ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DE COMPUTADORES

SISTEMA DE DETECCIÓN DE PRESENCIA BASADO EN MICROCONTROLADOR PARA CÁMARAS DE MOVIMIENTO PANORÁMICO UTILIZANDO REDES NEURONALES ARTIFICIALES

MICROCONTROLLER-BASED PRESENCE DETECTION SYSTEM FOR PANNING CAMERAS USING ARTIFICIAL NEURAL NETWORKS

Realizado por
JOSE JESÚS DE BENITO PICAZO
Tutorizado por
DR. D. ENRIQUE DOMÍNGUEZ MERINO
DR. D. ESTEBAN JOSÉ PALOMO FERRER
Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Julio 2017

Fecha Defensa:
El Secretario del Tribunal

Resumen: La detección de movimiento es el primer proceso esencial en la extracción de información de imágenes con objetos que se desplazan sobre un fondo estático. A la hora de realizar tareas de detección de movimiento, las técnicas basadas en diferencias con el fondo son las más utilizadas debido a la alta calidad conseguida en los procesos de segmentación. No obstante, los requisitos de tiempo real y los altos costes computacionales hacen muy complicado para la mayoría de los algoritmos propuestos en la literatura existente el aprovechamiento de la diferencia con el fondo a la hora de utilizar dichos algoritmos en aplicaciones del mundo real.

En este trabajo se presenta un nuevo algoritmo basado en redes neuronales artificiales que tiene como objetivo la detección de objetos en movimiento dentro de una escena tomada con cámaras de movimiento panorámico. Asimismo, tanto los requerimientos de memoria como el coste computacional del algoritmo de detección de movimiento se han optimizado para favorecer su despliegue en un microcontrolador modelo Broadcom BCM2837 montado en una placa Raspberry Pi, posibilitando el diseño e implementación de sistemas de monitorización y video-vigilancia de bajo coste.

Palabras clave: Detección de objetos en primer plano, modelado del fondo, mapas probabilísticos autoorganizados, características del fondo.

Abstract: Motion detection is the first essential process in the extraction of information regarding moving objects. The approaches based on background difference are the most used with fixed cameras to perform motion detection, because of the high quality of the achieved segmentation. However, real time requirements and high costs prevent most of the algorithms proposed in literature to exploit the background difference with panning cameras in real world applications. In this work, a new artificial neural network-based algorithm, to detect moving objects within a scene acquired by panning cameras, is presented. In order to reduce both memory requirements and power computing costs, the algorithm for motion detection presented here has been optimized to support its deployment on a Broadcom BCM2837 microcontroller embedded in a Raspberry Pi development board, which enables the design and implementation of a low-cost monitoring system.

Keywords: Foreground detection, background modeling, probabilistic self-organizing maps, background features.

Índice general

Capítulo 1

Introducción

En un mundo altamente globalizado, donde la movilidad de los individuos es la tónica general y donde la conducta de un solo individuo puede afectar a una gran cantidad de personas, las tareas de vigilancia se han convertido en una prioridad para gobiernos e instituciones.

Figura 1.1: Cámara de videovigilancia situada en entorno aeroportuario

Una de las actividades principales que se engloba en dichas tareas consiste en el control de la actividad que se produce dentro de un entorno físico limitado, mediante el análisis de las imágenes procedentes de una o varias cámaras de vídeo que se sitúan en dicho entorno. Actualmente, la forma más extendida de realizar esta tarea consiste en la asignación de la misma a una persona que se encargará de ella. El problema es que cualquier tarea asignada a un ser humano, especialmente si se trata de algo monótono que se prolonga durante una cantidad considerable de tiempo, está sujeta a

errores producidos tanto por distracciones del operario como al cansancio del mismo. Estas circunstancias motivan la automatización, en la medida de lo posible, de las tareas de videovigilancia mediante sistemas de detección de presencia automáticos en tiempo real, rápidos, baratos y con una precisión razonable.

A la hora de afrontar el diseño de un sistema de videovigilancia automatizado, la detección de objetos en movimiento dentro de una escena se convierte en una de las principales tareas a realizar. Dicha tarea es bien conocida en el ámbito investigador por constituir un problema de difícil resolución en muchos entornos del mundo real. y consistirá en advertir un cambio en la posición de un objeto en relación con su entorno, o un cambio en la posición del entorno en relación con determinado objeto.

Los sistemas de videovigilancia se han convertido en un importante campo de estudio debido a los niveles cada vez más altos de conflictos sociales que han desembocado en una mayor preocupación de los ciudadanos en materia de seguridad. Lo cual ha motivado el desarrollo de sistemas automáticos de videovigilancia más robustos y precisos que se han convertido en importantes herramientas para la seguridad tanto en el ámbito público como en el privado.

Uno de los algoritmos más comunes a la hora de afrontar esta tarea consiste en comparar el frame o fotograma actual obtenido con algún dispositivo de captura de imágenes con el frame previo. Si la diferencia de píxeles es mayor que un umbral predefinido, se genera un aviso de movimiento detectado. El fondo estimado es sencillamente el fotograma anterior. Este sistema presenta un buen rendimiento en imágenes donde los objetos en primer plano están claramente diferenciados y bajo condiciones favorables de velocidad de movimiento y tasa de fotogramas por segundo, pero es muy sensible a los umbrales de detección por lo cual en una imagen con una tasa de ruido alta, se detectará movimiento en muchas zonas de la imagen incluso si en la realidad no hay movimiento en absoluto. Por el contrario, si el objeto se mueve suavemente, se obtendrán pequeños cambios entre fotogramas que serán menores que el umbral establecido, con lo que el movimiento del objeto no será detectado. Las razones expuestas anteriormente son las que motivan el desarrollo de algoritmos más avanzados de detección de movimiento capaces de detectar objetos cuyo movimiento sea suave y que a la vez sean capaces de evitar los falsos positivos producidos por una umbralización demasiado restrictiva.

Por otra parte, los microcontroladores son dispoitivos versátiles, pequeños y económicos que se utilizan frecuentemente en sistemas de detección de movimiento debido a su bajo consumo energético y su coste reducido. Además de las ya conocidas, cada día surgen nuevas importantes aplicaciones para sistemas basados en microcontroladores. Un ejemplo son los interfaces de comunicación Hombre-Máquina especialmente adaptados a personas físicamente impedidas [1]. Otro ejemplo es

el que constituye el sistema de estimación de movimiento y proximidad basado en el diseño de un prototipo de Placa de Circuito Impreso (PCB) que incorpora un microcontrolador [2] el cual recibe la información proporcionada por ocho fotodiodos que actúan como sensores de luminosidad.

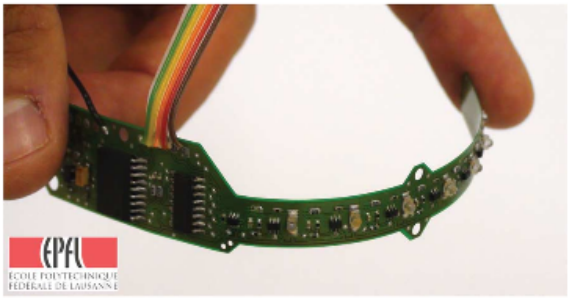

Figura 1.2: Sistema de visión y proximidad basado en fotodiodos.

La eficiencia de las plantas de energía solar también puede ser mejorada con la incorporación de pequeños sistemas de bajo consumo eléctrico que son capaces de estimar el movimiento de las nubes [3]. En dicho sistema, un microcontrolador se encarga de aproximar los vectores de movimiento de las nubes de forma que la colocación de los paneles solares puede ser optimizada para maximizar la producción de electricidad de dichos paneles. En la misma línea, [4] muestra como se puede implementar un sistema de alumbrado que ahorre energía dentro del contexto de las "Smart Cities" mediante detectores de movimiento de bajo consumo eléctrico equipados con microcontroladores de bajo coste y sistemas de comunicación inalambricos. De esta forma, las farolas solamente se encenderán cuando haya alguna persona en sus inmediaciones. Finalmente, en [5] se ilustra el desarrollo de un sistema de detección de movimiento basado en Mapas Autoorganizados (SOMs) utilizando un microcontrolador del tipo Arduino DUE. En dicho trabajo, la implementación del algoritmo basado en SOMs se emplea como un detector de movimiento para cámaras estáticas en un sistema automático de video-vigilancia (figura 1.3).

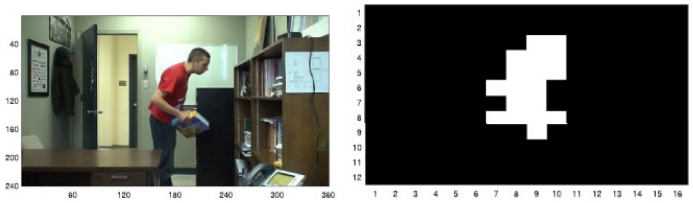

Figura 1.3: Sistema de video-vigilancia basado en microcontrolador.

Propio de la terminología de los sistemas de seguridad que comprenden tareas de video-vigilancia, el concepto "Cámara PTZ" (*Pan-Tilt-Zoom Camera*) hace alusión a un tipo especial de cámara robotizada normalmente teleoperada, capaz de rotar sobre sí misma en la dirección del eje horizontal (Movimiento panorámico o *Pan*) en la dirección del eje vertical (*Tilt*) y también es capaz de acercar o alejar la imagen que se está encuadrando en determinado momento (*Zoom*). Durante muchos años, los sistemas de visión por computador basados en cámaras PTZ han constituido un importante objeto de estudio para los investigadores [6, 7, 8, 9]. Sin embargo existe una falta de comprensión teórica que pueda establecer los cimientos para el desarrollo práctico de sistemas de este tipo. Existen trabajos que se limitan a resolver algunas partes del problema, pero por sí solos no ofrecen una alternativa clara que indique cómo combinarlos para obtener sistemas completos y fiables que puedan ser utilizados en distintas situaciones y contextos. En nuestro proyecto, nos centraremos en el movimiento panorámico de una cámara PTZ, que será capaz de abarcar todo el entorno de dicha cámara.

