

Universidad de Costa Rica

Detector de fuego con Arduino nano para predecir incendios
con Machine Learning

IE-0624 Laboratorio de Microcontroladores

Prof. MSc. Marco Villalta

Isaac Rojas Hernández

B76693

Amy Herrera Mora

B53473

6 de julio de 2023

Índice

1. Resumen	3
2. Objetivo General y Específicos	4
3. Alcances	4
4. Justificación	4
5. Marco Teórico	5
5.1. Arduino Nano 33 BLE Sense Lite	5
5.2. Microcontrolador nRF52840	8
5.3. APDS-9960	10
5.4. TensorFlow y TensorFlow Lite	10
5.5. TinyML	12
5.6. Google Colab	12
5.7. Diseño del proyecto	12
5.7.1. Diseño del circuito	13
5.8. Tabla de componentes para el desarrollo del proyecto	14
6. Desarrollo/Análisis de resultados	14
6.1. Desarrollo	14
6.2. Análisis de Resultados	19
7. Conclusiones, Recomendaciones y Observaciones	21
7.1. Conclusiones	21
7.2. Recomendaciones y observaciones	22
Bibliografía	23

1. Resumen

El presente proyecto muestra el desarrollo de aprendizaje automático con un Arduino Nano para poder detectar incendios por medio del sensor de color en la placa del microcontrolador. Este funciona de forma que si se detecta un incendio, se activa un zumbador. Su importancia radica en el hecho de que es mejor que un detector de llamas o de humo, puesto que estos utilizan el calor y el humo para detectar el incendio. La aplicación presente hace uso del valor RGB para detectar el fuego, siendo un poco más preciso, ya que no conlleva las consideraciones necesarias de las aplicaciones de calor y humo. El sensor de calor utilizado viene incorporado en la placa Arduino Nano BLE 33 Sense, como lo muestra su información en [2] y la placa tiene el sistema operativo mbed OS incorporado.

Para su debida implementación, primero se recopiló la información de los valores RGB de la habitación con ninguna llama de fuego utilizando la biblioteca APDS9960 [4]. Seguidamente, se capturo información de la habitación con una llama encendida para que por medio de estos datos se entrenara la red neuronal. Para dicho entrenamiento se utilizo la biblioteca tensorflow [6] de Google junto al entorno Google Colab [8]. Finalmente, la red después de entrenada se implementa dentro del microcontrolador para poder predecir un incendio de forma precisa realizando las pruebas correspondientes de cuando hay fuego o no cerca.

Como conclusiones mas relevantes se destacan que implementar un modelo de red neuronal que provea datos de predicción e identificación adecuado, requiere de una captura de datos de entrenamiento extenso. También, la creación de modelos de redes neuronales buenos, requieren de mucha potencia computacional, sin embargo, con el uso de la herramientas de desarrollo como TensorFlow, Google Colab y algunas bibliotecas reconocidas el desarrollo, entrenamiento y exportación de redes neuronales se vuelve mas fácil y eficiente.

2. Objetivo General y Específicos

Objetivo General:

- Desarrollar un detector de fuego con un Arduino Nano para predecir incendios utilizando Machine Learning.

Objetivos específicos:

- Implementar un circuito junto al microcontrolador Arduino Nano que sirva como alerta.
- Construir un modelo de red neuronal y entrenarlo con los datos obtenidos con ayuda de bibliotecas disponibles en Python para la detección de fuego.
- Exportar la red neuronal creada a un modelo TensorFlow lite para su utilización en el Arduino Nano.
- Implementar un programa que utilice la red neuronal para detectar fuego y alertar de la presencia del mismo.

3. Alcances

Dentro de los posibles alcances para este proyecto se encuentran:

1. La detección temprana de incendios: Como este proyecto desarrolla una predicción de fuego con Machine Learning, uno de los alcances podría ser desarrollar un sistema completo capaz de detectar de forma temprana algún incendio y activar alarmas como forma de prevención.
2. Monitorización remota: Otro alcance que permite este trabajo es la monitorización remota del sistema de detección de incendios. A esto se le podría agregar el acceso mediante alguna aplicación móvil, como por ejemplo, un bot de Telegram donde las personas puedan verificar la situación de emergencia incluso si no están presentes en el lugar de los hechos.
3. Análisis de los datos para la prevención de incendios: Los datos recopilados por este proyecto pueden utilizarse para analizar de manera mas profunda los patrones y factores de riesgo asociados a los incendios para poder identificarlos en el futuro y poder prevenir desastres o incluso salvar vidas.

4. Justificación

Los detectores de incendios son un elemento fundamental de seguridad. El poder detectar un incendio en su fase inicial es de suma importancia para lograr minimizar al máximo los daños ocasionados por el fuego que pueden ser tanto de vidas humanas como materiales. Por esta razón es que los detectores de incendios son esenciales en las instalaciones contra incendios. Estos dispositivos son fundamentales para la seguridad, ya que es la forma más rápida y eficaz de identificar un incendio cuando esta iniciando. Esto permite activar las medidas necesarias para su control y facilita la evacuación de la zona afectada. Asimismo, tener detectores de incendios en los hogares reduce el riesgo de morir por fuego en un 54% en comparación con hogares sin estos detectores o algún tipo de alarma que no se encuentre operativa. [1]

La importancia de esta propuesta radica en que, aunque no supera los sensores típicos de fuego o humo convencionales, esta utiliza los valores RGB para poder detectar el fuego. Este hecho permite que este detector sea mas preciso que los anteriormente mencionados.

