SISTEMA *IOT* PARA RECONOCIMIENTO DE PATRONES DE MARCHA MEDIANTE UN ACELERÓMETRO

CARLOS HUMBERTO MONSALVE HERRERA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2021

SISTEMA *IOT* PARA RECONOCIMIENTO DE PATRONES DE MARCHA MEDIANTE UN ACELERÓMETRO

CARLOS HUMBERTO MONSALVE HERRERA

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
JAIME GUILLERMO BARRERO PEREZ
MAG. en Potencia Eléctrica

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2021

Dedicado

A mis padres, Yolanda y Humberto por ser una guía y ejemplo, por sus palabras de apoyo y la ayuda que me brindaron estos años.

A mis amigos, quienes estuvieron conmigo durante cada semestre y me ayudaron a cumplir todos los retos que presentó el mundo universitario.

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

RESUMEN

TÍTULO: SISTEMA *IOT* PARA RECONOCIMIENTO DE PATRONES DE MARCHA MEDIANTE UN ACELERÓMETRO[1]

AUTOR: CARLOS HUMBERTO MONSALVE HERRERA[2]

PALABRAS CLAVE: RECONOCIMIENTO DE ACTIVIDADES, INTERNET DE LAS COSAS, INTELIGENCIA ARTIFICIAL, MICROPROCESADOR.

DESCRIPCIÓN:

Según la Organización Mundial de la Salud, las caídas son la segunda causa principal de muerte por lesiones accidentales o no intencionales en todo el mundo. Los adultos mayores de 65 años sufren el mayor número de caídas mortales. Gracias al rápido desarrollo de los sensores, la inteligencia artificial y el Internet de las cosas (IoT)[3], la interacción entre humanos y computadoras mediante la fusión de sensores se ha considerado un método eficaz para abordar el problema de la detección de caídas. Este documento presenta un sistema de detección de actividades (ADL)[4] con enfoque en la detección de caídas que monitorea en tiempo real al usuario a través de un microcontrolador. Se almacenan los datos recibidos en caché en una ventana deslizante para luego mapearlos en una imagen RGB y mediante una red neuronal convolucional entrenada usando un *dataset* público realizar la predicción. El sistema define dos componentes principales: un dispositivo portátil y un teléfono celular. El dispositivo portátil tiene la capacidad de comunicarse con un teléfono celular indicando la actividad realizada por el usuario. Una vez que el dispositivo portátil detecta una caída, envía una alerta al teléfono celular.

---

1 Trabajo de grado.

2 Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Jaime Barrero, MAG, en Potencia Eléctrica.

3 siglas para ”Internet of Things”

4 siglas para ”Activities of Daily Living”

ABSTRACT

TITLE: IOT SYSTEM FOR GAIT PATTERNS RECOGNITION USING AN ACCELEROMETER.[5]

AUTHOR: CARLOS HUMBERTO MONSALVE HERRERA [6]

KEYWORDS: ACTIVITY RECOGNITION, INTERNET OF THINGS, ARTIFICIAL INTELLIGENCE, MICROPROCESSOR.

DESCRIPTION:

According to the World Health Organization, falls are the second leading cause of death from accidental or unintentional injuries worldwide. Adults over 65 suffer the highest number of fatal falls. Thanks to the rapid development of sensors, artificial intelligence and the Internet of Things (IoT), human-computer interaction through sensor fusion has been considered an effective method to address the problem of fall detection. This document presents an activity detection system (ADL) focusing on the detection of falls monitoring the user in real time through a microcontroller. The received data is cached in a sliding window and then mapped into an RGB image and through a convolutional neural network trained using a public dataset perform the prediction. The system defines two main components: a portable device and a cell phone. The portable device has the ability to communicate with the cell phone indicating the activity performed by the user. Once the portable device detects a fall, it sends an alert to the cell phone.

---

[5] Bachelor Thesis

[6] Faculty of Physical-Mechanical Engineering; School of Electrical, Electronic and Telecommunications Engineering. Director: JAIME BARRERO, MAG, in Electrical Power.

INTRODUCCIÓN

Internet de las cosas es un tema importante en la industria de la tecnología, las políticas y los círculos de ingeniería y se ha convertido en noticia de primera plana, tanto en la prensa especializada como en los medios populares. Los avances logrados en la fabricación permiten incorporar tecnología de cómputo y comunicaciones de vanguardia en objetos muy pequeños.[7] Junto con una mayor economía en la capacidad de cómputo y avance en el aprendizaje automático, se ha impulsado el desarrollo de sensores pequeños y de bajo costo que a su vez promueven muchas aplicaciones de la *IoT* al ofrecer nuevas capacidades que antes no eran posibles como lo es en el campo del reconocimiento de actividad, donde se busca de comprender y clasificar los movimientos significativos de las partes del cuerpo humano y mediante la cual, es posible desarrollar una amplia gama de aplicaciones, pero se ha empezado a prestar especial atención al aumento de la calidad de vida, especialmente para las personas mayores. En este contexto, el desarrollo de diversas tecnologías de asistencia es un dominio importante en el que los investigadores realizan importantes intentos. Entre estos, los esfuerzos para desarrollar sistemas de detección de caídas eficientes y precisos.

El porcentaje de caída en la edad de 65 años es 28-35 %, que aumenta a 32-42 % en la edad de más de 70 años. Aproximadamente, 30 a 50 % de las personas que viven en centros de cuidado geriátrico se caen cada año, y 40 % experimentan caídas recurrentes, las cuales se atribuyen en mayor medida a los cambios biológicos relacionados con la edad, que si no se detectan o tratan a tiempo, pueden provocar

---

[7] KOOMEY, Jon. “How green is the Internet?” En: *https://youtu.be/O8-LDLyKaBM* (2013).

lesiones graves o incluso la muerte. Tales accidentes conducen a 20-30 % de las lesiones leves a severas, y 10-15 % de todos los casos de emergencia se deben a una caída. El porcentaje de hospitalizaciones relacionadas con lesiones entre las personas mayores de 65 años es más del 50 %. En caso de muerte por lesiones, las caídas representan el 40 % del total [8]. En las zonas urbanas y suburbanas, en la mayoría de los casos, los miembros de la familia están fuera de casa durante gran parte del día, lo que dificulta la situación en caso de emergencias. Por lo tanto, es una necesidad social centrarse más en el desarrollo de sistemas innovadores de detección de caídas con la ayuda de tecnologías emergentes como la *IoT*.

[8] WORLD HEALTH ORGANIZATIONS. "WHO Global Report on Falls Prevention in Older Age. Aging and Life Course, Family and Community Health". En: *https://apps.who.int/iris/handle/10665/43811* (2008).

1. OBJETIVOS

1.1. OBJETIVO GENERAL

- Diseñar e implementar un sistema *IoT* capaz de reconocer, analizar y procesar patrones de marcha por medio de inteligencia artificial.

1.2. OBJETIVOS ESPECÍFICOS

- Construir una base de datos que tenga diferentes patrones de marcha para entrenar un modelo aprendizaje automático.
- Mediante un modelo de aprendizaje automático reconocer patrones de movimiento de una persona a partir de datos de sensores de inercia.
- Diseñar e implementar un prototipo de hardware portable de bajo consumo que permita, con la información de los sensores, reconocer patrones de marcha.
- Crear un enlace de comunicación con un teléfono inteligente para enviar señales remotamente, indicando el tipo de movimiento que se está presentando por parte de la persona monitoreada.

2. MARCO TEÓRICO

El gran avance en el poder computacional de los microcontroladores, la capacidad de procesamiento y análisis de grandes cantidades de datos ha impulsado el uso de dispositivos portables, abriendo las puertas para la investigación y el análisis de datos de dispositivos *IoT*. Los acelerómetros y giroscopios son los sensores más utilizados para la detección de caídas con supervisión basada principalmente en umbrales y aprendizaje automático con enfoque en análisis predictivo. En las técnicas basadas en umbrales, se detecta una caída cuando los valores de los datos monitoreados exceden los valores de umbral predefinidos. Por el contrario, en las técnicas de aprendizaje automático se analizan los datos y se intentan aprender patrones ocultos para realizar la predicción.

2.1. ALGORITMOS DE DETECCIÓN DE ACTIVIDADES COTIDIANAS (ADL)

Las tecnologías de detección de caídas existentes se pueden dividir aproximadamente en tres categorías según el tipo de sensor: sensores portátiles [9], sensores ambientales [10] y sensores basados en visión [11]. Los sensores de los dos últimos

---

[9] BUKE, Ao, *et al.* "Healthcare algorithms by wearable inertial sensors: A survey". En: *China Commun* (2015), págs. 1-12.

[10] ZIGEL, Yaniv; LITVAK, Dima y GANNOT, Israel. "A method for automatic fall detection of elderly people using floor vibrations and sound—Proof of concept on human mimicking doll falls". En: *IEEE Trans. Biomed. Eng* (2009), págs. 2858-2867.

[11] MIAO, Yu, *et al.* "A posture recognition-based fall detection system for monitoring an elderly person in a smart home environment". En: *IEEE Trans. Inf. Technol. Biomed* (2012), págs. 1274-1286.

tipos deben implementarse en un entorno particular [12]. Aunque estos sistemas son de alta precisión e implementan una captura de movimiento sencilla, el despliegue del sensor y los algoritmos de detección son complejos. Además, el rango de monitoreo es bastante limitado y la privacidad del usuario podría incluso verse expuesta.

Con el desarrollo de sistemas microelectromecánicos (MEMS, por sus siglas en inglés), los investigadores han integrado sensores inerciales en pequeños dispositivos portátiles para lograr la detección de caídas [13]. El método de detección de caídas basado en MEMS presenta un bajo costo de implementación, consumo y protección de la privacidad del usuario. Por lo tanto, se ha convertido en un campo de investigación popular en la tecnología de detección de caídas [14]. La implementación de la detección de caídas basada en MEMS se puede clasificar de la siguiente manera:

- Los datos de movimiento se transmiten directamente a una computadora *host* con una gran capacidad de procesamiento y almacenamiento, en la que se implementa el algoritmo de detección de caídas. Por ejemplo, en el sistema RAReFall desarrollado por Gjoreski [15], se colocan módulos con acelerómetro integrado en el abdomen y el muslo derecho del sujeto. Los datos de aceleración triaxial recopilados por los módulos se transmiten a una computadora

---

[12] ARIANI, Arni, *et al.* "Simulated unobtrusive falls detection with multiple persons". En: *IEEE Trans. Biomed. Eng* (2012), págs. 3185-3196.

[13] TWOMEY, Niall, *et al.* "A comprehensive study of activity recognition using accelerometers". En: *Informatics* (2008), pág. 27.

[14] BECKER, Clemens, *et al.* "Proposal for a multiphase fall model based on real world fall recordings with body-fixed sensors". En: *Zeitschrift für Geron- tologie Geriatrie* (2012), págs. 707-715.

[15] GJORESKI, Hristijan, *et al.* "RAReFall— Real-time activity recognition and fall detection system". En: *Proc. IEEE Int. Conf. Pervasive Comput. Commun* (2014), págs. 145-147.

portátil a través de *Bluetooth*, y la caída se determina por la diferencia entre el máximo y el mínimo de la aceleración en la ventana deslizante que es mayor que un umbral. Benocci [16] construyó un módulo sensor de aceleración triaxial y transfirió los datos del flujo de aceleración al *host* a través de *ZigBee*. Dado que la unidad de procesamiento en el módulo debe recopilar datos continuamente y transmitirlos al *host* de forma inalámbrica, el principal inconveniente de este enfoque es un gran consumo de energía. Además, dada la gran cantidad de transmisión de datos, es muy probable que se produzca un retraso en la red.