1.1. Objetivo del proyecto

El objetivo del presente trabajo será el diseño de un algoritmo de detección de movimiento y su implantación para funcionamiento en tiempo real en un microcontrolador. Dicho algoritmo será capaz, con la ayuda de una red neuronal artificial, de analizar una imagen que proviene de una cámara PTZ que vendrá realizando un movimiento panorámico a velocidad constante y será optimizado para su implantación en un sistema SoC (System-on-Chip) de bajo precio y fácil disponibilidad.

La estructura del presente documento será la siguiente:

- El capítulo 2 corresponderá a la parte hardware del sistema, basada en un microcontrolador Raspberry PI modelo 3 B y la arquitectura software empleada.
- En el capítulo 3, se presenta el algoritmo de detección de movimiento junto con la base matemática en la que se apoya.
- En el capítulo 4 se expondrán los resultados obtenidos a partir de los experimentos realizados con material de vídeo real.
- Por último, el capítulo 5 corresponderá a la parte de conclusiones obtenidas y posibles desarrollos futuros basados en el presente diseño.

Capítulo 2

Arquitectura del sistema

Cuando se trata de aplicaciones de visión por computador que van a ser implantadas en un microcontrolador, la elección del hardware es de una importancia crítica. En general, cualquier tipo de sistema que implique algoritmos de visión por computador necesitará una cantidad mínima de potencia computacional sin la cual no podrá desempeñar su tarea. Esto se hace más patente en sistemas de videovigilancia que deben realizar detección de movimiento en tiempo real. Al mismo tiempo, para realizar la monitorización de un determinado entorno físico se necesitará un número de dispositivos que irá acorde tanto con el área a monitorizar como a la distribución física de la misma. Además es fácil que los puntos del entorno donde se vayan a colocar los dispositivos de videovigilancia no tengan fácil acceso a la red eléctrica, así que será necesario utilizar un hardware que permita implementar un sistema barato y que tenga un bajo consumo energético.

Todas estas razones apoyan la elección de microcontroladores para la implantación de nuestro software. El sistema elegido debe ser barato, tener un bajo consumo energético y además debe contar con un mínimo de potencia computacional para que nuestro sistema pueda funcionar correctamente y a una velocidad que le permita operar en tiempo real.

A continuación, se presentan una serie de alternativas que se han considerado de cara a la elección del hardware para la implantación del sistema.

2.1. Elección del microcontrolador a utilizar

De forma general, llamamos microcontrolador a un ordenador pequeño que cuyas unidades funcionales básicas están incluidas en un sólo circuito integrado. Además, disponen generalmente también de una gran variedad de dispositivos de entrada/salida,

como convertidores analógico digital, temporizadores, UARTs y buses de interfaz serie especializados, como I2C y CAN que le confieren la capacidad de interactuar con el medio físico lo que los convierte en dispositivos de una gran versatilidad que los hacen susceptibles de ser utilizados en gran cantidad de aplicaciones. Así, representan la inmensa mayoría de los chips de ordenadores vendidos y pueden encontrarse en casi cualquier dispositivo electrónico o que tenga partes electrónicas; tales como automóviles, lavadoras, hornos microondas, teléfonos, etc. Por regla general, son sistemas diseñados para reducir el coste económico y el consumo de energía de un sistema en particular. Por eso, el tamaño de la unidad central de procesamiento y sus características, la cantidad de memoria y los periféricos incluidos dependerán de la aplicación concreta para la que se utilicen.

En nuestro caso concreto el microcontrolador se va a utilizar en una tarea de gran peso computacional como es el procesamiento de imágenes, sin embargo, no será necesario generar ni interpretar señales externas a parte de la enviada por una cámara de vídeo, por lo que para la elección de nuestro dispositivo daremos mayor importancia a su velocidad de procesamiento y la cantidad de información que sea capaz de manejar simultáneamente que a la cantidad de interfaces de entrada/salida y la naturaleza de las mismas.

Dada la gran variedad de microcontroladores existente en el mercado, es necesario ajustar nuestra búsqueda a los sistemas que presenten características adecuadas a los requerimientos de nuestro problema. Además, necesitamos que el microcontrolador vaya instalado en una placa de desarrollo que permita utilizarlo desde el primer momento. Para ello se ha realizado un estudio que nos deja una serie de posibles alternativas cuyas características más importantes se indican brevemente a continuación:

- Raspberry Pi 3 Model B.

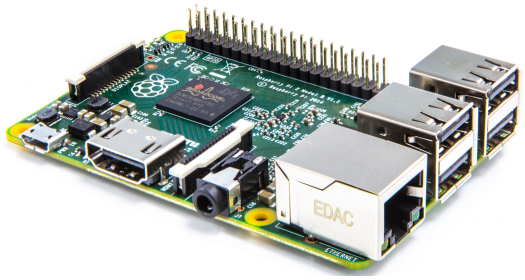

Figura 2.1: Placa de desarrollo Raspberry Pi.

Es una placa de desarrollo que consta de las siguientes características técnicas:

- SoC: Broadcom BCM2837
- CPU Quad-core ARM Cortex-V8. 1.2 Ghz
- 1GB de RAM
- 40 GPIO Pins
- Precio: 35 euros

- ODROID C1+.

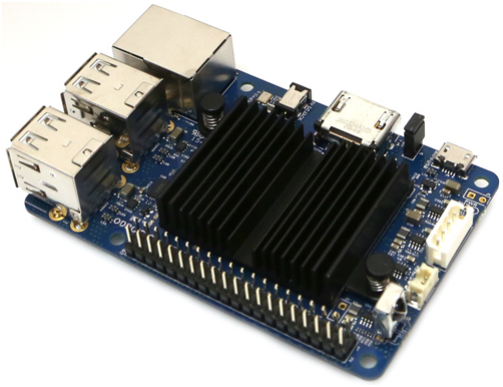

Figura 2.2: Placa de desarrollo ODROID c1+.

La placa de desarrollo ODROID C1+ desarrollada por la empresa Hardkernel es la más parecida por prestaciones a la Raspberry Pi 3 Model B. Aunque es un poco más potente, también tiene un coste superior. Sus características técnicas son las siguientes:

- SoC: Amlogic S805
- CPU Quad-core ARM Cortex-A5. 1.5 Ghz
- 1GB de RAM
- 40 GPIO Pins
- Precio: 45 euros

- Banana Pi M3

  Banana Pi M3 es la placa de desarrollo más potente que se ha considerado como candidata a su utilización en el presente proyecto. También tiene un precio mayor que el resto de las candidatas.

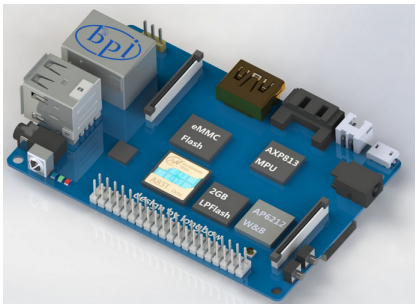

Figura 2.3: Placa de desarrollo Banana Pi.

- SoC: Allwinner A83T
- ARM Cortex-A7 octa-core. 2.0 Ghz
- 2GB de RAM
- 40 GPIO Pins
- Precio: 80 euros

- Orange Pi.

Figura 2.4: Placa de desarrollo Orange Pi.

Orange Pi es una placa de desarrollo con especificaciones parecidas a Banana Pi M3 salvo porque monta un procesador Quad-Core mientras que Banana Pi M3 monta un procesador Octa-Core. El precio de ésta es, no obstante, más económico.

- Allwinner H3
- Quad-core Cortex-A7. 2.0 Ghz
- 2GB de RAM

- 40 GPIO Pins
- Precio: 66 euros

Dado que el SOC Broadcom BCM2837 presenta características que incluyen una gran velocidad de procesamiento, un consumo bajo, alta disponibilidad y accesibilidad del chip al ir montado en la placa de desarrollo Raspberry Pi 3 Modelo B y la gran cantidad de información para su uso que existe en la red, además de ser el dispositivo con una mejor relación calidad-precio, se ha elegido dicha placa de desarrollo para el despliegue del algoritmo que se desarrolla en el presente trabajo.

2.2. Dispositivo de captura de imágenes

El seguimiento de objetos en primer plano con cámaras PTZ ha venido siendo objeto de estudio en visión por computador durante muchos años. Llamamos cámara PTZ a un tipo de cámara robotizada que es capaz de realizar movimiento en la dirección horizontal, vertical y movimiento de acercamiento a la escena (movimientos Pan Tilt y Zoom o PTZ). No obstante, las experiencias realizadas por investigadores, demuestran que es muy complicado realizar una evaluación de los progresos que se han venido realizando por que no existe una metodología estandarizada de evaluación. La dificultad a la hora de evaluar procesos que incluyan cámaras PTZ, como en cualquier sistema automático que implique dispositivos mecánicos, reside en su naturaleza intrínsecamente dinámica. En contraste con otros métodos de seguimiento, el seguimiento realizado con cámaras PTZ implica la localización del objetivo en la imagen y el control de los motores de la cámara para apuntar el objetivo de forma que dicho objetivo permanezca correctamente encuadrado. Este tipo de seguimiento sólo puede realizarse online de forma que es muy complicado que comparar el rendimiento de dos algoritmos que utilizan la misma cámara PTZ real ya que el experimento no va a ser repetible en las mismas condiciones. Incluso en un escenario bien controlado en el cual los "actores" tengan asignadas sus acciones de forma muy minuciosa, las condiciones de seguimiento nunca van a ser totalmente idénticas.