5. Marco Teórico

5.1. Arduino Nano 33 BLE Sense Lite

Este dispositivo pertenece a las placas de HW+SW libre con microcontrolador programable. La mayoría de estos MCUs pertenecen a la misma familia AVR. Tiene su propio IDE inspirado en Processing y se programa en lenguaje C/C++. Su hardware se inspira en la placa libre Wiring del 2003 y nace en Italia en el Instituto Ivrea en el año 2005. En la figura (1) se muestra esta placa. [2]



Figura 1: Placa del microcontrolador Arduino Nano 33 BLE Sense [2]

Dentro de la descripción general para el Arduino Nano BLE 33 Sense Lite TinyML kit se encuentran varias características que se explicaran a continuación de acuerdo a la hoja de datos específica para este laboratorio. [2]

- Es una placa basada en el microcontrolador nRF52840 de Nordic Semiconductor.
- Es versátil y compacta ya que incluye 11 sensores.
- Esta diseñada para que se puedan desarrollar aplicaciones de bajo consumo y que incluso requieran de conectividad BLE (Bluetooth).
- Posee un giroscopio, acelerómetro y magnetómetro en la placa (IMU LSM9DS1 de 9 ejes).
- Además, tiene un sensor de proximidad, intensidad de luz, color RGB y detección de gestos APDS9960, micrófono digital y cámara OV7675.
- Aunque la versión Lite no presenta el sensor HTS221 de temperatura y humedad, si posee el LPS22HB, el cual es un sensor de presión y temperatura.
- Presenta soporte de MicroPython.

Las especificaciones técnicas para el Arduino Nano BLE 33 Sense Lite también se encuentran descritas en la hoja de datos general para este microcontrolador y se pueden observar en la figura (2).

Board	Name	Arduino Nano 33 BLE Sense Lite
	SKU	ABX00031
Microcontroller	nRF52840	
USB connector	Micro USB	
	Built-in LED Pin	13
	Digital I/O Pins	14
Pins	Analog input pins	8
	PWM pins	5
	External interrupts	All digital pins
Connectivity	Bluetooth®	NINA-B306
	IMU	LSM9DS
	Microphone	MP34DT05
Sensors	Gesture, light, proximity	APDS9960
	Barometric pressure	LPS22HB
	Temperature, humidity	HTS221
Communication	UART	RX/TX
	I2C	A4 (SDA), A5 (SCL)
	SPI	D11 (COP), D12 (CIPO), D13 (SCK). Use any GPIO for Chip Select (CS).
	I/O Voltage	3.3V
Power	Input voltage (nominal)	5-18V
	DC Current per I/O Pin	10 mA
Clock speed	Processor	nRF52840 64MHz
Memory	nRF52840	256 KB SRAM, 1MB flash
	Weight	5gr
Dimensions	Width	18 mm
	Length	45 mm

Figura 2: Especificaciones técnicas para el Arduino Nano BLE 33 Sense. [2]

El diagrama de pines para esta placa se puede apreciar en la figura (3), mientras que la topología de la placa se muestra en la figura (4).



ARDUINO NANO 33 BLE

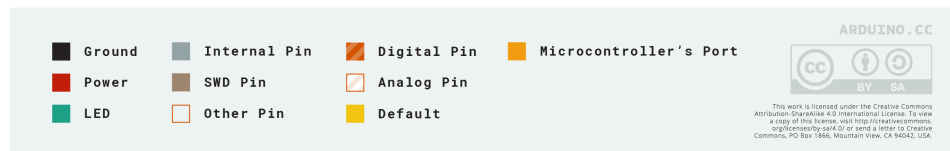
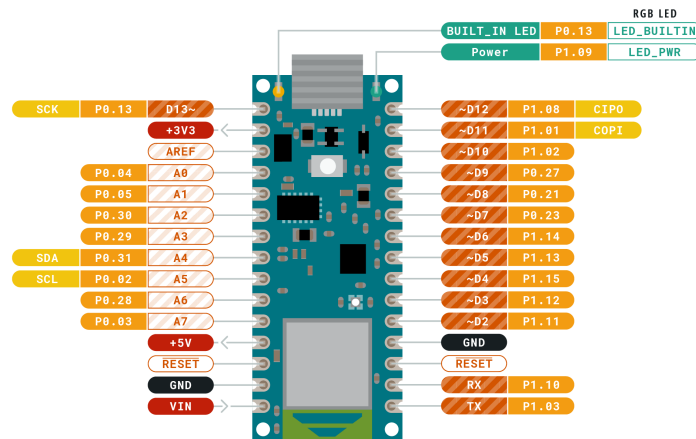


Figura 3: Diagrama de pines para esta placa. [2]

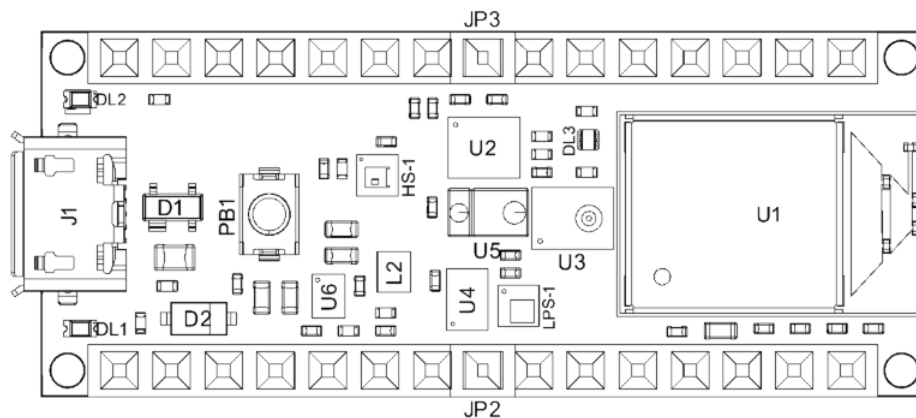


Figura 4: Topología para este microcontrolador. [2]

5.2. Microcontrolador nRF52840

Este microcontrolador es un dispositivo de Nordic Semiconductor, la cual es una compañía que se especializa en conectividad inalámbrica. Es parte de la familia de microcontroladores nRF52. Su diagrama de bloques se puede apreciar en la figura (5). A continuación, se explicaran sus características principales de acuerdo a la hoja de datos del fabricante. [3]