- Los datos se extraen y preprocesan en dispositivos portátiles y luego se transmiten a una computadora *host* con alta capacidad informática. Por ejemplo, Yuan [17] diseñó un módulo de detección de caídas impulsado por interrupciones combinando el método de interrupción con el método de umbral. Este módulo envía datos de actividad a una computadora a través de *Bluetooth* y esta clasifica las caídas de las actividades diarias a través de un algoritmo de árbol de decisiones. Este enfoque es uno de los más populares y se enfrenta a algunos desafíos como el retraso y la distancia de transmisión de datos.

- Se implementa un algoritmo de detección de caídas directamente en dispositivos portátiles con recursos y consumo de energía limitados. Por ejemplo, Guanyi [18]. utilizó una máquina de soporte vectorial para determinar el umbral de aceleración y velocidad angular de caída, y diseñó un sistema para detec-

---

[16] BENOCCI, Marco, *et al.* "Accelerometer-based fall detection using optimized Zig- Bee data streaming". En: *Microelectron* (2010), págs. 703-710.

[17] JIAN, Yuan, *et al.* "Power-efficient interrupt-driven algorithms for fall detection and classification of activities of daily living". En: *IEEE Sensors J* (2015), págs. 1377-1387.

[18] GUANGYI, Shi, *et al.* "Mobile human airbag system for fall protection using MEMS sensors and embedded SVM classifier". En: *IEEE Sensors* (2009), págs. 495-503.

tar y prevenir caídas utilizando acelerómetros, giroscopios y *airbags* de 3 ejes. Wang [19] implementó un dispositivo de detección de caídas usando un acelerómetro de 3 ejes y un barómetro. Este enfoque es probablemente el más prometedor y utiliza de manera efectiva el *edge computing*, el cual es un tipo de informática que ocurre en la ubicación física o cerca del usuario. Y cada vez más investigadores están trabajando en esta área. Sin embargo, este modelo aún enfrenta algunos desafíos, como diseñar un algoritmo de preprocesamiento simple, comprimir datos sin procesar en valores de características pequeños y extender el tiempo de uso de la batería.

2.2. CARACTERISTICAS Y FASES DE UNA CAÍDA

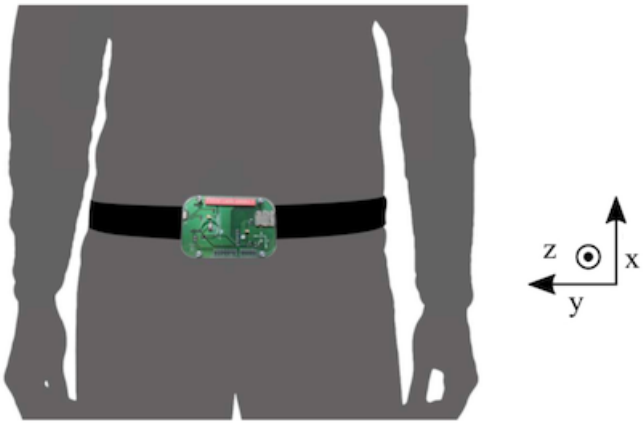
En el curso de un movimiento, la aceleración del cuerpo humano cambia en tiempo real. En un estudio de Erdogan y Bilgin [20], se demostró que la parte superior del torso del cuerpo humano (es decir, por encima de la cintura y por debajo del cuello) es el lugar óptimo para adquirir datos de aceleración y para distinguir las caídas de otras actividades. Teniendo en cuenta la comodidad del dispositivo portátil y la confiabilidad del sistema, el mejor lugar para colocar el dispositivo es la cintura y establecer un modelo de movimiento humano basado en un sistema de coordenadas de aceleración a lo largo de los ejes x, y y z como se observa en la figura 1.

---

[19] CHANGHONG, Wang, *et al.* "Low-power fall detector using triaxial accelerometry and barometric pressure sensing". En: *IEEE Trans. Ind. Informat* (2016), págs. 2302-2311.

[20] ERDOGAN, Zafer y BILGIN, Turgay. "A data mining approach for fall detection by using k-nearest neighbor algorithm on wireless sensor network data". En: *IET Commun* (2012), págs. 3281-3287.

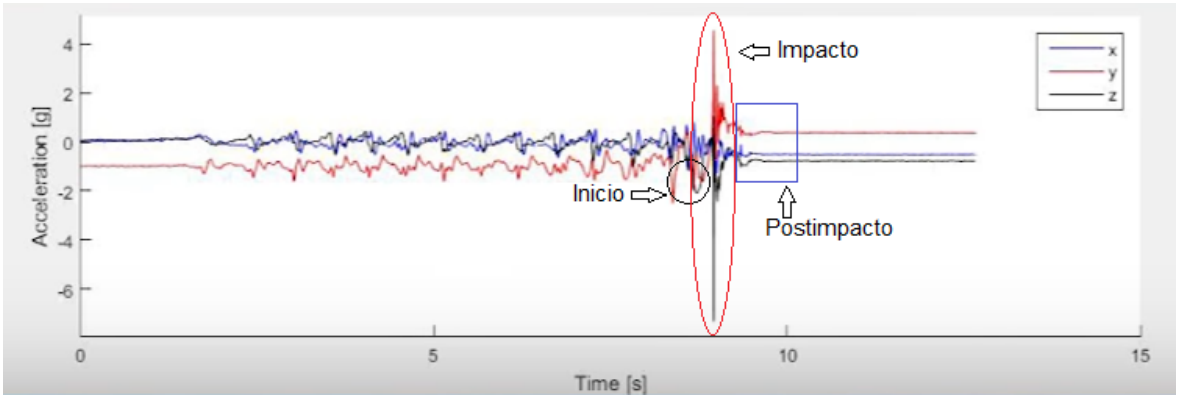
Figura 1. Modelo de movimiento.



SisFall: A Fall and Movement Dataset [figura]. [Consultado: 30 de Mayo de 2021] Disponible en: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5298771/

En la figura 2 se observan las fases de una caída sobre la señal de aceleración de un sensor ubicado en la cintura del usuario de acuerdo a la figura 1. Las etapas de una caída se describen de la siguiente manera:

Figura 2. Fases de una caída.



Fuente: Elaboración propia

- Inicio: se pierde el equilibrio y sobre la señal de aceleración puede verse reflejado una disminución en el valor de la gravedad que se registra mientras el usuario aún se encuentra de pie.

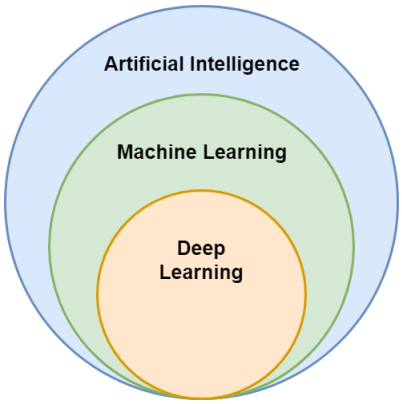
- Impacto: ocurre el impacto contra el suelo y se alcanza un pico de aceleración.
- Postimpacto: durante esta fase, el sujeto se mantiene inmóvil independientemente de la gravedad de la caída.

2.3. INTELIGENCIA ARTIFICIAL

En su forma más simple, la inteligencia artificial es un campo que combina la informática y conjuntos de datos para permitir la resolución de problemas. También abarca subcampos de aprendizaje automático y aprendizaje profundo, que se mencionan con frecuencia junto con la inteligencia artificial. Estas disciplinas están compuestas por algoritmos de inteligencia artificial que buscan crear sistemas expertos que hacen predicciones o clasificaciones basadas en datos de entrada.

2.3.1. Aprendizaje profundo (*deep learning*) vs. aprendizaje automático (*machine learning*) Dado que el aprendizaje profundo y el aprendizaje automático tienden a usarse indistintamente, vale la pena señalar los matices entre los dos. Como se mencionó anteriormente, tanto el aprendizaje profundo como el aprendizaje automático son subcampos de la inteligencia artificial, y el aprendizaje profundo es en realidad un subcampo del aprendizaje automático.

Figura 3. Interrelación entre diferentes técnicas computacionales inteligentes.



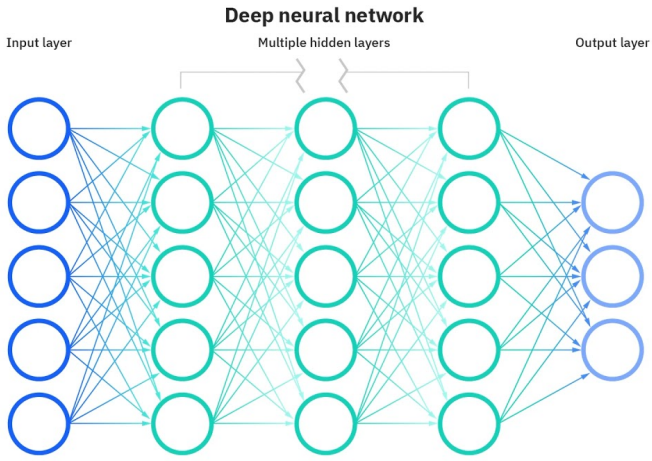
Fuente: What is artificial intelligence [figura]. [Consultado: 30 de Mayo de 2021] Disponible en: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

La forma en que el aprendizaje profundo y el aprendizaje automático difieren es en cómo aprende cada algoritmo. El aprendizaje profundo automatiza gran parte de la parte de extracción de características del proceso, eliminando parte de la intervención humana manual requerida y permitiendo el uso de conjuntos de datos más grandes. Se puede pensar en el aprendizaje profundo como un ”aprendizaje automático escalable”, como señaló Lex Fridman en una conferencia del MIT. El aprendizaje automático clásico o ”no profundo” depende más de la intervención humana para aprender. Los desarrolladores determinan la jerarquía de características para comprender las diferencias entre las entradas de datos, lo que generalmente requiere datos más estructurados para aprender.

2.3.2. Red neuronal Las redes neuronales, también conocidas como redes neuronales artificiales (ANN, por sus siglas en inglés), son un subconjunto del aprendizaje automático y están en el corazón de los algoritmos de aprendizaje profundo. Su nombre y estructura están inspirados en el cerebro humano, imitando la forma en que las neuronas biológicas se comunican entre sí.

Las redes neuronales artificiales se componen por un conjunto de nodos conocidos como neuronas artificiales que están conectadas y transmiten señales entre sí. Contienen una capa de entrada, una o más capas ocultas y una capa de salida. Cada nodo, o neurona artificial, se conecta a otro y tiene un peso y un umbral asociados. Si la salida de cualquier nodo individual está por encima del valor de umbral especificado, ese nodo se activa y envía datos a la siguiente capa de la red. De lo contrario, no se transmiten datos a la siguiente capa de la red.

Figura 4. Arquitectura de una red neuronal.

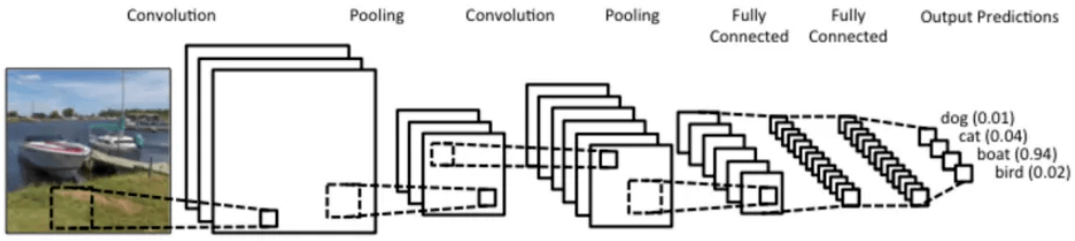


Fuente: What is artificial intelligence [figura]. [Consultado: 30 de Mayo de 2021] Disponible en: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

Las redes neuronales se basan en los datos de entrenamiento para aprender y mejorar su precisión con el tiempo. Una vez que estos algoritmos de aprendizaje se ajustan con precisión, son herramientas poderosas en informática e inteligencia artificial, lo que permite clasificar y agrupar datos a alta velocidad. Las tareas de reconocimiento de voz o reconocimiento de imágenes pueden tardar minutos en lugar de horas en comparación con la identificación manual realizada por humanos. Una de las redes neuronales más conocidas es el algoritmo de búsqueda de *Google*.