El principal problema con los trabajos realizados hasta ahora con seguimiento utilizando cámaras PTZ reales es que sólo son capaces de evaluar la localización del objetivo en un sólo fotograma. Aunque importante, no tiene en cuenta la restricción del seguimiento en línea con una cámara PTZ. Por ejemplo, en un escenario con una cámara PTZ, si un algoritmo procesa un fotograma por segundo, el sistema permanecerá ciego durante dicho tiempo, tiempo durante el cual la cámara PTZ ha continuado su movimiento programado lo cual puede llevar a que exista una gran

distancia entre el escenario capturado por la cámara en un fotograma y el siguiente. Es más, centrar la cámara en su localización previa puede acabar produciendo que el objetivo abandone el campo de visión de la propia cámara o *FOV*. En general, los algoritmos de seguimiento utilizando cámaras PTZ que se han realizado hasta ahora, siguen dos estrategias bien diferenciadas:

- Como primera alternativa, están aquellas técnicas que se apoyan en un algoritmo de seguimiento rápido capaz de procesar todos los fotogramas sin descartar ninguno y así siempre centrar la cámara en la posición previa del objetivo (que será una buena aproximación a la siguiente posición del mismo, ya que la tasa de procesamiento de fotogramas por segundo es alta). Esta técnica, no obstante, puede verse abocada a realizar una pobre localización del objetivo.

- La segunda alternativa se apoya en algoritmos de seguimiento más lentos pero más sofisticados que pueden localizar el objetivo de forma precisa. Como el hecho de ser lento implica que el sistema va a estar ciego por períodos más largos de tiempo, para realizar el seguimiento de los objetivos más rápidos, la opción más conveniente conlleva el diseño de un algoritmo específico para realizar el control de la cámara. Una aproximación típica al problema consiste en determinar la localización más probable del objetivo en el fotograma siguiente, y centrar el campo de visión de la cámara en dicha localización.

En resumen, el tiempo de procesamiento será muy importante en los sistemas de seguimiento que incorporen cámaras PTZ porque su naturaleza online y la lentitud de procesamiento implica observaciones perdidas lo que podría ser un problema crítico para obtener resultados precisos.

Por estas razones, el segundo componente de la arquitectura de nuestro sistema es un framework emulador de cámara PTZ llamado *Virtual PTZ* [10]. Dicho software consta de 3 componentes:

- Una librería escrita en C++ que simula el comportamiento principal de una cámara PTZ.

- Una colección de vídeos esféricos panorámicos para escenarios diferentes.

- Las secuencias correspondientes al ground-truth de dichos vídeos con sus correspondientes anotaciones.

El simulador de cámara PTZ recibe imágenes panorámicas desde un archivo de vídeo, construye el modelo del escenario y provee un *frustum* típico de visualización para el algoritmo de seguimiento basado en los parámetros de la cámara. El evaluador utiliza

datos del ground-truth básico y los mismos parámetros de la cámara para generar bounding boxes del ground-truth para *FOV* actual y las compara con el seguimiento del algoritmo que se está probando.

El framework está diseñado basándose en vídeos capturados por una cámara esférica modelo *Point Grey Ladybug 3* y la librería *OpenGL* utilizada para proyectar vídeos en una esfera.

Figura 2.5: Cámara *Point Grey Ladybug 3*

La cámara Ladybug es capaz de proporcionar una vista casi esférica de 360º de la escena que puede ser mapeada en en dicha superficie mediante OpenGL. Lo cual ha hecho posible el diseño de una cámara virtual que sea capaz de observar porciones de dicha esfera. De lo cual se obtiene una cámara PTZ virtual que puede ser controlada como se desee para realizar el seguimiento de objetos en vídeos pregrabados. Por conveniencia, el centro del modelo esférico se sitúa en el origen de coordenadas del mundo proporcionado por OpenGL.

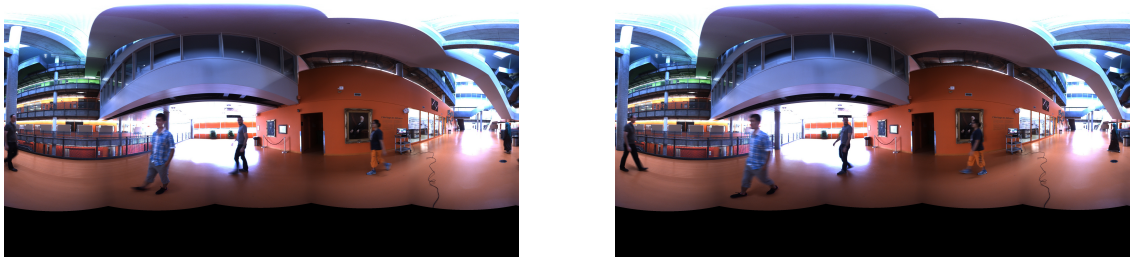

Figura 2.6: Imágenes esféricas de 360° proporcionadas por la *Point Grey Ladybug 3 Spherical camera*.

En resumen, para el propósito del presente proyecto se ha propuesto el citado framework *Virtual PTZ* por su capacidad para sustituir una cámara PTZ real proporcionando al usuario la posibilidad de mover el encuadre de la cámara virtual que incorpora de forma totalmente libre y con control absoluto por todo el fotograma

de 360 grados sin los inconvenientes dinámicos ni las limitaciones físicas que puede tener una cámara PTZ real. Como la única salida que nuestro sistema necesita de la cámara PTZ es un flujo de fotogramas de vídeo desde una cámara con capacidad de movimiento panorámico, el software *Virtual PTZ* se consolida como un sustituto válido de la propia cámara PTZ para nuestro trabajo.

2.3. Elección de la red neuronal

A grandes rasgos, nuestro sistema de detección de movimiento puede ser catalogado como un clasificador que consiste en un algoritmo que se encarga de obtener muestras de entrenamiento de un conjunto de imágenes consecutivas y proporcionárselas como entradas a una red neuronal artificial que será quien decida si hay o no objetos en primer plano que están en movimiento en un flujo de fotogramas de vídeo proporcionados por una cámara PTZ [1].

Por su adecuación a la mecánica de nuestro problema, su simplicidad de uso y su rápida respuesta, se ha decidido elegir el perceptrón multicapa como modelo de red neuronal que utilizará nuestro sistema.

2.3.1. El perceptrón multicapa

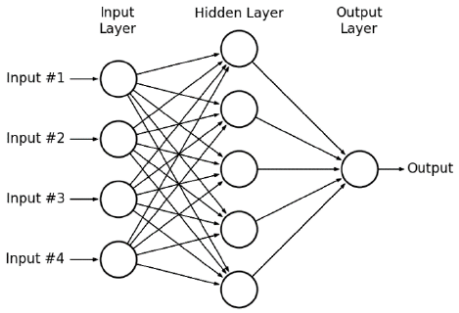

Figura 2.7: Arquitectura básica de un perceptrón multicapa

Llamamos perceptrón multicapa o MLP (Multilayer Perceptron) a una red neuronal artificial no cíclica que está formada por diferentes capas de neuronas conectadas en un grafo dirigido, donde cada capa está conectada completamente con la capa siguiente. Excepto los nodos de la capa de entrada, cada nodo es una neurona con

[1] O, en nuestro caso, un software capaz de simular la propia cámara PTZ

una función de activación no lineal. A la hora de entrenar la red, por regla general, el perceptrón multicapa utiliza una técnica de entrenamiento supervisado llamada Retropropagación. El perceptrón multicapa es una modificación del perceptrón estándar y es capaz de clasificar datos que no son linealmente separables.

Función de activación

Si un MLP tuviera una función de activación lineal en todas las neuronas, es decir, una función lineal que mapea las entradas con sus distintos pesos hacia la salida de cada neurona, entonces está demostrado que cualquier número de capas puede ser reducido al modelo de perceptrón estándar de dos capas entrada-salida.

Lo que diferencia al perceptrón multicapa es que algunas de las neuronas utilizan funciones no lineales de activación que fueron desarrolladas para modelar la frecuencia de los potenciales de acción, o disparo de las neuronas biológicas en el cerebro. Esta función se puede modelar de diversas formas y las dos principales funciones de activación que se usan en la actualidad están descritas por las siguientes expresiones:

$$y(v_i) = tanh(v_i) \tag{2.1}$$

$$y(v_i) = (1 + e^{-v_i})^{-1} \tag{2.2}$$

En la primera expresión, la función de activación es una sigmoide llamada tangente hiperbólica cuyo rango va desde -1 hasta 1. La segunda expresión es otra sigmoide conocida como función logística, tiene una forma parecida pero su rango va desde 0 hasta 1. En las expresiones mencionadas y_i corresponde a la salida de la neurona *i-ésima* y v_i es la suma ponderada de las sinapsis de entrada.

Capas

Como ya se dijo anteriormente, el perceptrón multicapa está compuesto por tres o más capas (una de entrada, una de salida y una o mas capas ocultas de neuronas) de nodos que se activan con funciones no lineales y está considerada por tanto como una *Deep Neural Network* o red neuronal de aprendizaje profundo. Dado que el MLP es una Red Totalmente Conectada, cada uno de los nodos de cada capa está conectado mediante un determinado peso w_{ij} a todos los nodos de la capa siguiente.

Algoritmo de Retropropagación

El aprendizaje se da en el perceptrón cambiando los pesos de las conexiones entre neuronas después de que cada muestra de entrada se ha procesado basándose en el

valor del error cometido a la salida comparado con el resultado esperado. Este es un ejemplo de aprendizaje supervisado y se lleva a cabo mediante retropropagación, una generalización del algoritmo de la media de los mínimos cuadrados o LMS (Least Mean Squares), que se utiliza en el perceptrón lineal.

Aplicaciones

Los perceptrones multicapa que utilizan retropropagación son algoritmos que se vienen utilizando de forma muy frecuente en procesos de reconocimiento de patrones mediante aprendizaje supervisado y constituyen un sujeto de estudio en neurociencia computacional y procesamiento paralelo distribuido. Han demostrado tener gran utilidad gracias a su habilidad para resolver problemas de manera estocástica, lo que frecuentemente permite aproximar soluciones a problemas muy complejos.

Según demuestra el teorema de Cybenko, los perceptrones multicapa son aproximadores universales de funciones, así que pueden ser utilizador para la creación de modelos de regresión. Debido a que la clasificación es un problema particular de regresión donde la respuesta es categorizada, los perceptrones multicapa también son clasificadores fiables.

