Sistema de bajo coste para detectar personas con mascarilla y su temperatura a través de redes neuronales

Low-cost system to detect people who wear a mask and their temperature through neural networks

Trabajo Fin de Máster
Curso 2020-2021

AUTOR
Vinicio Jose Valbuena Bracho

DIRECTORES
Alberto Antonio del Barrio García
Guillermo Botella Juan

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Sistema de bajo coste para detectar personas con mascarilla y su temperatura a través de redes neuronales

Low-cost system to detect people who wear a mask and their temperature through neural networks

AUTOR
Vinicio Jose Valbuena Bracho

DIRECTORES
Alberto Antonio del Barrio García
Guillermo Botella Juan

CONVOCATORIA: JUNIO 2021
CALIFICACIÓN: 9

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso 2020-2021

Resumen

El presente trabajo propone un sistema para la detección de personas que hagan uso de mascarilla, y a su vez para medir la temperatura cutánea mediante un dispositivo AIoT de bajo costo, compuesto por el MaixCube y el sensor AMG8833.

En lo que se refiere a la detección de personas con mascarilla, se ha hecho uso de redes neuronales convolucionales, las cuales reciben como entrada una imagen tomada con la cámara del MaixCube.

Para la clasificación de objetos, se han generado dos modelos de redes neuronales convolucionales de clasificación, basados en MobileNet y Tiny Yolo v2, como arquitectura interna para el sistema YOLO. Dichas redes neuronales ejecutan la fase de inferencia sobre el acelerador específico del MaixCube, consiguiendo 5-6 FPS con una precisión mayor del 80% en el caso de MobileNet.

Además de conseguir un sistema funcional, entre las principales contribuciones del trabajo se encuentran la modificación del firmware MaixPy y la creación de un driver para el sensor AMG8833, los cuales se han integrado en el repositorio oficial de MaixPy.

Palabras clave

Visión por computador, red neuronal convolucional, AIoT, MaixCube.

Abstract

This document proposes a system for the detection of people who use a mask; also, to measure skin temperature using a low-cost AIoT device made of the MaixCube and the AMG8833 sensor.

Regarding detection of people who use a mask, convolutional neural networks have been used, which receive an image taken with the MaixCube camera as input.

For object classification, two models of convolutional neural networks have been generated, based on MobileNet and Tiny Yolo v2 as internal architecture for the YOLO system. These neural networks execute the inference phase on the specific Maixcube neural network accelerator, achieving 5-6 FPS with an accuracy higher than 80% in the case of MobileNet.

In addition to achieving a functional system, among the main contributions of the work are the modification of the MaixPy firmware and the creation of a driver for the AMG8833 sensor. These have been uploaded to the official MaixPy repository.

Keywords

Computer vision, convolutional neural network, AIoT, MaixCube.

Índice

Agradecimientos

En primer lugar, un agradecimiento especial a mi esposa Maria Victoria Gonzalez (Vicky), por su apoyo y tener las fuerzas de soportar a alguien tan obstinado como yo.

También quiero agradecer a mis padres José Vinicio Valbuena y Matilde de Valbuena por darme todo su apoyo, a pesar de que en este momento están luchando contra la enfermedad del covid 19.

A mis hermanos, por su apoyo en cualquier proyecto que busque emprender.

A mis tutores, Alberto Antonio del Barrio García y Guillermo Botella Juan que no solo me apoyaron en todo el camino para completar el proyecto, sino que también estuvieron para motivarme en los momentos de dificultad para seguir adelante con el proyecto.

Un agradecimiento especial a Andrez Quiroga por todo su apoyo y ayuda a la hora de llevar a cabo este trabajo.

Capítulo 1 - Introducción

Con la necesidad de resolver problemas cada vez más complejos se han creado aplicaciones de inteligencia artificial en distintos ámbitos, como son las redes neuronales convolucionales, las cuales nos permiten dotar a una máquina de la capacidad de ver e interpretar una imagen de forma similar al proceso del sistema de la percepción visual humana.

En el campo de la visión por computadoras, en la actualidad se busca resolver principalmente problemas como la clasificación de objetos y detección de objetos en tiempo real.

En los últimos años se ha desatado una pandemia a nivel mundial la cual ha restringido el contacto físico de persona a persona para evitar la propagación de dicho virus (covid-19). Es por esto que se crea la necesidad de disponer de nuevas herramientas que trabajen a distancia y nos permitan medir la temperatura de las personas, así como comprobar si están cumpliendo con las medidas sanitarias básicas, como el cumplimiento de la distancia social, o el uso de mascarilla.

Este trabajo se centra en el procesamiento de imágenes para la detección de personas con mascarilla, para posteriormente tomar su temperatura cutánea a distancia, a través de un sistema basado en redes neuronales convolucionales.

1.1 Objetivos

Con este trabajo se busca crear un sistema AIoT de bajo costo que sea capaz de detectar personas que tienen mascarillas como también las personas que no llevan mascarillas, haciendo uso de la inteligencia artificial, a su vez medir su temperatura cutánea (o temperatura superficial) para validar si es posible con un cámara termográfica de bajo costo indicar si una persona podría tener fiebre o no.

Con dicha información, el sistema deberá ser capaz de generar secciones de cada fotograma donde se localiza cada uno de los rostros detectados e indicarnos si este utiliza mascarilla. Una vez

detectado un rostro se tomará su temperatura cutánea y se mostrará en pantalla.

El producto será capaz de detectar el rostro, medir su temperatura cutánea y almacenar la fotografía con su resultado en una tarjeta microsd, para un estudio futuro o para reentrenar nuestra red neuronal convolucional. También avisará con un recuadro verde sobre el rostro cuando una persona se considera segura y roja cuando ésta no lleve la mascarilla colocada.

1.2 Organización de la memoria

La memoria está organizada en 6 capítulos los cuales se hará una breve mención a continuación:

En el capítulo uno se encuentra la introducción, objetivos y esta misma sección.

En el capítulo dos se describen las técnicas utilizadas a nivel teórico, este está muy enfocado a las redes neuronales.

En el capítulo tres se encuentra la información sobre el hardware y detalles del firmware utilizados para la realización del proyecto.

En el capítulo cuatro se presenta el diseño tanto de la aplicación como también su implementación junto a los métodos utilizados.

El capítulo cinco está enfocado en la presentación y análisis de los resultados obtenidos con nuestro producto.

Por último en el capítulo seis se presentan las conclusiones y posibles trabajos futuros.

Capítulo 2 - Métodos utilizados

En este capítulo se describen las técnicas utilizadas a nivel teórico en las que se basa nuestro proyecto para dar una solución al problema planteado a la hora de detectar el uso de mascarillas con un dispositivo de bajo consumo.

2.1 Inteligencia Artificial

La inteligencia artificial es la capacidad de un sistema para interpretar correctamente datos externos, para aprender de dichos datos y emplear esos conocimientos para lograr tareas y metas concretas a través de la adaptación flexible.

Según Takeyas (2007) la inteligencia artificial es una rama de las ciencias computacionales encargada de estudiar modelos de cómputo capaces de realizar actividades propias de los seres humanos con base en dos de sus características primordiales: el razonamiento y la conducta [1].

2.2 Redes Neuronales Artificiales

Las redes neuronales artificiales son un modelo computacional basado en el comportamiento biológico de las neuronas, donde se tienen varias neuronas interconectadas entre sí. Por lo tanto, una red neuronal no sólo establece cómo se relacionan las neuronas entre sí para realizar de forma colectiva un determinado cálculo, sino también cómo las propias neuronas procesan la información que reciben de otras neuronas en la red.

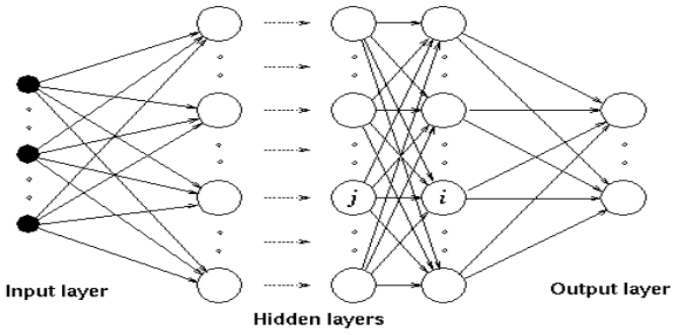


Figura 2.1. Red neuronal multicapa.

En la Figura 2.1 cada nodo representa el modelo matemático más simple de una neurona, esto es, un perceptrón.

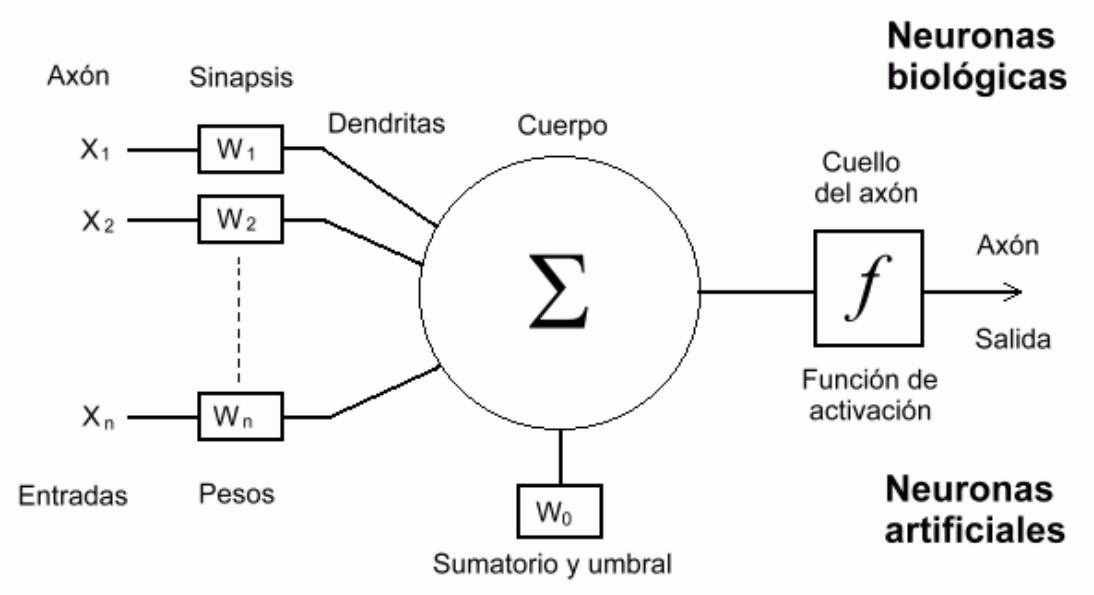


Figura 2.2. El perceptrón.

2.2.1 Función de activación

La función de activación se escoge de acuerdo a la tarea que se quiera realizar por la neurona. Entre las más comunes nos encontramos las siguientes:

- Identidad / Identity

$$f(x) = x$$

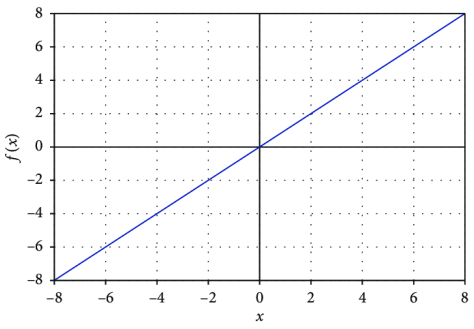


Figura 2.3. Función de activación Identidad.

La función identidad, como se puede observar, nos retorna el mismo valor de entrada como el valor de salida. Por tanto, sus valores van desde [-∞, +∞].

- Escalón / Binary Step

$$f(x) = \begin{cases} 0 & for \ \ x < 0 \\ 1 & for \ \ x \geq 0 \end{cases}$$

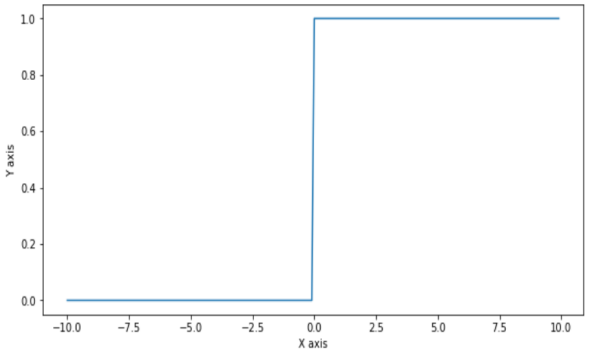


Figura 2.4. Función de activación Escalón.

La función escalón mostrada en la Figura 2.4, nos devuelve un resultado de 0 si su valor de entrada es menor a 0 y en caso contrario su salida sería de 1. Sus valores van entre [0, 1].

- ReLU

$$f(x) = max(0, x)$$

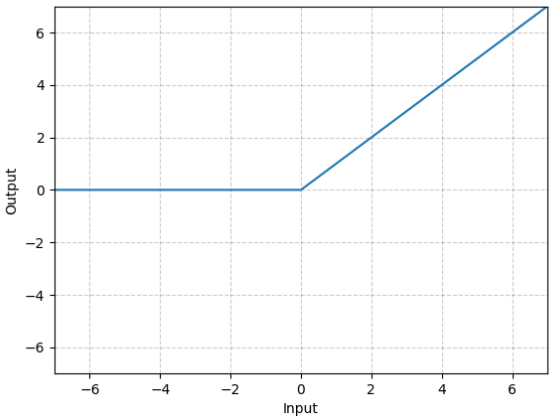


Figura 2.5. Función de activación ReLU.

La función de activación ReLU presentada en el Figura 2.5 nos devuelve un resultado del valor máximo entre 0 y un número x, esto quiere decir que su valor mínimo siempre será el 0.

2.2.2 Regla de aprendizaje

El aprendizaje es el proceso por el cual una red neuronal modifica sus parámetros en respuesta a una información de entrada. En los sistemas biológicos existe una continua creación y destrucción de conexiones. En los modelos de redes neuronales artificiales la creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero. De la misma forma, una conexión se destruye cuando su peso pasa a ser cero.
Durante el proceso de aprendizaje los pesos de las conexiones de la red sufren modificaciones, por tanto se puede afirmar que este proceso ha terminado (la red ha "aprendido") [2].

La regla de aprendizaje está basada en fórmulas matemáticas, utilizando comúnmente técnicas de optimización para la minimización del error, se modifican los valores de los pesos en función a las entradas disponibles y con ello optimizan la respuesta de la red a las salidas que deseamos.

El aprendizaje se basa en el entrenamiento de la red con patrones (ej. imágenes), que normalmente son llamados patrones de entrenamiento. El proceso del algoritmo que realiza la red se basa en ejecutar los patrones iterativamente, ajustar los parámetros, para disminuir el error hasta que la red logre converger, es decir aprender. Posteriormente el modelo entrenado se utilizará con un conjunto de patrones diferentes a los que se usaron durante la fase de entrenamiento.

2.2.3 Métodos de optimización

La optimización consiste en minimizar o maximizar una función, optimizando iterativamente los parámetros de entrada para encontrar un mínimo local/global en el caso de buscar minimizar la función o un máximo local/global si se está buscando maximizar la función.

El optimizador más conocido dentro de las redes neuronales es el descenso del gradiente.

2.2.3.1 El Descenso del Gradiente

Es un algoritmo de optimización que permite converger hacia el valor mínimo de una función mediante un proceso iterativo. En aprendizaje automático básicamente se utiliza para minimizar una función que mide el error de predicción del modelo en el conjunto de datos [6].

Para encontrar el mínimo local/global en función del descenso del gradiente se calcula la derivada parcial respecto a cada parámetro en el punto de evaluación. La derivada indica el valor y sentido que se debe mover para encontrar el mínimo más próximo.

$$\begin{aligned}&\text{repeat until convergence } \{\\ &\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)\\ &\qquad (\text{for } j = 1 \text{ and } j = 0)\\ &\}\end{aligned}$$

Figura 2.6. Algoritmo Descenso del Gradiente.

El resultado de la derivada se le resta a cada parámetro y se multiplica por la velocidad de aprendizaje α. La velocidad de aprendizaje tiene un valor entre [0, 1] y permite configurar lo rápido que converge el algoritmo. Para buscar un valor mínimo es necesario escoger muy bien la velocidad de aprendizaje. Ya que un valor muy bajo puede provocar que nunca se encuentre un mínimo. Por otra parte, un valor demasiado alto podría pasar por encima del mínimo.