2.3.3. Red neuronal convolucional Las redes neuronales convolucionales (CNN, por sus siglas en ingles), son un tipo de red neuronal con aprendizaje supervisado que procesa sus capas imitando la corteza visual del ojo humano para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos y ”ver”.

Figura 5. Arquitectura de una red neuronal convolucional.



Fuente: Redes Neuronales Convolucionales en Profundidad [figura]. [Consultado: 30 de Mayo de 2021] Disponible en: https://datasmarts.net/es/redes-neuronales-convolucionales-en-profundidad/

El comportamiento de cada neurona se define por sus pesos. Cuando se alimentan con los valores de los píxeles, las neuronas artificiales de una CNN seleccionan varias características visuales.

Cuando ingresa una imagen en una CNN, cada una de sus capas genera varios mapas de activación. Los mapas de activación resaltan las características relevantes de la imagen. Cada una de las neuronas toma un parche de píxeles como entrada, multiplica sus valores de color por sus pesos, los suma y los ejecuta a través de la función de activación.

La primera capa (o inferior) de la CNN generalmente detecta características básicas

como bordes horizontales, verticales y diagonales. La salida de la primera capa se alimenta como entrada de la siguiente capa, que extrae características más complejas, como esquinas y combinaciones de bordes. A medida que avanza en la red neuronal convolucional, las capas comienzan a detectar características de nivel superior, como objetos, caras y más.

La operación de multiplicar los valores de los píxeles por pesos y sumarlos se llama ”convolución” (de ahí el nombre de red neuronal convolucional). Una CNN generalmente se compone de varias capas de convolución, pero también contiene otros componentes. La capa final de una CNN es una capa de clasificación, que toma la salida de la capa de convolución final como entrada.

Según el mapa de activación de la capa de convolución final, la capa de clasificación genera un conjunto de puntuaciones de confianza (valores entre 0 y 1) que especifican la probabilidad de que la imagen pertenezca a una ”clase”.

2.4. UNIDAD DE CONTROL

Uno de los componentes importantes a considerar al desarrollar una aplicación es la selección de la unidad de control (MCU, por sus siglas en inglés). Una MCU es un circuito integrado semiconductor inteligente que consta de una unidad de procesador, módulos de memoria, interfaces de comunicación y periféricos. La MCU se utiliza en una amplia gama de aplicaciones, incluidas lavadoras, robots, drones, radio y controladores de juegos.

Si bien una MCU tiene una unidad de procesador, es más que realizar operaciones aritméticas en valores binarios. El verdadero valor de una MCU es su capacidad para interactuar con el mundo físico con sus periféricos y comunicación incorporados.

Cuando se enciende, la MCU comenzará a ejecutar la instrucción cargada como datos de programa. Utiliza completamente la RAM para almacenar variables de tiempo de ejecución como lo indica el programa. Como se mencionó, las MCU están diseñadas para interactuar con el mundo físico. En la forma más simple, una MCU detectará las entradas y controlará las salidas de acuerdo con la lógica que fue programada.

Es importante que se preste especial atención a la elección de una MCU para el diseño. Hay numerosos factores a tener en cuenta al elegir una MCU para una aplicación portable de inteligencia artificial como lo son la velocidad de procesamiento y el bajo consumo.

2.4.1. La arquitectura Kendryte K210 Kendryte K210 es un sistema en *chip* (SoC, por sus siglas en inglés) que integra capacidades de procesamiento para visión y audio. El K210 implementa dos soluciones de inteligencia artificial y una tercera híbrida; la primera es la visión automática, con capacidad para detectar objetos, clasificar imágenes, detectar y reconocer rostros, obtener tamaño y coordenadas de un objetivo en tiempo real y obtener el tipo del objetivo detectado también en tiempo real. La segunda solución es audición automatizada, ya que el *chip* viene con un procesador de audio de matriz de micrófonos de alto rendimiento para la orientación y formación de la fuente en tiempo real, la cual permite detectar la orientación de la fuente del sonido, despertador de voz y reconocimiento de voz. La tercera solución es una forma híbrida de las dos anteriores que combina ambas para lograr un mayor rendimiento. Además el SoC ofrece un rendimiento de 0.25 TOPS (TeraOperaciones Por Segundo) en su frecuencia base 400 MHz en procesamiento de redes neuronales. A grandes rasgos, incorpora los siguientes elementos principales, tal y como se observa en la figura 6:

- CPU: Equipa dos núcleos, cada uno con una FPU independiente de 64 bits de alto rendimiento y bajo consumo basado en RISC-V. La frecuencia de funcionamiento es regulable desde 400 MHz hasta 800 MHz, y una SRAM en chip de 8 MB, además de soportar cálculos de coma flotante de precisión doble y simple. Tiene 2 memorias caches, una para instrucciones y otra para datos, ambas de 32 kB por núcleo.

- KPU: Es un procesador de redes neuronales capaz de acelerar procesos de inferencia sobre modelos previamente entrenados. Alguna de sus características son la integración de operaciones de convolución, normalización y aplicación de funciones de activación. El tamaño máximo del modelo en punto fijo para el trabajo a tiempo real es de 5 MB a 5.9 MB y además soporta kernels de convolución de 1x1 y 3x3.

- APU: Es un módulo de preprocesamiento cuya tarea es el preproceso de la dirección de la voz y su salida de datos. Algunas de sus características son la capacidad para 8 canales de datos de entrada y el soporte de datos de ancho de entrada de 12, 16, 24 y 32 bits. Tiene una frecuencia de muestreo de hasta 182 KHz y utiliza DMAC para almacenar en la memoria del sistema datos de salida.

Estos son los periféricos que incorpora el chip:

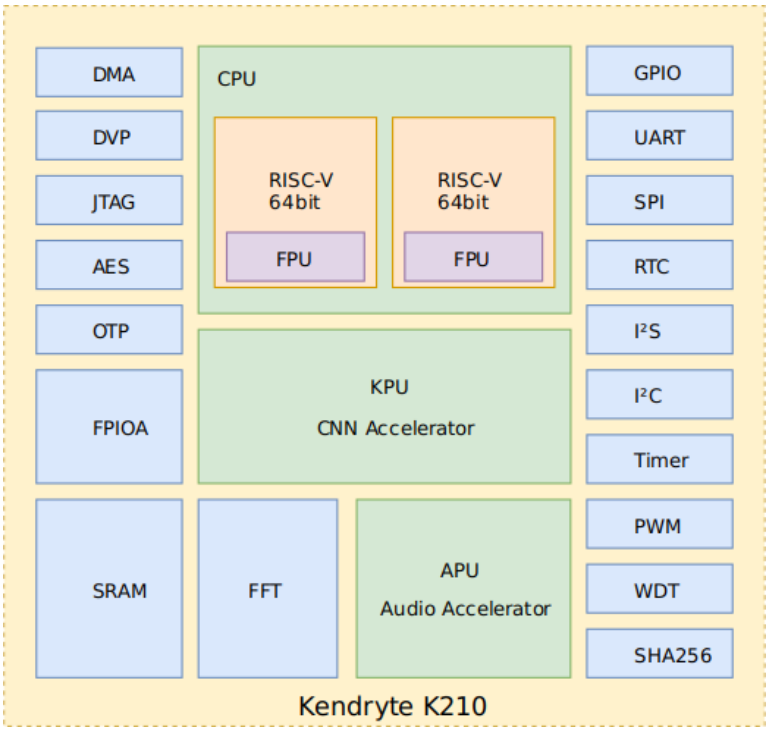
- Memoria estática de acceso aleatorio (SRAM) de 8 MB, dividida en 2 partes: una de uso general de 6 MB donde se almacenarán los pesos del modelo y otros parámetros y otra de 2 MB donde se almacenarán los mapas de características de entrada y salida.

- Memoria programable de una sola vez (OTP).

- Acelerador AES.

- Puerto de video digital (DVP).
- Acelerador FFT.
- Acelerador SHA256.
- Puerto UART.
- Temporizador de vigilancia (WDT).
- Interfaz de entrada / salida de uso general (GPIO).
- Controlador de acceso directo a memoria (DMAC).
- Bus de circuito inter-integrado (I2C).
- Sonido Inter-Integrado (I2S).
- Interfaz periférica en serie (SPI).
- Temporizador.
- Memoria de solo lectura (ROM).
- Reloj de tiempo real (RTC).
- Modulación de ancho de pulso (PWM).

Figura 6. Arquitectura del chip Kendryte K210.

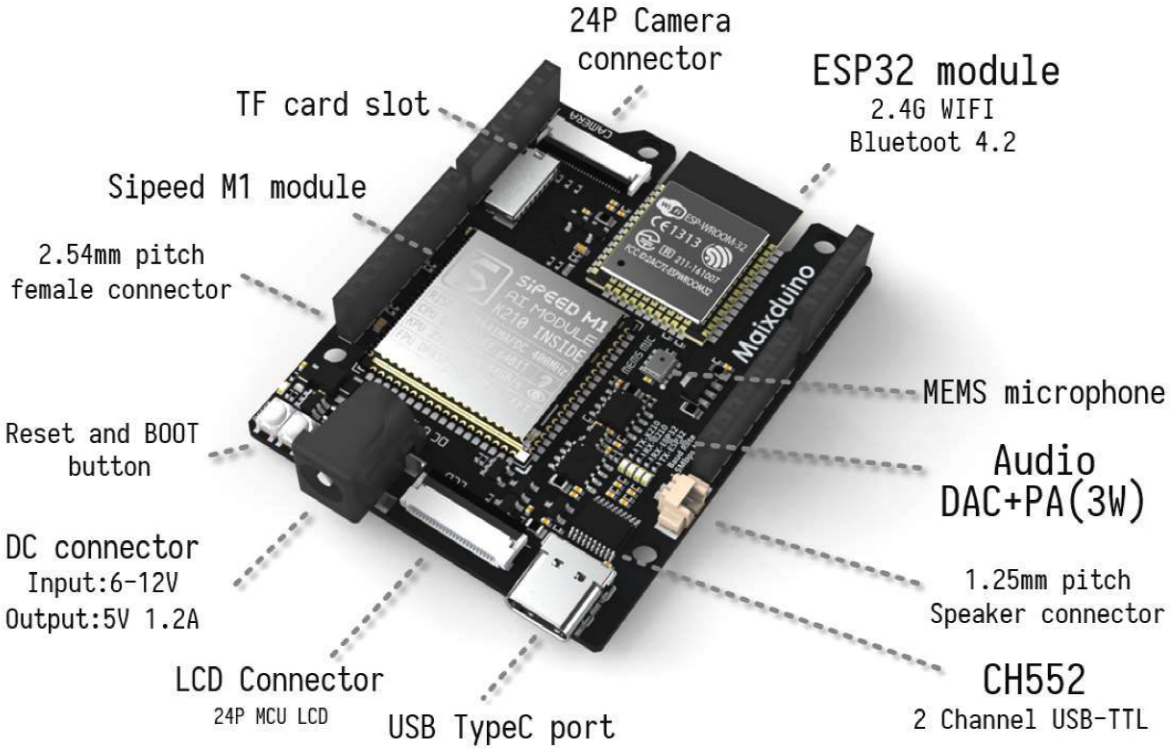


Fuente: Arquitectura del chip Kendryte K210 [figura]. [Consultado: 30 de Mayo de 2021] Disponible en: https://github.com/kendryte/kendryte-doc-datasheet/blob/master/en/001.md

2.4.2. Sipeed Maixduino Maix es la serie de productos de Sipeed especialmente diseñados para ejecutar inteligencia artificial usando *edge computing*. Al establecer servicios de computación cerca de sus ubicaciones, los usuarios obtienen servicios más rápidos, confiables, privados y a menor costo. Maix no es solo una solución de *hardware*, combina *hardware* personalizado, *software* abierto y algoritmos de inteligencia artificial de última generación. Diferentes tipos de placas de desarrollo, kits, periféricos y una amplia compatibilidad permiten un desarrollo de prototipos rápido y ágil, hacen que los proyectos *AIoT* sean mucho más fáciles. Y gracias al rendimiento, tamaño reducido, bajo consumo de energía y bajo costo de Maix, permite la implementación amplia de *Edge AI* de alta calidad.