- **Arquitectura y rendimiento:** El nRF52840 está basado en una arquitectura de 32 bits ARM Cortex-M4. Esto le permite alto rendimiento y capacidad de procesamiento. Además, trabaja a una velocidad de reloj de hasta 64 MHz para poder ejecutar tareas de manera rápida y eficiente.
- **Memoria y almacenamiento:** Posee una memoria flash integrada de 1 MB para poder almacenar el firmware del dispositivo. También, tiene una memoria SRAM de 256 KB para la ejecución de código y almacenamiento temporal de datos.
- **Periféricos:** Este dispositivo incluye varios periféricos, como por ejemplo: 48 Puertos GPIO (General Purpose Input/Output) para la interfaz con componentes externos, interfaces de comunicación como UART, I2C y SPI; y convertidores analógico-digital (ADC) con 8 canales para la lectura de sensores.
- **Conectividad inalámbrica:** El nRF52840 soporta diversos protocolos como Bluetooth Low Energy (BLE) y Bluetooth 5. Además de las características propias del BLE, este dispositivo permite velocidades de datos de 2 Mbps para mejorar la transferencia de datos inalámbrica. Finalmente, posee criptografía acelerada por hardware (ARM TrusZone Cryptocell 310 security subsystem) para garantizar la seguridad de la comunicación inalámbrica.
- **Alimentación y características energéticas:** El diseño del nRF52840 busca ser eficiente en términos de consumo de energía. Para esto, incluye modos de bajo consumo que puedan prolongar la duración de la batería en dispositivos que se alimentan de ella.

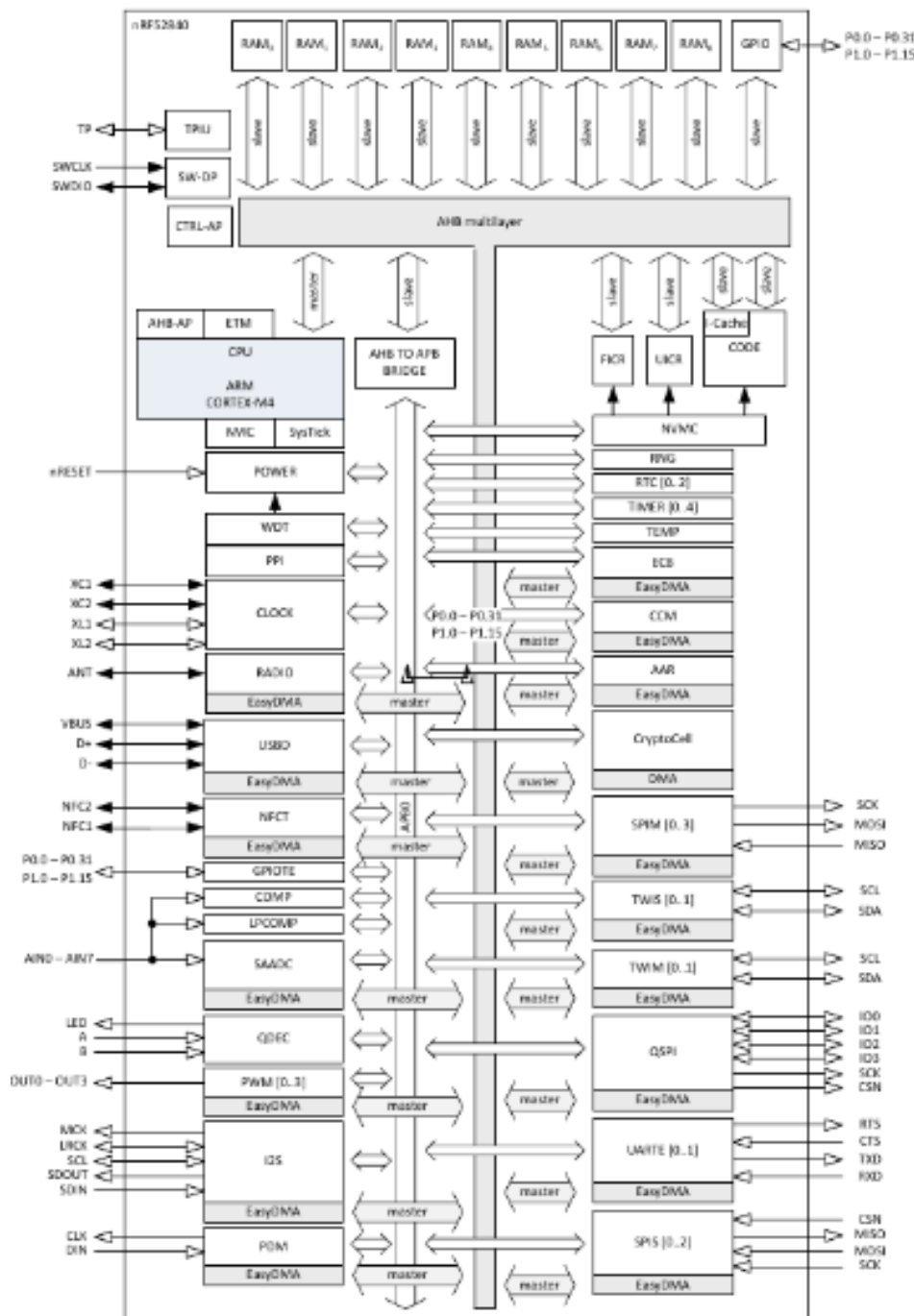


Figura 5: Diagrama de bloques para el nRF52840. [3]

5.3. APDS-9960

El dispositivo APDS-9960 mostrado en la figura (6) es un sensor de detección, reconocimiento de gestos y proximidad desarrollado por Broadcom. Ahora, se explicaran a detalle sus características principales de acuerdo a la hoja de datos para este. [4]

- Reconocimiento de gestos: Este sensor, cuya funcionalidad es controlada mediante una maquina de estados, puede reconocer una variedad de gestos realizados cerca de él tales como deslizamiento hacia arriba, hacia abajo, hacia la derecha, hacia la izquierda, movimientos circulares tanto en sentido de las manecillas del reloj como antihorario.
- Detección de proximidad: Este utiliza un emisor de luz infrarroja y un receptor para detectar objetos cercanos. Con esto puede medir la distancia entre el sensor y cierto objeto en un rango 100 mm aproximadamente.
- Detección de luz ambiente y color: Este dispositivo puede medir la luz ambiental a su alrededor para poder brindar información sobre la intensidad de la luz en el ambiente. Además, este sensor puede detectar y medir el color de la luz ambiente. Esto ultimo permite que sea utilizado en aplicaciones que requieran el uso de RGB. Adicionalmente, cuenta con una función que compensa la luz ambiental para adaptarse a diferentes condiciones de iluminación y garantizar mediciones precisas.
- Interfaz de comunicación: El APDS-9960 se puede comunicar con otros dispositivos por medio de una interfaz I2C (Inter-Integrated Circuit) de dos hilos, lo que facilita su integración en sistemas existentes. Dicha interfaz I2C permite configurar el sensor, recibir datos de detección y controlar estas funciones.



Figura 6: Sensor APDS-9960. [4]

5.4. TensorFlow y TensorFlow Lite

TensorFlow es una biblioteca de Machine Learning de código abierto desarrollada por Google. Permite proporcionar un conjunto de herramientas y mas bibliotecas para desarrollar modelos de

aprendizaje automático en muchas plataformas. Es compatible con diversos lenguajes y además con una gran cantidad de aplicaciones desde el procesamiento de imágenes hasta la detección de objetos. Esta utiliza un modelo de grafo de flujo de datos para así poder definir, entrenar y ejecutar modelos de Machine Learning. [5]

Por otro lado, TensorFlow Lite es una versión optimizada, y como su nombre lo indica, mas ligera que TensorFlow. Se diseñó para ayudar a los desarrolladores a ejecutar sus modelos en dispositivos móviles y sistemas integrados que tienen recursos limitados. Brinda una biblioteca mas agilizada y un intérprete que permite una mejor implementación de modelos de aprendizaje automático en estos dispositivos. Además de que utiliza una API mas sencilla y eficiente. En la figura (7) se muestra la arquitectura para esta versión, y en la figura (8) su flujo de trabajo. [6]

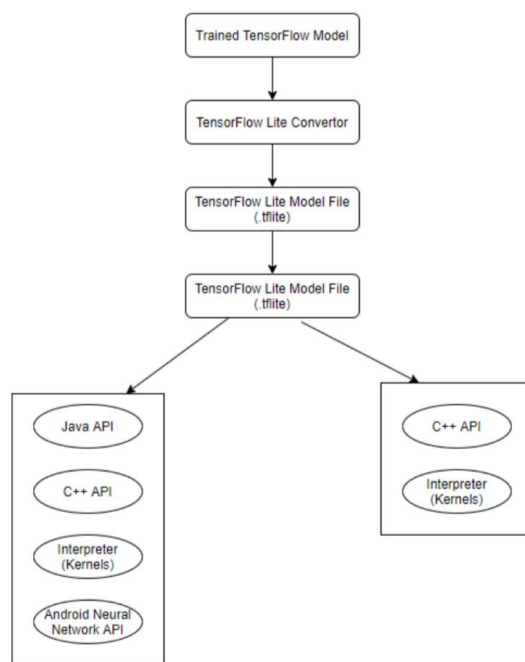


Figura 7: Arquitectura para TensorFlow. [5]

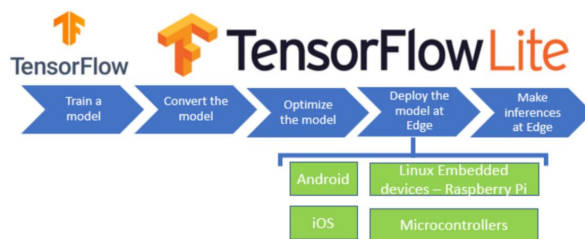


Figura 8: Flujo de trabajo para TensorFlow Lite. [6]

5.5. TinyML

Este concepto de TinyML se refiere a Machine Learning en dispositivos que presentan limitaciones y muy poca potencia tales como: microcontroladores, sistemas embebidos y dispositivos IoT (Internet de las cosas). Es un campo novedoso de las aplicaciones que incluyen algoritmos, hardware y software capaces de analizar datos en dispositivos con sensores. Seguidamente, se detallaran sus principales características. [7]

- Aplicaciones de TinyML: Es utilizada en muchísimas aplicaciones donde el procesamiento de datos y la toma de decisiones en tiempo real son primordiales tales como: reconocimiento de voz, detección de gestos, monitoreo de salud, sistemas de seguridad, entre otros. Con esta nueva tecnología de aprendizaje automático para dispositivos de borde, se logra menor latencia, mayor privacidad de datos y una capacidad mejorada para que pueda funcionar en entornos con conectividad limitada o incluso desconectados.
- Implementación de modelos mas agilizados: Usualmente, los modelos de Machine Learning son grandes y requieren gran cantidad de recursos computacionales para ejecutarse. Para TinyML, se busca que dichos modelos sean más ligeros y optimizados para que puedan adaptarse a las limitaciones de los dispositivos, pero sin alterar mucho la precisión para estas aplicaciones.

5.6. Google Colab

Google Colab es un entorno de desarrollo que permite escribir, ejecutar y colaborar en lenguaje de programación Python. Dicha plataforma es gratuita y ofrece recursos como Jupyter notebooks útiles para Machine Learning, análisis de datos y educación en general. Esto facilita el acceso a recursos potentes de procesamiento sin necesidad de configurarlos localmente. [8]

Este entorno también permite la integración con otros servicios de Google para facilitar la importación y exportación de datos hacia Google Drive y Google Cloud Storage. Además, los notebooks pueden ser compartidos con otros usuarios para que haya colaboración en tiempo real y una de las ventajas es que solo se necesita tener una cuenta en Google. [8]

5.7. Diseño del proyecto

El diseño del presente proyecto, comenzó por la división de los componentes que iban a conformar el total del funcionamiento deseado. En la figura (9), se pueden ver los diferentes componentes que conforman el diseño: El arduino nano BLE 33 Sense, que es la placa principal utilizada, la misma incluye el sensor APDS-9960 y además la Red Neuronal con el programa implementado para la captura e interpretación de datos; El Circuito con alarma, que es el encargado de sonar la alarma cuando la neuronal detecte la posibilidad de fuego mayor a 95 %; por último la PC se utilizó como fuente de alimentación para el Arduino y receptor de datos por serial, ya que ella recibe por consola las estadísticas en tiempo real de la probabilidad que hay de que haya fuego en el ambiente.

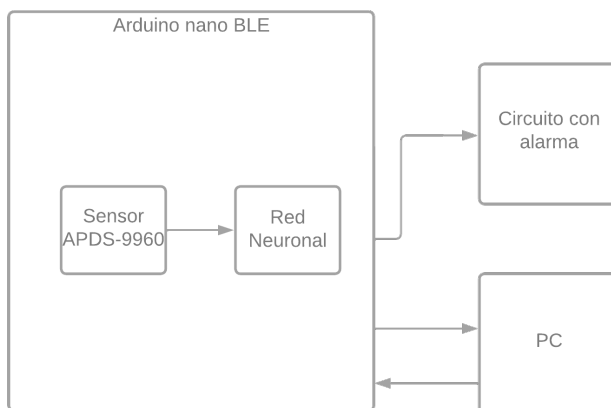


Figura 9: Diagrama de bloques del circuito detector de fuego (Creación propia).

5.7.1. Diseño del circuito

Para la alarma, se implementó un diseño de circuito sencillo, que contase únicamente con 3 componentes externos a la placa principal. Una bocina, que será la que realice el sonido de alerta en caso de que se detecte fuego cerca, un transistor NPN modelo BC547 y una resistencia para controlar la corriente que pasa a través del circuito. La topología del mismo se puede apreciar en la figura (10)

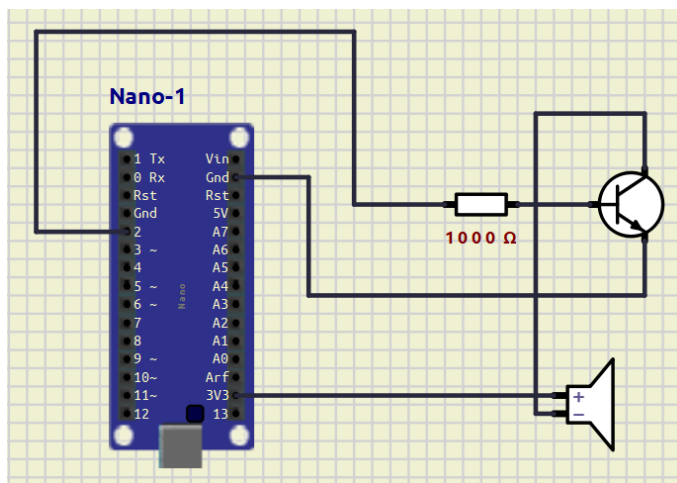


Figura 10: Topología del circuito de alarma para detector de fuego(Creación propia).

Se utilizó un Buzzer de 5V, por lo que la conexión del negativo del buzzer al colector del transistor limita la corriente cuando el transistor no se encuentre en saturación. Para poder saturar el transistor según la hoja de datos del fabricante se requiere de una corriente de base $I_B = 5mA$, además la tensión de salida de el Arduino Nano es de 3.3V en alta, por lo que se tiene la siguiente ecuación:

$$R = \frac{V}{I} = \frac{3,3V}{5mA} = 660\Omega \quad (1)$$

En términos comerciales, se eligió una resistencia de $1k\Omega$, que contiene también la corriente mínima de saturación dentro de sí dando una corriente equivalente a:

$$I = \frac{V}{R} = \frac{3,3}{1000\Omega} = 3,3mA \quad (2)$$

Lo cual sigue saturando al transistor y protege de corrientes altas al mismo. El funcionamiento del circuito reside en la resistencia limitando la corriente del transistor, al D2 estar en alto, el transistor puede conducir, lo que permite que pase corriente a través del buzzer lo que emite sonido, pero cuando el transistor se apaga el buzzer también.

5.8. Tabla de componentes para el desarrollo del proyecto

Componentes	Precio
Arduino Nano BLE 33 Sense Lite TinyML kit	45 350 colones
Resistencia 1kΩ	40 colones
Buzzer 5V	580 colones
Transistor BC547	220 colones
Breadboard de 400 perforaciones	2200 colones
Jumper Macho-Macho (20 uds)	1000 colones
Total	49390 colones

Tabla 1: Tabla precios para los componentes del detector de fuego

6. Desarrollo/Análisis de resultados

6.1. Desarrollo

Para poder implementar el programa de detección de fuego fue necesario primeramente el desarrollo de diferentes bloques, como se mostró en la figura (9), que además permitiese crear el programa principal, que sigue el flujo de la figura (11).

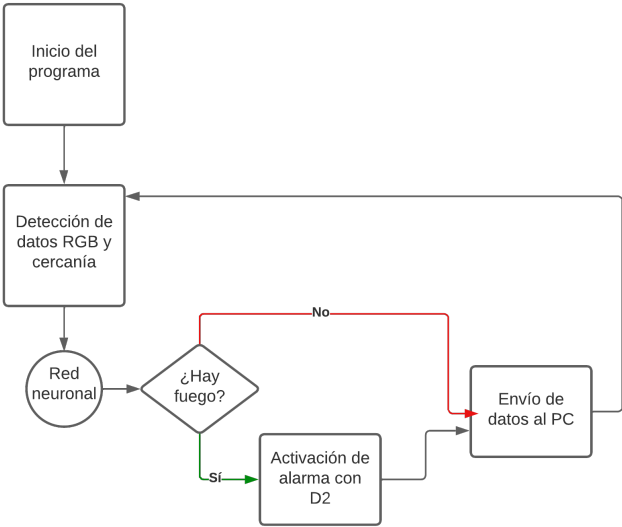


Figura 11: Diagrama de flujo del programa principal(Creación propia).

En primer lugar, se construyó en conjunto un programa de Arduino, que permitiese la captura de los datos del sensor RGB y los enviara por serial. Este código primero incluye la biblioteca APDS9960 para poder controlar el sensor. Seguidamente, dentro de la función `loop()`, se utilizan varias funciones como `colorAvailable()` para verificar si el sensor detecto algún color para leerlo, `APDS.readproximityAvailable()` para revisar si hay objetos cerca.

Luego, la función `APDS.readColor()` permite almacenar los datos de color leídos en las variables `r`, `v` y `a`. Despues, con el condicional `if` se determina si el objeto de interes esta mas proximo para poder con esto calcular la relación RGB. Una vez que se obtienen estos valores en las variables `redRatio`, `greenRatio` y `blueRatio` se imprimen en el monitor serial con sus encabezados correspondientes.

Con este programa es que ya se puede realizar la captura tanto de los datos sin fuego como de los datos con fuego. Este programa se muestra a continuación.

```
1 #include <Arduino_APDS9960.h>
2
3 void setup() {
4   Serial.begin(9600);
5   while(!Serial){};
6
7   if (!APDS.begin()) {
8
9     Serial.println("No se inicio correctamente");
10
11   }
12   Serial.println("Rojo , Verde , Azul");
13 }
14
15
16 void loop() {
17   int r, v, a, c, p;
18   float sum;
19
20   while (!APDS.colorAvailable() || !APDS.proximityAvailable()) {}
21
22
23   APDS.readColor(r,v,a,c);
24   sum = r + v + a;
25
26   p = APDS.readProximity();
27
28   if(p >= 0 && c > 10 && sum > 0){
29
30     float redRatio = r/sum;
31     float greenRatio = v/sum;
32     float blueRatio = a/sum;
33
34     Serial.print(redRatio, 3);
35     Serial.print(',');
36     Serial.print(greenRatio, 3);
37     Serial.print(',');
38     Serial.print(blueRatio, 3);
39     Serial.println();
40 }
```

```

41
42     }
43
44 }
```

Listing 1: Código en Arduino para realizar la captura de los datos.

Seguidamente, se desarrolló un script de Python para la obtención de dichos datos para posteriormente escribirlos en un archivo csv. Por medio de este script se capturaron 6300 datos para cada una de las muestras.

Una vez capturados los datos se implementó un cuaderno de jupyter por medio de Google Colab, que realizara la creación, el entrenamiento y la exportación de la red neuronal para el modelo de TensorFlow Lite. El script creado utiliza como base el ejemplo mostrado en [9], ya que la base que posee es bastante sólida y muestra de forma concreta como se puede instanciar y exportar la red. Dentro del cuaderno de jupyter también se incluyó el manejo de los datos de los CSV, que primero fueron leídos y luego almacenados en data frames para poder utilizarlos como datos de entrenamiento.

```

1 # Se importan las bibliotecas necesarias
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import pandas as pd
5 import tensorflow as tf
6 import os
7 import fileinput
8
9 #Esto ser relevante más adelante para la exportación a tensorflow lite
10 print(f"Versión de tensorflow = {tf.__version__}")
11
12 #Primero se va a definir una semilla aleatoria que no cambie cada vez que se
    instancie el programa dentro del notebook
13 semilla = 6942
14 np.random.seed(semilla)
15 tf.random.set_seed(semilla)
16
17 CLASES = []
18 #Se van a buscar todos los archivos con extensión .csv en el directorio del colab
    para así poder tomar los datos de muestra
19 for file in os.listdir("/content/"):
20     if file.endswith(".csv"):
21         CLASES.append(os.path.splitext(file)[0])
22 CLASES.sort()
23
24 SAMPLES_WINDOW_LEN = 1
25 NUM_CLASSES = len(CLASES)
```

Listing 2: Script para la lectura de datos de los CSV

Para la definición de la red neuronal, se implementó por medio de código una instanciación de una red secuencial, es decir las capas están apiladas, por lo que la capa dos tiene como entrada las salidas de la capa uno y así sucesivamente. En este caso se crearon tres capas, la primera de 8 neuronas con modelo ReLU, que evita linealidades entre los datos de la siguiente manera:

$$f(x) = \begin{cases} y = 0 & \text{if } x < 0 \\ y = x & \text{if } x > 0 \end{cases} \quad (3)$$

Esta definición de función permite que las relaciones entre los datos sean más complejas y la red aprenda a asociarlos de dicha forma, eliminando así la linealidad que permite relaciones directas entre los datos. La segunda capa posee 5 neuronas y tiene el mismo modelo, mientras que la última capa posee 2 neuronas, una por cada resultado que se quiere obtener. El código de la instanciación de la red neuronal es la siguiente:

```

1
2 #Definicion del modelo de tensorflow:
3 modelo = tf.keras.Sequential()
4 #Definicion de la capa 1
5 modelo.add(tf.keras.layers.Dense(8, activation='relu'))
6 #Definicion de la capa 2
7 modelo.add(tf.keras.layers.Dense(5, activation='relu'))
8 #Definicion de la capa 3, con una neurona por cada salida deseada.
9 modelo.add(tf.keras.layers.Dense(NUM_CLASSES, activation='softmax'))
10 modelo.compile(optimizer='rmsprop', loss='mse')
11 history = modelo.fit(inputs_train, outputs_train, epochs=400, batch_size=4,
    validation_data=(inputs_validate, outputs_validate))

```

Listing 3: Código de definición de la red neuronal.

En este mismo código mostrado anteriormente, se realizó el entrenamiento de la red por medio de los datos provistos en los CSVs. Para finalizar con la implementación del cuaderno de jupyter, se utilizó el proceso de exportación visto en [9], para poder pasar el modelo creado de tensorflow a tensorflow lite y este último poder exportarlo a un header file que sea utilizable en el Arduino Nano. El siguiente paso, fue implentar un código que capturase los datos por medio del sensor RGB, como el mostrado para la captura de datos de entranmiento, pero que ahora enviase estos datos al modelo exportado y este los pudiese clasificar.

Si bien el algoritmo consigue el mismo objetivo del anterior, que es la toma de datos por medio del sensor, a partir de que se tienen los datos en vez de enviarlos por serial, se envían directamente como entradas a la red neuronal. La red neuronal da salidas de la interpretación de dichos datos y dependiendo del nivel de certidumbre obtenido se activa la alarma. En este caso, por temas de diseño, se escogió un umbral de 95 % de certidumbre para que active la alarma. El código que implementa este programa es el siguiente:

```

1 #include <Arduino_APDS9960.h>
2
3 #include <TensorFlowLite.h>
4 #include <tensorflow/lite/micro/all_ops_resolver.h>
5 #include <tensorflow/lite/micro/micro_interpreter.h>
6 #include <tensorflow/lite/schema/schema_generated.h>
7
8 #include "model.h"
9
10 unsigned long buzzerStartTime = 0;
11 const unsigned long buzzerDuration = 2000;
12 bool toneActive = false;
13
14 // pull in all the TFLM ops, you can remove this line and
15 // only pull in the TFLM ops you need, if would like to reduce
16 // the compiled size of the sketch.
17 tf::AllOpsResolver tflOpsResolver;
18
19 const tf::Model* tflModel = nullptr;

```

```
20 tflite::MicroInterpreter* tflInterpreter = nullptr;
21 TfLiteTensor* tflInputTensor = nullptr;
22 TfLiteTensor* tflOutputTensor = nullptr;
23
24 // Create a static memory buffer for TFLM, the size may need to
25 // be adjusted based on the model you are using
26 constexpr int tensorArenaSize = 8 * 1024;
27 byte tensorArena[tensorArenaSize] __attribute__((aligned(16)));
28
29 //Arreglo para mostrar las dos posible salidas que quiero identificar.
30 const char* CLASSES[] = {
31     "fuego",
32     "no fuego"
33 };
34
35 void setup() {
36     Serial.begin(9600);
37     while (!Serial);
38
39     pinMode(2, OUTPUT);
40
41     if (!APDS.begin()) {
42         Serial.println("Error inicializando el sensor de RGB");
43     }
44
45     //get the TFL representation of the model byte array
46     tflModel = tflite::GetModel(model);
47     if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
48         Serial.println("La versi n del modelo no coincide!");
49         while (1);
50     }
51
52     // Create an interpreter to run the model
53     tflInterpreter = new tflite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena, tensorArenaSize);
54
55     // Allocate memory for the model's input and output tensors
56     tflInterpreter->AllocateTensors();
57
58     // Get pointers for the model's input and output tensors
59     tflInputTensor = tflInterpreter->input(0);
60     tflOutputTensor = tflInterpreter->output(0);
61 }
62
63 void loop() {
64     int r, g, b, c, p;
65     float suma;
66
67     // Esperar hasta que los datos de color y proximidad est n disponibles
68     while (!APDS.colorAvailable() || !APDS.proximityAvailable()) {};
69
70     // Leer los valores de color y proximidad
71     APDS.readColor(r,g,b,c);
72     suma = r+g+b;
73     p = APDS.readProximity();
74 }
```

```

75 // Verificar la condición mínima para evaluar los valores capturados
76 if (p >= 0 && c > 10 && suma > 0){
77     float red = r/suma;
78     float green = g/suma;
79     float blue = b/suma;
80
81     // Asignar los valores de entrada al tensor del modelo
82     tflInputTensor->data.f[0] = red;
83     tflInputTensor->data.f[1] = green;
84     tflInputTensor->data.f[2] = blue;
85
86     // Llamar al modelo para poder utilizar sus resultados
87     TfLiteStatus invokeStatus = tflInterpreter->Invoke();
88     if (invokeStatus != kTfLiteOk){
89         Serial.println("Fallo el llamado a la red");
90         while(1);
91         return;
92     }
93
94     // Imprimir las certidumbres
95     for (int i = 0; i < 2; i++){
96         Serial.print(CLASES[i]);
97         Serial.print(" ");
98         Serial.print(int(tflOutputTensor->data.f[i]*100));
99         Serial.print("%\n");
100     }
101
102     Serial.println();
103
104     // Verifica el umbral establecido para poder activar el tono de alarma
105
106     if(int(tflOutputTensor->data.f[0]*100) > 95){
107         if (!toneActive){
108             tone(2, 1000, 2000);
109         }
110     }
111     else if(int(tflOutputTensor->data.f[1]*100) > 95){
112         noTone(2);
113     }
114
115 }
116
117 }

```

Listing 4: Código de detección de fuego para el Arduino Nano con la red neuronal exportada

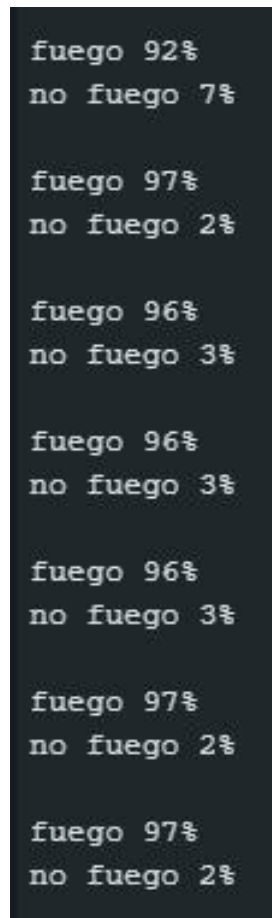
6.2. Análisis de Resultados

Para poder analizar los resultados del proyecto, se compiló el programa mostrado anteriormente y se cargó al Arduino Nano BLE 33 Sense, con el modelo de red neuronal ya implementado. Luego, se volvió a realizar la captura de datos pero esta vez para verificar la precisión del modelo creado y analizar si estaba prediciendo correctamente cuando había o no fuego. En la figura (12) se muestran los resultados obtenidos de cuando se realizan las pruebas con el sensor de la placa en un ambiente normal. Se aprecia como se predice de forma correcta que no existe fuego cerca.

```
fuego 16%  
no fuego 83%  
  
fuego 16%  
no fuego 83%  
  
fuego 5%  
no fuego 94%  
  
fuego 5%  
no fuego 94%  
  
fuego 5%  
no fuego 94%  
  
fuego 5%  
no fuego 94%  
  
fuego 10%  
no fuego 89%
```

Figura 12: Muestra de datos sin fuego

Por otra parte, en la figura (13), se observan las predicciones que realiza el modelo entrenado cuando se hacen las pruebas con fuego presente. Se muestra para este caso que efectivamente se logra predecir con éxito la existencia de fuego.



```
fuego 92%  
no fuego 7%  
  
fuego 97%  
no fuego 2%  
  
fuego 96%  
no fuego 3%  
  
fuego 96%  
no fuego 3%  
  
fuego 96%  
no fuego 3%  
  
fuego 97%  
no fuego 2%  
  
fuego 97%  
no fuego 2%
```

Figura 13: Muestra de datos con fuego

Gracias a la gran cantidad de datos capturados, además de la correcta implementación y modificación del código del ejemplo provisto por Arduino, se puede concluir que la implementación para este proyecto se alcanzo satisfactoriamente.

7. Conclusiones, Recomendaciones y Observaciones

7.1. Conclusiones

- Implementar un modelo de red neuronal que provea datos de predicción e identificación adecuado, requiere de una captura de datos de entrenamiento extenso.
- Crear modelos de redes neuronales buenos, requieren de mucha potencia computacional.
- Ejecutar un modelo de redes neuronales no requiere de mucha potencia computacional si se realiza de forma correcta.
- El uso de la herramienta de desarrollo de google collab, para desarrollar, instanciar, entrenar y exportar redes neuronales es de suma importancia.
- Herramientas como TensorFlow Lite son indispensables para desarrollar un proyecto de este tipo considerando que se tienen recursos limitados.

- Utilizar bibliotecas reconocidas permiten una mejor implementación de AI por toda la documentación disponible y trabajos previamente realizados.

7.2. Recomendaciones y observaciones

- La recopilación de datos de entrenamiento es fundamental para desarrollar un modelo de aprendizaje automático preciso. Se recomienda capturar dichos datos en una variedad de condiciones con bastante luz y presencia de fuego para obtener resultados más seguros. Además, es indispensable tomar en cuenta la calidad y cantidad de los datos de entrenamiento para obtener un modelo mas preciso.
- El hecho de trabajar con fuego y electricidad se vuelve riesgoso. Es por esto que se deben tomar las medidas y precauciones apropiadas, como por ejemplo, tener un extintor cerca y un entorno con ventilación.
- Es necesario considerar otros sensores además del utilizado en este proyecto. Esto puede ser una buena manera de complementar el desarrollo para una detección mas completa, como por ejemplo: agregar un sensor de temperatura y un sensor de humo para evitar confusiones con los colores del entorno.

Bibliografía

1. Google. *Los detectores de humo, elemento fundamental de seguridad*, 2021. [Online]. Disponible en: <https://www.nfpajla.org/blog/1747-la-importancia-de-las-alarmas-de-humo-la-planificacion-de-evacuacion-y-los-rociadores> [Accesado en: Abril 30, 2023].
2. Arduino Nano 33 BLE Sense Datasheet. (2022). Retrieved from Arduino website: <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf> [Accesado: Junio 16, 2023].
3. Nordic Semiconductor. (2018). nRF52840 Product Specification v1.4. Retrieved from Nordic Semiconductor website: https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf [Accesado: Junio 16, 2023].
4. Broadcom Limited. (2016). APDS-9960 Proximity, Light, RGB, and Gesture Sensor. Retrieved from Broadcom Limited website: <https://docs.broadcom.com/doc/AV02-4191EN> [Accesado en: Abril 29, 2023].
5. Google Inc. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. [Online]. Disponible en: <https://www.tensorflow.org/>. [Accesado en: Mayo 2, 2023].
6. Google. *TensorFlow Lite for Microcontrollers Library*, 2021. [Online]. Disponible en: <https://www.tensorflow.org/lite/microcontrollers>. [Accesado en: Abril 30, 2023].
7. TinyML Foundation, *We are tinyML*, 2023. [Online]. Disponible en: <https://www.tinyml.org/about/> [Accesado: Junio 18, 2023].
8. Google. *Welcome to Colaboratory*, 2022. [Online]. Disponible en: https://colab.research.google.com/?utm_source=scs-index. [Accesado en: Abril 30, 2023].
9. Arduino. *Get Started With Machine Learning on Arduino*, 2023. [Online]. Disponible en <https://docs.arduino.cc/tutorials/nano-33-ble-sense/get-started-with-machine-learning> [Accesado en: Junio 22, 2023]