El perceptrón multicapa fue una solución popular al problema del aprendizaje computacional en la década de los años 80 del pasado siglo, encontrándosele aplicaciones en diversos campos tales como el reconocimiento del lenguaje y el reconocimiento de imágenes. No obstante, desde los años 90 han venido soportando una competencia fuerte de las Máquinas Vectoriales de Soporte o *SMV*. Hoy en día, gracias al crecimiento de la capacidad computacional de los ordenadores en la actualidad, se han consolidado como una herramienta que goza de una increíble cantidad de aplicaciones y gran popularidad.

El diseño e implementación *ex profeso* de un perceptrón multicapa es algo que se sale de los objetivos del presente trabajo por lo que se ha decidido utilizar alguna de las implementaciones que ya existen en forma de librerías *Open Source*.

Después de realizar un estudio sobre las distintas implementaciones de redes neuronales de tipo perceptrón multicapa que se están utilizando, en el cual se han tenido en cuenta distintos factores tales como la eficiencia, el espacio ocupado en memoria la versatilidad de la red y la cantidad de documentación existente, se ha convenido en utilizar la Red Neuronal Artificial Rápida (Fast Artificial Neural Network o FANN) diseñada por Nissen [11], una implementación del perceptrón multicapa muy utilizada que además goza de total vigencia y actualizaciones periódicas.

Capítulo 3

Diseño e implementación del algoritmo de detección

Como ya se ha mencionado, el objetivo de nuestro trabajo es diseñar un sistema que sea capaz de detectar objetos en primer plano que estén en movimiento con respecto al entorno en las imágenes proporcionadas por una cámara PTZ que a su vez esté ejecutando un movimiento panorámico a velocidad constante.

En el presente capítulo se realizará una descripción matemática que formalice el algoritmo utilizado para posteriormente continuar con una descripción práctica de la dinámica de dicho algoritmo.

3.1. Formalización matemática del algoritmo

Consideremos la imagen tomada por la cámara como una función que responde a la siguiente formulación:

$$f \ : \ \mathbb{R}^3 \to \mathbb{R}^3 \tag{3.1}$$

$$f\left(x_1, x_2, t\right) = \left(y_1, y_2, y_3\right) \tag{3.2}$$

donde $\mathbf{x} = (x_1, x_2) \in [-A, A] \times [-B, B]$ son las coordenadas de vídeo en píxeles dentro del fotograma considerando $(0, 0)$ como el centro de la imagen y un tamaño de imagen de $(2A) \times (2B)$ píxeles; t es el instante de tiempo; e $\mathbf{y} = (y_1, y_2, y_3)$ comprende el valor del conjunto de valores RGB para un determinado píxel en el lugar del fotograma y el instante de tiempo dados.

Para una cámara PTZ moviéndose horizontalmente se cumple que:

$$(x_1, x_2), (x_1 + \delta, x_2) \in [-A, A] \times [-B, B] \Rightarrow$$

$$f(x_1, x_2, t) \approx f(x_1 + \delta, x_2, t + \epsilon) \tag{3.3}$$

Donde δ es el desplazamiento horizontal observado de la imagen transcurridas ϵ unidades de tiempo y la igualdad no se mantiene debido a efectos ópticos tales como la aberración en la lente de la cámara y el movimiento de los objetos que están en primer plano en la escena.

La precondición implica que la aproximación se aplica a los puntos de la imagen que son visibles tanto en el instante t como en el instante $t + \epsilon$; de esta forma, los restantes píxeles de la imagen serán ignorados para nuestro propósito y el error en la aproximación puede ser calculado de la siguiente manera:

$$\mathcal{E}(x_1, x_2, t) = f(x_1, x_2, t) - f(x_1 + \delta, x_2, t + \epsilon) \tag{3.4}$$

Para valores dados de ϵ y velocidad de la cámara, el valor de δ puede ser estimado de forma experimental encontrando el valor de δ que minimiza $\|\mathcal{E}\|$, donde $\|\cdot\|$ representa cualquier norma aplicable. Ahora es importante remarcar que el error viene de dos fuentes distintas: Los efectos ópticos, como la aberración en los extremos del fotograma que produce la deformación de la lente de la cámara, y la presencia de objetos en primer plano:

$$\mathcal{E}(x_1, x_2, t) = \mathcal{E}_{opticos}(x_1, x_2, t) + \mathcal{E}_{objetos}(x_1, x_2, t) \tag{3.5}$$

Donde los efectos ópticos se asumen como pequeños respecto al efecto producido por los objetos en primer plano, si es que esos objetos están presentes:

$$Fore(t) \Leftrightarrow \|\mathcal{E}_{optical}(x_1, x_2, t)\| \ll \|\mathcal{E}_{objects}(x_1, x_2, t)\| \tag{3.6}$$

donde $Fore(t)$ significa que existen objetos en primer plano en el instante t. Es más, si no hay objetos en primer plano, entonces el error asociado es cero:

$$Fore(t) \Leftrightarrow \mathcal{E}_{objects}(x_1, x_2, t) \neq 0 \tag{3.7}$$

Por eso, la norma del error esperado debería ser mayor cuando hay objetos en primer plano que están en movimiento:

$$E[\|\mathcal{E}(x_1, x_2, t)\| \mid Fore(t)] > E[\|\mathcal{E}(x_1, x_2, t)\| \mid \neg Fore(t)] \tag{3.8}$$

La propuesta que se indica en este trabajo aprovecha este hecho entrenando un clasificador basado en una red neuronal de tipo perceptrón multicapa que será la encargada de estimar la probabilidad de que estén presentes objetos en primer plano, $P(Fore(t))$. En este punto, la norma del error es computada por columnas de píxeles de forma que se pueda calcular la suma de las normas de las diferencias entre píxeles en las columnas x_1 y $x_1 + \delta$. A continuación se añaden los sumatorios de las normas de los errores para columnas de píxeles contíguas de forma que se pueda obtener un conjunto reducido de sumatorios de las normas de los errores. Estas sumas serán proporcionadas como entradas a la red neuronal, mientras que la salida deseada $z(t)$ será 1 cuando $Fore(t)$ se mantenga, y -1 en otro caso. De esta forma, la probabilidad será, por tanto, estimada de la siguiente manera:

$$P(Fore(t)) = \frac{1}{2}(z(t) + 1) \tag{3.9}$$

A continuación se aplicará un umbral de probabilidad para determinar si se ha detectado algún objeto en primer plano en movimiento:

$$Detection(t) \Leftrightarrow P(Fore(t)) > \theta \tag{3.10}$$

3.2. Implementación del algoritmo

Para tener mayor control sobre el proceso de adquisición de imágenes, entrenamiento de la red neuronal a utilizar y prueba de muestras no pertenecientes a los conjuntos de entrenamiento, se ha decidido implementar el diseño formal matemático propuesto en la sección anterior por medio de tres programas distintos escritos en C++, cada uno de ellos con una misión específica. Antes de pasar a explicar el funcionamiento de dichos programas hay que recordar que el vídeo utilizado para realizar los experimentos se ha separado en forma de secuencia de fotogramas generando para cada uno de ellos una imagen *.jpg* que será sobre la que trabajarán los distintos programas del sistema.

3.2.1. Programa *GeneradorEntrenamientoAvanzado*

Este programa [1] será el encargado de generar las muestras que servirán de entrenamiento a la red neuronal encargada de reconocer la presencia de objetos en primer

[1] Todos los programas incluido éste se pueden encontrar en el directorio *sources* del CD que se adjunta a este documento.

plano que se encuentren en el escenario. Por tanto, será el encargado de generar un conjunto de entrenamiento válido para la red neuronal del sistema a partir de una secuencia de fotogramas que se encuentran en una localización específica. Para ello, realizará un preprocesamiento de los fotogramas para simplificarlos y formatearlos en forma de conjuntos de muestras que pueda entender la red neuronal del sistema. De los tres, este programa es el único que tendrá contacto real con los fotogramas del vídeo ya que el resto de ellos trabajarán sobre los ficheros que va a devolver como salida este programa. A continuación se ofrece una descripción simplificada del funcionamiento del programa *GeneradorEntrenamientoAvanzado*.

En primer lugar hay que destacar que la implementación de red neuronal elegida para el sistema [2] sólo acepta muestras en el formato que se indica en la figura 3.1:

```
1  4 4 1
2  1084.663208 996.501221 1719.120117 4296.963379
3  1
4  1354.261108 392.868683 151.668320 51.481251
5  1
6  351.547363 149.918915 139.659363 61.548744
7  -1
8  753.491394 470.317963 352.225342 206.497360
9  -1
```

Figura 3.1: Formato de muestras aceptadas por la red neuronal

En la primera fila se indican por orden los valores correspondientes al número de muestras que contiene el archivo, el número de entradas de la red y el número de salidas. En el ejemplo de la figura 3.1 se observa un archivo de entrada que indica que hay 4 muestras con 4 entradas y 1 salida cada una [3]. En las filas posteriores se pueden ver cada una de las muestras donde las líneas pares contienen los valores de entrada de las neuronas de la capa de entrada de la red neuronal y las líneas impares contienen los valores correspondientes a la salida esperada de la red para dicha entrada.

A continuación el programa lanza el método *GenerateFannTrainingAdvanced* que realizará las siguientes tareas:

- En primer lugar, abre los directorios donde se encuentran los fotogramas que se van a procesar.

[2] Fast Artificial Neural Network por Lenikssen.

[3] Para facilitar la comprensión del fichero de entrada de la red, el fichero que se ha indicado en la figura 3.1 es mucho más pequeño que el fichero que genera el programa GeneradorEntrenamientoAvanzado, pero el funcionamiento general es el mismo.

- En segundo lugar, el tomará dos fotogramas consecutivos de $640x480$ píxeles, a los que llamaremos n y $n+1$, y lanzará el método *compareImages* que, en primer lugar, desplazará el fotograma $n+1$ a la izquierda una distancia de δ píxeles para compensar el movimiento horizontal de la cámara y recortará el sobrante de píxeles que quedará a la derecha del fotograma. Además también recortará δ píxeles a la derecha del fotograma n ya que, dado que el movimiento se realiza hacia la izquierda, esos píxeles no nos aportarán nada a la hora de comparar los dos fotogramas y de esta forma, ambos tendrán las mismas dimensiones tal y como se puede observar en la figura 3.2.

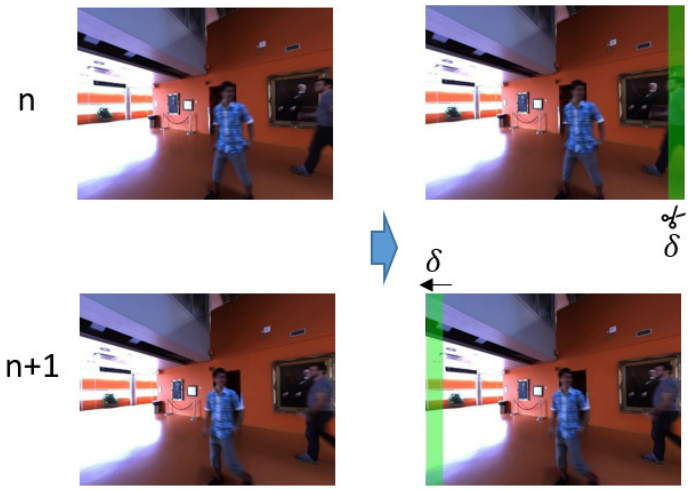

Figura 3.2: Primera fase del algoritmo de comparación de imágenes

A continuación, el algoritmo dividirá dichos fotogramas en columnas de píxeles de una anchura fija como se puede ver en la figura 3.3

realizando la comparación de forma sucesiva de la columna x del fotograma n con la columna x del fotograma $n+1$ utilizando como norma el Error Cuadrático Medio (MSE) entre los píxeles de los tres canales RGB de dichas columnas. Esta fórmula se puede ver en la ecuación 3.11

$$MSE = \frac{1}{M}\sum_{i=1}^{M}\|\mathbf{w}_i - \mathbf{x}_i\|^2 \tag{3.11}$$

Seguidamente, guardará los valores obtenidos como una muestra en el fichero que se ha habilitado para ello seguidas de una línea que representará, como ya se ha dicho, el valor del ground-truth para esos dos fotogramas en el archivo

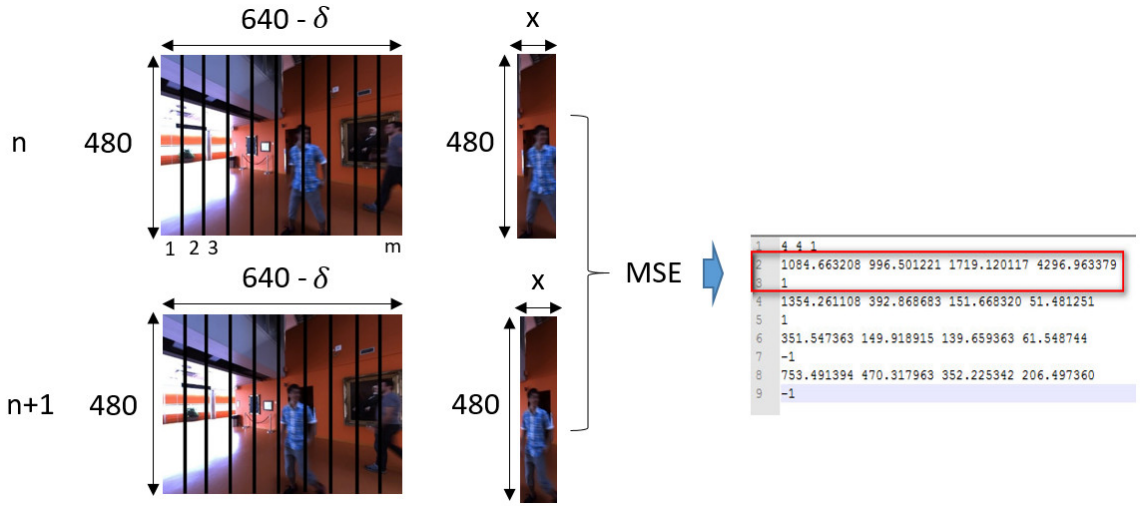

Figura 3.3: Segunda fase y resultado del algoritmo de comparación de imágenes

cuyo formato se explica en la imagen 3.1. Todo ello viene especificado en la imagen 3.3

Por último, el método *compareImages* terminará y el programa *GenerateFannTrainingAdvanced* al cual pertenece, realizará el mismo proceso descrito tomando los fotogramas $n+2$ y $n+3$. Este proceso será realizado de forma sucesiva mientras queden fotogramas sin procesar en el directorio.

Una vez el programa *GeneradorEntrenamientoAvanzado* termine habrá generado el fichero de muestras que será el que pasarán a procesar los programas que se describen en los ítems subsiguientes.

3.2.2. Programa *GeneradorConjuntosEntrenamiento*

Con el objetivo de obtener unas medidas de rendimiento del sistema que sean lo más precisas posible, el proceso de entrenamiento y validación de la red neuronal asociada al sistema se realizará utilizando el método *10-folds cross-validation*, bien conocido en los ámbitos de los sistemas de clasificación. Dado que dicho método se utilizará en la parte experimental del presente trabajo, se ha creído más conveniente explicar su dinámica en una sección habilitada para ello en el capítulo 4. Por ahora, lo que debemos saber para comprender el funcionamiento de este programa es que será el encargado de separar el conjunto inicial de muestras que vienen en el archivo generado

por el programa *GeneradorEntrenamientoAvanzado* en una serie de conjuntos de Entrenamiento, Validación y Test que se utilizarán para verificar la precisión en la detección de objetos en movimiento por parte de la red neuronal que se utilizara en el sistema.

La salida del programa estará compuesta de un total de 10 baterías de test almacenadas en 10 directorios distintos cada uno de ellos conteniendo 10 conjuntos de entrenamiento, 10 conjuntos de validación y 1 conjunto de test haciendo un total de 190 ficheros que se utilizarán en la generación y optimización de las distintas redes neuronales que serán necesarias en el proceso de experimentación que se llevará a cabo [4]. Todo este proceso dará lugar a una estructura como la que se puede ver en la figura 3.4.

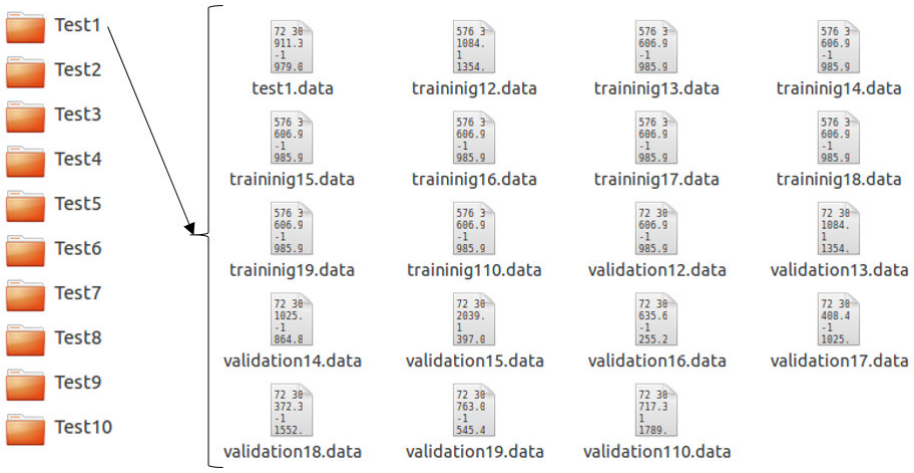

Figura 3.4: Estructura de directorios generada por el programa GeneradorConjuntosEntrenamiento

Como se puede ver, cada batería de test consta de un fichero de muestras de test y 9 duplas de ficheros de entrenamiento-validación. Cada una de estas duplas de ficheros entrenamiento-validación se utilizarán para entrenar una red neuronal diferente, lo cual nos da un número de 90 redes neuronales que luego habrá que multiplicar por el número de diferentes valores que se le den a cada parámetro de la red neuronal que se quiera variar. Además, para evitar un proceso demasiado largo de experimentación, el único parámetro que se va a variar para captar la evolución del rendimiento y la precisión de dicha red, será el número de neuronas en la capa oculta del perceptrón.

[4]Ver más información acerca del método de validación cruzada en el capítulo 4

3.2.3. Programa *EntrenamientoTimeStats*

Este programa consiste en un algoritmo que es capaz de automatizar todos los procesos de entrenamiento, validación y tests necesarios para el presente proyecto. Estará a cargo de la calibración de los distintos modelos de red neuronal que tomarán parte en los procesos de experimentación y además será el encargado de recoger estadísticas temporales para cada uno de los procesos que se llevan a cabo en dicho proceso de experimentación. El funcionamiento de dicho programa será el siguiente:

- Como suele ser normal, el punto de entrada del programa será el módulo **main** el cual comienza estableciendo los valores iniciales de todas las variables de tipo cadena de caracteres que contendrán las rutas base de los ficheros que se van a generar.

- A continuación, se establecen los valores iniciales de los parámetros del perceptrón multicapa y los valores de las variables que se van a utilizar para sacar estadísticas del tiempo que tardan en realizarse los distintos procesos de experimentación.

- En este punto, entramos en el bucle principal del programa que se ejecutará tantas veces como valores distintos se le den al número de neuronas en la capa oculta del perceptrón multicapa. En este bucle se encuentran anidados dos bucles for más, que serán los encargados de ir recorriendo los diferentes ficheros de entrenamiento, validación y test que fueron generados por el programa *GeneradorConjuntosEntrenamiento* y al mismo tiempo realizarán el entrenamiento de cada una de las redes resultantes de dichos conjuntos de muestras tomando valores estadísticos del tiempo que tardan en realizarse cada uno de los pasos de los procesos de entrenamiento, validación y test [5].

El programa arrojará como resultado una estructura de archivos de la forma que se puede ver en la figura 3.5.

En ella se observan tres directorios llamados *resultadosEntrenamiento*, *resultadosTest* y *resultadosValidacion*.

- El directorio *resultadosEntrenamiento* contendrá los resultados que devuelve el perceptrón multicapa en los procesos de entrenamiento para cada una de las baterías de entrenamiento que se han descrito y para cada valor del número de neuronas en capa oculta del perceptrón multicapa. Dichos resultados se ofrecen

[5] Ver código fuente adjunto.

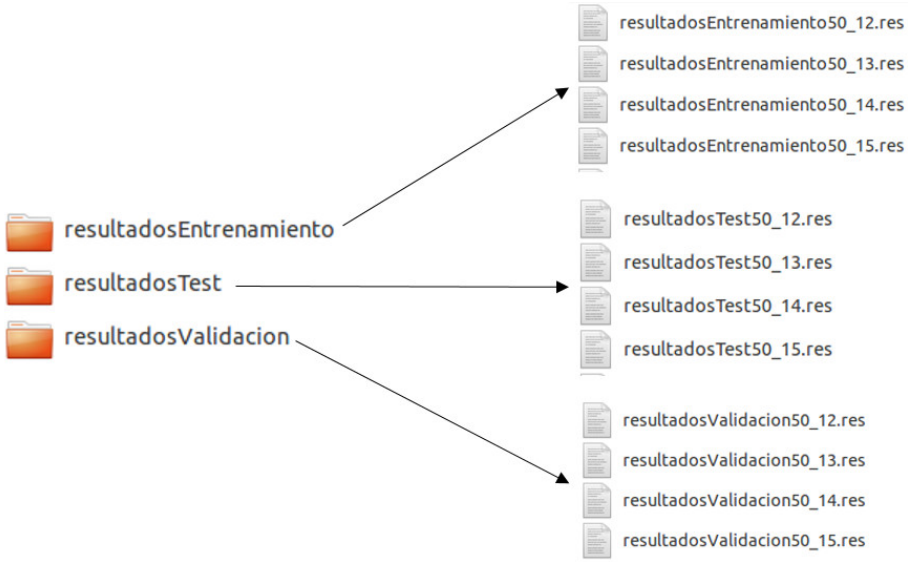

Figura 3.5: Estructura de directorios generada por el programa EntrenamientoTimeStats

comparados con los resultados del ground-truth del sistema para su posterior procesado.

- El directorio *resultadosTest* contendrá los resultados que devuelve el perceptrón multicapa en los procesos de testeo para cada una de las baterías de entrenamiento que se han descrito y para cada valor del número de neuronas en capa oculta del perceptrón multicapa. Dichos resultados se ofrecen comparados con los resultados del ground-truth del sistema para su posterior procesado.

- El directorio *resultadosValidacion* contendrá los resultados que devuelve el perceptrón multicapa en los procesos de validación para cada una de las baterías de entrenamiento que se han descrito y para cada valor del número de neuronas en capa oculta del perceptrón multicapa. Dichos resultados se ofrecen comparados con los resultados del ground-truth del sistema para su posterior procesado.

Por último, el programa también devuelve una serie de ficheros con resultados estadísticos temporales en los cuales se ven reflejados los valores medios de tiempo para los procesos de entrenamiento, validación y test de cada red neuronal y para cada valor del número de neuronas en capa oculta del perceptrón. Los ficheros generados ofrecerán la información que se puede ver en la figura 3.6.

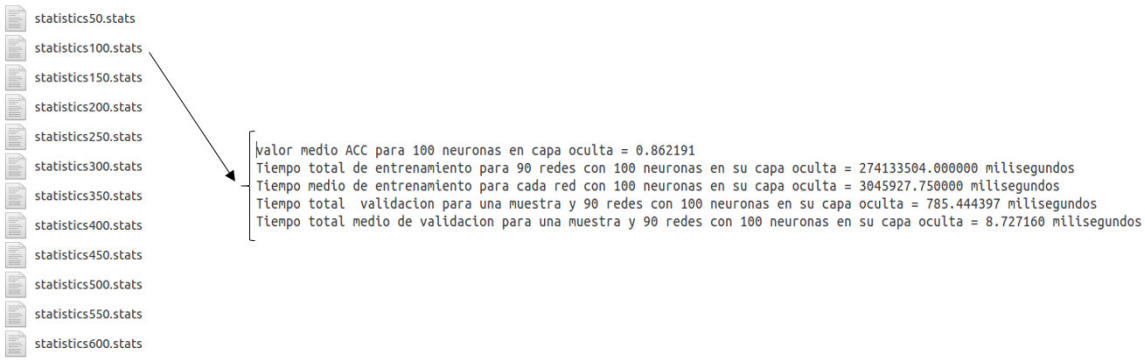

Figura 3.6: Estructura de ficheros de estadísticas temporales generada por el programa EntrenamientoTimeStats

En el siguiente capítulo, se describe el proceso de experimentación que se ha llevado a cabo en el sistema.

Capítulo 4

Resultados experimentales

4.1. Descripción del proceso de experimentación

Como se dijo en el capítulo 2, con motivo de mejorar nuestro control sobre el proceso de experimentación, todas las pruebas se han realizado utilizando vídeos grabados con una cámara de 360º en combinación con el software *Virtual PTZ*. Por la misma razón, como los vídeos proporcionados por la cámara 360º tienen una tasa de 16 fotogramas por segundo, se ha ajustado la rotación en sentido panorámico de la cámara PTZ a una velocidad de 16 grados sexagesimales por segundo en sentido antihorario. Después de realizar una serie de tests, se ha ajustado el valor δ a 5 píxeles/grado y se ha considerado el error cuadrático medio (MSE en inglés) como la norma de error $\|\mathcal{E}\|$ (ver capítulo 3). Hechas las anteriores consideraciones, el proceso de comparación de un frame con el siguiente se puede ver descrito en la figura 4.1:

En primer lugar, con el objetivo de compensar el cambio en el fondo de la imagen provocado por la rotación de la cámara, el fotograma $n + 1$ será desplazado δ píxeles a la izquierda con respecto al fotograma n. Este desplazamiento hará que se pierdan los 5 píxeles más a la izquierda del fotograma, de modo que el fotograma que procesaremos será de $635x480$ píxeles. A continuación, los dos fotogramas serán divididos en 10 franjas de 63 píxeles de ancho (esta será la dimensión de todas las franjas salvo la última que tendrá que, como 635 no es divisible entre 10 tendrá que recoger los píxeles que sobran pasando a ser de 68 píxeles de ancho) y se calculará el error cuadrático medio para cada franja de los dos fotogramas. Finalmente, un vector de 30 componentes (10 por cada color de los canales RGB) más un número, que será 1 si la muestra es positiva y -1 si es negativa, será guardado como una muestra que se pasará al perceptrón en el proceso de entrenamiento, validación y test.

Con el objetivo de evaluar el rendimiento y precisión del sistema descrito a la hora

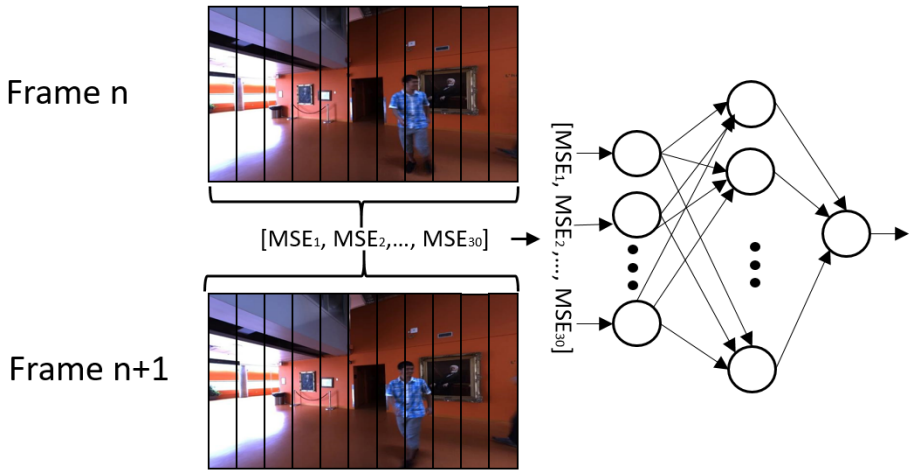

Figura 4.1: Ejemplo de como son obtenidas las muestras de vídeo y proporcionadas como entradas para el perceptrón.

de detectar movimiento de objetos en primer plano en el flujo de vídeo proporcionado por la *Virtual PTZ*, se han realizado una serie pruebas que consisten en el cálculo de medidas generales de rendimiento que se utilizan habitualmente en procesos de visión artificial, para diferentes topologías de la red neuronal utilizada.

Los perceptrones multicapa pueden ser calibrados mediante la modificación de sus diferentes parámetros con el objetivo de obtener tasas de rendimiento superiores según la tarea a la que se estén aplicando. No obstante, como el número de combinaciones de los distintos parámetros del perceptrón crecería de manera exponencial, realizar pruebas del sistema variando todos los parámetros a la vez no sería práctico, por lo que para este trabajo, se ha considerado la modificación de un sólo parámetro que sabemos que tenga una importante repercusión en el funcionamiento del perceptrón mientras que el resto se mantendrán inalterados. Por todo ello, el parámetro seleccionado será el número de neuronas en la capa oculta del perceptrón, quedando las características de la red neuronal como se indica en la tabla 4.1

Para asegurar la correcta evaluación del rendimiento de la red neuronal del sistema, se ha considerado un proceso de validación cruzada del tipo *10-fold cross validation* o validación cruzada de 10 iteraciones.

Tipo de red neuronal	Perceptrón multicapa
Número de entradas	30
número de neuronas en la capa oculta	50-600
Número de salidas	1
Algoritmo de aprendizaje	Retropropagación
Número máximo de etapas en el entrenamiento	10000
Tasa de aprendizaje	0.7

Cuadro 4.1: Parametros de prueba par el perceptron multicapa

4.2. Validación cruzada

La validación cruzada o cross-validation es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Se utiliza en entornos donde el objetivo principal es la predicción y se quiere estimar cómo de preciso es un modelo que se llevará a cabo en la práctica. Es una técnica muy utilizada en proyectos de inteligencia artificial para validar modelos generados.

4.2.1. Validación cruzada de K iteraciones (*10-folds cross-validation*)

En la validación cruzada de K iteraciones o *K-fold cross-validation* los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos se utiliza como datos de prueba y el resto (K-1) como datos de entrenamiento. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado. Este método es muy preciso puesto que evaluamos a partir de K combinaciones de datos de entrenamiento y de prueba, pero aun así tiene una desventaja, y es que, a diferencia del método de retención, es

lento desde el punto de vista computacional. En la práctica, la elección del número de iteraciones depende de la medida del conjunto de datos. Lo más común es utilizar la validación cruzada de 10 iteraciones (10-fold cross-validation) que es la que se ha utilizado para nuestros experimentos.

4.2.2. Explicación detallada del método de validación cruzada seguido en el proyecto

Como ya se ha indicado anteriormente, en este proyecto se ha utilizado el método 10-fold Cross-Validation o validación cruzada de 10 iteraciones. En primer lugar, se divide el conjunto de muestras de que se dispone en 10 subconjuntos disjuntos (figura 4.2):

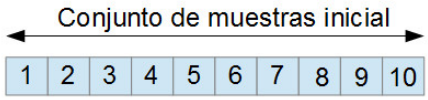

Figura 4.2: Conjunto de todas las muestras de que se dispone dividido en 10 subconjuntos.

A continuación se separa uno de los subconjuntos y se aparta del resto. Éste será nuestro conjunto de test , el cual sólo se utilizará estrictamente para probar la red. Nunca en el proceso de entrenamiento ni validación. A continuación se realizan las 10 iteraciones del proceso de validación cruzada con el resto de los subconjuntos tal y como se indica en la figura 4.3.

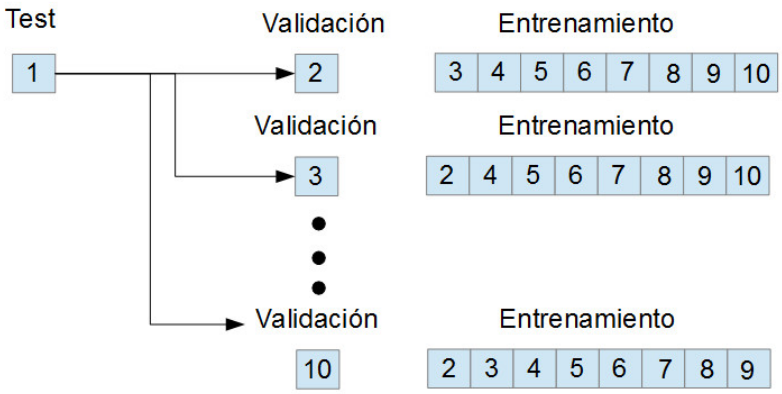

Figura 4.3: Proceso de validación cruzada con el resto de subconjuntos.

En cada una de las k iteraciones de este tipo de validación se realiza un cálculo de error. El resultado final lo obtenemos a partir de realizar la media aritmética de los K valores de errores obtenidos, según la fórmula:

$$E = \frac{1}{K} \sum_{i=1}^{K} E_i \tag{4.1}$$

Es decir, se realiza el sumatorio de los K valores de error y se divide entre el valor de K.

De este modo, se obtienen los valores que serán necesarios para el cálculo de las medidas de rendimiento del sistema. Medidas que se explican en el apartado siguiente.

4.3. Medidas de rendimiento

4.3.1. Exactitud (*Accuracy*)

Por su simplicidad, una de las medidas de rendimiento más utilizada es la Exactitud (en adelante, *Accuracy*). La Accuracy es una medida que resulta de dividir el número de predicciones acertadas por el número total de muestras de prueba [12] como se indica en la fórmula siguiente:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

Donde:

- TP es el número de valores positivos que la red ha señalado como positivos.
- TN es el número de valores negativos que la red ha señalado como negativos.
- FP es el número de valores negativos que la red ha señalado como positivos.
- FN es el número de valores positivos que la red ha señalado como negativos.

Para calcular este valor y poder representarlo en forma gráfica de manera más cómoda, se ha utilizado el framework Matlab. En concreto, los programas *accuracyEntrenamiento.m*, *accuracyValidacion.m* y *accuracyTest*, adjuntos a esta memoria en el directorio *matlabSources*, serán los encargados de generar de forma definitiva estos valores y el programa *calculoEstadísticasAccuracy.m* será el encargado de representarlos en forma de gráfico de barras de error. Así, la gráfica de la izquierda de

la figura 4.4 muestra los valores de la precisión en la predicción para los conjuntos de muestras de entrenamiento, validación y prueba según va variando el número de neuronas en la capa oculta del perceptrón.

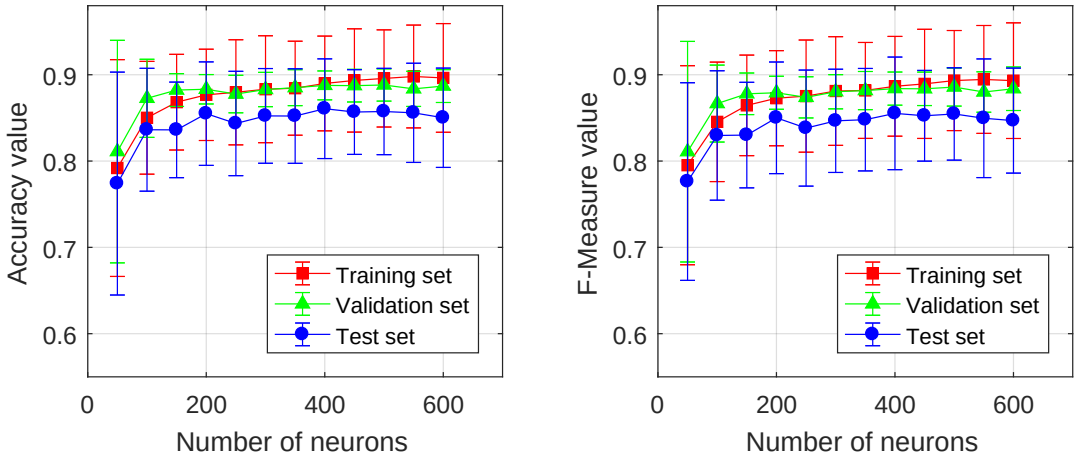

Figura 4.4: Gráfico de barras de error correspondiente a la Precisión y F-Medida para los conjuntos de entrenamiento, validación y test versus número de neuronas en capa oculta del perceptron.

4.3.2. F-Medida (*F-Measure*)

Aunque, la Exactitud es una medida que está ampliamente bien considerada como un criterio aceptable para la medida de rendimiento de un clasificador, especialmente en casos como el que nos ocupa, donde el número de muestras positivas y negativas está bien equilibrado, es interesante ampliar nuestros resultados experimentales con la F-Medida (*F-measure* en inglés) de rendimiento del clasificador ya que la está también bien considerada como medida de rendimiento siendo en algunos casos incluso más fiable que la misma Exactitud o Accuracy [12]. La deducción de la expresión que se utiliza para el cálculo de esta medida es la siguiente:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.4}$$

$$F - Measure = 2\frac{Precision * Recall}{Precision + Recall} \tag{4.5}$$

Y el resultado obtenido al realizar el cálculo de la F-Medida se puede ver en la parte derecha de la figura 4.4. Los programas encargados del cálculo de la F-Medida serán los siguientes: *fEntrenamiento.m*, *fValidacion.m* y *fTest.m*, adjuntos a esta memoria en el directorio *matlabSources*. El programa *calculoEstadísticasF.m* será el encargado de representarlo en forma de gráfico de barras de error.

4.3.3. Área Bajo la Curva ROC (*AUC*)

Como ya se ha dicho, la función de transmisión de las neuronas del perceptrón multicapa que utilizamos en nuestro sistema corresponde a una sigmoide que devolverá un valor entre 1 y -1 a su salida. Como se recordará, entrenamos nuestra red neuronal estableciendo que las muestras donde el sistema detectara movimiento de objetos en primer plano se corresponderá con el valor 1, asignando el valor -1 a aquellas muestras en las que no hay movimiento de objetos en primer plano. El problema es que la salida de la red neuronal no sólo será un valor correspondiente al conjunto $\{-1, 1\}$, sino que será un valor perteneciente al intervalo $[-1, 1]$, lo cual hace necesario que se establezca un valor de umbral por debajo del cual, la salida del perceptrón se considerará negativa, considerándose positiva por encima de dicho umbral. De forma que para calcular tanto la medida de Exactitud como la F-Medida se ha situado el umbral en el valor 0 del intervalo, considerado que la salida de la red neuronal será negativa si pertenece al intervalo $[-1, 0)$ y se considera positiva si pertenece al intervalo $[0, 1]$.

Dado que la tanto la Exactitud como la F-medida limitan su medición del rendimiento a un solo valor de umbral se ha considerado necesario completar los resultados mostrados en la figura 4.4 con una medida que tenga en cuenta todos los valores posibles de umbral. Con este objetivo en mente se ha incorporado el cálculo del *Área Bajo la Curva* o AUC para todos los modelos de redes neuronales considerados en este trabajo. La expresión matemática que formaliza el cálculo de esta medida puede ser observada a continuación:

$$AUC = \int TPR(T) - (FPR'(T))dT \tag{4.6}$$

Donde:

- TPR es la tasa de verdaderos positivos.

- FPR es la tasa de falsos negativos.
- T es el umbral.

Los valores obtenidos para esta medida de rendimiento después de realizar las pruebas se pueden ver en la figura 4.5.

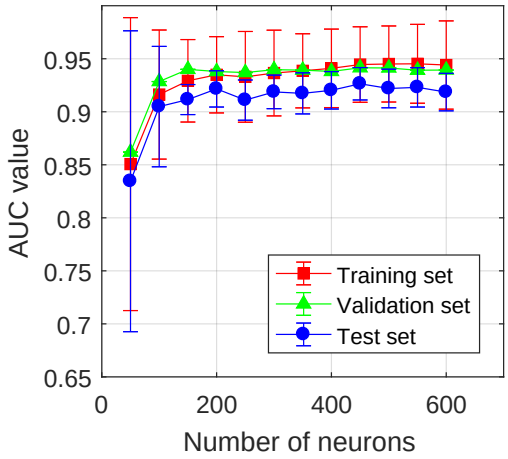

Figura 4.5: Gráfico de barras de error de la AUC para los conjuntos de validación y prueba (mejor cuanto más alto sea).

De la misma forma que para el caso de las medidas anteriores, los programas que se han utilizado para calcular la AUC serán *AUCEntrenamiento.m*, *AUCValidacion.m* y *AUCTest.m* y su representación en forma de gráfico de barras de error será llevada a cabo por el programa *calculoEstadísticasAUC.m*, todos ellos adjuntos a esta memoria en el directorio *matlabSources*.

Las tres gráficas presentadas revelan cómo el modelo alcanza altos niveles de rendimiento para una cantidad de 200 neuronas en adelante en su capa oculta y a partir de ese punto aumenta su estabilidad conforme el número de neuronas en capa oculta crece. Los resultados obtenidos también muestran cómo para los tests en configuraciones de 200 neuronas o más, todas las medidas de rendimiento presentadas presentan valores por encima del 80 %. Una consideración importante que hay que hacer es que con el objetivo de prevenir la pérdida de información y después de observar que no afecta de forma dramática al rendimiento del clasificador, todas las medidas de rendimiento obtenidas anteriormente se han calculado a partir del flujo de vídeo tal y como viene de la cámara PTZ. Esto implica que no se ha realizado ningún preprocesamiento de la imagen para corregir tanto el ruido proveniente del

sensor CMOS de la cámara como la aberración producida en los bordes de la imagen por la lente de la cámara.

4.4. Medidas de tiempo

El rendimiento del sistema respecto al tiempo es una cuestión de importancia capital cuando se trata de procesamiento de vídeo en tiempo real. Esto se hace más patente cuando se utilizan microcontroladores para llevar a cabo procesos basados en redes neuronales artificiales que consisten en detección de movimiento a partir de flujos de vídeo provenientes de una cámara PTZ como el que nos ocupa. Por tanto, el algoritmo utilizado no sólo tiene que ser fiable, sino que también tiene que probar que los tiempo de entrenamiento está dentro de unos límites aceptables y la clasificación de las muestras de prueba es suficientemente rápida como para proporcionar detección de movimiento de objetos en primer plano cuando se ejecuta en una placa Raspberry Pi con las características que se señalaron en la sección 2. En la tabla número 4.2 se puede observar el tiempo medio de entrenamiento para cada una de las redes entrenadas y el tiempo de clasificación de una sola de las muestras de prueba respecto al número de neuronas en capa oculta que se puede observar cuando se ejecuta el algoritmo en la placa Raspberry Pi 3 Modelo B.

# Neurons	50	100	200	300	400	500	600
Training Avg. Time (s)	39.49777	26.49671	36.32062	53.55619	54.05586	62.48622	93.27951
Processing Avg. Time (s)	0.01959	0.01963	0.01969	0.01975	0.01982	0.01988	0.01995
Fps	51.03369	50.95048	50.78488	50.62035	50.45434	50.29447	50.13059

Cuadro 4.2: Tiempo medio de entrenamiento y testeo de muestras respecto al número de neuronas en capa oculta.

Para proporcionar una idea más clara del rendimiento del sistema que se documenta en esta memoria, la tabla 4.2 también proporciona la velocidad media (medida

en fotogramas por segundo o *fps*) a la que puede trabajar el sistema cuando recibe un flujo directo de vídeo desde la *Virtual PTZ*.

El tiempo medio de entrenamiento ha sido calculado a partir de los valores obtenidos al lanzar el proceso de entrenamiento un total de 90 veces, cada una correspondiendo al mismo número de neuronas en la capa oculta. La media del tiempo de clasificación obtenido y la velocidad de procesamiento en *fps* han sido calculados a partir de los valores obtenidos después de pasar al algoritmo un total de 72 muestras de test combinadas con cada una de las redes neuronales entrenadas. Los resultados mostrados en la tabla 4.2 reflejan una velocidad de procesamiento aproximada mayor de 50 fotogramas por segundo, la cual constituye una tasa de fotogramas por segundo excelente en términos de procesamiento de vídeo en tiempo real.

Capítulo 5

Conclusiones y trabajo futuro

En el presente trabajo se ha llevado a cabo el diseño e implementación de un sistema de detección de movimiento de objetos en primer plano en tiempo real para sistemas de videovigilancia basados en cámaras de movimiento panorámico.

- El sistema incorpora un algoritmo capaz de procesar una secuencia de imágenes proveniente de un software de simulación de una cámara PTZ dividiendo cada imagen en un conjunto de franjas de una anchura determinada y comparando cada una de ellas con la franja equivalente en el siguiente fotograma para obtener un vector de números que pueda ser introducido como patrones de entrenamiento, validación y test en una red neuronal del tipo perceptrón multicapa que estará encargada de indicar cuando hay o no objetos en movimiento en primer plano en dicho flujo de vídeo.

- Con el objetivo de incrementar la eficiencia energética del sistema y su portabilidad, el algoritmo ha sido optimizado para su implantación un microcontrolador modelo Broadcom BCM2837 montado sobre una placa Raspberry Pi 3 Modelo B.

- Después de realizar determinadas baterías de pruebas variando el número de neuronas en la capa oculta del perceptrón, los resultados obtenidos en medidas de rendimiento ampliamente utilizadas, indican que es posible conseguir un sistema con una precisión aceptable y un funcionamiento correcto en tiempo real.

- Las pruebas de rendimiento del sistema respecto a los tiempos de ejecución también indican tiempos de entrenamiento razonables y velocidades de procesamiento de vídeo que superan los 50 *fps*, reafirmando el sistema descrito en

esta memoria como una alternativa válida para la detección de movimiento en tiempo real cuando se combina con cámaras de movimiento panorámico.

5.0.1. Trabajo Futuro

Teniendo en cuenta el rendimiento del sistema en su estado de desarrollo actual, las líneas futuras de investigación del proyecto serán las siguientes:

- Actualmente el sistema sólo está preparado para detectar movimiento en flujos de vídeo que provengan de una cámara PTZ que se mueva en sentido panorámico, no estando preparado para realizar la detección de movimiento cuando la cámara se mueve en sentido vertical o cuando la cámara hace zoom. Una línea de desarrollo futura será la adaptación del algoritmo de detección de movimiento para que también funcione cuando la cámara se mueva en sentido vertical, cuando haga zoom y con combinaciones de dichos movimientos.

- Por otra parte, actualmente el algoritmo está diseñado para realizar la detección de movimiento cuando la cámara se mueve a velocidad constante. De modo que una futura línea de desarrollo pasa por adaptar el algoritmo para que funcione cuando la velocidad de la cámara en cualquier dirección sea variable.

- La linea de desarrollo futuro más importante para nuestro proyecto consistirá en hacer que nuestro sistema no sólo sea capaz de detectar movimiento de objetos en primer plano sino que además sea capaz de reconocer y clasificar dichos objetos. Este problema es mucho más ambicioso, pero podría ser resoluble adaptando el algoritmo para que utilice una red neuronal convolucional convenientemente optimizada para que funcione en un microcontrolador con una velocidad aceptable.

Bibliografía

[1] K. Papadimitriou, A. Dollas, and S. N. Sotiropoulos. Low-cost real-time 2-D motion detection based on reconfigurable computing. *IEEE Transactions on Instrumentation and Measurement*, 55(6):2234–2243, Dec 2006.

[2] M. K. Dobrzynski, R. Pericet-Camara, and D. Floreano. Vision tape-a flexible compound vision sensor for motion detection and proximity estimation. *IEEE Sensors Journal*, 12(5):1131–1139, May 2012.

[3] V. Fung, J. L. Bosch, S. W. Roberts, and J. Kleissl. Cloud shadow speed sensor. *Atmospheric Measurement Techniques*, 7(6):1693–1700, 2014.

[4] L.H. Adnan, Y.M. Yussoff, H. Johar, and S.R.M.S. Baki. Energy-saving street lighting system based on the waspmote mote. *Jurnal Teknologi*, 76(4):55–58, 2015.

[5] Francisco Ortega-Zamorano, Miguel A. Molina-Cabello, Ezequiel López-Rubio, and Esteban J. Palomo. Smart motion detection sensor based on video processing using self-organizing maps. *Expert Systems with Applications*, 64:476 – 489, 2016.

[6] T.E. Boult, X. Gao, R. Micheals, and M. Eckmann. Omni-directional visual surveillance. *Image and Vision Computing*, 22(7):515–534, 2004.

[7] K.-T. Song and J.-C. Tai. Dynamic calibration of pan-tilt-zoom cameras for traffic monitoring. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(5):1091–1103, 2006.

[8] C. Micheloni, B. Rinner, and G.L. Foresti. Video analysis in pan-tilt-zoom camera networks. *IEEE Signal Processing Magazine*, 27(5):78–90, 2010.

[9] C. Ding, B. Song, A. Morye, J.A. Farrell, and A.K. Roy-Chowdhury. Collaborative sensing in a distributed PTZ camera network. *IEEE Transactions on Image Processing*, 21(7):3282–3295, 2012.

[10] G. Chen, P. St-Charles, W. Bouachir, G. Bilodeau, and R. Bergevin. Reproducible evaluation of pan-tilt-zoom tracking. In *Proceedings - International Conference on Image Processing (ICIP)*, pages 2055–2059, 2015.

[11] Steffen Nissen. Fast Artificial Neural Network. `http://leenissen.dk/fann/wp/`, 2016. [Online; accessed 10-January-2017].

[12] C. Parker. An analysis of performance measures for binary classifiers. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 517 – 526, 2011.