mantener el recuadro obtenido
        rect(1,4)=rect(1,4)*proporcion; % se aplica el coeficiente de proporcion solo a la altura
        rect(1,1)=x_centro-rect(1,3)/2; % xmin = x_centro-ancho imagen / 2

        if rect(1,1)<1 % si el valor de xmin es inferior a 1 se limita a 1
            rect(1,1)=1;
        end
        if rect(1,1)+rect(1,3)>size(I1,2)% si xmin+ancho ajustado es mayor que la anchura original se limita valor
            rect(1,3)=size(I1,2)-rect(1,1);
        end

else % si no se trata de una nueva detección, se mantienen los valores de altura y anchura iniciales
        rect(1,4)=featureStruct(position).rect(1,4); % mantiene valor anterior
        rect(1,3)=featureStruct(position).rect(1,3); % mantiene valor anterior
        rect(1,2)=y_centro - rect(1,4)/2; % ymin =y_centro - alto imagen / 2


        if rect(1,2) < 1 % si ymin es mayor que el valor de alto de la imagen original se limita valor.
            rect(1,2)=1;
        end
```

```
    if rect(1,4)+rect(1,2)>size(I1,1) % se verifica que no supere anchura de imagen original
        rect(1,4)=size(I1,1)-rect(1,2); % si supera se ajusta al máximo peromitido
    end

    rect(1,1)=x_centro-rect(1,3)/2; % xmin = x_centro-ancho imagen / 2

    if rect(1,1)<1 % si el valor de xmin es inferior a 1 se limita a 1
        rect(1,1)=1;
    end

    if rect(1,1)+rect(1,3)>size(I1,2)% si xmin+ancho ajustado es mayor que la anchura original se limita valor
        rect(1,3)=size(I1,2)-rect(1,1);
    end

  end

  I1=imcrop(I1,rect(1,:)); % se recorta imagen original con valores calculados para extraer secuencia
  %I2=imresize(I1, [320 180]); % Utilizar con GBA
  I2=I1;
  %I2=imresize(I1, [540 300]); % Utilizar con GBA2018
  %I2=imresize(I1, [240 320]); % Utilizar con KTH
  %I2=imresize(I1, [288 360]); % Utilizar con Weizzman
  I1=imresize(I1, [227 227]); % se ajusta para adaptar a la entrada de la capa encargada de obtener el
  % descriptor. En Alexnet 227x227 en Resnet50 224x224