El kit de desarrollo Maixduino está basado en el procesador K210 para aplicaciones de inteligencia artificial e internet de las cosas, con un precio en la página oficial de 28.40 Dólares (Consultado el 30 de mayo de 2021)

Figura 7. Tarjeta Maixduino.



Fuente: Placa de desarrollo Sipeed Maixduino [figura]. [Consultado: 30 de Mayo de 2021] Disponible en: https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html

La tarjeta Maixduino cuenta con un módulo ESP32 para realizar conexiones *Wifi* 2.4G y *Bluetooth* 4.2, conector DC de alimentación entre 6 - 12 V, botón de reinicio y de arranque, conector LCD, puerto USB tipo C, conector para cámara de 24P, puerto para tarjeta micro SD y micrófono.

La placa se puede programar mediante los *sotfware* MaixPy IDE, Arduino, OpenMV IDE y PlatformIO IDE.

Dentro de las principales aplicaciones de esta tarjeta de desarrollo tenemos:

- Desarrollos de hogar inteligentes, como robots de limpieza, altavoces, cerraduras electrónicas.
- Aplicaciones en la industria médica como diagnóstico, reconocimiento de imágenes médicas, alarmas de emergencia.
- Desarrollo para la industrial en general como clasificaciones inteligentes, supervisión de plagas, supervisión de equipos electrónicos.

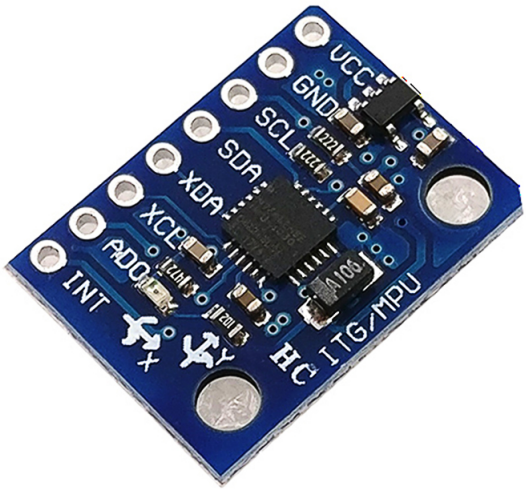
2.5. SENSOR DE MOVIMIENTO

2.5.1. Sensor inercial MPU6050 Los sensores miden los eventos detectados o cambios en el entorno e informan al dispositivo maestro para tomar decisiones. También miden las cantidades físicas y lo convierte en digital, por lo que los circuitos electrónicos entienden la información que están recibiendo. Las tecnologías de sensores han desempeñado un papel crucial en la transformación de la automatización del hogar, automóvil, la aviación, los dispositivos médicos y manufactura. Por lo tanto, los sensores de los sistemas integrados actuales deben fabricarse con precisión desde un punto de vista eléctrico y mecánico, el consumo de energía y tamaño de un sensor también juega un papel importante cuando se desarrolla una aplicación de energía de batería portátil.

Los acelerómetros son sensores dedicados a medir la aceleración de un cuerpo a través del efecto de fuerzas sobre una masa, sistema ha sido reemplazado en los

sensores miniaturizados por un objeto de silicio muy pequeño y se mide su interacción con campos magnéticos, esto es gracias a que los recientes avances en la fabricación de sensores resolvieron el problema de potencia y tamaño al aumentar la relación de transistores en un micro. Los giroscopios tienen como función medir la velocidad angular de un cuerpo en función del llamado efecto coriolis, con base a estas mediciones es posible calcular los cambios de orientación de un objeto.

Figura 8. Módulo MPU6050.



El sensor MPU6050 contiene todo lo necesario medir movimiento en 6 grados de libertad, combinando un giroscopio de 3 ejes y un acelerómetro de 3 ejes en un mismo *chip*. Integra un procesador digital de movimiento capaz de realizar complejos algoritmos de captura de movimiento de 9 ejes.

Se comunica a través de una interfaz I2C y posee una librería muy difundida para su uso inmediato. Este sensor entrega 6 grados de libertad e incorpora un regulador

de tensión a 3.3V y resistencias *pull-up* para su uso directo por I2C.

MPU6050	Rango de medida	Frecuencia de muestreo
Acelerómetro	$\pm$2g, $\pm$4g, $\pm$8g, $\pm$16g	1Khz
Giroscopio	250°/s, 500°/s, 1000°/s, 2000°/s	8KHz

Tabla 1. Especificaciones del sensor MPU6050.

2.6. INFRAESTRUCTURA DEL SOFTWARE

2.6.1. TensorFlow y TensorFlow Lite TensorFlow [21] es una plataforma de código abierto que proporciona herramientas, librerías y una colección de flujos de trabajo para desarrollar y entrenar modelos usando lenguajes como Python o JavaScript sobre CPUs, GPUs o TPUs (Tensor Processing Units). Con ello, es capaz de construir y entrenar redes neuronales con el fin de desarrollar aplicaciones que puedan, por ejemplo, detectar y clasificar objetos. Esta plataforma proporciona una API de alto nivel llamada Keras [22], para construir y entrenar modelos de aprendizaje profundo. En las versiones actuales de Tensorflow (2.0 en adelante), Keras está integrado dentro de Tensorflow, caso contrario de las versiones anteriores a 2.0. Tensorflow Lite [23] es un marco de trabajo de aprendizaje profundo de código abierto para la inferencia en dispositivos, donde podemos convertir un modelo TensorFlow en un *buffer* plano comprimido con el convertidor de TensorFlow Lite; con ello, es posible partir de un comprimido en formato TFLite, cargarlo en un dispositivo móvil

---

[21] "TensorFlow". En: *www.tensorflow.org* (2021).

[22] "Keras". En: *https://keras.io/* (2021).

[23] "TensorFlow Lite". En: *https://www.tensorflow.org/lite* (2021).

y también si fuera necesario cuantizar el modelo convirtiendo los floats de 32 bits (la precisión por defecto) a enteros de 8 bits para una mayor optimización y eficiencia.

2.6.2. KModel Es un formato de modelo utilizado actualmente en placas Maix equipadas con el procesador Kendryte K210. Este modelo nos permite utilizar dos tipos de precisiones: float32, que ofrece mayor precisión a coste de más memoria y menos velocidad e int8 (enteros de 8 bits) que proporcionan una mayor velocidad a un bajo coste de memoria, pero con menor precisión. Podemos encontrar dos versiones de este modelo dependiendo de la versión del compilador que usemos al compilar el modelo en formato TFLite. Usando la versión 0.1.0 RC5 o anteriores del compilador NNcase resultaría en un KModel versión 3 y con la versión 0.2.0 Alpha1 en adelante del compilador un KModel versión 4. Hay que tener en cuenta que, por el momento, la versión 4 del KModel no soporta algunas de las operaciones claves en inferencia, y por tanto estas son mapeadas sobre la CPU; entre ellas destaca la operación MaTMul, por lo que se obtiene un peor rendimiento frente a la versión 3. Su consumo de memoria es, además, mayor. Sin embargo, soporta un número de modelos de red neuronal mucho mayor que las versiones anteriores.

2.6.3. La herramienta NNCase NNCase [24] es una herramienta para compilar redes neuronales para aceleradores de inteligencia artificial y para realizar inferencia, siendo ncc la orden de línea de comando ofrecido por la herramienta para interactuar con el usuario, ncc ofrece dos comandos diferenciados (véase la figura 9 para más información):

- *compile*: compila los modelos entrenados (TFLite, Caffemodel y Onnx) a KModel.

[24] "NNCase". En: *https://github.com/kendryte/nncase* (2021).

- *infer*: ejecuta el KModel; en este caso, ncc almacenará los tensores de salida del modelo en archivos bin en formato NCHW.

Figura 9. Parámetros del NNCase.

```
DESCRIPTION
NNCASE model compiler and inference tool.

SYNOPSIS
    ncc compile <input file> <output file> -i <input format> [-o <output
        format>] [-t <target>] [--dataset <dataset path>] [--dataset-format
        <dataset format>] [--inference-type <inference type>] [--input-mean
        <input mean>] [--input-std <input std>] [--dump-ir] [--input-type <input
        type>] [--max-allocator-solve-secs <max allocator solve secs>]
        [--calibrate-method <calibrate method>] [-v]

    ncc infer <input file> <output path> --dataset <dataset path>
        [--dataset-format <dataset format>] [--input-mean <input mean>]
        [--input-std <input std>] [-v]
```

Fuente: Parámetros de los comandos compile e infer [Consultado 30 de mayo del 2021] Disponible en: https://github.com/kendryte/nncase/blob/master/docs/USAGE_EN.md

2.7. VALIDACIÓN DE ALGORITMOS DE RECONOCIMIENTO DE PATRONES

Existen diferentes parámetros que se utilizan para evaluar el desempeño de un clasificador. Estos parámetros toman como criterio los aciertos y errores que se dan en la clasificación. Si se definen las clases A y B como positiva y negativa respectivamente, se manejan cuatro tipos de resultados que son:

- Verdaderos positivos (TP): Los verdaderos positivos son aquellos datos que pertenecen a la clase A y son clasificados dentro de esta.
- Falsos positivos (FP): Son los datos pertenecientes a la clase B que son clasificados dentro de la clase A.
- Verdaderos negativos (TN): Son los datos pertenecientes a la clase B que son clasificados dentro de esta.

- Falsos negativos (FN): Son aquellos datos pertenecientes a la clase A que son clasificados como pertenecientes a la clase B.

A partir de las definiciones de TP, FN, FP, y TN se plantea la matriz de confusión la cual es una representación que permite visualizar el desempeño de un clasificador. Las columnas representan el número de predicciones de cada clase y las representan las instancias en la clase real. La matriz de confusión queda definida así:

$$\begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix}$$

A partir de la matriz de confusión es posible calcular los parámetros de desempeño más populares que son:

- Precisión: Esta medida indica el porcentaje de efectividad del clasificador.

$$Precisión = \frac{TP}{TP + FP} \tag{1}$$

- Especificad: Esta medida también es conocida como la tasa de falsos positivos.

$$Especifidad = \frac{TN}{TN + FP} \tag{2}$$

- Sensibilidad: La sensibilidad mide el desempeño en la clasificación de verdaderos positivos, es decir miembros de la clase considerada verdadera.

$$Sensibilidad = \frac{TP + TN}{TP + FN + FP + TN} \tag{3}$$

3. DESARROLLO DEL PROYECTO

Para realizar el reconocimiento de patrones de movimiento, existen varias posibilidades, está la detección de objetos utilizando algoritmos basados en umbrales, métodos estadísticos de aprendizaje, máquinas de soporte vectorial, entre otras. Pero antes de escoger uno de estos métodos para desarrollar el proyecto, se consideraron varios sistemas de desarrollo en el que se iba a implementar. Raspberry Pi, por su capacidad de procesamiento y la amplia información que se encuentra en la red, era una de las principales opciones, sin embargo, el precio de esta tarjeta de desarrollo de 35 dólares (Consultado el 30 de mayo del 2021), su tamaño y su elevado consumo energético fue la principal causa para desistir y buscar otra alternativa. Sipeed estudio es una compañía que desde el año 2008 se especializa en soluciones de *hardware* para *IoT* e inteligencia artificial a bajo costo y bajo consumo, ideal para el desarrollo del proyecto, es por esta razón que se escogió la tarjeta de desarrollo Maixduino de esta compañía, la cual viene integrada con casi todo lo que se necesita para hacer el reconocimiento de actividades.