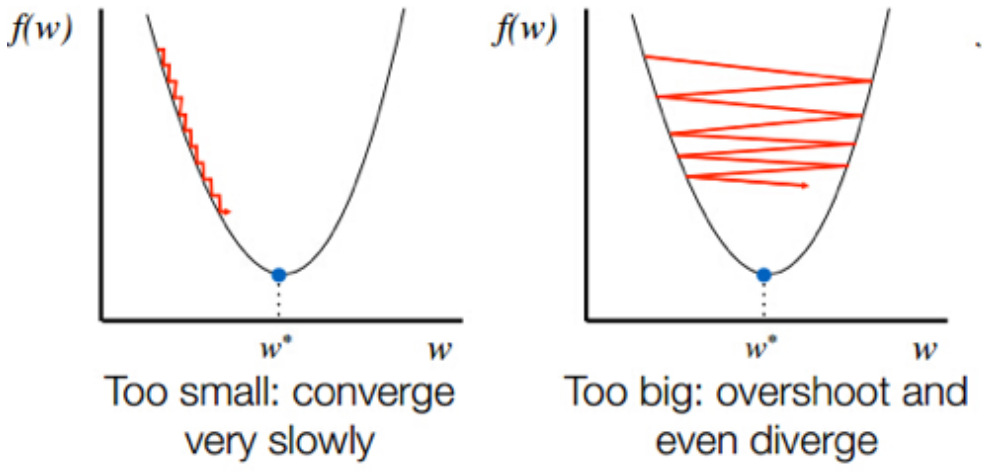


Figura 2.7. Ejemplo de velocidad de aprendizaje.

2.3 Redes Neuronales Convolucionales

Las redes neuronales convolucionales son un tipo especializado de redes neuronales. Son el algoritmo más utilizado en Aprendizaje Automático para dar la capacidad de "ver" al ordenador [18].

Utilizan el aprendizaje supervisado, que procesa sus capas imitando al córtex visual del ojo humano para identificar distintas características en las entradas que le permitirá "ver".

Las primeras capas pueden detectar líneas, curvas, etc., las siguientes pueden identificar figuras básicas, hasta llegar a capas más profundas que reconocen formas complejas como un rostro, un gato o cualquier otro objeto.

Desde el punto de vista computacional, una imagen es una matriz de entradas reales, donde a cada una de sus entradas se le conoce como pixel.

En este tipo de red neuronal toma como entradas los píxeles de una imagen. Pongamos como ejemplo la imagen de 320 x 240 píxeles, esto equivale a 76800 neuronas por canal, es decir que para los canales rgb tenemos un total de 320 x 240 x 3 = 230400 neuronas de entrada.

Es importante que antes de entregar la imagen a la red convolucional ésta se normalice, es decir, en lugar de trabajar con valores de píxel que van 0 a 255, trabajar con valores de 0 a 1.

2.3.1 Capa de Convolución de matrices

Sean A y B dos matrices de tamaño m x n y p x q.

Entonces la convolución de A x B, definida como C = A x B es la matriz C de tamaño (m + p - 1) x (n + q - 1), tal que

$$C(i,j) = [A * B](i,j) = \sum_r \sum_s A(r,s)B(i-r+1, j-s+1)$$

donde $r \in \{\max(1, i-p+1), \ldots, \min(i,m)\}$ y $s \in \{\max(1, j-q+1), \ldots, \min(j,n)\}$

- En el procesamiento de imagen las dimensiones de B normalmente son impares y de tamaño 3 x 3.
- La matriz B se conoce como **núcleo/kernel**, también se le llama **filtro de convolución**.

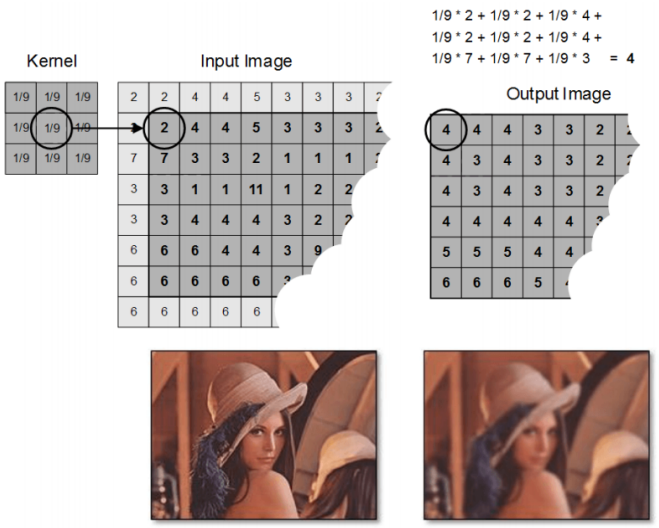


Figura 2.8. Ejemplo de convolución de matrices.

La imagen de salida generada en la Figura 2.7 será nuestra matriz de entrada a la nueva capa de neuronas ocultas [4].

Es importante aclarar que en realidad no se aplica un solo kernel, sino que se aplican varios kernels en paralelo, los cuales se denominan filtros. Por ejemplo para la primera convolución podríamos tener 32 filtros, esto quiere decir que en realidad tendríamos 32 matrices de salida que se conocen como “feature mapping”.

Si volvemos al primer ejemplo donde teníamos una imagen de 320 x 240 x 3 y aplicamos 32 filtros tendríamos un total de 7372800 neuronas de entrada solo en nuestra primera capa oculta.

Estas 32 imágenes nuevas obtenidas nos permiten identificar ciertas características de la imagen original. Lo cual ayudará en un futuro a identificar cualquier objeto, como un rostro o un gato por ejemplo.

2.3.2 Capa de reducción o pooling

La capa de reducción o pooling se coloca generalmente después de la capa convolucional. Su utilidad principal se enfoca en la reducción de las dimensiones espaciales del volumen de entrada para la siguiente capa convolucional.

Esta capa también se le conoce como reducción de muestreo, ya que se pierde información por la reducción de tamaño. Aun así, esta pérdida también nos proporciona beneficios para la red. Las razones se muestran a continuación:

1. La disminución en la dimensión conduce a una menor sobrecarga de cálculo para las siguientes capas ocultas de la red neuronal convolucional.
2. Reducir el sobreajuste.

2.3.3 Muestreo con Max-Pooling

La operación de max-pooling busca un valor máximo entre una ventana de muestra y envía este valor como resumen de características sobre esa área. Como resultado se consigue que el tamaño de los datos se reduzca por un factor igual al tamaño de la muestra con la que se trabaja.

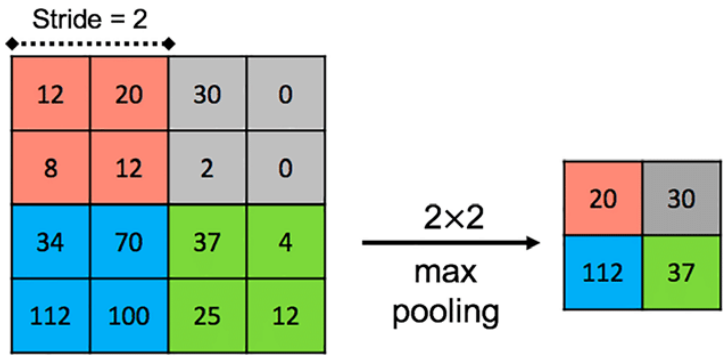


Figura 2.9. Max-Pooling de 2x2 stride 2.

Vamos a explicarlo con un ejemplo: supongamos que tengamos un max-pooling de 2 x 2. Esto quiere decir que recorremos cada una de las 32 imágenes de características obtenidas en el ejemplo anterior,

que eran matrices de 320 x 240 píxeles y al aplicar el max-pooling esta se reduce a la mitad y obtenemos 32 matrices de salidas con dimensión de (320/2) x (240/2) píxeles. Esto quiere decir que reducimos las 320 x 240 x 3 x 32 = 7372800 neuronas que pasarían a la siguiente capa oculta de la red neuronal, ahora tenemos 160 x 120 x 3 x 32 = 1843200 neuronas como entrada para la siguiente capa oculta.

2.3.4 Arquitectura de una Red Convolucional

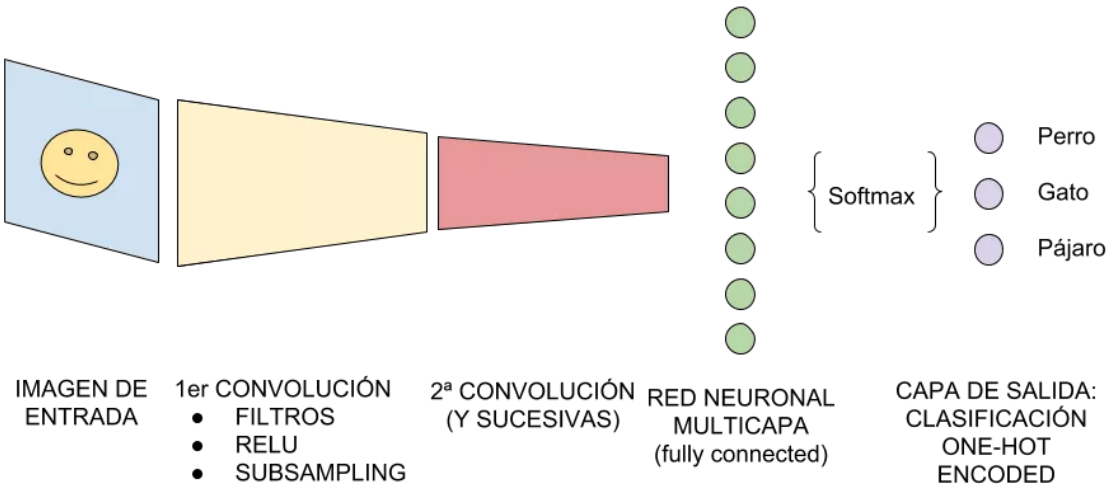


Figura 2.10. Red Convolucional.

Seguimos con el ejemplo de una imagen de entrada de 320 x 240 x 3 píxeles para visualizar el flujo de una red convolucional paso a paso.

Cabe anotar que el ejemplo que se mostrará a continuación está basado en el encontrado en la referencia [3].

1.- Se preprocesa la imagen y se normalizan los pixeles para trabajar en un rango de 0 a 1.

2.- los 320 x 240 x 3 = 230400 pixeles normalizados se pasan como entrada a la red neuronal convolucional.

3.- Primera convolución

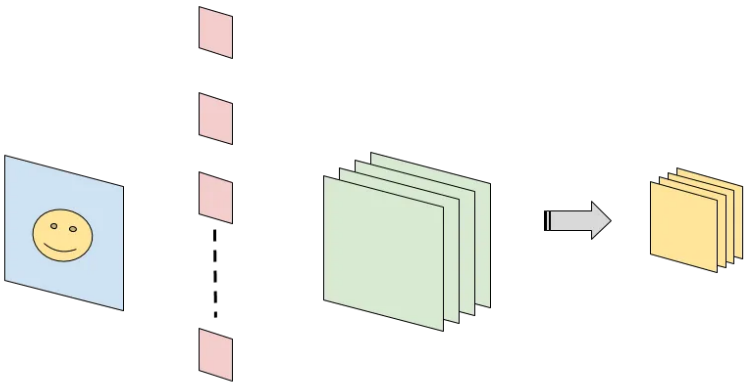


Figura 2.11. Capa de convolución.

1 - Entrada	2 - Filtros	3 - Feature Mapping	4 - Max-Pooling	5 - salida
320 x 240 x 3	32 de 3 x 3	320 x 240 x 3 x 32	2 x 2 paso 2	160 x 120 x 3 x 32

La función de activación aplicada en este ejemplo es ReLU y la salida solo puede tomar como su mínimo el valor cero, de acuerdo a su definición.

$$ReLU(x) = max(0, x)$$

Ya que nuestra entrada está normalizada, nuestras salidas estarán en el rango de 0 a 1.

4.- Segunda convolución

Al aplicar la segunda convolución obtenemos el resultado que se muestra a continuación:

1 - Entrada	2 - Filtros	3 - Feature Mapping	4 - Max-Pooling	5 - salida
160 x 120 x 3 x 32	64 de 3 x 3	160 x 120 x 3 x 64	2 x 2 paso 2	80 x 60 x 3 x 64

5.- Se aplican tantas convoluciones como tenga la arquitectura del modelo de nuestra red neuronal convolucional, y como se puede observar en la tabla anterior la entrada de la tercera convolución será de tamaño 80 x 60 píxeles por canal.

6.- Se conecta a una red neuronal multicapas.

Para pasar los valores tridimensionales de nuestra red neuronal convolucional a la red neuronal multicapas, es necesario aplanar sus valores y transformarlos en una matriz de 1 x n para pasarlos como entrada a la red tradicional.

7.- La salida de la red neuronal tradicional se pasa por la función softmax lo que nos permite saber la probabilidad de que un objeto pertenezca a una de las clases para la que se entrenó dicho modelo.

8 .- La salida de la función softmax se conecta directamente a la capa de salida final, donde la cantidad de neuronas es equivalente al número de clases a buscar. Por ejemplo, si clasificamos rostros con mascarilla o sin mascarillas, tendremos 2 neuronas de salida.

La salida tiene un formato conocido como one-hot-encoding, es decir que devuelve valores en un rango del 0 al 1.

2.3.5 Función Softmax

La función softmax, o función exponencial normalizada es una generalización de la función logística. En probabilidad la salida de la función softmax se emplea para representar una distribución categórica.

Esta se define de la siguiente manera:

$$\sigma : \mathbb{R}^K \to [0,1]^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Gracias a esta función podemos saber que probabilidad tiene un rostro de llevar o no una mascarilla lo cual lo reflejaremos con el siguiente ejemplo:

Supongamos que en la capa de la red neuronal tradicional nos da como salida [1.2, 2.0], donde el primer valor corresponde a un rostro con mascarilla y el segundo valor para un rostro sin mascarilla. Aplicando la función exponencial normalizada (softmax) tendremos como resultado la probabilidad.

La función softmax se calcula aplicando la siguiente fórmula:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

donde K es igual al número de neuronas de salida de la capa de la red neuronal tradicional.

Volviendo a los valores de salida [1.2, 2.0] mencionados en el punto anterior y aplicando la fórmula, obtenemos lo siguiente:

$$e^{1.2} + e^{2.0} = 10.7091$$

La probabilidad para detectar si la persona tiene colocada una mascarilla se calcula de la siguiente manera:

$$\frac{e^{1.2}}{10.7091} = 0.3100$$

Y en el caso de no tener una mascarilla, su probabilidad se calcularía de la siguiente forma:

$$\frac{e^{2.0}}{10.7091} = 0.6899$$

2.3.6 Batch normalization

Es un proceso tiene como objetivo hacer que las redes neuronales sean más rápidas y estables mediante la adición de capas adicionales en una red neuronal. La nueva capa realiza las operaciones de normalización en la entrada de una capa procedente de una capa anterior [19].

Cuando se ingresan datos a una red neuronal, se busca escalar los valores a una escala equilibrada. La razón por la que se busca normalizar los datos es en parte para asegurar que el modelo sea capaz de generalizar adecuadamente.

Una red neuronal se entrena utilizando un conjunto recopilado de datos de entrada llamado batch/lotes. El proceso de normalización en batch normalization se realiza en lotes, no solo en la capa de entrada de la red neuronal.

2.4 Modelos de Detección de Objetos

Un algoritmo de Machine Learning de detección de objetos, para clasificarse como tal debería como mínimo cumplir los siguientes requisitos [3]:

- Detectar múltiples objetos.
- Devolver la posición del objeto en la imagen.
- Devolver su clasificación, ejemplo (con mascarilla o sin mascarilla).

Comparándolo con los algoritmos de reconocimiento, el algoritmo de detección no solo predice la clase de un objeto, sino también la localización de los objetos en la imagen de entrada.

En este documento nos centraremos en las redes neuronales convolucionales de detección de objetos.

Por otra parte, se debe tener en cuenta que las arquitecturas que se mencionan a continuación serán utilizadas como backend (red convolucional que se utilizará para clasificar) en combinación con

el sistema de YOLO, que se encargará de la detección de de los objetos.