end
```

Pliego de condiciones

A continuación, se detallan los elementos físicos y las herramientas software utilizadas durante el desarrollo de este proyecto.

9.1 *Equipos físicos*

- Ordenador portátil HP Omen con procesador Core i7-7700HQ @ 2.80GHz con 16Gb RAM y una GPU Nvidia GeForce GTX 1050 con 4Gb.

9.2 *Software*

- Sistema operativo Windows 10
- Matlab 2017a, 2017b y 2018a
- Parallel Computing Toolbox
- Driver CUDA 9.2
- Deep Learning Toolbox de Matlab
- Image Acquisition Toolbox de Matlab
- Procesador de textos (Microsoft Word)

Presupuesto

En esta parte del trabajo se incluye una estimación del coste total que supone la ejecución del mismo. En los apartados siguientes aparecen los gastos agrupados según su origen, y en el último apartado se detalla el presupuesto total.

10.1 *Recursos hardware*

CONCEPTO	PRECIO UNIT.	CANTIDAD	SUBTOTAL
Ordenador portátil HP Omen con procesador Core i7-7700HQ @ 2.80GHz con 16Gb RAM y una GPU Nvidia GeForce GTX 1050 con 4Gb.	1.200,00€	1	1.200,00€
		TOTAL	1.200,00€

10.2 *Recursos software*

CONCEPTO	PRECIO UNIT.	CANTIDAD	SUBTOTAL
Licencia perpetua Matlab 2018a	2.000,00€	1	2.000,00€
Licencia perpetua Parallel Computing Toolbox	1.000,00€	1	1.000,00€
Licencia perpetua Deep Learning Toolbox	1.150,00€	1	1.150,00€
Licencia perpetua Image Acquisition Toolbox	1.000,00€	1	1.000,00€
Microsoft Office Word 2010	60,00€	1	60,00€
		TOTAL	5.210,00€

10.3 *Coste de la mano de obra*

La realización de este proyecto ha sido llevada a cabo por las siguientes personas:

CONCEPTO	PRECIO UNIT.	CANTIDAD	SUBTOTAL
Ingeniero	50,00€	600	30.000,00€
		TOTAL	30.000,00€

10.4 *Presupuesto de ejecución material*

Es la suma total de los importes del coste de materiales y de la mano de obra.

CONCEPTO	PRECIO
Coste recursos hardware	1.200,00€
Coste recursos software	5.210,00€
Coste mano de obra	30.000,00€
TOTAL	36.410,00€

10.5 *Importe de la ejecución por contrata*

A continuación, se incluyen los gastos derivados del uso de las instalaciones donde se ha llevado a cabo el proyecto, cargas fiscales, gastos financieros, tasas administrativas y derivados de las obligaciones de control del proyecto. De igual forma se incluye el beneficio industrial. Para cubrir estos gastos se establece un recargo del 22% sobre el importe del presupuesto de ejecución material.

CONCEPTO	PRECIO
22% coste total de ejecución material	8.010,20€

10.6 *Honorarios facultativos*

Se ha fijado en este proyecto un porcentaje del 7% sobre el coste total de ejecución por contrata.

CONCEPTO	PRECIO
7% coste total de ejecución material	2.548,70€

10.7 *Presupuesto total*

CONCEPTO	PRECIO
Presupuesto de ejecución material	36.410,00€
Importe de la ejecución por contrata	8.010,20€
Honorarios facultativos	2.548,70€
TOTAL (sin IVA)	46.968,90€
IVA (21%)	9.863,47€
TOTAL	56.832,37€

El presupuesto total del proyecto asciende a la cantidad de cincuenta y seis mil ochocientos treinta y dos euros con treinta y siete céntimos.

Alcalá de Henares, a 17 de junio de 2019.

Firmado: Pedro López Miguel

Ingeniero Industrial

Referencias

[1] J. Macías Guarasa y M. Marrón Romera, Proyecto de Investigación: Detección semántica multisensorial de situaciones anómalas en entornos sin restricciones (HEIMDAL), Entidad financiadora: MINISTERIO DE ECONOMIA Y COMPETITIVIDAD: Referencia: TIN2016-75982-C2-1-R, 2016.

[2] «http://www.geintra-uah.org/,» Grupo de Ingeniería Electrónica Aplicada a Espacios Inteligentes y Transportes. [En línea].

[3] D. Xu, E. Ricci, Y. Yan, J. Song y N. Sebe, «Learning Deep Representations of Appearance and Motion for Anomalous Event Detection.,» *CoRR,* vol. abs/1510.01553, 2015.

[4] T. Xiao, C. Zhang y H. Zha, «Learning to Detect Anomalies in Surveillance Video.,» *IEEE Signal Process. Lett.,* vol. 22, nº 9, pp. 1477-1481, 2015.

[5] U. Ravale, N. Marathe y P. Padiya, «Feature Selection Based Hybrid Anomaly Intrusion Detection System Using K Means and RBF Kernel Function,» vol. 45, nº 428-435, 2015.

[6] S. Ren, K. He, R. B. Girshick y J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.,» de *NIPS*, 2015.

[7] A. Krizhevsky, I. Sutskever y G. E. Hinton, «Imagenet classification with deep convolutional neural networks,» de *Advances in neural information processing systems*, 2012.

[8] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell y K. Saenko, «Long-term recurrent convolutional networks for visual recognition and description.,» de *CVPR*, 2015.

[9] M. B. Ríos, Anomalous Behaviour Detection in Video Surveillance Scenes, Universidad de Alcalá: TFM Máster en Ingeniería Industrial, 2017.

[10] I. Laptev y B. Caputo, «Recognition of human actions Action Database,» 2005 Enero 2005. [En línea]. Available: http://www.nada.kth.se/cvap/actions/. [Último acceso: 8 Septiembre 2018].