El módulo K210 de las tarjetas de desarrollo Maix, procesa redes neuronales en formato KModel de hasta 5 MB de tamaño, por lo que se debía escoger una opción de entrenamiento cuyo resultado fuera ligero y se pudiera convertir al formato necesitado.

3.1. ALGORITMO DE DETECCIÓN DE CAIDAS

Según lo expuesto en [25] [26], las señales relacionadas con el movimiento de una persona se encuentran entre los 0.5 y 20 Hz, por lo que muestrear a 100 Hz permite conservar los componentes de frecuencia necesarios cumpliendo el criterio de Nyquist.

Como el tiempo desde la caída hasta el contacto con el suelo suele ser inferior a 2 segundos y el resultado experimental de [27] mostró que una ventana deslizante de 2 segundos es mejor que una de 1 y 3 segundos, se selecciona una ventana deslizante de 2 segundos para almacenar la aceleración triaxial.

3.1.1. Visualización de los datos de la actividad En el microcontrolador, la ventana deslizante almacena en caché los 200 datos de aceleración triaxial equivalente de la actividad humana durante 2 segundos. Si los 3 ejes del modelo movimiento humano se consideran como los 3 canales de una imagen RGB, los datos de los ejes x, y y z se pueden convertir y asignar como el valor de un píxel de una imagen RGB, respectivamente. Es decir, las 200 piezas de datos triaxiales almacenados en caché en la ventana deslizante se pueden ver como una imagen con un tamaño de 20 por 10 píxeles. El esquema de la figura 10 ilustra la forma de mapear las aceleraciones triaxiales en una imagen.

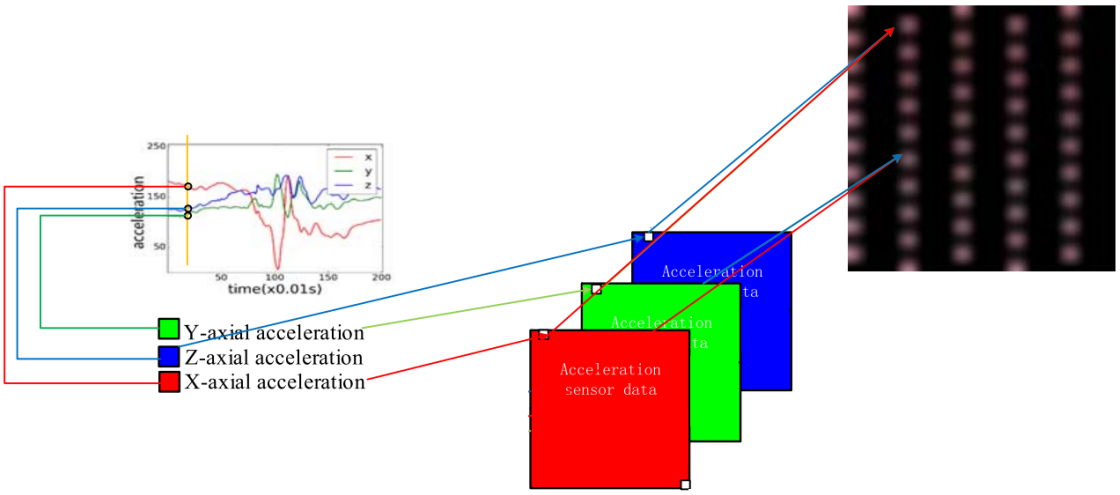
---

[25] HARLE, Robert. “A survey of indoor inertial positioning systems for pedestrians”. En: *IEEE Communications Surveys and Tutorials* (2013), págs. 1281-1293.

[26] LADETTO, Quentin. “On foot navigation: continuous step calibration using both complementary recursive prediction and adaptive Kalman ltering”. En: *Proceedings of the 13th International Technical Meeting of the Satellite Division of The Institute of Navigation* (2000), págs. 1735-1740.

[27] ORDOÑEZ, Francisco y ROGGEN, Daniel. “Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition”. En: *Sensors* (2016), pág. 115.

Figura 10. Ilustración esquemática del mapeo de datos triaxiales en una imagen RGB.



Fuente: Elaboración propia

Debido a que el rango de los datos de una imagen RGB va desde 0 a 255 enteros, y el rango de los datos del acelerómetro va desde -20 a 20 flotantes, se usa una conversión lineal para normalizar los datos del acelerómetro al rango de 0 a 255 de acuerdo a la ecuación 8.

$$OldRange = (OldMax - OldMin) = (20 - (-20)) = 40 \tag{4}$$

$$NewRange = (NewMax - NewMin) = (255 - 0) = 255 \tag{5}$$

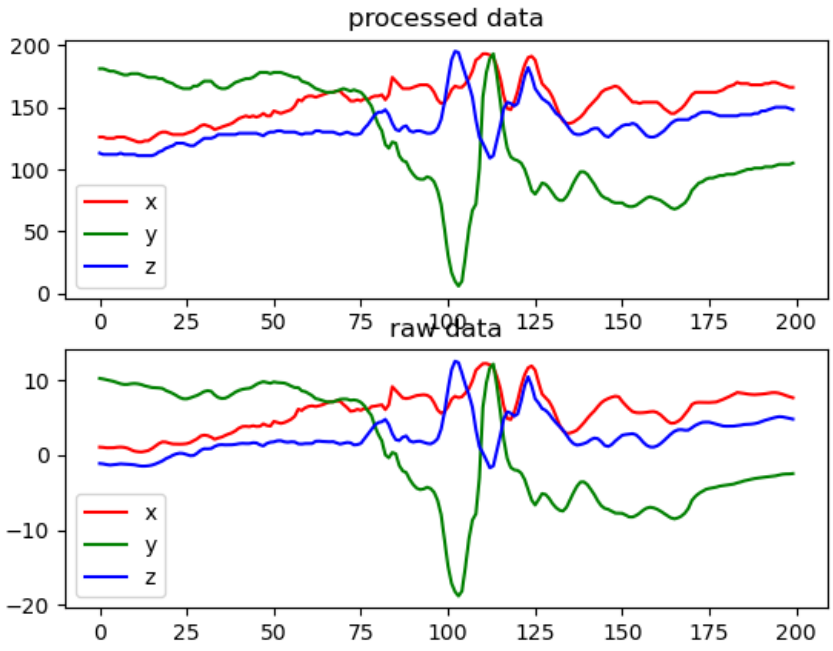
$$NewValue = \frac{(OldValue - Oldmin) * NewRange}{OldRange} + NewMin \tag{6}$$

$$NewValue = \frac{(OldValue + 20) * 255}{40} \tag{7}$$

$$NewValue \approx int((OldValue + 20) * 6) \quad (8)$$

La figura 11 compara el diagrama de líneas de las aceleraciones triaxiales sin procesar de 2 segundos para una caída con los datos triaxiales normalizados.

Figura 11. Comparación de los datos sin procesar y normalizados para una caída.



Fuente: Elaboración propia

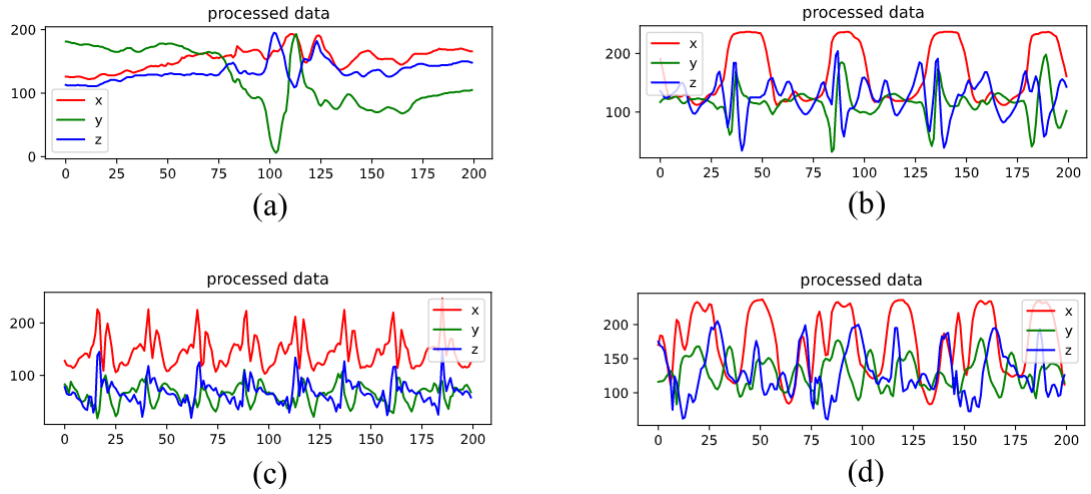
3.1.2. Comparación entre caídas y ADL Sobre la base de la normalización de rango de los datos de actividad, se comparan y analizan 4 tipos de actividades es decir, estar de pie, caminar, saltar y caer. De esta última se desprenden caídas específicas como:

- Caerse hacia adelante al caminar causado por un resbalón.
- Caerse hacia atrás al caminar causado por un resbalón.

- Caída lateral al caminar provocada por un resbalón.
- Caída vertical al caminar causada por desmayos.
- Caída lateral al estar sentado, causada por desmayos o por quedarse dormido.

La figura 12 muestra el gráfico de líneas de aceleración para los distintos tipos de actividades diarias. Se puede ver que las aceleraciones triaxiales de estar de pie, caminar y saltar muestran variaciones periódicas. La figura 12 muestra que la dirección de la aceleración de una caída presenta una gran diferencia respecto a las de otras ADL. Por lo tanto, las caídas se pueden distinguir de otras ADL mediante la selección de algoritmos de clasificación apropiados.

Figura 12. Diagrama de líneas de aceleraciones de ADL y caída.
a) Aceleración de una caída. b) Aceleración de saltar. c) Aceleración de caminar. d) Aceleración de trotar.

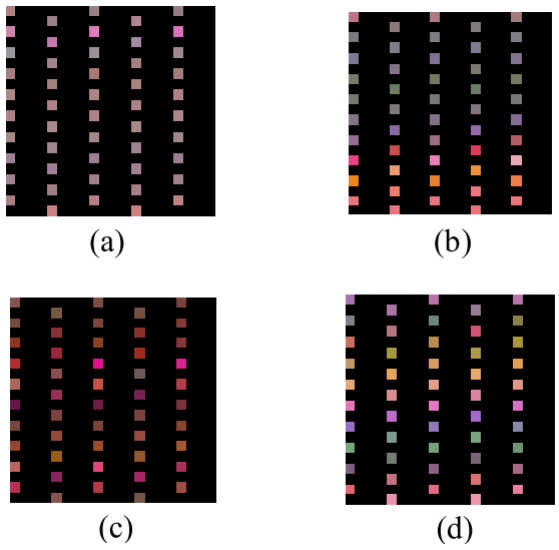


Fuente: Elaboración propia

La figura 13 muestra las imágenes correspondientes después de convertir los datos de una caída y las actividades diarias de la figura 12 en píxeles RGB. En la figura 13, se puede encontrar que la imagen de la caída es diferente al de las actividades

diarias, lo que proporciona la base para usar un algoritmo de clasificación basado en el reconocimiento de imágenes para identificar las caídas. Una red neuronal convolucional muestra una excelente precisión de reconocimiento para la detección y el reconocimiento de imágenes donde las “neuronas” corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico. Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes.

Figura 13. Imágenes de aceleraciones de ADL y caída.
a) Imagen de una caída. b) Imagen de saltar. c) Imagen de caminar. d) Imagen de trotar.



Fuente: Elaboración propia

3.2. BASE DE DATOS

La investigación sobre la detección de caídas y movimientos con dispositivos portátiles ha experimentado un crecimiento prometedor. Sin embargo, hay pocos *data-*

set disponibles públicamente, la gran mayoría registrados con teléfonos inteligentes, que son insuficientes para probar nuevas propuestas debido a la ausencia de población objetiva y la falta de actividades realizadas.

SisFall [28] es un *dataset* hecho en la facultad de ingeniería de la Universidad de Antioquia, el cual contiene datos de caídas y actividades de la vida diaria adquiridos con un dispositivo de desarrollo compuesto por dos tipos de acelerómetro y un giroscopio. Consta de 19 ADL y 15 tipos de caídas realizadas por 23 adultos jóvenes, 15 tipos de ADL realizadas por 14 participantes sanos e independientes mayores de 62 años y datos de un participante de 60 años que realizó todas las ADL y caídas.

El *dataset* se transformó de acuerdo con el sistema de coordenadas de la figura 1, para asegurar que los datos estén en las mismas coordenadas. Además, los datos son escalados de acuerdo a la ecuación 4 y luego segmentados cada 200 datos para crear un nuevo *dataset* con imágenes de 20 x 10 píxeles.

Actividad	**Muestras de entrenamiento**	**Muestras de prueba**
Caminar	1000	200
Trotar	1000	200
Saltar	1000	200
Caer	1000	200
De pie	500	100
Total	4500	900

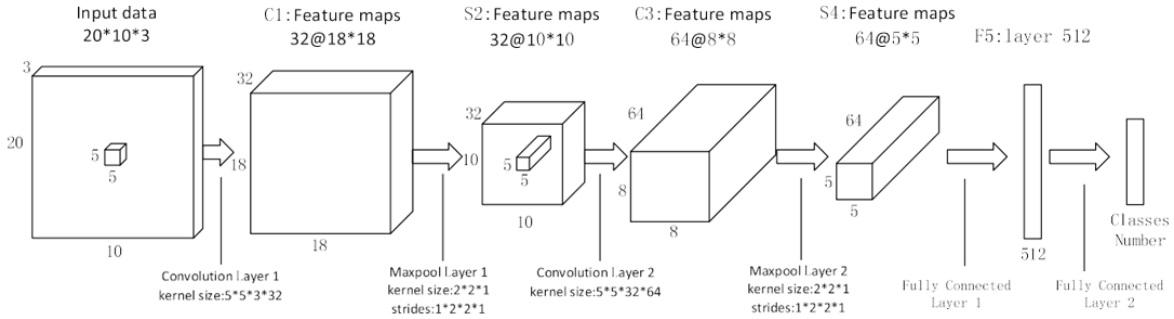
Tabla 2. *Dataset* para entrenamiento y prueba.

[28] SUCERQUIA, Angela; LÓPEZ, José y VARGAS BONILLA, Jesús. “SisFall: A fall and movement dataset”. En: *Sensors* (2017), pág. 198.

3.3. ARQUITECTURA DE LA RED NEURONAL CONVOLUCIONAL

Para la fase de entrenamiento se construyó la red neuronal desde cero usando como base *Google Colab*, el cual ofrece un entorno interactivo en línea llamado *notebook colab* el cual cuenta con acceso gratuito a una unidad de procesamiento gráfico (GPU, por sus siglas en inglés) ideal para realizar el entrenamiento de las redes neuronales, ya que no se necesita contar con una máquina potente. Además, *Colab* cuenta con la posibilidad de conectarse con *Google Drive* para cargar o guardar imágenes, modelos, resultados, entre otras cosas.

Figura 14. Arquitectura de la red neuronal convolucional.



Fuente: Elaboración propia

En la figura 8, la capa convolucional está etiquetada como Cx, la submuestra de la capa de submuestreo está etiquetada como Sx, y y la capa completamente conectada se etiqueta como Fx, donde x es el índice de la capa. La entrada es una imagen RGB de 20 × 10 de 3 canales.

La capa C1 es una capa convolucional con 32 mapas de características. Debido a que el K210 usa un método de relleno diferente al predeterminado de Keras, se hace la debida adaptación, por lo cual se rellenan los ceros de alrededor (izquier-

da, arriba, derecha, abajo), contrario a como lo hace Keras que utiliza un relleno de derecha y abajo. Para ello usaremos la función ZeroPadding2D para establecer el método de relleno que deseamos.

El tamaño del kernel convolucional es 5 × 5 × 3, solo se usa un kernel de convolución común en cada mapa de características. El kernel convolución avanza un píxel a la vez, el tamaño del mapa de características es de 18 × 18. Cada kernel de convolución tiene una unión de 5 × 5 × 3 parámetros, es decir, tiene 76 parámetros, y cada unidad se activa usando la función ReLu después de la convolución. C1 contiene 2432 parámetros entrenables.

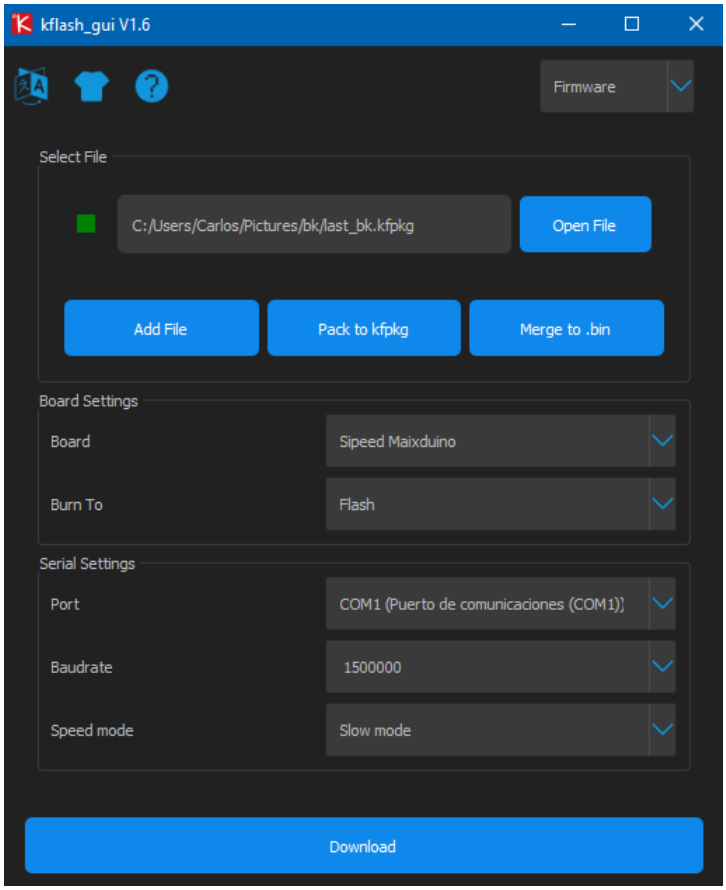
La capa S2 es una capa de submuestreo con 32 mapas de características. La capa C3 es una capa convolucional con 64 mapas de características. Antes de la convolución, cada mapa de características se expande rellenando con ceros.

F5 contiene 512 unidades y está completamente conectada a C4. Cada unidad se activa usando ReLu después de estar completamente conectada. Se usa la técnica de *dropout* donde las neuronas seleccionadas al azar se ignoran durante el entrenamiento para evitar él *overfitting* durante el entrenamiento de la red. Finalmente, la capa de salida está completamente conectada a F5 con 4 unidades. Softmax se usa para calcular la probabilidad de cada unidad, y la que tenga la probabilidad máxima será el resultado de la predicción.

3.4. CONFIGURACIÓN DEL MICROCONTROLADOR

3.4.1. *Firmware* y modelo Para quemar el *Firmware* y el modelo en la memoria *flash* se usó la herramienta kflash [29] como se puede observar en la figura 15.

Figura 15. Ejemplo configuración de la herramienta kflash.



Fuente: Elaboración propia

Para cargar el modelo entrenado se tienen 2 opciones. La primera opción es cargar el modelo a la memoria *flash* utilizando la aplicación kflash. La segunda forma es copiar el modelo a una memoria SD y desde allí correrlo.

---

[29] "kflash gui". En: *https://github.com/sipeed/kflash_gui* (2021).

3.4.2. Programación Maixduino Para la programación de la tarjeta se usó el *software* MaixPy IDE versión 0.2.5. Este programa utiliza como motor el entorno de Micropython y se necesitan dos librerías externas:

- network_esp32 versión 1.0 para la conexión *Wifi*.
- BlynkLib versión 0.2.0 para la conexión *IoT* con el dispositivo móvil.

El código está distribuido de la siguiente manera:

- Se importan las librerías base para el correcto funcionamiento entre ellas tenemos:
  - *machine*: conexión I2C.
  - *time*: permite introducir retrasos p. ej. se usa un retraso de 10 milisegundos para muestrear a 100 Hz.
  - *image*: permite crear las imágenes a partir de los datos del acelerómetro.
  - *kpu*: activa el procesador de redes neuronales.
- Se especifican las credenciales para las respectivas conexiones.
- Usando la librería network_esp32 se hace la respectiva conexión *wifi*.

```
#--------------------------------------------------------------------
import machine, utime, time, network, image
from machine import I2C
import BlynkLib
from BlynkTimer import BlynkTimer
import KPU as kpu

SSID = " WIFI ID "
```

```
PASW = " WIFI PASSWORD "
BLYNK_AUTH = ' BLYNK KEY '

def enable_esp32():
    from network_esp32 import wifi
    if wifi.isconnected() == False:
        for i in range(5):
            try:
                wifi.reset(is_hard=True)
                print('try AT connect wifi...')
                wifi.connect(SSID, PASW)
                if wifi.isconnected():
                    break
            except Exception as e:
                print(e)
    print('network state:', wifi.isconnected(), wifi.ifconfig())

enable_esp32()
#--------------------------------------------------------------------------
```

- Se inicia la conexión I2C especificando los pines del microcontrolador y la velocidad deseada de 400 kHz.
- Se especifican los registros de configuración del MPU6050.
- Se crean las funciones que permiten obtener los datos de aceleración usando los registros especificados.

```
#-------------------------------------------------------------------------
i2c = I2C (I2C.I2C0, freq = 400000, scl = 30, sda = 31)

#Direccion MPU6050
MPU6050_ADDR = 0x68

#Registros acelerometro
MPU6050_ACCEL_XOUT_H = 0x3B
MPU6050_ACCEL_XOUT_L = 0x3C
MPU6050_ACCEL_YOUT_H = 0x3D
MPU6050_ACCEL_YOUT_L = 0x3E
MPU6050_ACCEL_ZOUT_H = 0x3F
MPU6050_ACCEL_ZOUT_L = 0x40

#Enable
MPU6050_PWR_MGMT_1 = 0x6B

#Escala acelerometro = 16g
MPU6050_LSBG = (9.81 / 2048.0)

#SMPLRT_DIV
i2c.writeto_mem(MPU6050_ADDR, 0x19, 0b00000000)

#DLPF_FILTER 44Hz
i2c.writeto_mem(MPU6050_ADDR, 0x1A, 0b00000011)
```

```
#GYRO 2000
i2c.writeto_mem(MPU6050_ADDR, 0x1B, 0b00011000)

#ACEL 16g
i2c.writeto_mem(MPU6050_ADDR, 0x1C, 0b00011000)

#FIFO DISABLED
i2c.writeto_mem(MPU6050_ADDR, 0x23, 0b00000000)

def mpu6050_init(i2c):
    i2c.writeto_mem(MPU6050_ADDR, MPU6050_PWR_MGMT_1, 0b00001000)

def combine_register_values(h, l):
    if not h[0] & 0x80:
        return h[0] << 8 | l[0]
    return -(((h[0] ^ 255) << 8) |  (l[0] ^ 255) + 1)

def mpu6050_get_accel(i2c):
    accel_x_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_H, 1)
    accel_x_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_L, 1)
    accel_y_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_H, 1)
    accel_y_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_L, 1)
    accel_z_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_H, 1)
    accel_z_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_L, 1)

    return [combine_register_values(accel_x_h, accel_x_l) *
    ↪  MPU6050_LSBG,
```

```
            combine_register_values(accel_y_h, accel_y_l) *
            ↪ MPU6050_LSBG,
            combine_register_values(accel_z_h, accel_z_l) *
            ↪ MPU6050_LSBG]
#--------------------------------------------------------------------
```

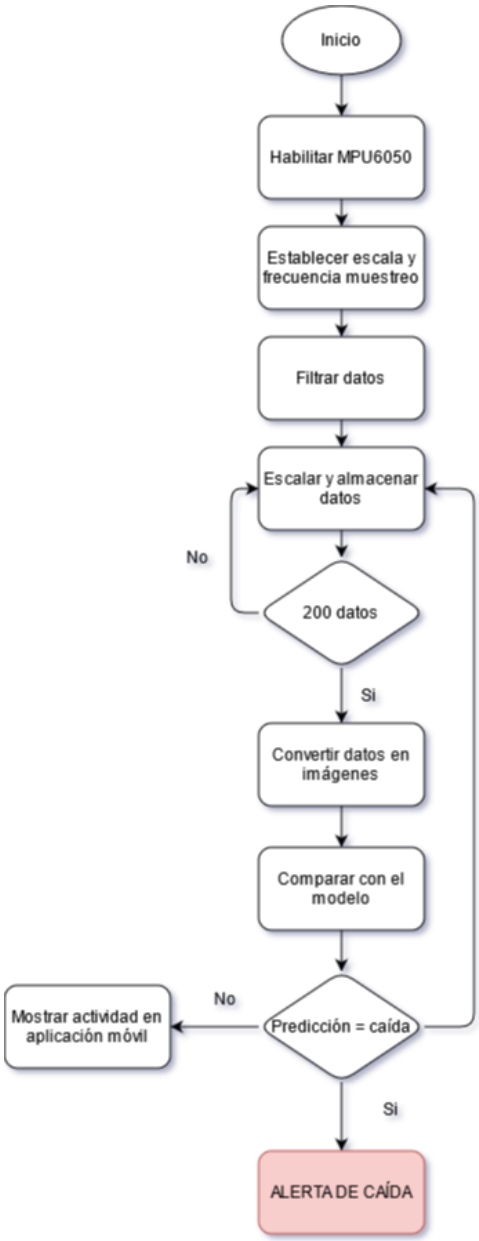
- Se carga el modelo previamente guardado en la memoria flash en la dirección de memoria 0x300000.
- Usando la librería *image* se crea una imagen vacía de 20 x 10 píxeles a la cual se le agregaran más adelante los datos escalados del acelerómetro.
- Se crean las funciones que permiten obtener los datos de aceleración usando los registros especificados.
- En el bucle principal se va a leer y escalar los datos de aceleración. Una vez haya 200 datos se va a formar la imagen de 20 x 10 píxeles.
- Con la imagen ya formada se procede a comparar con el modelo y a hacer la respectiva predicción.
- Finalmente el resultado va a ser enviado al dispositivo móvil indicando la actividad realiza y en caso de presentarse una emergencia envía una alerta.

```
#--------------------------------------------------------------------
mpu6050_init(i2c)
blynk = BlynkLib.Blynk(BLYNK_AUTH)
timer = BlynkTimer()
task = kpu.load(0x300000)
dummyImage = image.Image()
dummyImage = dummyImage.resize(20, 10)
```

```
image_data_array = []
counter = 0
notify = 0
LABELS = ['Walking', 'Jumping', 'Fall', 'Standing']

while True:
    blynk.run()
    timer.run()
    counter = counter + 1
    data = mpu6050_get_accel(i2c)
    accel_x = int((data[0] + 20) * 6)
    accel_y = int((data[1] + 20) * 6)
    accel_z = int((data[2] + 20) * 6)
    image_data_array.append([accel_x, accel_y, accel_z])
    if counter >= 200:
        for h in range(0, 10):
            for w in range(0, 20):
                dummyImage.set_pixel(w, h,
                (image_data_array[h * 20 + w][0], image_data_array[h *
                ↪  20 + w][1], image_data_array[h * 20 + w][2]))
        image_data_array.clear()
        dummyImage.pix_to_ai()
        fmap = kpu.forward(task, dummyImage)
        plist = fmap[:]
        pmax = max(plist)
        max_index = plist.index(pmax)
```

```
        def ResetNotification():
            global notify
            notify = 0

        def Notification():
            global notify
            if notify == 0:
                blynk.set_property(2, 'color', '#FF0000')
                blynk.notify('Se ha producido una emergencia')
                timer.set_timeout(30, ResetNotification)
                notify = 1

        @blynk.VIRTUAL_READ(2)
        def my_read_handler():
            blynk.set_property(2, 'color', '#FFFFFF')
            blynk.virtual_write(2, LABELS[max_index])
            if LABELS[max_index] == 'Fall':
                Notification()
            if LABELS[max_index] != 'Fall':
                ResetNotification()

        print (LABELS[max_index])
        counter = 0

    utime.sleep_ms(10)
#------------------------------------------------------------------------
```

Figura 16. Diagrama de flujo del algoritmo.



Fuente: Elaboración propia

El código, *dataset*, librerías y todo lo que se necesita para desarrollar este proyecto se puede encontrar en el siguiente repositorio GitHub. **(https://github.com/carlos-monsalve/Human-gait-pattern-and-fall-recognition-using-a-CNN/tree/main)**.

4. RESULTADOS

Para desarrollar un clasificador en tiempo real, debe realizarse previamente el entrenamiento del mismo. Esto para calcular su desempeño y tener un estimado del posterior funcionamiento en tiempo real.

En este capítulo se muestra el entrenamiento y la implementación en tiempo real del algoritmo de detección de actividades y caídas, así como el análisis de desempeño en cada caso y una comparación de los mismos.

4.1. ENTRENAMIENTO DE LA RED NEURONAL CONVOLUCIONAL

La red neuronal convolucional fue entrenada con 4500 imágenes como se especifica en la tabla 2. Durante el entrenamiento, el tamaño del lote se estableció en 512 y se utilizó el optimizador Adam. La tasa de aprendizaje fue 0,001 y el número de pasos de entrenamiento fue 10000. Además, Softmax se utilizó en la última capa de la red para calcular la probabilidad de los resultados de salida. Finalmente, la pérdida se calculó usando entropía cruzada. Cuando el entrenamiento de la red se iteró durante 8 epochs, la precisión del conjunto de verificación se mantuvo en 99,6 %, y la precisión del entrenamiento posterior se mantuvo estable. Entonces el entrenamiento de la red se completó y se guardó.

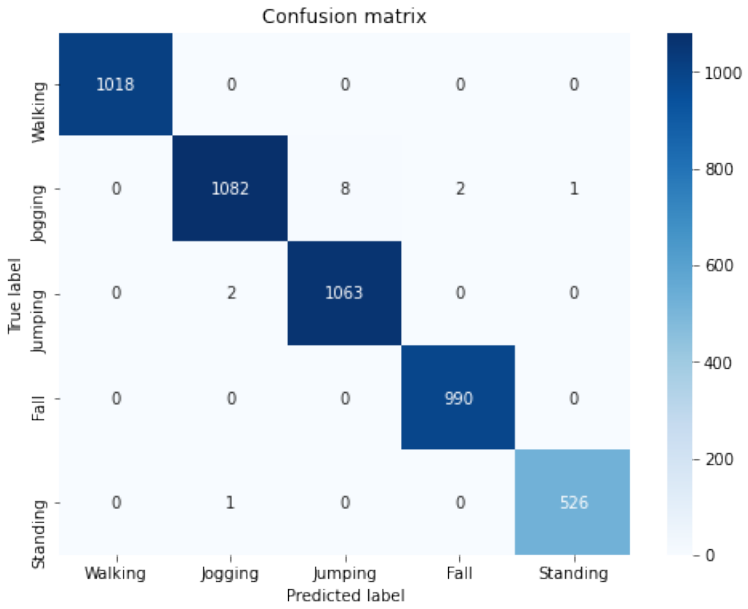
Figura 17. Entrenamiento de la red neuronal convolucional.

```
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_valid, y_valid), epochs=8, batch_size=512, verbose=2)
```

```
Train on 33746 samples, validate on 8437 samples
Epoch 1/8
33746/33746 - 13s - loss: 4.1957 - accuracy: 0.6924 - val_loss: 0.0885 - val_accuracy: 0.9720
Epoch 2/8
33746/33746 - 12s - loss: 0.1078 - accuracy: 0.9659 - val_loss: 0.0580 - val_accuracy: 0.9801
Epoch 3/8
33746/33746 - 11s - loss: 0.0671 - accuracy: 0.9793 - val_loss: 0.0392 - val_accuracy: 0.9883
Epoch 4/8
33746/33746 - 11s - loss: 0.0401 - accuracy: 0.9882 - val_loss: 0.0181 - val_accuracy: 0.9944
Epoch 5/8
33746/33746 - 11s - loss: 0.0361 - accuracy: 0.9890 - val_loss: 0.1166 - val_accuracy: 0.9603
Epoch 6/8
33746/33746 - 11s - loss: 0.0366 - accuracy: 0.9876 - val_loss: 0.0193 - val_accuracy: 0.9925
Epoch 7/8
33746/33746 - 11s - loss: 0.0207 - accuracy: 0.9935 - val_loss: 0.0112 - val_accuracy: 0.9959
Epoch 8/8
33746/33746 - 11s - loss: 0.0133 - accuracy: 0.9961 - val_loss: 0.0111 - val_accuracy: 0.9963
```

Fuente: Elaboración propia

Figura 18. Matrix de confusión del entrenamiento.



Fuente: Elaboración propia

4.2. IMPLEMENTACIÓN EN TIEMPO REAL DEL ALGORITMO

La prueba del sistema de detección de caídas se ha realizado basándose en el *hardware* y estructura de *software* descritos en el capítulo anterior. El prototipo fue diseñado y probado en un entorno controlado. Los datos se analizaron a partir de los resultados obtenidos de las pruebas hechas por un sujeto de pruebas que realiza 15 veces cada actividad. La figura 19 muestra el sistema diseñado para la detección de las actividades.

Figura 19. El modelo práctico.

Fuente: Elaboración propia

Figura 20. Ejemplo de pruebas realizadas para la validación de caídas.

Fuente: Elaboración propia

Actividad	Verdaderos Positivos Caídas	Verdaderos Positivos Caídas
Caída adelante	15	0
Caída atrás	13	2
Caída lateral	14	1
Caída vertical	15	0
Caída sentado	14	1
	Falsos Positivos Caídas	**Verdaderos Negativos Caídas**
De pie	0	15
Caminar	0	15
Saltar	1	14

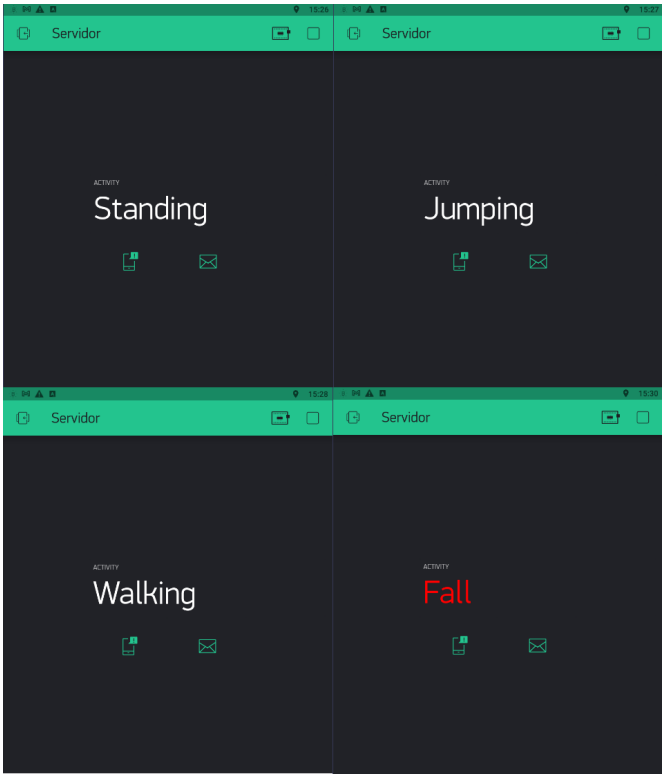
Tabla 3. Parámetros de desempeño para el algoritmo de detección de caídas.

La exactitud, sensibilidad y especificidad se determina utilizando ecuaciones 1, 2, 3 y los datos de prueba en la tabla 3. El sistema diseñado estima una precisión del 95.83 %, una sensibilidad del 94.66 % y una especificidad del 97.77 %.

4.3. APLICACIÓN MÓVIL

La aplicación fue desarrollada usando Blynk [30], el cual permite crear aplicaciones para dispositivos inteligentes basados en *Android* para controlar Arduino, Raspberry Pi y similares a través de *Internet*. Es un tablero digital donde puede crear una interfaz gráfica de manera sencilla simplemente arrastrando y soltando *widgets*.

Figura 21. Aplicación móvil.

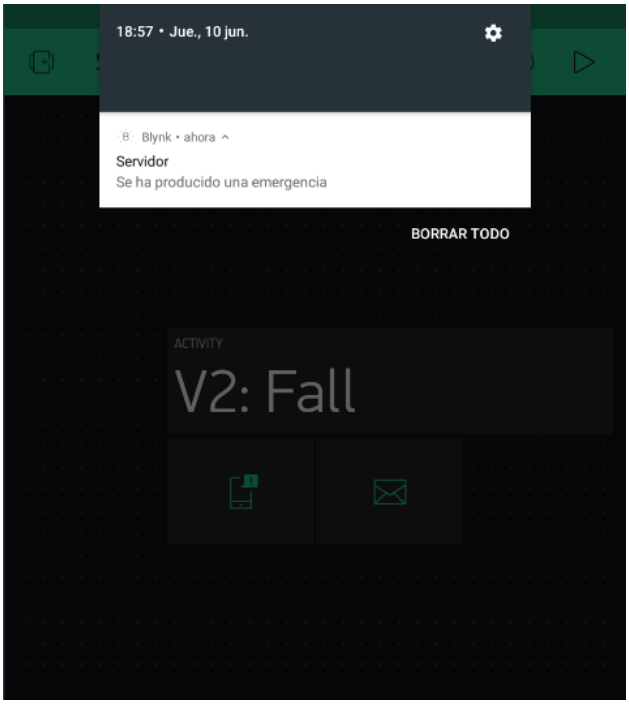


Fuente: Elaboración propia

[30] "Blynk". En: *https://blynk.io/* (2021).

En caso de presentarse una caída desde el sistema de detección se enviará una alerta de emergencia al teléfono del cuidador.

Figura 22. Notificación de alerta.



Fuente: Elaboración propia

5. CONCLUSIONES Y TRABAJO FUTURO

5.1. CONCLUSIONES GENERALES

En este trabajo de investigación, un dispositivo de detección de caídas, está diseñado para realizar la adquisición y transmisión de datos de actividad humana de manera eficiente. Se utilizaron unidades de medida inercial para tomar señales de aceleración con las cuales se realiza la detección de actividades. Una tarjeta de desarrollo Maixduino se utilizó para la adquisición y procesamiento de los datos, esta se programó usando el lenguaje Python, el cual brinda la posibilidad de usar librerías especializadas en el procesamiento de datos y una mayor simpleza en la implementación en comparación con otros lenguajes de programación existentes. Mientras tanto, inspirado en la idea de la codificación de imágenes RGB de 3 canales, los datos de aceleración de 3 ejes se mapean en una imagen RGB, y una red neuronal convolucional de detección de caídas está diseñada para distinguir las caídas de las ADL aprovechando las ventajas de la alta precisión para la detección de caídas, bajo consumo de energía, *IoT*, etc. Por tanto, es muy adecuado para la detección de caídas en la comunidad de personas mayores.

El sistema diseñado resultó ser óptimo cuando se usa en la cintura del usuario. Una de las limitantes del dispositivo se presenta en la detección de actividades más complejas como subir o bajar escaleras con actividades básicas como caminar o trotar, esto es debido a que solo un sensor no es suficiente para detectar las diferencias que se presentan entre estas actividades que siguen un patrón de movimiento similar produciendo así errores en la predicción. Para evitar esto, es necesario el uso de sensores auxiliares que ayuden a marcar diferencias entre actividades como un sensor de presión con el cual se pueden determinar cambios de altura.

El desarrollo de aplicaciones de inteligencia artificial a través de las tarjetas Maix de Sipeed trae ventajas desde el punto de vista económico respecto a otras tarjetas del mercado, ya que con una poca inversión se puede tener un sistema que ejecute algoritmos de inteligencia artificial. El mayor desafío al que se enfrenta un desarrollador que quiera usar una de estas tarjetas como base para su proyecto es la falta de documentación y soporte para librerías debido a que al ser tarjetas relativamente nuevas no hay una comunidad grande y activa apoyando el desarrollo de las mismas.

Google Colab es una excelente herramienta permite a los desarrolladores escribir y ejecutar código Python a través de su navegador enfocado para tareas de *deep learning*. Es un Jupyter alojado que no requiere configuración, trae las librerías más importantes preinstaladas y tiene una excelente versión gratuita, que brinda acceso gratuito a los recursos informáticos de *Google*, como GPU y TPU en caso de no contar con una máquina potente.

5.2. TRABAJO FUTURO

El algoritmo desarrollado se podrá complementar con el diseño de un dispositivo de menores dimensiones y probar su desempeño sobre adultos mayores para obtener información que se acerque más al desempeño real sobre la población objetivo. Además, puede estudiarse la implementación del algoritmo en teléfonos inteligentes, teniendo en cuenta que las especificaciones de algunos de estos dispositivos abren la puerta a ejecutar algoritmos cada vez más especializados. Cabe resaltar que sistemas operativos como *Android* cuentan con librerías para la implementación de algoritmos de aprendizaje de máquina.

BIBLIOGRAFÍA

ARIANI, Arni, *et al.* “Simulated unobtrusive falls detection with multiple persons”. En: *IEEE Trans. Biomed. Eng* (2012), págs. 3185-3196 (vid. pág. 15).

BECKER, Clemens, *et al.* “Proposal for a multiphase fall model based on real world fall recordings with body-fixed sensors”. En: *Zeitschrift für Geron- tologie Geriatrie* (2012), págs. 707-715 (vid. pág. 15).

BENOCCI, Marco, *et al.* “Accelerometer-based fall detection using optimized ZigBee data streaming”. En: *Microelectron* (2010), págs. 703-710 (vid. pág. 16).

“Blynk”. En: *https://blynk.io/* (2021) (vid. pág. 57).

BUKE, Ao, *et al.* “Healthcare algorithms by wearable inertial sensors: A survey”. En: *China Commun* (2015), págs. 1-12 (vid. pág. 14).

CHANGHONG, Wang, *et al.* “Low-power fall detector using triaxial accelerometry and barometric pressure sensing”. En: *IEEE Trans. Ind. Informat* (2016), págs. 2302-2311 (vid. pág. 17).

ERDOGAN, Zafer y BILGIN, Turgay. “A data mining approach for fall detection by using k-nearest neighbor algorithm on wireless sensor network data”. En: *IET Commun* (2012), págs. 3281-3287 (vid. pág. 17).

GJORESKI, Hristijan, *et al.* “RAReFall— Real-time activity recognition and fall detection system”. En: *Proc. IEEE Int. Conf. Pervasive Comput. Commun* (2014), págs. 145-147 (vid. pág. 15).

GUANGYI, Shi, *et al.* “Mobile human airbag system for fall protection using MEMS sensors and embedded SVM classifier”. En: *IEEE Sensors* (2009), págs. 495-503 (vid. pág. 16).

HARLE, Robert. “A survey of indoor inertial positioning systems for pedestrians”. En: *IEEE Communications Surveys and Tutorials* (2013), págs. 1281-1293 (vid. pág. 36).

JIAN, Yuan, *et al.* “Power-efficient interrupt-driven algorithms for fall detection and classification of activities of daily living”. En: *IEEE Sensors J* (2015), págs. 1377-1387 (vid. pág. 16).

“Keras”. En: *https://keras.io/* (2021) (vid. pág. 31).

“kflash gui”. En: *https://github.com/sipeed/kflash_gui* (2021) (vid. pág. 44).

KOOMEY, Jon. “How green is the Internet?” En: *https://youtu.be/O8-LDLyKaBM* (2013) (vid. pág. 11).

LADETTO, Quentin. “On foot navigation: continuous step calibration using both complementary recursive prediction and adaptive Kalman ltering”. En: *Proceedings of the 13th International Technical Meeting of the Satellite Division of The Institute of Navigation* (2000), págs. 1735-1740 (vid. pág. 36).

MIAO, Yu, *et al.* “A posture recognition-based fall detection system for monitoring an elderly person in a smart home environment”. En: *IEEE Trans. Inf. Technol. Biomed* (2012), págs. 1274-1286 (vid. pág. 14).

“NNCase”. En: *https://github.com/kendryte/nncase* (2021) (vid. pág. 32).

ORDOÑEZ, Francisco y ROGGEN, Daniel. “Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition”. En: *Sensors* (2016), pág. 115 (vid. pág. 36).

SUCERQUIA, Angela; LÓPEZ, José y VARGAS BONILLA, Jesús. “SisFall: A fall and movement dataset”. En: *Sensors* (2017), pág. 198 (vid. pág. 41).

“TensorFlow”. En: *www.tensorflow.org* (2021) (vid. pág. 31).

“TensorFlow Lite”. En: *https://www.tensorflow.org/lite* (2021) (vid. pág. 31).

TWOMEY, Niall, *et al.* “A comprehensive study of activity recognition using accelerometers”. En: *Informatics* (2008), pág. 27 (vid. pág. 15).

WORLD HEALTH ORGANIZATIONS. “WHO Global Report on Falls Prevention in Older Age. Aging and Life Course, Family and Community Health”. En: *https://apps.who.int/iris/han* (2008) (vid. pág. 12).

ZIGEL, Yaniv; LITVAK, Dima y GANNOT, Israel. “A method for automatic fall detection of elderly people using floor vibrations and sound—Proof of concept on human mimicking doll falls”. En: *IEEE Trans. Biomed. Eng* (2009), págs. 2858-2867 (vid. pág. 14).