2.4.1 YOLO: You Only Look Once

YOLO “You Only Look Once“ es un sistema de código abierto que se utiliza para la detección de objetos en tiempo real, el cual utiliza una única red neuronal convolucional de clasificación, como por ejemplo Tiny Yolo o MobileNet, para detectar objetos en imágenes o videos previamente procesados.

YOLO no requiere de iterar y puede conseguir velocidades muy rápidas gracias a que hace una única pasada a la red convolucional y detecta todos los objetos para los que ha sido entrenada a la hora de clasificar. Esto permite trabajar con video en tiempo real y la detección de cientos de objetos al mismo tiempo y hasta su ejecución en dispositivos móviles [3].

En el proyecto se utilizará una variante de YOLO que tiene como nombre Tiny Yolo v2, ya que está soportada por el propio MaixPy, explicado en el punto 3.6.1, y que ofrece un muy buen rendimiento en tiempo real, aunque se sacrifique algo de precisión [5].

Yolo puede utilizar cualquier red convolucional de clasificación y en este apartado se mostrarán las dos más utilizadas en AIoT como backend de Yolo (Tiny Yolo y MobileNet). La arquitectura de Tiny Yolo v2 se muestra en la Figura 2.13.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figura 2.13. Arquitectura Tiny Yolo.

2.4.1.1 Anchors

YOLO, utilizará la red neuronal convolucional de clasificación y utilizará las características (features) obtenidas en sus capas convolucionales de salida para realizar la detección de los objetos, es decir las posiciones x e y, alto y ancho. Para conseguir esto se valdrá de unas anclas (anchors), en nuestro caso serán 5. Las anclas son unas “cajas”, o bounding boxes de distintos tamaños, pequeños, mediano grande, rectangulares o cuadrados que servirán para hacer “propuestas de detección” [3].

2.4.1.2 Algoritmo k-means

K-means es un algoritmo no supervisado de agrupamiento o clustering. Se utiliza cuando tenemos una gran cantidad de datos sin etiquetar. El objetivo de este algoritmo es el de encontrar “K” grupos (clusters) entre los datos crudos [3].

Para saber si los datos son parecidos o diferentes el algoritmo K-means utiliza la distancia entre los datos.

Las características que se utilizan como entradas para aplicar el algoritmo K-means deberán ser de valores numéricos y para obtener mejores resultados es recomendable que los valores utilizados estén normalizados, manteniendo una misma escala.

El algoritmo utiliza un proceso iterativo en el que se van ajustando los grupos para producir el resultado final. Para utilizar el algoritmo se entrega como entrada el conjunto de datos y un valor de K. El conjunto de datos serán las características para cada punto. Las posiciones iniciales de los K centroides serán asignadas de manera aleatoria de cualquier punto del conjunto de datos de entrada [3].

2.4.2 MobileNet

Es una red neuronal convolucional de clasificación que funciona en aplicaciones AIoT de manera muy eficiente en dispositivos de bajo consumo.

La estructura de MobileNet se basa en convoluciones separables en profundidad explicado en el punto 2.4.2.3, excepto por la primera capa, que es una convolución completa [15].

Todas las capas van seguidas de un batch normalization explicado en el punto 2.3.6 y una no linealidad ReLU, con la excepción de la capa final, que está completamente conectada y no utiliza ReLU. Finalmente aplica una capa softmax para su clasificación.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figura 2.12. Arquitectura MobileNet.

En nuestro proyecto será una de las redes convolucionales de clasificación que se utilizará como backend en Tiny Yolo v2.

2.4.2.1 Pointwise convolution

La convolución puntual (Pointwise Convolution) lleva este nombre porque utiliza un kernel 1x1, o un kernel que itera a través de cada punto. Este kernel tiene una profundidad de muchos canales, dependiendo de los canales que tenga la imagen de entrada [19].

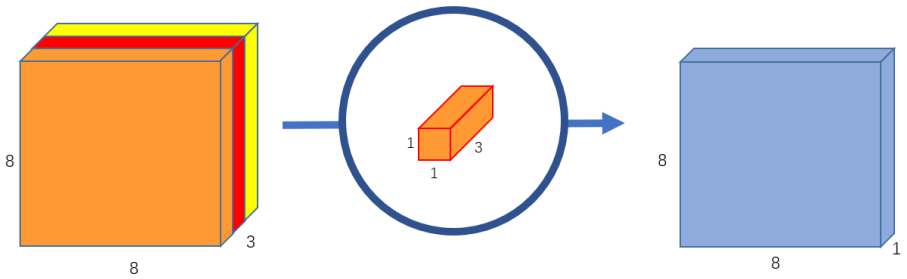


Figura 2.13. Pointwise Convolution, transforma una imagen de 3 canales en una imagen de 1 canal.

Se puede utilizar junto con convoluciones en profundidad (punto 2.4.2.2) para producir una clase eficiente de convoluciones conocidas como convoluciones separables en profundidad explicada en el punto 2.4.2.3.

2.4.2.2 Depthwise convolution

La convolución en profundidad (depthwise convolution) es un tipo de convolución en la que aplicamos un solo kernel convolucional para cada canal de entrada. En la convolución 2D normal realizada sobre múltiples canales de entrada, el kernel es tan profundo como la entrada y esto permite mezclar canales para generar cada elemento en la salida. Por el contrario, las convoluciones en profundidad mantienen cada canal separado.

- Divida la entrada y kernels en canales.
- Convoluciona cada entrada con el kernel respectivo.
- Apilamos las salidas convolucionadas juntas.

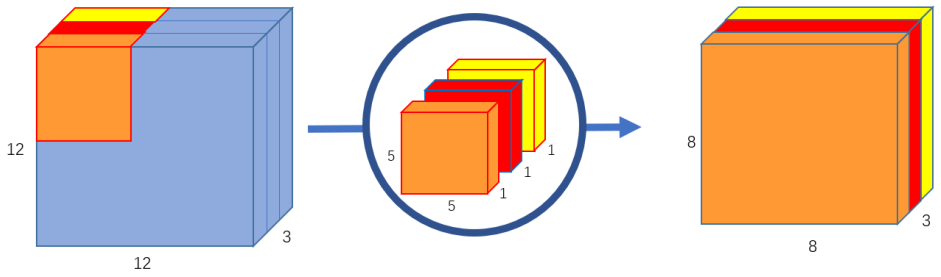


Figura 2.14. Depthwise convolution, utiliza 3 kernels para transformar una imagen de 12x12x3 a una imagen de 8x8x3.

2.4.2.3 Depthwise separable convolution

La convolución separable en profundidad (depthwise convolution) es una convolución en profundidad (punto 2.4.2.2) seguida por una convolución puntual (punto 2.4.2.1), como se muestra a continuación:

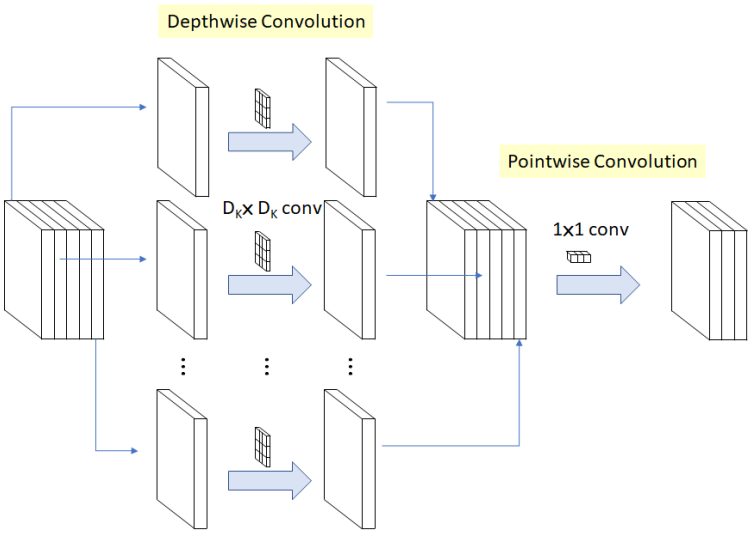


Figura 2.15. Depthwise Separable Convolution.

2.4.2.4 Width Multiplier

El Width Multiplier α permite controlar el número de canales o la profundidad del canal, lo que hace que M se convierta en αM. Y el costo de convolución separable en profundidad (explicado en el punto 2.4.2.3) se convierte en:

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

Figura 2.13. Coste de convolución separable en profundidad con Width Multiplier α.

donde alfa (α) va entre los valores de 0 y 1. Cuando alfa vale uno, es la línea de base de MobileNet. El costo computacional y el número de parámetros se pueden reducir cuadráticamente en aproximadamente alfa al cuadrado.

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Figura 2.16. Diferentes valores de Width Multiplier.

Capítulo 3 - Hardware y Firmware

En este capítulo se detalla tanto el hardware utilizado para el proyecto, como la placa de desarrollo AIoT, como los módulos externos y protocolos de comunicación utilizados.

Además se detalla la estructura interna del dispositivo y su firmware a nivel de directorios, que otorga toda la documentación necesaria para la creación de un custom firmware.

3.1 MaixCube

Sipeed MaixCube es un kit de desarrollo que integra una cámara, una ranura para tarjeta micro SD, tres botones generales configurables por el usuario, una pantalla ips de 1.3 pulgadas, una batería de litio de 200mAh, altavoz, micrófono, e interfaz SP-MOD y Grove [14].

MaixCube viene con Maixpy (firmware [17]) de forma predeterminada. Esto permite que los usuarios puedan trabajar fácilmente con la sintaxis de python (micropython [16]) para comenzar rápidamente con desarrollo de Inteligencia artificial para AIoT.

MaixCube nos permite desarrollar aplicaciones para reconocimiento facial, reconocimiento de objetos o cualquier otra aplicación de inteligencia artificial ya que cuenta con un acelerador convolucional (KPU).

El resto de especificaciones se detallan a continuación:

- Kendryte K210 Core: RISC-V Dual-Core 64-bit, with FPU
    - Frecuencia: 400MHz (overclockable to 600MHz)
    - SRAM: Built-in 8MByte
        - 2 MByte dedicado al KPU
        - 6 MByte de propósito general
    - Modelos convolucionales: TinyYOLO V2 y Mobilenet V1
    - Periféricos: FPIOA, UART, GPIO, SPI, PC, PS, TIMER
    - Unidad de aceleración de hardware
        - KPU acelerador de operaciones convolucionales
        - FPU Acelerador de punto flotante
        - APU Procesador de audio

        - FFT Acelerador de Fourier transform
- MaixCube Modulos
    - Periféricos
        - 3 botones configurables
        - 2 led rgb
        - IPS Screen de 1.3 pulgadas: Resolución 240*240
        - Memoria Flash: 16MiB
        - Cámara 30W (0V7740)
        - Microfono, y Reproductor de audio, driver IC (ES8374)
        - Tres ejes del sensor de aceleración (MSA301)
    - Interfaces
        - USB Type-C
        - TF card slot
        - Grove
        - SP-MOD
    - Fuente de alimentación
        - USB Type-C
        - AXP173 unidad de control
        - Batería de litio interna (200mAh)

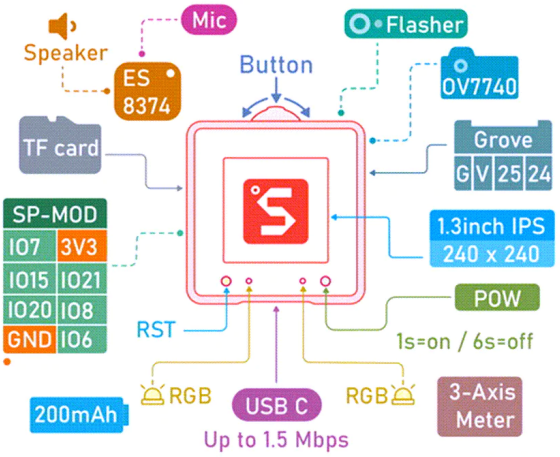


Figura 3.1. MaixCube conexiones.

MaixCube tiene 8M SRAM, de los cuales 2M están dedicados al uso exclusivo de la KPU, y 6M de propósito general. Sus direcciones se muestran en la Figura 3.2.

SRAM address map :

Region	Access	Start Address	End Address	Size
General-purpose SRAM	CPU cached	0x80000000	0x805FFFFF	0x600000
AI SRAM	CPU cached	0x80600000	0x807FFFFF	0x200000
General-purpose SRAM	CPU non-cached	0x40000000	0x405FFFFF	0x600000
AI SRAM	CPU non-cached	0x40600000	0x407FFFFF	0x200000

Figura 3.2, Direcciones de la memoria SRAM [8].

En la Figura 3.3 se muestra el MaixCube, donde se pueden ver tanto la pantalla como sus diferentes botones e interfaces disponibles.

Figura 3.3. MaixCube.

1.- Pantalla TFT 1.3”.
2.- Botón de reset.
3.- Botón de apagado.
4.- Interfaz Grove.
5.- Interfaz SP-MOD.
6.- Ranura para tarjeta micro SD.
7.- Cámara.
8.- Interfaz tipo C.
9.- Tres botones en uno tipo rueda.

3.2 Interfaces y Protocolos de Comunicación

En esta sección se explicarán las interfaces utilizadas en el proyecto, como también sus respectivos protocolos de comunicación.

3.2.1 Interfaz I2C

Abreviatura de Inter-IC (inter integrated circuits), es un tipo de bus diseñado por Philips Semiconductors a principios de los 80s, y este se utiliza para conectar circuitos integrados.

Es un estándar que facilita la comunicación entre microcontroladores, memorias y otros dispositivos. Este requiere dos líneas de señal: reloj (SCL, Serial Clock) y la línea de datos (SDA, Serial Data). Permite el intercambio de información entre muchos dispositivos a una velocidad aceptable, de unos 100 kbits por segundo en el modo estándar.

El I2C está diseñado como un bus maestro-esclavo. La transferencia de datos es siempre iniciada por un maestro y el esclavo reacciona. Es posible tener varios maestros mediante un modo multimaestro, en el que se pueden comunicar dos maestros entre sí, de modo que uno de ellos trabaja como esclavo. El arbitraje (control de acceso en el bus) se rige por las especificaciones, de este modo los maestros pueden ir turnándose.

El I2C precisa de dos líneas de señal: reloj (SCL, Serial Clock) y la línea de datos (SDA, Serial Data). Ambas líneas requieren resistencias de pull-up hacia VCC. Cualquier dispositivo conectado a estas líneas es de drenador o colector abierto (Open Collector), lo cual en combinación con las resistencias pull-up, crea un circuito Wired-AND. El nivel alto debe ser de al menos 0,7 x VCC y el nivel bajo no debe ser más de 0,3 x VCC [18].

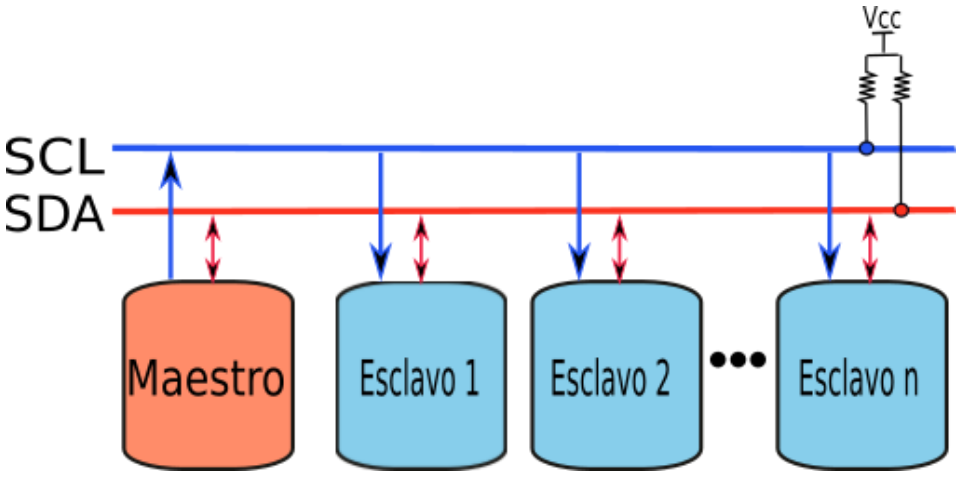


Figura 3.4. Interfaz I2C.

La señal de reloj siempre es generada por el maestro sin importar su modo de operación. En la tabla siguiente se muestran las velocidades de transmisión máximas de reloj en sus distintos modos de operación.

Modo	Velocidad de transmisión máxima	Dirección
Standard Mode (Sm)	0,1 Mbit/s	Bidireccional
Fast Mode (Fm)	0,4 Mbit/s	Bidireccional
Fast Mode Plus (Fm+)	1,0 Mbit/s	Bidireccional
High Speed Mode (Hs-mode)	3,4 Mbit/s	Bidireccional
Ultra Fast-mode (UFm)	5,0 Mbit/s	Unidireccional

Dirección

La dirección de I2C es el primer byte enviado por el maestro, los primeros 7 bits representan la dirección y el octavo bit indica al esclavo si debe enviar datos al maestro (Estado Alto) o recibir datos del maestro (Estado Bajo).

Estos 7 bits nos permiten tener conectados hasta 128 direcciones diferentes, aunque 16 posiciones están reservadas para situaciones especiales.

$2^7 = 128 - 16 = 112$ nodos que podemos utilizar en el mismo bus.

Los circuitos integrados que soportan I2C tienen una dirección predeterminada por el fabricante, de la cual los últimos tres bits (subdirección) pueden ser fijados por tres pines de control.

Protocolo de transferencia

El inicio de la transmisión es indicado por la señal de inicio del maestro, seguido de la dirección. Ésta se confirma por medio del bit ACK del esclavo correspondiente. En función del bit R/W se indica si el esclavo debe leer datos del maestro o enviar datos al maestro.

El ACK es enviado desde el esclavo al escribir, y desde el maestro al leer. El último byte recibido lo reconoce el maestro como un NACK (not acknowledge) que indica el final de una transmisión.

Una transmisión se puede finalizar por la señal de parada. También es posible enviar una señal de reset al arranque de una nueva transmisión, sin necesidad de parar la transmisión anterior con una señal de parada.

Todos los bytes son transferidos de esta manera como "Most Significant Bit First" (bit más significativo primero).

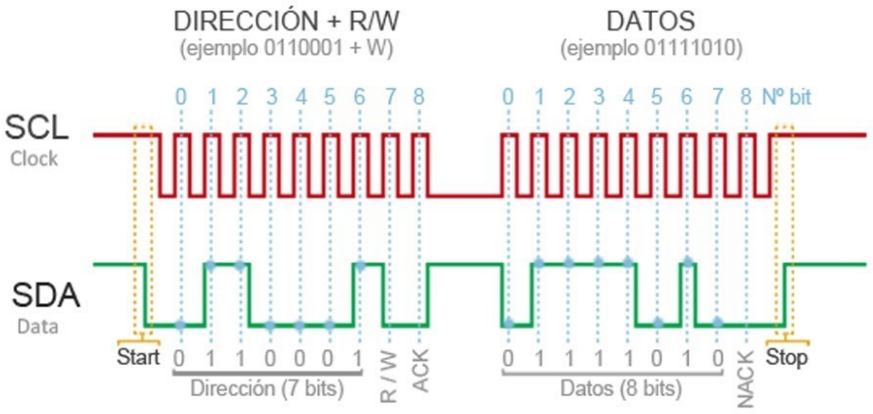


Figura 3.5. Ejemplo de interfaz I2C.

3.2.2 Interfaz Grove

Grove es un sistema de interfaz unificado utilizado por el equipo de Seeed y actualmente admite una gran cantidad de módulos.

Los cables de la interfaz Grove tienen 4 colores, y se pueden identificar rápidamente según los colores y su función.

Figura 3.6. Interfaz Grove.

Grove para interfaz I2C

Pin	Color	Función
1	Amarillo	SCL
2	Blanco	SDA
3	Rojo	VCC
4	Negro	GND

Grove para interfaz UART

Pin	Color	Función
1	Amarillo	RX
2	Blanco	TX
3	Rojo	VCC

4	Negro	GND

Grove para interfaz Digital

Pin	Color	Función
1	Amarillo	D0
2	Blanco	D1
3	Rojo	VCC
4	Negro	GND

3.2.3 Interfaz SP-MOD

SP-MOD es un módulo de Sipeed que simplifica la interfaz PMOD. La interfaz está conformada por 8 pines de 2.54mm distribuidos en dos filas con conexiones hembras que admite diferentes configuraciones para trabajar con distintos protocolos de comunicación en la misma interfaz. Sus diferentes configuraciones se muestran en la Figura 3.7.

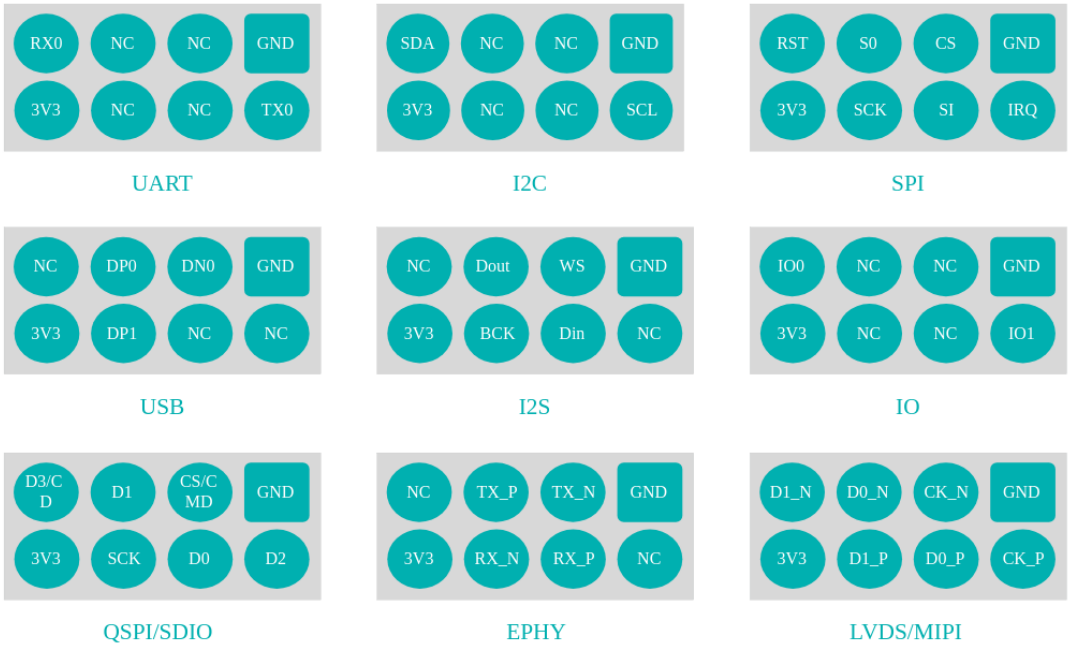


Figura 3.7. Interfaz SP-MOD.

3.3 Field Programmable IO Array (FPIOA/IOMUX)

El microprocesador K210 admite que cada periférico se asigne a cualquier pin a voluntad, utilizando la función FPIOA para lograrlo [8].

La función FPIOA permite asignar 255 funciones internas a 48 IO libres en el chip:

- Selección de función para el GPIO.
- Resistencias pull-up internas seleccionables.
- Resistencias pull-down internas seleccionables.
- disparador de Schmitt como opciones de entradas.
- Control de velocidad de respuesta (Slew rate control) como opciones de salidas.
- Nivel de entrada interno seleccionable.
- 8 Fuerza de impulso (Drive strength) como opciones de salidas.

Hay un total de 32 GPIO de alta velocidad (GPIOHS) y 8 GPIO de uso general. Cada GPIO se puede asignar a uno de los 48 pines del FPIOA.

Ball	Name	Type	Function	Reset State
A1	IO_37	I/O	Multifunctional IO (FPIOA) (Bank 6, Group C)	GPIOHS21
A2	IO_36	I/O	Multifunctional IO (FPIOA) (Bank 6, Group C)	GPIOHS20
A3	IO_35	I/O	Multifunctional IO (FPIOA) (Bank 5, Group B)	GPIOHS19
A4	IO_33	I/O	Multifunctional IO (FPIOA) (Bank 5, Group B)	GPIOHS17
A5	IO_31	I/O	Multifunctional IO (FPIOA) (Bank 5, Group B)	GPIOHS15
A6	IO_29	I/O	Multifunctional IO (FPIOA) (Bank 4, Group B)	GPIOHS13
A7	IO_27	I/O	Multifunctional IO (FPIOA) (Bank 4, Group B)	GPIOHS11
A8	IO_25	I/O	Multifunctional IO (FPIOA) (Bank 4, Group B)	GPIOHS9
A9	IO_23	I/O	Multifunctional IO (FPIOA) (Bank 3, Group B)	GPIOHS7
A10	IO_21	I/O	Multifunctional IO (FPIOA) (Bank 3, Group B)	GPIOHS5
A11	IO_19	I/O	Multifunctional IO (FPIOA) (Bank 3, Group B)	GPIOHS3
A12	IO_17	I/O	Multifunctional IO (FPIOA) (Bank 2, Group A)	GPIOHS1
B1	IO_39	I/O	Multifunctional IO (FPIOA) (Bank 6, Group C)	GPIOHS23
B2	IO_38	I/O	Multifunctional IO (FPIOA) (Bank 6, Group C)	GPIOHS22
B3	IO_34	I/O	Multifunctional IO (FPIOA) (Bank 5, Group B)	GPIOHS18
B4	IO_32	I/O	Multifunctional IO (FPIOA) (Bank 5, Group B)	GPIOHS16
B5	IO_30	I/O	Multifunctional IO (FPIOA) (Bank 5, Group B)	GPIOHS14
B6	IO_28	I/O	Multifunctional IO (FPIOA) (Bank 4, Group B)	GPIOHS12
B7	IO_26	I/O	Multifunctional IO (FPIOA) (Bank 4, Group B)	GPIOHS10
B8	IO_24	I/O	Multifunctional IO (FPIOA) (Bank 4, Group B)	GPIOHS8
B9	IO_22	I/O	Multifunctional IO (FPIOA) (Bank 3, Group B)	GPIOHS6
B10	IO_20	I/O	Multifunctional IO (FPIOA) (Bank 3, Group B)	GPIOHS4
B11	IO_18	I/O	Multifunctional IO (FPIOA) (Bank 3, Group B)	GPIOHS2
B12	IO_16	I/O	Multifunctional IO (FPIOA) (Bank 2, Group A)	GPIOHS0 (ISP)
C1	IO_41	I/O	Multifunctional IO (FPIOA) (Bank 6, Group C)	GPIOHS25
C2	IO_40	I/O	Multifunctional IO (FPIOA) (Bank 6, Group C)	GPIOHS24

Figura 3.8. K210: primeros 26 I/O del FPIOA [8].

MaixCube cuenta con varios módulos internos como la pantalla lcd o el micrófono los cuales ocupan los siguientes puertos GPIO de alta velocidad y no deberían ser utilizados por el aplicativo que se desee desarrollar, para evitar problemas de conflicto con los dispositivos internos.

GPIOHS	Función	Descripción
GPIOHS5	LCD_DC	Pin de señal de r/w del LCD
GPIOHS4	LCD_RST	Pin de chip de reinicio del LCD
GPIOHS29	SD_CS	Selección del chip SPI de la tarjeta SD
GPIOHS28	MIC_LED_CLK	SK9822_CLK
GPIOHS27	MIC_LED_DATA	SK9822_DAT

3.4 KPU convolution operation accelerator

Una KPU es un procesador de propósito específico para redes neuronales convolucionales, operaciones de normalización, activación y agrupación de lotes (batch). Las características específicas son las siguientes:

- Soporta 1x1 y 3x3 kernels convolucionales.
- Soporta cualquier función de activación.
- Cuando se trabaja en tiempo real, el tamaño máximo de parámetro de red neuronal admitido es de 5.5MiB a 5.9MiB.
- El tamaño máximo de parámetro de red admitido cuando no se trabaja en tiempo real es (tamaño del flash - tamaño del software).

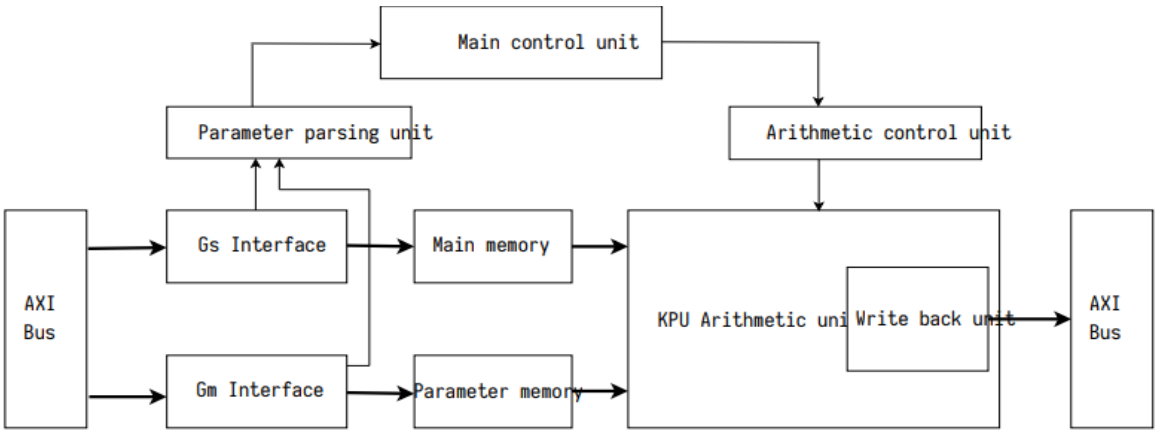


Figura 3.9. Estructura interna de la KPU [8].

3.5 Grove - AMG8833

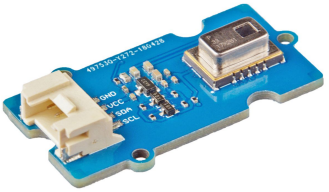


Figura 3.10. AMG8833.

Es un sensor de matriz de infrarrojos de alta precisión basado en tecnología MEMS avanzada, que puede medir temperaturas que van desde 0 °C a 80 °C (32 °F a 176 °F) y proporcionar una precisión de ±2.5℃ (4.5 °F). Puede admitir la detección de temperatura del área bidimensional de 8 × 8 (64 píxeles) y una distancia máxima de detección de 7 metros [7].

Especificaciones:

- Números de pixeles: 64 (8 x 8).
- Interfaz: Grove - I2C
- Frame Rate: 10 frames / seg
- Resolución de la temperatura: 0.25℃
- Dirección I2C: 0x68
- Rango de temperatura del objeto de medición: 0 ℃～80 ℃
- Precisión de la temperatura: ±2.5℃
- Campo de visión: 60° (Horizontal, Vertical)

3.6 MaixCube Firmware

MaixCube está basado en el microcontrolador K210, lo que permite trabajar directamente con el lenguaje C utilizando kendryte-standalone-sdk [11] o FreeRTOS.

La empresa Sipeed da soporte a MaixPy para sus placas de desarrollo como su firmware principal, lo que nos permite trabajar directamente con python.

3.6.1 MaixPy

MaixPy es un port de Micropython a K210; es un proyecto que admite el funcionamiento normal del microcontrolador e integra la aceleración hardware. Esto permite desarrollar rápidamente aplicaciones inteligentes en el campo AIoT con un costo extremadamente bajo y un tamaño práctico.

MaixPy hace que la programación en K210 sea más fácil y rápida, y sea mucho más fácil de modificar el aplicativo por usuarios terceros con pocos conocimientos en el desarrollo de aplicaciones embebidas.

3.6.1.1 Sistema de almacenamiento

El sistema de almacenamiento de MaixPy viene organizado de forma predeterminada de la siguiente manera:

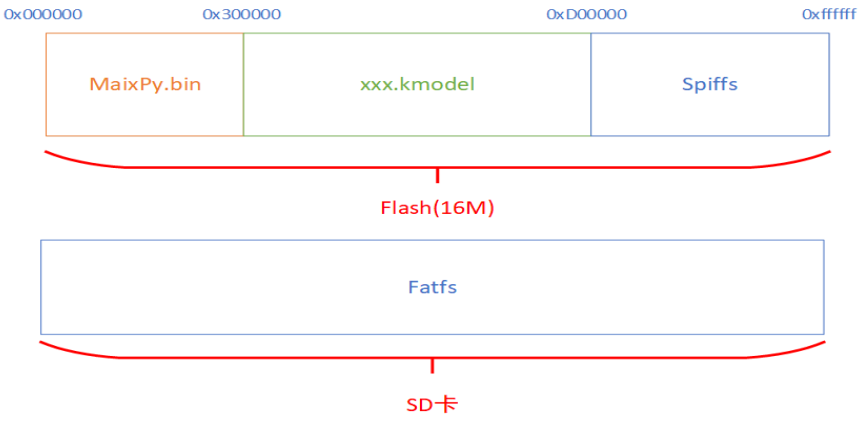


Figura 3.11. Organización del sistema de almacenamiento por defecto.

- **MaixPy.bin** indica la posición del firmware.
- **xxx.kmodel** indica la dirección del modelo de nuestra red neuronal convolucional.
- **Spiffs** es un sistema de archivos diseñado para funcionar en memorias flash conectadas por SPI, utilizado como memoria interna para subir nuestros scripts de python.
- **SD** para el almacenamiento externo.

A partir de la dirección 0xD00000, este área es administrada por el sistema de archivos. Reservamos el espacio 3MiB al final de Flash, que será administrado por spiffs, y también admite tarjeta FAT32(Fatfs) SD. Estos sistemas de archivos proporcionan interfaces para que podamos leer y escribir archivos a través del nombre del archivo en lugar de usar la dirección de inicio del archivo como en el área del modelo. Al mismo tiempo, también puede ayudarnos a gestionar de forma eficaz los medios de almacenamiento los cuales mencionan a continuación junto a el formato soportado:

- La tarjeta micro SD debe formatearse como sistema de archivos FAT (FAT32).
- La tarjeta de memoria formateada FAT se montará en /sd, y SPIFFS en Flash interno se montará en /flash.

3.6.1.2 Scripts de arranque

El sistema creará el archivo boot.py y main.py en el directorio /flash o /sd(externa), y lo ejecutará automáticamente: boot.py primero, y luego ejecutará main.py (si se detecta la tarjeta SD, ejecutará la tarjeta SD). Se recomienda no escribir un bucle infinito en el boot.py, ya que esto no permitirá la ejecución del programa principal main.py.

- **boot.py** se utiliza principalmente para configurar el hardware y solo debe configurarse una vez.
- **main.py** se puede utilizar para el programa principal en ejecución.

3.6.1.3 Estructura de directorio del firmware

La estructura del firmware MaixPy que se explica a continuación es la distribución del código fuente y sus módulos dentro del repositorio oficial que se puede encontrar al final de la página.

Conocer bien la estructura del firmware nos permite crear un custom firmware siguiendo la estructura de diseño original, lo cual nos permite trabajar junto al proyecto original, seguir evolucionando junto a la comunidad y así conseguir un producto más escalable y con la facilidad de que todos podamos seguir el código sin ningún inconveniente.

Con este conocimiento se ha logrado agregar el soporte para la placa MaixCube, que no estaba soportada en el proyecto de manera oficial, así como el driver de control para el AMG8833.

Código fuente de MaixPy está disponible en el siguiente enlace:

1. https://github.com/sipeed/MaixPy

La estructura del firmware se muestra a continuación:

principal	carpeta 1	carpeta 2	carpeta 3	nota
components	┐			componentes.
	boards	┐		soporte de placas.
		config		configuración de las placas, ejemplo: mapeo de los pines, asignar resolución de la pantalla, etc.
	drivers			controladores escritos directamente en lenguaje C.
	micropython	┐		código relacionado con micropython.
		core		código fuente de micropython.
		port	┐	código fuente personalizado de maixpy.
			builtin_py	maixpy default built-in class.
			include	cabeceras del port
			src	código fuente de los módulos accesibles desde el lenguaje python, escrito en lenguaje C. (binding)

Capítulo 4 - Diseño y desarrollo de la aplicación

En este capítulo se describe tanto el proceso realizado para la creación del diseño y desarrollo de la aplicación AIoT, como también los criterios que se tomaron en cuenta a la hora de seleccionar los componentes utilizados.

Figura 4.1. Prototipo en funcionamiento, captura tomada con el MaixCube.

A continuación se muestra un diagrama con el flujo que sigue el aplicativo, se ha omitido las interrupciones hardware que se activan por medio de los botones, los cuales se encargan de activar un menú de calibración o tomar una foto de una medición.

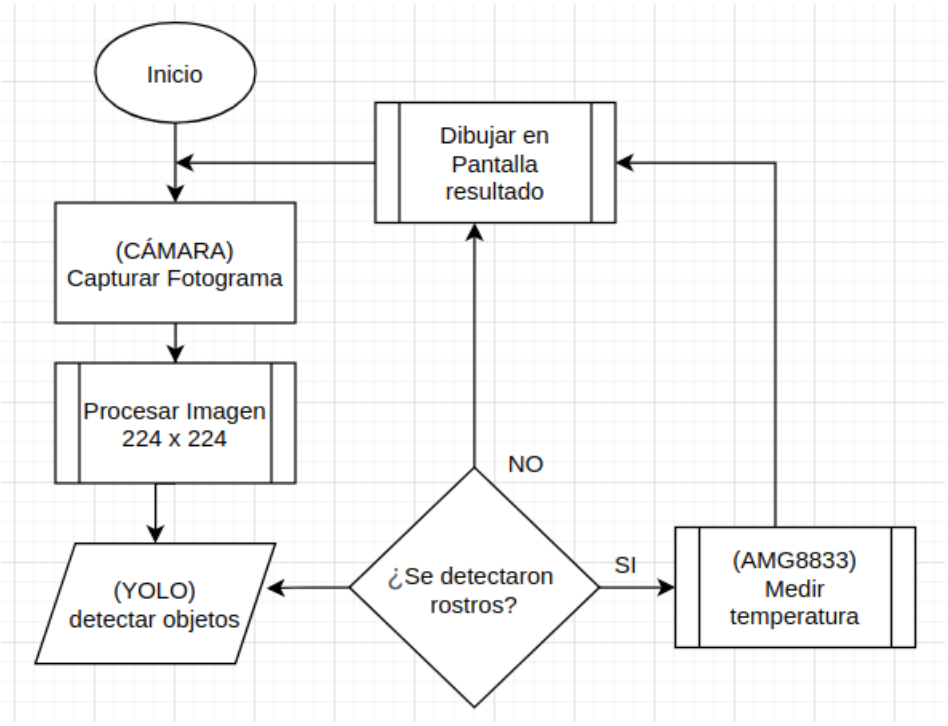


Figura 4.2. Flujo del aplicativo, sin interrupciones por hardware.

El diseño sigue el siguiente flujo:

- Captura de imagen en tiempo real desde la cámara.
- Detectar los rostros en la imagen utilizando redes neuronales convolucionales.
- Con la predicción de la red neuronal detectamos si los rostros tienen o no mascarilla.
- Para los rostros detectados se calcula su temperatura cutánea.
- Si no se cumplen los criterios de seguridad se pinta el cuadrado en el rostro color rojo.
- Si se cumplen los criterios de seguridad se pinta el cuadrado color verde.

En cualquier momento del flujo es posible acceder al módulo de calibración o al módulo de guardar fotografías de la medición. Los botones funcionan por interrupción por hardware y se explican a continuación:

Índice Botones.

- Botón 1: Cambia el modo de configuración hacia abajo.
- Botón 2: Entra al módulo de tomar fotografía de la medición.
- Botón 2: Si se deja pulsado 10 segundos entra al módulo de calibración.
- Botón 3: Cambia el modo de configuración hacia arriba.
- Botón 4: Reinicia el dispositivo.
- Botón 5: Apaga el dispositivo si se deja pulsado 10 segundos.

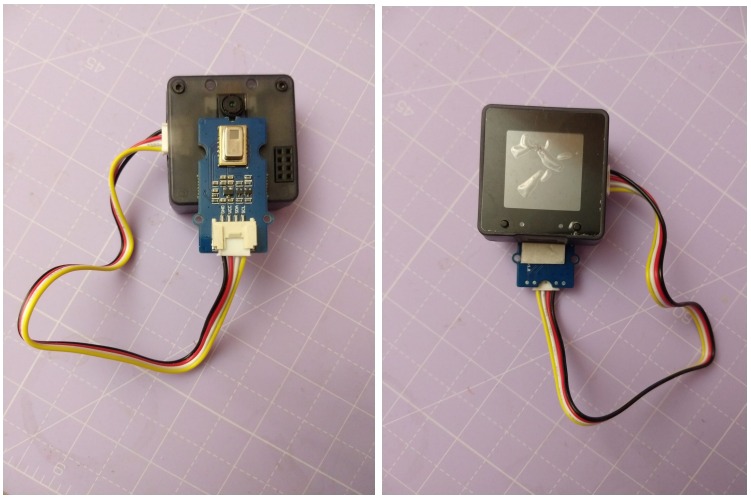


Figura 4.4. Fotografía del Prototipo Hardware.

4.1 Selección de componentes y estudio de costes

En este apartado se analizarán los distintos componentes que se tomaron en cuenta a la hora de seleccionar cual se utilizara en el proyecto, teniendo en cuenta tanto su costo, como las prestaciones que nos ofrece el mismo.

A. Opción con Raspberry Pi 4

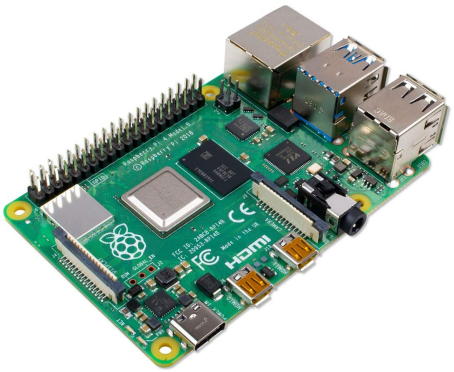


Figura 4.5. Raspberry Pi 4 Modelo B.

#	Nombre	Precio
1	Raspberry Pi 4 Modelo B / 4 GB SDRAM	69,15 €
2	Cámara Sensor Ov5647	9,33 €
3	AMG8833 8x8 Cámara termográfica	35,00 €
4	Pantalla 3.5“ Tft LCD 480X320	28,69 €

5	Set de botones 180 pcs	8,99 €
6	Set de cables 120 pcs	9,99 €
	Total	161.15 €

B. Opción con Raspberry Pi Zero

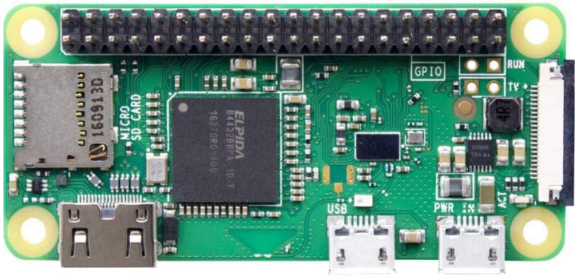


Figura 4.6. Raspberry Pi Placa Zero.

#	Nombre	Precio
1	Raspberry Pi Placa Zero WH 512 MB	27,22 €
2	Cámara Sensor Ov5647	9,33 €
3	AMG8833 8x8 Cámara termográfica	35,00 €
4	Pantalla 3.5“ Tft LCD 480X320	28,69 €
5	Set de botones 180 pcs	8,99 €
6	Set de cables 120 pcs	9,99 €
	Total	119.22 €

C. Opción con ESP32

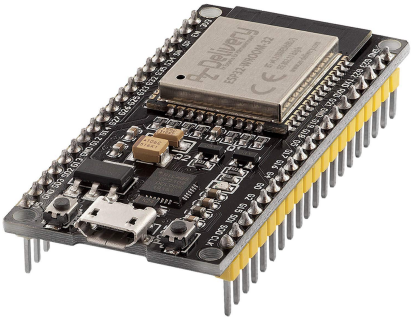


Figura 4.7. ESP32 ESP-WROOM-32.

#	Nombre	Precio
1	ESP32 ESP-WROOM-32	8,99 €
2	Cámara Sensor Ov5647	9,33 €
3	AMG8833 8x8 Cámara termográfica	35,00 €
4	Pantalla 3.5“ Tft LCD 480X320	28,69 €
5	Lector de tarjetas Micro SD TF	4,69 €
6	Set de botones 180 pcs	8,99 €
7	Set de cables 120 pcs	9,99 €
	Total	105.68 €

D. Opción con MaixCube

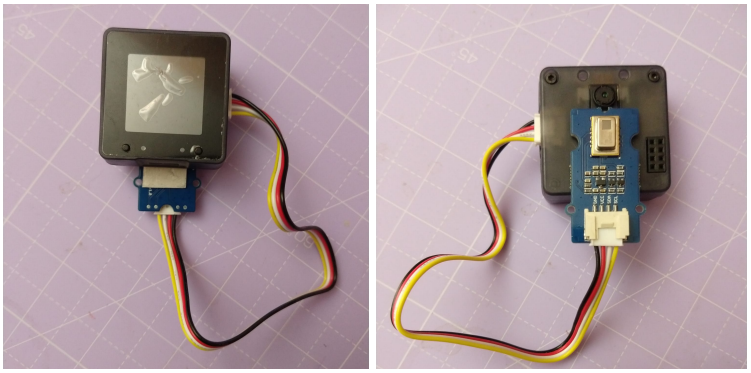


Figura 4.8. MaixCube con AMG8833 8x8 Cámara termográfica.

#	Nombre	Precio
1	MaixCube	51,44 €
2	AMG8833 8x8 Cámara termográfica	35,00 €
	Total	86.44 €

Para las opciones A, B y C se tuvieron que agregar componentes extras como los son: la cámara que se utilizaría para tomar fotografías, como también una pantalla LCD, botones y una ranura para entrada de micro SD.

En el caso de la opción D, como se muestra en la Figura 4.6, ya cuenta con pantalla, botones, una cámara y solo hizo falta comprar el módulo AMG8833 8x8, el cual trae un cable Grove que se conecta directamente con el MaixCube.

Como se puede apreciar la opción D es la más económica y como se detalla en el punto 3.4, además MaixCube cuenta con un procesador dedicado para trabajar con redes convolucionales.

Gracias a la interfaz Grove mencionada en el punto 3.2.2 la conexión con MaixCube y el módulo AMG8833 queda muy estable y con un acabado muy profesional como se muestra en la Figura 4.8.

Por las razones que se han mencionado, se ha elegido para el desarrollo del prototipo el MaixCube.

4.2 Crear soporte para la placa MaixCube

Al momento de trabajar con MaixPy nos encontramos con el problema de que nuestra placa no estaba soportada, lo que causaba que la pantalla presentara los colores invertidos y las entradas y salidas no estuvieran bien mapeadas.

Para solucionarlo se ha creado un custom firmware siguiendo el diseño mostrado en el punto 3.6.1.3.

La configuración para solucionar el problema de los colores invertidos y el nombre de todos los pines se han subido al repositorio oficial del firmware.

El mapeo de los pines de forma correcta también se ha subido al repositorio oficial y se puede encontrar el enlace al final de la Sección 4.3.

Actualmente nuestro código y placa MaixCube está disponible desde el repositorio oficial y puede instalarse sin necesidad de aplicar ningún cambio al firmware MaixPy.

4.3 Crear driver y binding para AMG88XX

Para medir la temperatura cutánea se hizo uso del módulo AMG8833 explicado en el punto 3.5. El firmware MaixPy no contaba con soporte para dicho módulo, con lo cual fue necesario crear un driver para el firmware y un binding para comunicar el driver escrito en c con micropython para poder comunicar el lenguaje python con el nuevo driver.

Driver AMG88xx

Como se mencionó en el punto 3.5, este módulo se comunica por I2C por la interfaz Grove del MaixCube.

Este driver tiene la siguiente estructura:

- **amg88xx_init**, iniciar el dispositivo.
- **amg88xx_snapshot**, solicitar la matriz de píxeles.
- **amg88xx_thermistor**, temperatura del propio módulo.
- **amg88xx_destroy**, desconecta el dispositivo.

La dirección I2C del módulo es 0x68, la cual es utilizada en el proceso de init y para todas las comunicaciones posteriores.

Para el snapshot se solicitan 128 bytes de memoria que comienzan en la dirección 0x80 de dicho dispositivo. Dicha solicitud se hace por comunicación I2C.

Cada dos bytes corresponden a la posición de la temperatura para cada píxel del módulo.

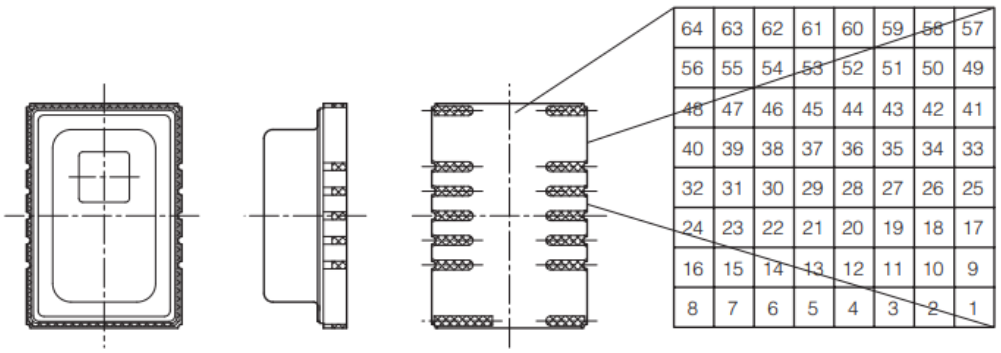


Figura 4.9. Pixel Array del 1 al 64.

El driver está ubicado en la siguiente ruta del firmware components/drivers/amg88xx

Binding AMG88xx

El binding es la parte de micropython que nos permite comunicarnos desde el lenguaje python a nuestro driver escrito en lenguaje C. Para la comunicación se sigue la siguiente estructura:

- **amg88xx**, inicia el dispositivo.
- **temperature**, nos retorna la matriz de 64 píxeles de una sola dimensión.
- **min_max**, nos da el valor mínimo y máximo dentro de la matriz de temperatura.
- **thermistor**, retorna la temperatura interna del módulo.
- **to_image**, permite transformar la temperatura en una imagen de 8x8 y aplicar filtros para escalarla.

```
import lcd

from board import board_info
from fpioa_manager import fm
from modules import amg88xx

fm.fpioa.set_function(board_info.GROVE1, fm.fpioa.I2C0_SCLK)
fm.fpioa.set_function(board_info.GROVE2, fm.fpioa.I2C0_SDA)

lcd.init()

dev = amg88xx(0)

while 1:
    time.sleep(1/10)

    # GET DATA
    temperature = dev.temperature()
    mn, mx, _, _ = dev.min_max()

    #                   min,   max,   scale,  method
    img   = dev.to_image(mn,    mx,    30,     dev.METHOD_NEAREST)
    # img = dev.to_image(mn,    mx,    30,     dev.METHOD_BILINEAR)

    lcd.display(img.to_rainbow(1))
```

Figura 4.10. Ejemplo del módulo accediendo desde micropython.

El binding está ubicado en la siguiente ruta del firmware components/micropython/port/src/modules/amg88xx

Publicación del modulo AMG88xx

Tanto el driver como el binding están publicados en el repositorio oficial del firmware MaixPy y los enlaces se pueden encontrar al final de la página.

En los siguientes enlaces se encuentran los pull requests del nuevo firmware con soporte a MaixCube enlace 1 y a su mapeo de pines disponible en el enlace 2.

1. https://github.com/sipeed/MaixPy/pull/328
2. https://github.com/sipeed/MaixPy/pull/326

4.4 Selección y Pruebas del conjunto de datos

Crear un dataset adecuado es una de las partes más importante para el correcto funcionamiento de una red neuronal. Es por esto, que hay que elegir el mismo, cuidadosamente.

Se ha utilizado parte del dataset que se encuentra al final de la página en el enlace 2, el cual tiene licencia de dominio libre (CC0) y se ha incrementado con fotografías tomadas desde el mismo dispositivo para tener un dataset variado, con diferentes resoluciones y calidades de imágenes.

Esta solución requiere de una gran dedicación de horas para seleccionar y generar un conjunto de imágenes suficientemente amplio y con imágenes suficientemente variadas para que la red neuronal pueda ser entrenada correctamente. Esto es debido a que demasiadas imágenes sobre fondos muy similares, o con la misma iluminación dificultan el entrenamiento de la red neuronal capaz de generalizar correctamente.

Al tener una gran variedad en los datos y la iluminación de las imágenes, se busca evitar el overfitting y que la red neuronal pueda aprender satisfactoriamente.

El dataset cuenta con 912 imágenes, las cuales se reetiqueraton manualmente, ya que el dataset original solo seleccionaba la parte de la mascarilla y lo que se buscaba era que la red neuronal detectara el rostro completo para así poder medir al tiempo la temperatura cutánea.

Los siguientes enlaces corresponden al driver del módulo AMG88XX en el enlace 1 y el dataset utilizado en el enlace 2:

1. https://github.com/sipeed/MaixPy/pull/329
2. https://www.kaggle.com/andrewmvd/face-mask-detection

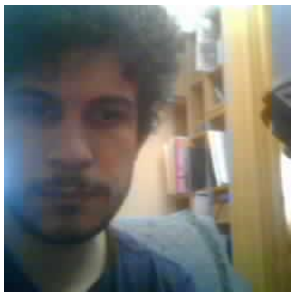
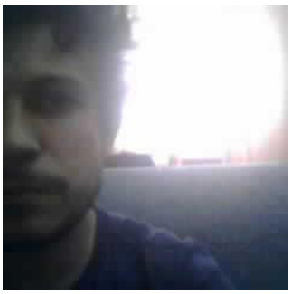
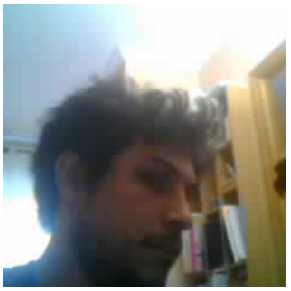
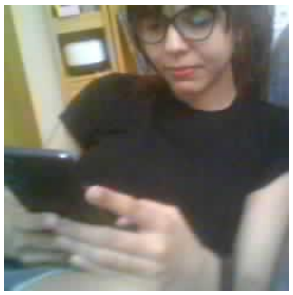
Las imágenes del dataset se separan en dos categorías:

- Rostros con mascarillas (with_mask)
- Rostros sin mascarillas (without_mask)

Se muestran algunos ejemplos de imágenes a continuación:

- **Con mascarilla**

- **Sin mascarilla**

- **Con y Sin mascarillas**





Para etiquetar las imágenes se utilizó el software labelimg (ubicado en el enlace 1 del pie de página), en donde se seleccionó en cada imagen qué rostro tenía mascarilla y cuáles no, como se muestra en la Figura 4.11.

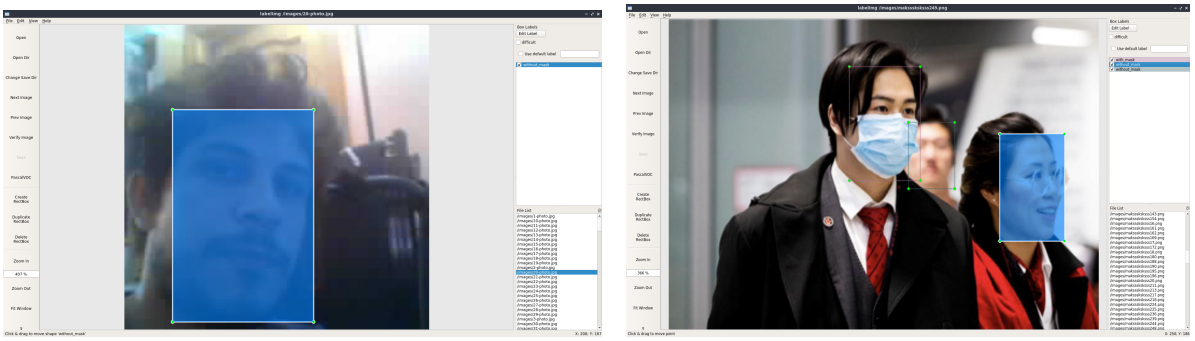


Figura 4.11. Programa LabelImg.

El conjunto de las imágenes seleccionadas se ha dividido de manera aleatoria en dos subconjuntos en una proporción de 8 a 2, el conjunto de entrenamiento y el conjunto de validación.

4.5 Entrenamiento de redes neuronales y generación del modelo

Para el entrenamiento de las redes neuronales, se ha hecho uso del ecosistema de TensorFlow bajo el framework aXeleRate (se puede ver en el enlace 2 al final de la página) [13], ya que éste nos permite tanto entrenar a nuestra red como también generar un modelo compatible para diferentes plataformas de AIoT.

En el proyecto se ha utilizado el formato kmodel(K210) v3 [12], el cual es soportado por el firmware MaixPy y se puede utilizar directamente con el acelerador KPU mencionado anteriormente en el punto 3.4.

El proceso de entrenamiento se realiza a partir de un archivo de configuración en formato json, el cual se muestra en la Figura 4.10.

Los siguientes enlaces corresponden a labelImg se puede acceder desde el enlace 1 y el framework aXeleRate disponible en el enlace 2.

1. https://github.com/tzutalin/labelImg
2. https://github.com/AIWintermuteAI/aXeleRate

Figura 4.12. Ajustes para el entrenamiento con aXeleRate.

Los campos más relevantes son:

- **type**: indicar si queremos clasificar o detectar imágenes.
- **architecture**: se indica que red neuronal queremos utilizar.
- **input_size**: el tamaño de la entrada de la red neuronal, en el ejemplo mostrado corresponde a 224x224.
- **anchors**: tamaños pre calculados para la detección de objetos específicos de la red.
- **labels**: etiquetas de nuestro dataset.
- **weghts/backend**: se indican los pesos preentrenados.
- **train/actual_epoch**: se indica cuántas épocas se quiere entrenar.
- **train/batch_size**: es el número de datos que tiene cada iteración (epoch).
- **train/train_image_folder**: conjuntos de imágenes para el entrenamiento.
- **train/train_annot_folder**: etiquetado de las imágenes de entrenamiento.

- **train/valid_image_folder**: conjuntos de imágenes para la validación.
- **train/valid_annot_folder**: etiquetado de las imágenes de validación.
- **train/augumentation**: escala, rota, recorta y hace zoom a las imágenes para aumentar el dataset.
- **train/saved_folder**: dónde queremos guardar la salida.
- **converter/type**: formatos de modelo que queremos generar.

El algoritmo K-means detallado en el punto 2.4.1.2 se ha utilizado para la generación de los anchors explicado en el punto 2.4.1.1.

El dataset se ha dividido de manera aleatoria en dos subconjuntos 80% el conjunto de entrenamiento y 20% el conjunto de validación.

Se han generados modelos basados en las siguientes arquitecturas, que se utilizaran como la red neuronal convolucional de clasificación para YOLO mencionado en el punto 2.4.1:

- **MobileNet7_5**: MobileNet, alpha: 7.5, epoch: 100.
- **Tiny Yolo**: TinyYolo, epoch: 100.

El siguiente enlace es el algoritmo k-means utilizado en el proyecto.

1. https://github.com/formatcom/kmeans-anchor-boxes

4.6 Módulo para el cálculo de temperatura

Este módulo es el encargado de utilizar el driver creado anteriormente en el punto 4.3. Este módulo está escrito en python y hace uso directamente del binding detallado en el punto 4.3 y se utiliza para leer la temperatura cutánea en un punto dato.

En la Figura 4.13 se muestra un rectángulo blanco que representa el punto en el cual se toma la temperatura con el dispositivo:

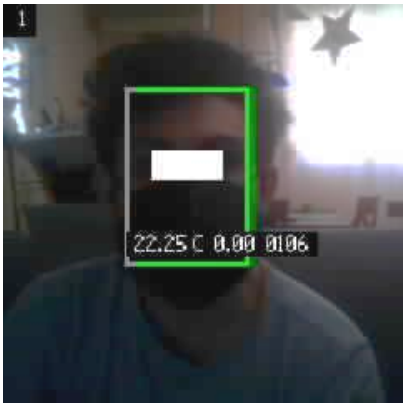


Figura 4.13. Temperatura captada con el módulo AMG8833.

4.7 Módulo de configuración

Se ha implementado un sistema que nos permite tener diferentes configuraciones en un mismo dispositivo, lo que nos permitirá tener diferentes perfiles para cambiar de módulos y asignar una precisión al mismo.

Con esto conseguimos tener diferentes niveles de precisión (en el caso de tener diferentes módulos conectados a la interfaz grove), varios sistemas de calibración para la temperatura y cualquier implementación que se desee evaluar en el futuro.

A continuación, en la Figura 4.14, se muestra la tabla de configuración del sistema junto a su identificador.

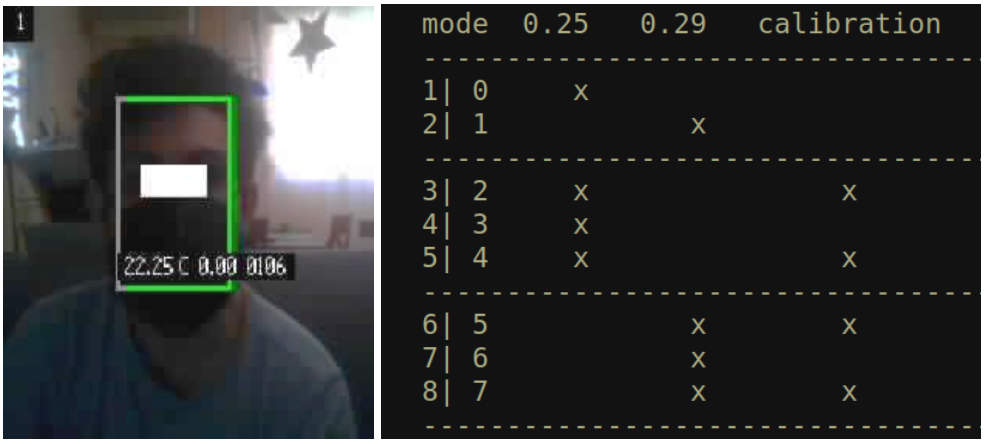


Figura 4.14. Módulo de configuración.

El número en la parte superior izquierda indica qué configuración se está utilizando en el momento de tomar las medidas, lo que nos permite poder analizar la imagen en un futuro o utilizarla para un posterior entrenamiento de nuestra red convolucional.

Los perfiles que se dejaron activados son los siguientes:

- modo 1: Resolución 0.25 utilizada en el modulo AMG8833.
- modo 2: Resolución 0.29. No utilizada.
- modo 3: Resolución 0.25 y calibración activada.
- modo 4: Igual al modo 1. Perfil de respaldo.
- modo 5: Igual al modo 3. Perfil de respaldo.
- modo 6: Resolución 0.29 y calibración activada. No utilizada.
- modo 7: Igual al modo 2. Perfil de respaldo.
- modo 8: Igual al modo 6. Perfil de respaldo.

4.8 Módulo de calibración para ajustar temperatura

El sistema de calibración básicamente nos permite establecer un offset entre la temperatura obtenida y un valor asignado en el menú de calibración.

Con este pequeño sistema podemos tomar varias medidas a diferentes personas con un detector de temperatura en el mercado y ajustar la diferencia entre las temperaturas tomadas con nuestro dispositivo.

Figura 4.15. Módulo de calibración.

En este módulo los botones funcionan ligeramente diferente. El funcionamiento de los mismos se explica a continuación:

Figura 4.16. Botones Índice.

- Botón 1: Selecciona el número que se quiere ajustar desplazándose a la izquierda.
- Botón 2: Botón de seleccionar o aceptar.
- Botón 3: Selecciona el número que se quiere ajustar desplazándose a la derecha.
- Botón 4: Reinicia el dispositivo.
- Botón 5: Apaga el dispositivo si se deja pulsado 10 segundos.

4.9 Módulo para tomar fotografías

Este módulo permite tomar una fotografía en cualquier momento, que puede ser guardada si insertamos una tarjeta SD al dispositivo. En el caso de no tener una tarjeta SD, el sistema muestra un error en pantalla.

La intención con este módulo es poder tener nuevas imágenes para el análisis futuro, como también poder utilizarlas para aumentar el tamaño de nuestro dataset y poder reentrenar nuestra red neuronal. De esta forma se obtendrá una versión más precisa conforme sigamos utilizando el dispositivo.

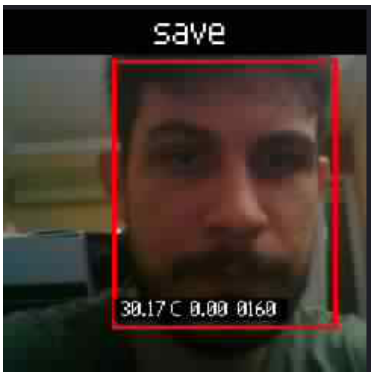


Figura 4.17. Módulo para guardar fotografías.

En este módulo los botones funcionan ligeramente diferente. El funcionamiento de los mismos se explica a continuación:

Figura 4.18. Botones Índice.

- Botón 1: Cambia entre las opciones exit y save.
- Botón 2: Botón de aceptar.
- Botón 3: Cambia entre las opciones exit y save.
- Botón 4: Reinicia el dispositivo.
- Botón 5: Apaga el dispositivo si se deja pulsado 10 segundos.

4.10 Integración de todos los componentes previos

En este punto del proyecto se han unido todos los módulos para poder cumplir con el siguiente flujo:

- Capturar fotograma desde la cámara del dispositivo.
- Procesar el fotograma para generar una imagen de resolución 224 x 224, ya que este es el tamaño de entrada de nuestra red neuronal convolucional.
- Utilizar la imagen generada como entrada a nuestro modelo.
- Según el resultado se verifica si se detectan rostros.
- Si el rostro tiene mascarilla se pinta un cuadrado verde, en el caso de no tenerla se pinta un cuadrado rojo.
- En las posiciones donde se detectaron rostros se calcula la temperatura cutánea.
- En cualquier momento del flujo es posible tomar una fotografía para ser analizada o guardada en una tarjeta SD.

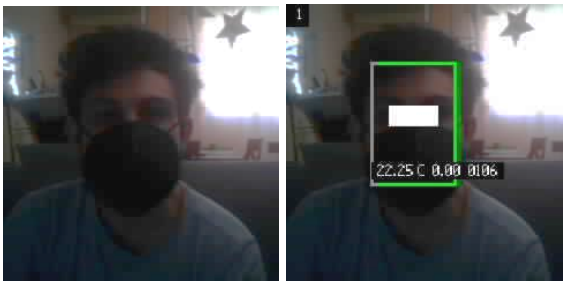


Figura 4.19. Imagen capturada desde el dispositivo (MaixCube).

Capítulo 5 - Resultados

Es necesario realizar un análisis de los resultados, así como la comparación entre las diferentes arquitecturas de redes neuronales convolucionales utilizadas. Con esto se busca comprobar cuál nos brinda el mejor rendimiento y precisión en dispositivos de AIoT de bajo consumo.

Es importante destacar que todas las pruebas de rendimiento se han realizado en el mismo dispositivo MaixCube con la configuración en modo 3 como se puede ver en la Figura 4.1.

Para cada red convolucional se analizará la fase de entrenamiento, la métrica mAP, el peso del modelo generado en formato kmodel v3 y por último los FPS [20] a los que se ejecuta la predicción en el dispositivo AIoT y su diferencia cuando se calcula junto con la detección de la temperatura.

5.1 MobileNet como red convolucional de clasificación en YOLO

Los resultados que se mostrarán a continuación parten de utilizar la red convolucional de clasificación MobileNet como la arquitectura de clasificación en el sistema YOLO explicado en el punto 2.4.1.

5.1.1 Fase de entrenamiento

MobileNet se ha configurado para estas pruebas con un alpha de 0.75, se asignó las épocas a 100 y un tamaño de batch de 4 con una velocidad de aprendizaje de 1e-4.

El dataset utilizado cuenta con 912 imágenes de personas con y sin mascarillas y se ha realizado una separación aleatoria con una proporción del 80% para las imágenes que se utilizaran en el entrenamiento y un 20% para las utilizadas en la validación.

Las imágenes se procesan antes de ser enviadas a la red convolucional y se escalan a la resolución 224x224.

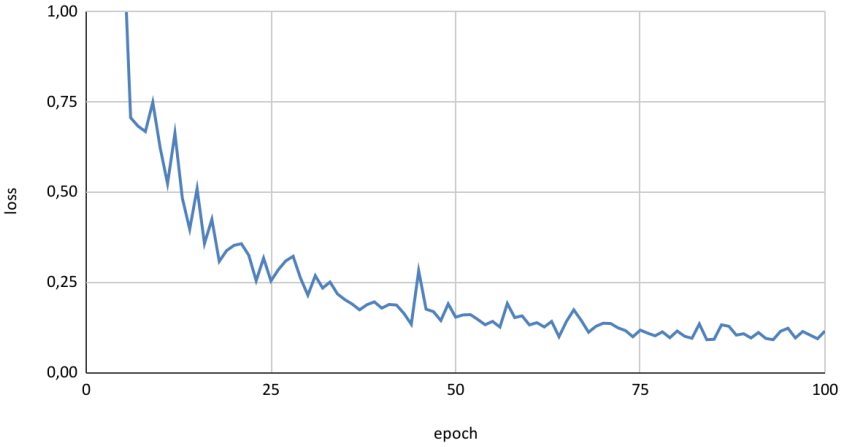


Figura 5.1. MobileNet: loss por época, alpha 0.75.

Como se puede observar, desde la época 70 nuestra red convolucional (MobileNet) casi no presenta pérdidas, se mantiene estable más o menos en el valor 0.12. A continuación se analizará el gráfico de la Figura 5.2.

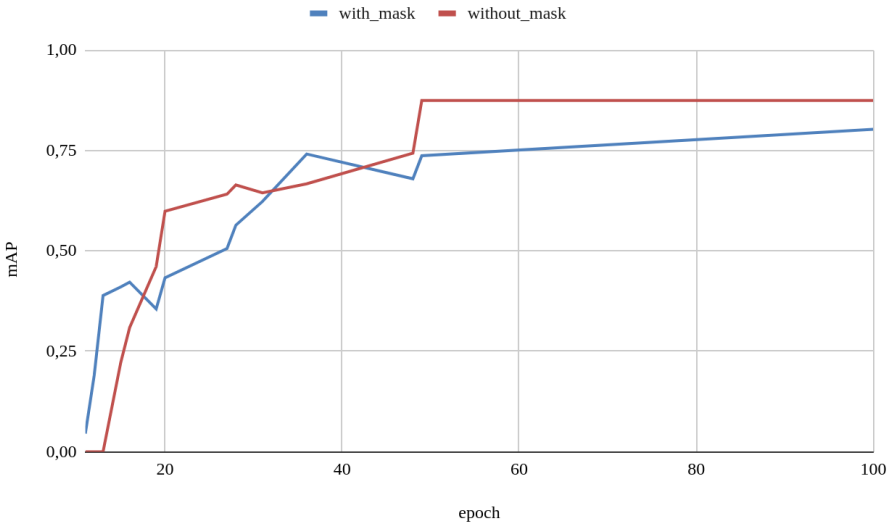


Figura 5.2. MobileNet: mAP de las clases with_mask, without_mask.

En la Figura 5.2, vemos que la precisión de la red sobre nuestro conjunto de datos. Como se observa, detecta casi la misma cantidad de personas con mascarillas que personas sin mascarillas. Para la clase with_mask el mAP aumenta progresivamente a lo largo de las épocas hasta establecerse en el valor 0.74, y para la clase without_mask un valor de 0.85.

5.1.2 Modelo generado

El modelo MobileNet se ha utilizado con el alpha igual a 0.75, ya que con una configuración de arquitectura alpha igual a 1.0 el modelo en formato kmodel v3 tenía un peso de 3.3MB. Teniendo en cuenta el dato anterior, se explica la limitación de espacio en el dispositivo.

MaixCube tiene 16MB de memoria flash interna y es posible almacenar dicho modelo de 3.3MB en la memoria flash, junto al firmware que ocupa 1.2MB.

Teniendo en cuenta que disponemos de 6MB de SRAM para propósito general, de los cuales 1.5 MB los consume el propio firmware, y el KPU requiere 3.3MB libres para almacenar el modelo, esto nos deja con 1.2MB libres para crear todos los buffer del sistema.

Al momento de tomar fotografías y procesar las imágenes capturadas, al contar con tan solo 300kb libres, esto provoca que la KPU congele el sistema, con una configuración de alpha 1.0.

Por tanto, se descarta la opción de utilizar MobileNet con alpha=1.0 por las propias limitaciones del sistema.

Con el punto anterior aclarado, nuestro modelo final con la arquitectura MobileNet con alpha=0.75, tiene un peso de 1.9MB, lo que nos deja un margen bastante grande para trabajar.

5.2 Tiny Yolo v2 como red convolucional de clasificación en YOLO

Los resultados que se presentan a continuación se realizaron utilizando el sistema YOLO. Se ha configurado su red convolucional de clasificación con la arquitectura de Tiny Yolo v2 (explicando anteriormente en el punto 2.4.1).

5.2.1 Fase de entrenamiento

Tiny Yolo v2 se ha configurado para estas pruebas con épocas a 100 y un tamaño de batch de 4 con una velocidad de aprendizaje de 1e-4.

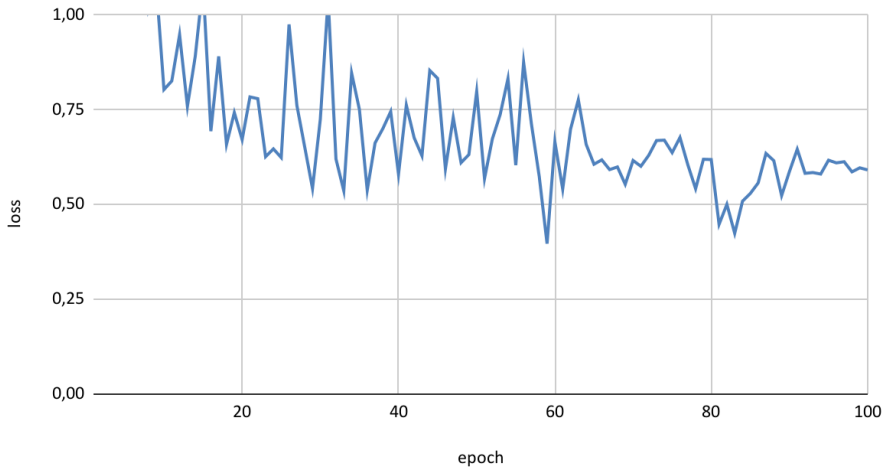


Figura 5.3. Tiny Yolo: loss por época.

A diferencia del MobileNet, se puede ver que aquí la pérdida es mucho más inestable y con nuestro conjunto de datos para 100 épocas no logra bajar la perdida por debajo de 0.40.

Veamos cómo se refleja este comportamiento a la hora de predecir con el gráfico de la Figura 5.4.

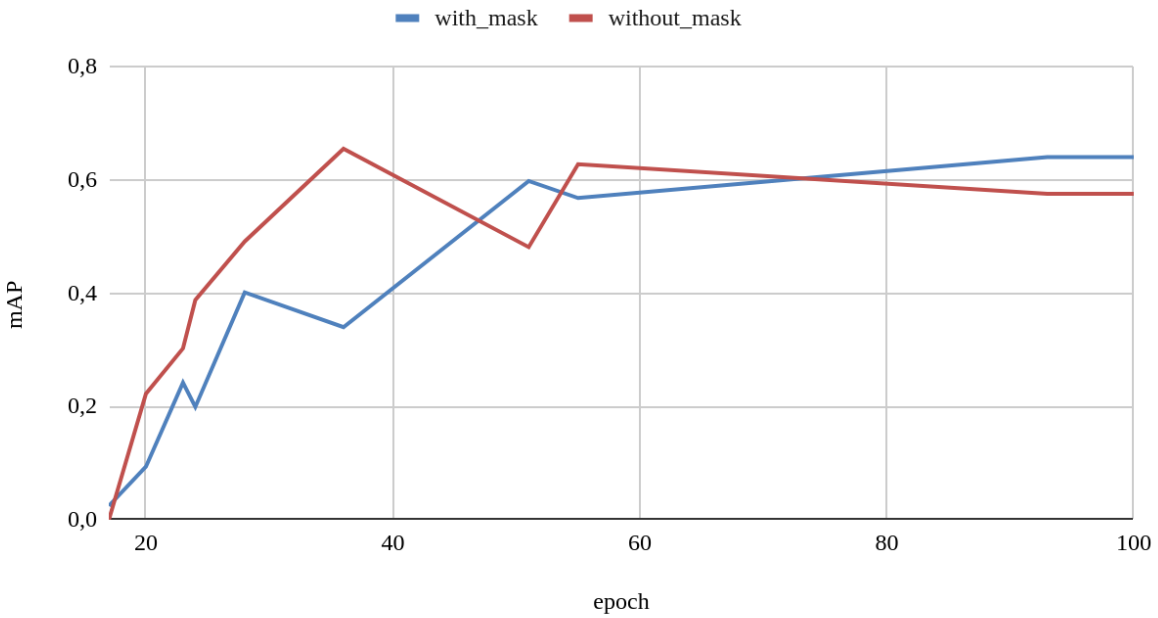


Figura 5.4. Tiny Yolo: mAP de las clases with_mask, without_mask.

Como se puede ver en el gráfico anterior, Tiny Yolo v2 puede detectar apenas 0.64 personas con mascarilla y un 0.58 personas sin mascarilla.

Este comportamiento se debe a que Tiny Yolo v2 está más enfocado en la detección de objetos en tiempo real, sacrificando un poco la precisión a la hora de detectar dichos objetos.

5.2.2 Modelo generado

Para este caso el modelo de Tiny Yolo v2 ocupa 2.2MB y nos deja con un margen de 2.3MB SRAM para trabajar, realizando el siguiente cálculo 6 - 1.5 - 2.2 = 2.3 MB SRAM libres para la ejecución del aplicativo que desarrollamos para nuestro dispositivo.

5.3 Comparativa entre MobileNet y Tiny Yolo

Se destaca que ambos modelos están corriendo bajo el mismo aplicativo en un dispositivo AIoT (MaixCube) de bajo consumo y bajo costo.

Modelo	FPS	loss	tamaño	mAP con mascarilla	mAP sin mascarilla
MobileNet	5	0,1913	1,9M	0,8023	0,8740
Tiny Yolo	6	0,2054	2,3M	0,6399	0,5754

Tabla 5.1. Resultados de rendimiento de los modelos convolucionales.

La Tabla 5.1 nos presenta que, aunque Tiny Yolo sea un fps más rápido que MobileNet. El módulo MobileNet cuenta con mayor precisión.

Por otra parte, MobileNet ocupa menos espacio tanto en memoria flash, como en memoria ram.

5.4 Comparativa entre termómetro infrarrojo y cámara termográfica (AMG8833)

En este apartado se busca realizar una comparativa tomando 8 muestras (medidas de la temperatura cutánea o de superficie) y

analizar los resultados obtenidos, utilizando los dispositivos que se muestran a continuación:

- Termómetro infrarrojo
- Sensibilidad ± 0.1 ºC

Figura 5.5. Termómetro infrarrojo.

- Cámara termográfica
- Sensibilidad ± 0.25 ℃

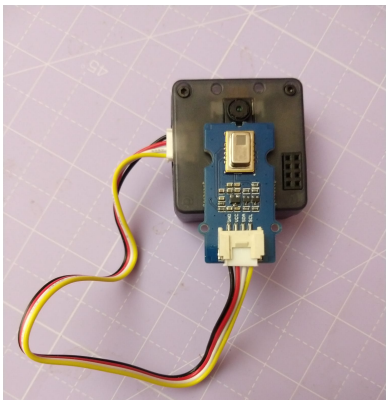


Figura 5.6. Nuestro dispositivo (AMG8833).

34.80 ºC	34.83 ºC
34.40 ºC	34.00 ºC
34.70 ºC	34.08 ºC
34.60 ºC	35.00 ºC
34.30 ºC	35.05 ºC
34.40 ºC	35.00 ºC
34.50 ºC	34.96 ºC
34.80 ºC	34.58 ºC

Tabla 5.2. Comparación de temperatura cutánea sin mascarilla.

En la Tabla 5.2 se puede observar que los valores de la temperatura cutánea tanto para el termómetro infrarrojo y nuestro dispositivo son muy similares, en torno a los 35 ºC. Para tener un mejor entendimiento de los valores presentados en la Tabla 5.2, se calculó a partir de sus datos la media y el error cuadrático medio que se muestra a continuación.

- Termómetro infrarrojo

  - MEDIA: 34.5625 ºC
  - ECM: 0.03234
  - PRECISIÓN: ±0.3 ℃

- Nuestro dispositivo (AMG8833)

  - MEDIA: 34.6875 ºC
  - ECM: 0.15931
  - PRECISIÓN: ±2.5 ℃

Cabe notar que aunque los resultados promedio de ambos dispositivos son similares, sus precisiones son muy diferentes. En el caso del termómetro infrarrojo es de tan sólo ±0.3 °C, en cambio la del dispositivo AMG8833 es ±2.5 °C.

A continuación se muestra el resultado de la toma de temperatura cutánea con mascarilla y sin mascarilla capturada con nuestro dispositivo AMG8833.

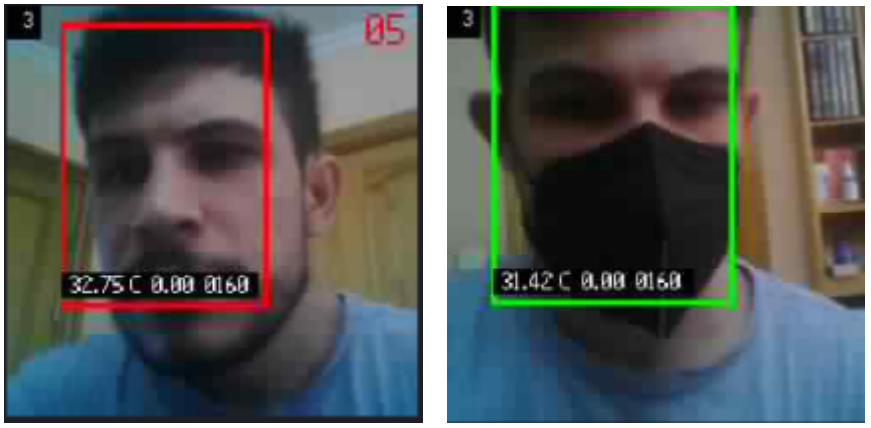


Figura 5.7. Medir temperatura con y sin mascarilla.

Como se muestra en la Figura 5.7, normalmente con la mascarilla se tiene un grado celsius por debajo de la temperatura que cuando no se tiene colocada una mascarilla. Este factor es importante, ya que debemos tomarlo en cuenta a la hora de analizar dichas mediciones.

Capítulo 6 - Conclusiones y trabajo futuro

Considerando los temas abordados en el trabajo, se indicarán los aspectos más importantes que merecen una especial consideración en relación a los resultados obtenidos, así como también a donde se podrían enfocar los trabajos futuros en relación al proyecto.

6.1 Conclusiones generales

Para concluir, tomando en cuenta los objetivos especificados en el punto 1.1, se han logrado alcanzar con resultados satisfactorios, así que el trabajo desarrollado cumple las expectativas planteadas en la prueba de concepto realizada.

6.1.1 MaixCube

MaixCube es un dispositivo con mucho potencial a la hora de trabajar con inteligencia artificial y de muy bajo costo, lo cual permitió realizar todas las pruebas con pocos inconvenientes una vez se implementó el soporte para MaixPy (firmware), que oficialmente no venía soportado.

La experiencia de trabajar junto al repositorio oficial, tanto para dar soporte al MaixCube dentro de MaixPy como a la hora de desarrollar un diver para el mismo, detallado en el punto 4.2, fue muy satisfactoria y junto a una comunidad muy activa.

Aun así, MaixPy a día de hoy aun no esta maduro y tiene muchos problemas con la memoria ram.

El lenguaje de programación utilizado para codificar en MaixPy es Python; Además, tener conocimientos de programación en C y sobre la arquitectura interna de MaixCube es necesario para poder utilizarlo de forma óptima, así como para solucionar los problemas que se presenten.

Además, hay que destacar que actualmente no se encuentra mucha documentación del mismo aparte de su propio código fuente.

6.1.2 Redes neuronales

En el trabajo realizado utilizando redes neuronales, específicamente Tiny Yolo v2 y MobileNet, el resultado ha sido satisfactorio, y se ha conseguido realizar un sistema robusto, escalable y además se logró llevar a un dispositivo IoT con recursos limitados. También se consigue detectar personas que utilizan mascarillas con muy buena precisión.

Es importante anotar que YOLO utilizando MobileNet da muchos mejores resultados que utilizado al propio Tiny Yolo v2, como se puede ver en el punto 5.3.

6.1.3 Detectar fiebre

A la hora de validar si era posible medir la fiebre con nuestra cámara termográfica (AMG8833). Esta no es capaz de medir la temperatura interna del cuerpo.

Con una cámara termográfica de apenas 8x8 píxeles, se puede detectar solo la temperatura cutánea o temperatura superficial.

El módulo AMG8833 se ve muy afectado por la iluminación de la habitación y tiene una precisión de ±2.5 °C, con esto la mayoría de los resultados pueden darnos muchos falsos positivos, a menos que la persona tenga una temperatura tan elevada que los ±2.5 °C no sean relevantes.

Teniendo en cuenta lo mencionado se concluye que con el dispositivo AMG8833 no es posible detectar fiebre de una manera precisa ya que esto acabaría dando demasiados falsos positivos.

6.2 Trabajo futuro

El proyecto que se llevó a cabo en este documento, hace uso de varias tecnologías y se basa principalmente en el procesamiento de imágenes, trabajo con redes convolucionales y detección de la temperatura por medio de un modelo de cámara termográfica. Basado en lo anterior mencionado, se presentan a continuación algunas de las

áreas más relevantes donde se puede mejorar o complementar aspectos dentro del mismo proyecto.

6.2.1 Evolucionar el conjunto de datos

Aunque se tiene un conjunto de datos de licencia libre y el dataset se ha incrementado con algunas imágenes capturadas desde el propio dispositivo, aún sigue siendo muy limitado con apenas 912 imágenes. Lo ideal sería incrementar este conjunto de datos con fotografías capturadas desde el propio dispositivo, haciendo uso del módulo que nos permite grabar fotografías en la microsd, según lo explicado en el punto 4.8.

Para generar un buen conjunto de datos que sea relevante para el aplicativo se debe tener en consideración lo siguiente:

- Tener imágenes donde las personas estén a diferentes distancias.
- Tomar fotografías con diferentes iluminaciones y entornos.
- Etiquetar correctamente cada una de las imágenes generadas.
- Hacer fotos donde aparezcan no solo personas con mascarillas si no también, grupos de personas que estén utilizando o mascarillas.

6.2.2 Redes neuronales

En este proyecto se trabajó con el sistema YOLO y MobileNet como la red neuronal convolucional de clasificación como alternativa a Tiny Yolo v2. Con esto se consiguió detectar personas con mascarilla a una velocidad que ronda los 5-6 FPS.

Como evolución se plantea dar soporte al sistema YOLO v3 en el firmware MaixPy para que trabaje directamente con la KPU. Así se podrían hacer pruebas con el formato de los modelos kmodel v4 (actualmente en beta) para validar si es posible alcanzar más FPS y con una mejor precisión a la hora de detectar objetos.

Como también se pueden probar alternativas a las redes convoluciones de clasificación utilizadas en este documento (MobileNet y Tiny Yolo).

6.2.3 Medir temperatura

Uno de los puntos más importante a mejorar es la precisión a la hora de detectar las temperaturas y para este punto sería conveniente probar con distintos módulos de cámaras termográficas con una mejor precisión y resolución, con el fin de tener una herramienta realmente útil a la hora de validar si una persona está dentro de un rango de temperatura que se considere normal o si esta presenta fiebre sin llegar a tener tantos falsos positivos y así sea mucho más estable.

Capítulo 7 - Introduction

With the need to solve increasingly complex problems, artificial intelligence applications have been created in different areas, such as convolutional neural networks, which provide a machine with the ability to see and interpret an image in a way similar to the process of the human visual perception system.

In the field of computer vision, at present it is mainly sought to solve problems such as the classification of objects and detection of objects in real time.

In recent years, a global pandemic has been unleashed which has restricted physical contact from person to person to prevent the spread of said virus (covid-19). This is why the need for new tools that work remotely and allow us to measure the temperature of people, as well as check if they are complying with basic sanitary rules, such as compliance with social distance, or the use of a mask.

This document focuses on image processing for the detection of people with a mask, to later take their skin temperature remotely, through a system based on convolutional neural networks.

7.1 Objetives

With this document it is sought to create a low-cost AIoT system that is capable of detecting people who wear masks as well as people who do not wear masks, making use of artificial intelligence, in turn measure their skin temperature (or surface temperature) to validate whether it is possible with an inexpensive thermographic camera to indicate whether a person might have a fever or not.

With this information, the system should be able to generate sections of each frame where each of the detected faces is located and tell us if it wears a mask. Once a face is detected, its skin temperature will be taken and displayed on the screen.

The product will be able to detect a face, measure its skin temperature and store the picture with its result on a MicroSD card

for future study or to retrain our convolutional neural network. It will also warn with a green box on the face when a person is considered safe and red when they are not wearing the mask.

Capítulo 8 - Conclusions

Considering the topics addressed in the work, the most important aspects that deserve special consideration in relation to the obtained results will be indicated.

8.1 General conclusions

Taking into account the objectives specified in point 1.1, we can conclude that they have been achieved with satisfactory results, so the developed work meets the proposed expectations in the proof of concept carried out.

8.1.1 MaixCube

MaixCube is a device with a lot of potential when it comes to working with artificial intelligence and at a very low cost,which allowed us to carry out all the tests with few inconveniences once the support for MaixPy (firmware) was implemented, which was not officially supported.

The experience of working together with the official repository team, both to support the MaixCube within MaixPy and when developing a diver for it, detailed in the point 4.2, was very satisfactory and together with a very active community.

Even so, MaixPy to this day is still in the Beta development phase and has problems with the RAM memory.

The used programming language for coding in MaixPy is Python; also, to have knowledge of C programming and about the MaixCube internal architecture is necessary to be able to use it in an optimal way, as well as to solve any problems that arise.

In addition, it should be noted that currently there is not much documentation about it other than its own source code.

8.1.2 Neural networks

In work done using neural networks,specifically Tiny Yolo v2 and MobileNet, the result has been satisfactory and a robust and scalable system has been developed. It was also possible to build an IoT device with limited resources. It is also possible to detect people who use masks with very good precision.

It is important to note that using YOLO and MobileNet gives better results than using Tiny Yolo V2 and MobileNet, as you described in point 5.3.

8.1.3 Detect fever

When validating whether it was possible to detect fever with the thermographic camera (AMG8833), it is found that it is not able to measure the internal temperature of the body.

With a thermographic camera of 8x8 pixels, only skin temperature or surface temperature can be detected.

The AMG8833 module is greatly affected by room lighting and has an accuracy of ± 2.5 °C; with this most of the results can give us many false positives, unless the person has a temperature so high that ± 2.5 °C are not relevant.

Taking into account the aforementioned, it is concluded that with the AMG8833 device it is not possible to detect fever accurately as this would end up giving too many false positives.

Bibliografía

[1] Bruno López Takeyas; “Introducción a la inteligencia artificial”, Instituto Tecnológico de Nuevo Laredo Reforma Sur 2007, C.P. 88250, Nuevo Laredo, Tamps. México.

[2] J. R. Hilera; V. J. Martínez; "Redes neuronales artificiales. Fundamentos, modelos y aplicaciones”; ISBN 10: 8478971556 ISBN 13: 9788478971558; Editorial: RA-MA S.A; 1995.

[3] Juan Ignacio Bagnato; “Aprende Machine Learning Teoría + Práctica Python”; ISBN-10: 8409258161 ISBN-13: 978-8409258161; 2020

[4] Vincent Dumoulin and Francesco Visin; A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285, 2016.

[5] R. Huang, J. Pedoeem, and C. Chen, “Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers,” in 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 2503–2510.

[6] Luis Rodríguez Ojeda; “Método del gradiente de máximo descenso”; Vol. 14 Núm. 1 (2016): Volumen 2016.

[7] AMG88XX datasheet. [Online]. Disponible: https://cdn-learn.adafruit.com/assets/assets/000/043/261/original/Grid-EYE_SPECIFICATIONS%28Reference%29.pdf.

[8] K210 datasheet. [Online]. Disponible: https://m5stack.oss-cn-shenzhen.aliyuncs.com/resource/docs/datasheet/core/kendryte_datasheet_en.pdf.

[9] MaixCube Schemematic. [Online]. Disponible: https://dl.sipeed.com/fileList/MAIX/HDK/Sipeed-Maix-Cube/Maix-Cube-2757/Maix-Cube-2757(Schematic).pdf.

[10] MaixCube Datasheet. [Online]. Disponible: https://dl.sipeed.com/fileList/MAIX/HDK/Sipeed-Maix-Cube/ProductSpecification/Sipeed%20Maix%20Cube%20Datasheet%20V1.0.pdf.

[11] Github page for the standalone sdk for kendryte k210. [Online]. Disponible: https://github.com/kendryte/kendryte-standalone-sdk.

[12] Github page for nncase: Open deep learning compiler stack for kendryte k210. [Online]. Disponible: https://github.com/kendryte/nncase.

[13] Github page for axelerate: Keras-based framework for ai on the edge. [Online]. Disponible: https://github.com/AIWintermuteAI/aXeleRate.

[14] Sipeed home page, https://www.sipeed.com. [Online]. Disponible: https://www.sipeed.com

[15] Andrew G. Howard, , Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." (2017).
[16] Github page for micropython: The MicroPython project. [Online]. Disponible: https://github.com/micropython/micropython.
[17] Github page for MaixPy: MicroPython for K210 RISC-V, let's play with edge AI easier. [Online]. Disponible: https://github.com/sipeed/MaixPy.
[18] Juan Mas, Teodoro Panadero, Guillermo Botella, Alberto A. Del Barrio, Carlos García, CNN Inference acceleration using low-power devices for human monitoring and security scenarios, Computers & Electrical Engineering, Volume 88, 2020, 106859, ISSN 0045-7906.
[19] M. S. Kim, A. A. Del Barrio Garcia, H. Kim and N. Bagherzadeh, "The Effects of Approximate Multiplication on Convolutional Neural Networks," in IEEE Transactions on Emerging Topics in Computing, doi: 10.1109/TETC.2021.3050989.
[20] D.G. Fernández, A.A. Del Barrio, G. Botella, C. García, M. Prieto, R. Hermida, Complexity reduction in the HEVC/H265 standard based on smooth region classification, Digital Signal Processing, Volume 73, 2018, Pages 24-39, ISSN 1051-2004.

Apéndice A - Manual de instalación

En este anexo se exponen los pasos de instalación del proyecto en el dispositivo MaixCube desde en un sistema Linux. Concretamente en Fedora 33.

1.- Requisitos del sistema:

- Python 3

2.- Pasos de instalación:

Clonar repositorio.

```
$ git clone https://github.com/formatcom/tfm-mask-detect.git
$ cd tfm-mask-detect
```

Crear entorno virtual.

```
$ python3 -m venv env
$ source env/bin/activate
```

Instalar dependencias de Python 3.

```
$ pip install -r requirements.txt
```

Conectar maixcube al pc y cargar el firmware con el siguiente comando.

```
$ kflash                  \
         -p /dev/ttyUSB0  \
         -b 1500000       \
                            maixpy_k210_minimum_v0.6.2_mask.kfpkg
```

Por último copiar la aplicación (app.py) en la raíz de la tarjeta MicroSD que se utilizará en nuestro MaixCube.