[11] L. Gorelick, M. Blank, E. Shechtman, M. Irani y R. Basri, «Actions as Space-Time Shapes.,» *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 29, nº 12, pp. 2247-2253, 2007.

[12] C. Martínez, M. Baptista, C. Losada, M. Marrón y V. Boggian, «Human action recognition in realistic scenes base on Action Bank",,» *Grupo GEINTRA, Universidad de Alcalá,* 2016.

[13] P. Dollár, R. Appel, S. J. Belongie y P. Perona, «Fast Feature Pyramids for Object Detection.,» *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 36, pp. 1532-1545, 2014.

[14] C. Wojek, G. Dorkó, A. Schulz y B. Schiele, «Sliding-windows for rapid object class localization: A parallel technique,» *DAGM,* 2008.

[15] T. Tang, S. Zhou, Z. Deng, H. Zou y L. Lei, «Vehicle Detection in Aerial Images Based on Region Convolutional Neural Networks and Hard Negative Example Mining.,» *Sensors,* vol. 17, nº 2, p. 336, 2017.

[16] V. Chandola, A. Banerjee y V. Kumar, «Anomaly detection: A survey,» *ACM Computing Surveys (CSUR) 41 , no. 3: 15,* 2009.

[17] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg y F.-F. Li, «ImageNet Large Scale Visual Recognition Challenge.,» *CoRR,* vol. abs/1409.0575, 2014.

[18] R. B. Girshick, J. Donahue, T. Darrell y J. Malik, «Rich feature hierarchies for accurate object detection and semantic segmentation.,» *CoRR,* vol. abs/1311.2524, 2013.

[19] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers y A. W. M. Smeulders, «Segmentation as selective search for object recognition.,» de *ICCV*, 2011.

[20] D. Lowe, «Distinctive image features from scale-invariant keypoints,» de *International Journal of Computer Vision*, 2003.

[21] R. Girshick, *Fast R-CNN,* 2015.

[22] G. Dishashree, «https://www.analyticsvidhya.com,» 7 Diciembre 2017. [En línea]. Available: https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/. [Último acceso: 28 Julio 2018].

[23] C. Olah, «http://colah.github.io/,» 27 Agosto 2015. [En línea]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Último acceso: 28 Julio 2018].

[24] Y. Lu y F. M. Salem, «Simplified Gating in Long Short-term Memory (LSTM) Recurrent Neural Networks.,» *CoRR,* vol. abs/1701.03441, 2017.

[25] S. Hochreither y J. Schmidhuber, «Long Short-Term Memory,» 1997.

[26] M. Schuster y K. K. Paliwal, «Bidirectional recurrent neural networks,» *IEEE Transactions on Signal Processing,* vol. 45, pp. 2673-2681, November 1997.

[27] Z. Cui, R. Ke y Y. Wang, «Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction.,» *CoRR,* vol. abs/1801.02143, 2018.

[28] F. S. Caparrini, «Entrenamiento de Redes Neuronales: mejorando el Gradiente Descendiente,» 23 Abril 2017. [En línea]. Available: http://www.cs.us.es/~fsancho/?e=165. [Último acceso: 8 Septiembre 2018].

[29] B. Yao, X. Jiang, A. Khosla, A. L. Lin, L. J. Guibas y F.-F. Li, «Human action recognition by learning bases of action attributes and parts.,» de *ICCV*, 2011.

[30] V. Boggian Arévalo, Diseño, generación y anotación de base de datos de secuencias de imágenes en interiores para aplicaciones de video-vigilancia, Universidad de Alcalá. TFG: Grado en Ingeniería Electrónica y Automática Industrial, 2016.

[31] «The PASCAL Visual Object Classes Homepage,» 2005-2012. [En línea]. Available: http://host.robots.ox.ac.uk/pascal/VOC/.

[32] «Common Objects in Context,» 2018. [En línea]. Available: https://cocodataset.org/#home.

[33] S. V. Lab, «ImageNet,» Stanford University, Princeton University, 2016. [En línea]. Available: http://www.image-net.org/.

[34] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition.,» *CoRR,* vol. abs/1512.03385, 2015.

[35] «ImageNet Large Scale Visual Recognition Challenge 2015 (ILSVRC2015),» 2015. [En línea]. Available: http://image-net.org/challenges/LSVRC/2015/index#cite.

[36] S. Sudhakaran y O. Lanz, «Learning to Detect Violent Videos using Convolutional Long

Short-Term Memory.,» *CoRR,* vol. abs/1709.06531, 2017.

[37] D. Navneet, «INRIA Person Dataset,» 17 Julio 2006. [En línea]. Available: http://pascal.inrialpes.fr/data/human/.

[38] G. Welch, G. Bishop y C. Hill, «{An introduction to the Kalman filter},» pp. 1-16, 1995.

[39] F. Chollet, «https://keras.io/,» Keras, 23 Marzo 2015. [En línea]. [Último acceso: 29 Enero 2019].

[40] Google, «https://www.tensorflow.org/,» 9 Noviembre 2015. [En línea]. [Último acceso: 29 Enero 2019].

[41] L. L. Université de Montréal, «http://deeplearning.net/software/theano/,» 15 Noviembre 2017. [En línea]. [Último acceso: 29 Enero 2019].

[42] Microsoft, «https://www.microsoft.com/en-us/cognitive-toolkit/,» 25 Enero 2016. [En línea]. [Último acceso: 29 Enero 2019].

[43] Y. Bengio, I. Goodfellow y A. Courville, «Deep Learning,» 2016.

-

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR