

# Despliegue de un Modelo de IA en dispositivo Tiny Machine Learning

Cristian P., Ivan S.

Especialización IA, Universidad Autónoma de Occidente. Cali, Colombia

(Noviembre de 2022)

## Resumen -

En el presente documento presenta los resultados del laboratorio con los temas de en una tarjeta de desarrollo como el TinyML Kit de Arduino u otro dispositivo de Edge Computing, para un data set personalizado cinco clases sobre información de audio y de movimiento.

Como objetivo final se escogió el desarrollo de una aplicación de detección de movimientos básicos para el estiramiento de los músculos del cuerpo, la cual el algoritmo fue entrenado utilizando Colab, cuyo modelo fue desplegado sobre hardware Arduino Nano 33 BLE. La inferencia del despliegue en hardware tiene un interfaz visual web, el cual visualiza si el movimiento es realizado correctamente dentro la rutina de estiramiento. En total se trabajo sobre 5 clases de movimientos, los cuales se les capturo los datos, gracias al hardware del acelerómetro y giroscopio contiene el hardware del Arduino

*Palabras Clave: captura, entrenamiento, despliegue, clasificación, y movimiento.*

## I. INTRODUCCIÓN

El Tiny ML se ha convertido en una opción más cercana para el despliegue de aplicaciones en hardware, buscando interactividad con los usuarios a un bajo coste, en dispositivos integrados basados en microcontroladores y alimentados por batería que pueden realizar tareas de ML con respuesta en tiempo real.



Figura 1 Kit tarjeta Arduino Nano 33 BLE. [1]

Para el desarrollo de modelo se utiliza el software TensorFlow lite para microcontroladores, esta herramienta fue desarrollado por Google para ejecutarse en esta clase de dispositivos de hardware.

La motivación de este trabajo es aprovechar los conceptos vistos en clase sobre el procesamiento de datos secuenciales, en este caso la captura de datos de movimiento [2], entrenar un modelo, y realizar su despliegue sobre hardware, en este caso el dispositivo Arduino Nano 33 BLE.

## II. APLICACIÓN DE CLASIFICACIÓN DE MOVIMIENTO

La primera parte del entregable en el punto No 1. el proyecto contempla capturar los 5 movimientos (categorías) para generar un dataset propio, se requiere entrenar dos modelos

basado en RNN, uno basado en convolución 1D y otro en redes MLP.

### PROCESO DE OBTENCIÓN DE LOS DATOS

Para el proceso de captura y creación del dataset con los datos de movimiento se realizó a través del siguiente procedimiento:

Se utilizó el hardware de desarrollo Arduino Nano 33 BLE. Este dispositivo conectado al pc y utilizando IDE de desarrollo de Arduino se copilo y transfirió el programa "IMU\_Capture.ino", para utilizar el sensor inercial (acelerómetro y giroscopio)

Captura de los datos generado por el sensor inercial y transmitidos por el puerto serial, para su visualización se utilizó una sesión terminal, cada movimiento realizado arroja un bloque de datos con las coordenadas aX, aY, aZ; y gX, gY, gZ, se produjo un dataset con un total de 10 capturas por cada uno de los movimientos propuestos. El archivo al final con los datos, se guardo con la extensión .csv para el entrenamiento del modelo.

### MODELO BASADO EN CONVOLUCIÓN 1D

Para entrenar este modelo, se validan el tipo de datos de entrada.

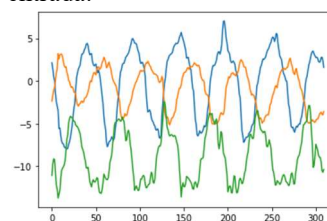


Figura 2. Datos de entrada en coordenadas X, Y Z (acelerómetro)

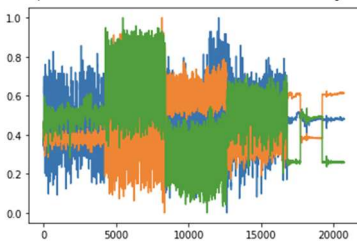


Figura 3. Visualización nformacion capturada

El modelo propuesto para entrenar la red.

```
modelo = keras.models.Sequential()
modelo.add(keras.layers.Conv1D(16, 3,
activation="relu",padding="same", input_shape=(90,3)))
modelo.add(keras.layers.Conv1D(32, 3,
activation="relu",padding="same"))
modelo.add(keras.layers.Conv1D(64, 3,
activation="relu",padding="same"))
modelo.add(keras.layers.MaxPooling1D(pool_size=2, strides=2,
padding='same'))
modelo.add(keras.layers.Flatten())
modelo.add(keras.layers.Dense(5, activation = 'softmax'))
# Se muestra el resumen de la arquitectura del modelo
modelo.summary()
# Se muestra un esquema del modelo
keras.utils.plot_model(modelo, to_file='model_plot3.png',
show_shapes=True, show_layer_names=True)
```

Model: "sequential_16"		
Layer (type)	Output Shape	Param #
conv1d_48 (Conv1D)	(None, 90, 16)	160
conv1d_49 (Conv1D)	(None, 90, 32)	1568
conv1d_50 (Conv1D)	(None, 90, 64)	6208
max_pooling1d_16 (MaxPoolin g1D)	(None, 45, 64)	0
flatten_16 (Flatten)	(None, 2880)	0
dense_16 (Dense)	(None, 5)	14405
-----		
Total params: 22,341		
Trainable params: 22,341		
Non-trainable params: 0		

Figura 4. Parámetros de l modelo

Entrenamiento del modelo.

```
modelo.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
historia = modelo.fit(Xtrain, Ytrain, epochs=250, batch_size=None)
```

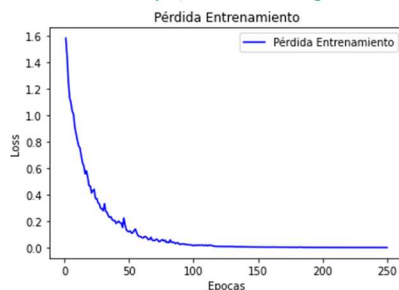


Figura 5. Resultado modelo -Loss

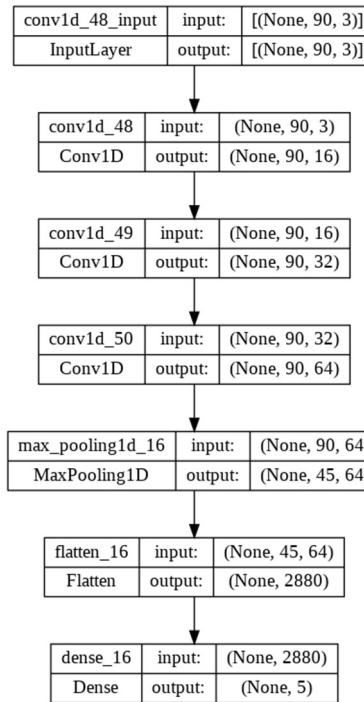


Figura 6. Diagrama del modelo

Evaluación del modelo.

```
# Se evalua el modelo con los datos de testeo
modelo.evaluate(XVal, YVal)
```

3/3 [===] - 0s 5ms/step - loss: 0.2910 - accuracy: 0.9333

Prueba de test.

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
ypredic=modelo.predict(XVal)
y_test_class = np.argmax(YVal,axis=1)
y_pred_class = np.argmax(ypredic,axis=1)
#Accuracy of the predicted values
print(classification_report(y_test_class, y_pred_class)) # Precision ,
Recall, F1-Score & Support
cm = confusion_matrix(y_test_class, y_pred_class)
print(cm)
# visualize the confusion matrix in a heat map
df_cm = pd.DataFrame(cm)
heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
```

	precision	recall	f1-score	support
0	0.81	0.94	0.87	18
1	1.00	0.94	0.97	18
2	1.00	1.00	1.00	18
3	0.94	0.83	0.88	18
4	0.94	0.94	0.94	18
accuracy			0.93	90
macro avg	0.94	0.93	0.93	90
weighted avg	0.94	0.93	0.93	90

Figura 7. Resultados de test

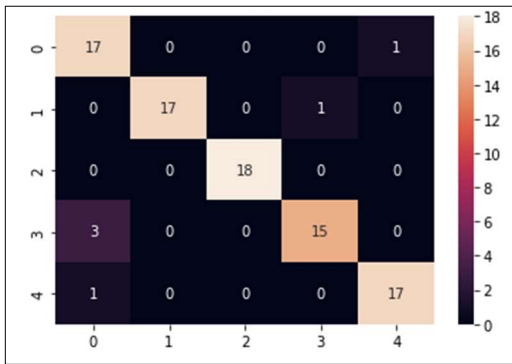


Figura 8. Matriz de confusión

### MODELO BASADO EN MLP

Para entrenar este modelo, se validan el tipo de datos de entrada.

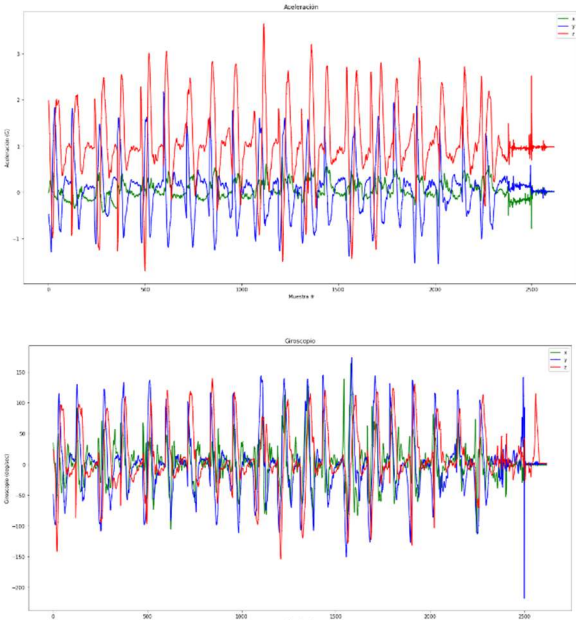


Figura 9. Datos entrada acelerómetros y giroscopio

Modelo propuesto para entrenar la red.

```
# Definición y entrenamiento del modelo
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(50, activation='relu'))
model.add(tf.keras.layers.Dense(15, activation='relu'))
model.add(tf.keras.layers.Dense(NUM_GESTURES, activation='softmax'))
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
history = model.fit(inputs_train, outputs_train, epochs=600, batch_size=1,
validation_data=(inputs_validate, outputs_validate))
```

Layer (type)	Output Shape	Param #
dense (Dense)	(1, 50)	35750
dense_1 (Dense)	(1, 15)	765
dense_2 (Dense)	(1, 5)	80
=====		
Total params:	36,595	
Trainable params:	36,595	
Non-trainable params:	0	

Figura 10. Parámetro del modelo

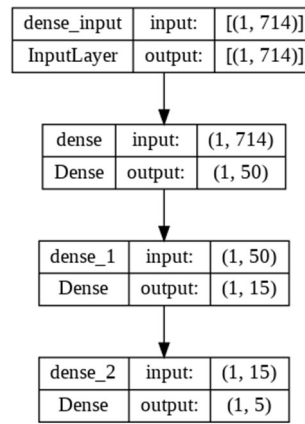


Figura 11. Diagrama del modelo

### Resultados entrenamiento del modelo.

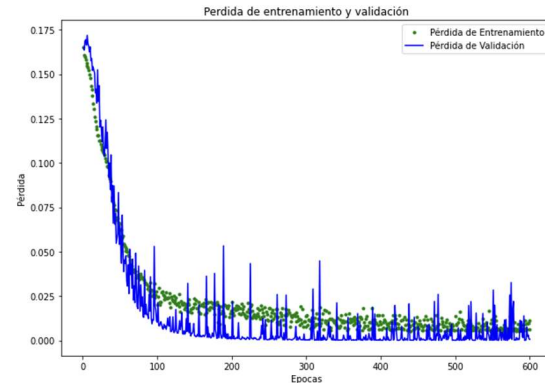


Figura 12. Resultado entrenamiento -Loss

### Test al modelo.

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

y_test_class = np.argmax(outputs_test,axis=1)
y_pred_class = np.argmax(predictions,axis=1)
#Exactitud de los valores predichos
print(classification_report(y_test_class, y_pred_class)) # Precision ,
Recall, F1-Score & Support
cm = confusion_matrix(y_test_class, y_pred_class)
print(cm)
# visualizar la matriz de confusión en un mapa de calor
df_cm = pd.DataFrame(cm)
heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
```

	precision	recall	f1-score	support
0	1.00	0.75	0.86	4
1	1.00	1.00	1.00	3
2	0.83	1.00	0.91	5
3	1.00	1.00	1.00	4
4	1.00	1.00	1.00	4
accuracy			0.95	20
macro avg	0.97	0.95	0.95	20
weighted avg	0.96	0.95	0.95	20

Figura 13. Resultado test modelo

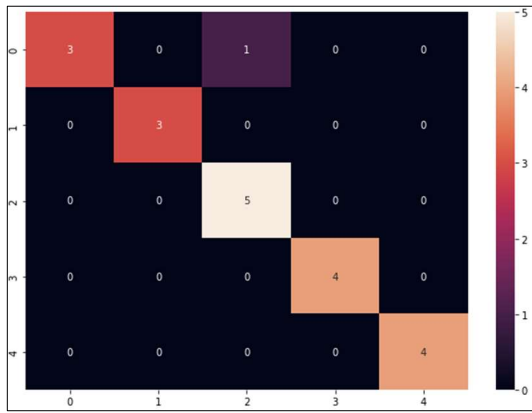


Figura 14. Matriz de confusión

#### ENTRENAMIENTO MODELO CON EDGE IMPULSE

Realizando el mismo procedimiento capturando los datos desde Edge impulse [3] y realizando el entrenamiento desde la herramienta se obtiene los siguientes resultados:

(<https://studio.edgeimpulse.com/public/160464/latest/learning/keras/17>)

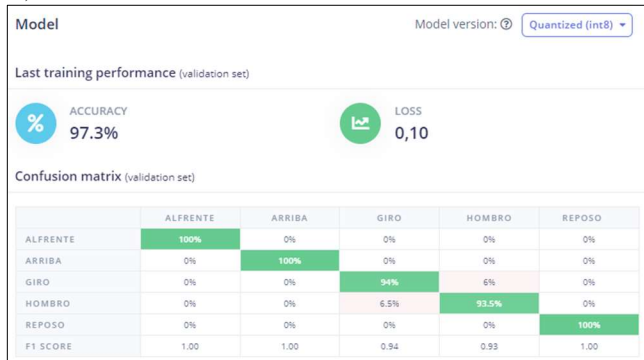


Figura 15. Entrenamiento con edge impulse

Los resultados para la parte de test

(<https://studio.edgeimpulse.com/public/160464/latest/validation>)



Figura 16. Test edge impulse

### III. APLICACIÓN CLASIFICACIÓN DE AUDIO

La segunda parte del entregable contempla capturar 5 clases de audio para generar un dataset propio, se requiere entrenar dos modelos basados en RNN, uno utilizando convolución 2D con una entrada utilizando espectrograma y otro en MFCC

#### PROCESO DE OBTENCIÓN DE LOS DATOS

Para el proceso de captura y creación del dataset con datos de audio se utilizó el Arduino como dispositivo y la aplicación edge impulse.

Los audios capturados hacen referencia a comandos de voz utilizados en el adiestramiento canino (ABAJO, QUIETO, RUIDO, SENTADO, VEN)

#### MODELO BASADO EN CONVOLUCIÓN 2D

Para entrenar este modelo, se validan el tipo de datos de entrada.

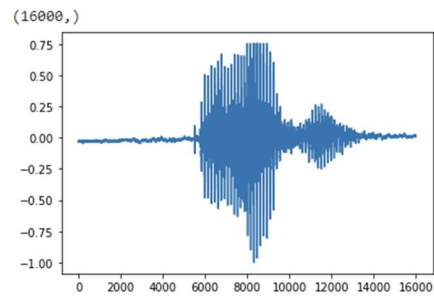


Figura 17. Archivo de audio

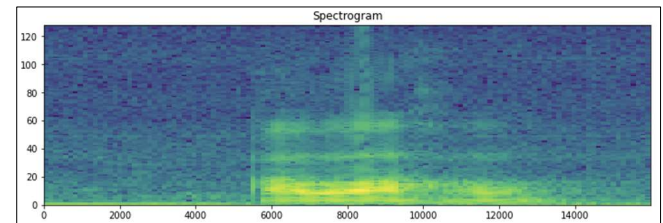


Figura 18. Espectrograma

El modelo propuesto para entrenar la red

#Definición del modelo

```
modelo = keras.models.Sequential()
modelo.add(keras.layers.Conv2D(8, 3,
activation="relu",padding="same", input_shape=(124,129,1)))
modelo.add(keras.layers.Conv2D(16, 3,
activation="relu",padding="same"))
modelo.add(keras.layers.Conv2D(32, 3,
activation="relu",padding="same"))
modelo.add(keras.layers.Conv2D(64, 3,
activation="relu",padding="same"))
#modelo.add(keras.layers.Conv2D(128, 3,
activation="relu",padding="same"))
modelo.add(keras.layers.MaxPooling2D(pool_size=2, strides=2,
padding='same'))
modelo.add(keras.layers.Flatten())
modelo.add(keras.layers.Dense(5, activation = 'softmax'))
modelo.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 124, 129, 8)	80
conv2d_1 (Conv2D)	(None, 124, 129, 16)	1168
conv2d_2 (Conv2D)	(None, 124, 129, 32)	4640
conv2d_3 (Conv2D)	(None, 124, 129, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 62, 65, 64)	0
flatten (Flatten)	(None, 257920)	0
dense (Dense)	(None, 5)	1289605
Total params: 1,313,989		
Trainable params: 1,313,989		
Non-trainable params: 0		

Figura 19. Parametros del modelo

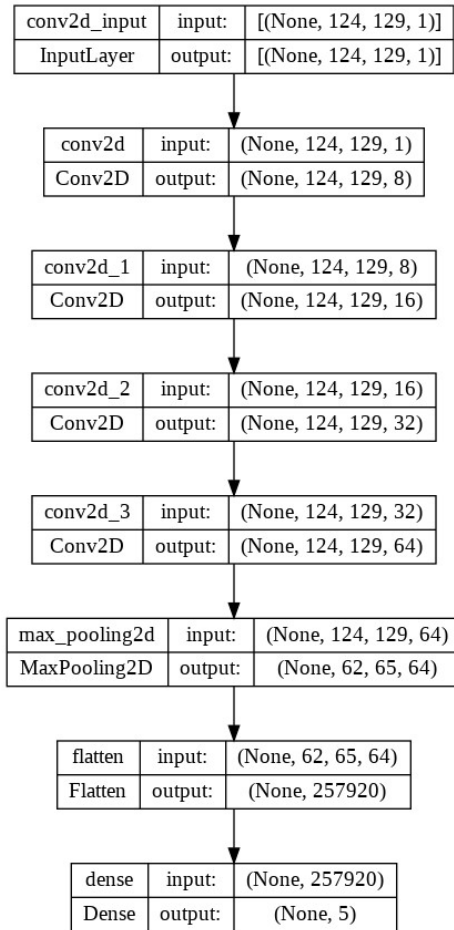


Figura 20. Diagrama del modelo

### Entrenamiento del modelo.

```

modelo.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
history =
modelo.fit(Xtrain,Ytrain,epochs=70,batch_size=None,validation_data=(XVal,
YVal),verbose=1)

```

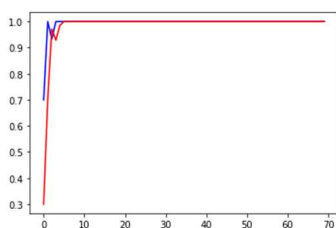


Figura 21. Val accuracy -accuracy

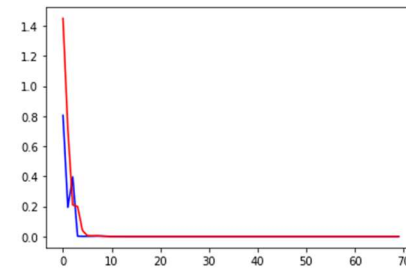


Figura 22. Grafica de val-Loss

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	6
2	1.00	1.00	1.00	6
3	1.00	1.00	1.00	6
4	1.00	1.00	1.00	6
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Figura 23. Resultado test modelo

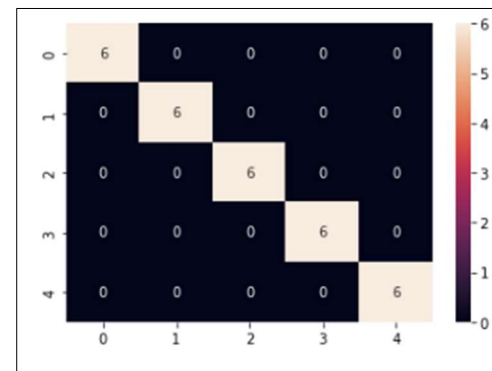


Figura 24. Matriz de confusión

### MODELO BASADO EN MFCC

Para entrenar este modelo, se hace uso de la librería librosa y numba

El modelo propuesto para entrenar la red

```

modelo_mfcc= keras.models.Sequential()
modelo_mfcc.add(keras.layers.Dense(50,activation="relu",
input_shape=(50,)))
modelo_mfcc.add(keras.layers.Dense(50,activation="relu"))
modelo_mfcc.add(keras.layers.Dense(20,activation="relu"))
modelo_mfcc.add(keras.layers.Dense(20,activation="relu"))
modelo_mfcc.add(keras.layers.Flatten())
modelo_mfcc.add(keras.layers.Dense(5, activation = 'softmax'))
modelo_mfcc.summary()

```



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 50)	2550
dense_2 (Dense)	(None, 50)	2550
dense_3 (Dense)	(None, 20)	1020
dense_4 (Dense)	(None, 20)	420
flatten_1 (Flatten)	(None, 20)	0
dense_5 (Dense)	(None, 5)	105
Total params: 6,645		
Trainable params: 6,645		
Non-trainable params: 0		

Figura 25. Parametros del modelo

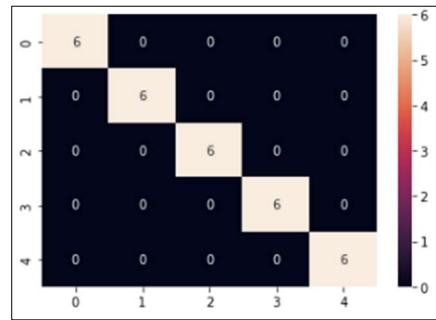


Figura 29. Matriz de confusión

## Entrenamiento del modelo

```
modelo_mfcc.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
history_mfcc =
modelo_mfcc.fit(xtrain_mfcc, Ytrain, epochs=100, batch_size=None, validation
_data=(x_test_mfcc, YVal), verbose=1, callbacks=[cb])
```

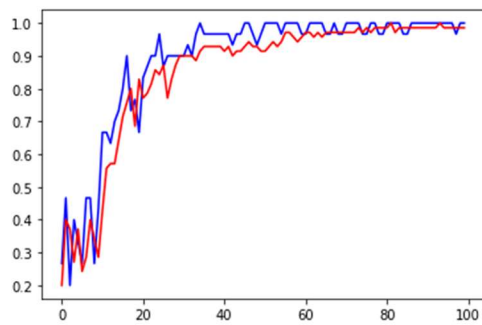


Figura 26. Val accuracy . accuracy

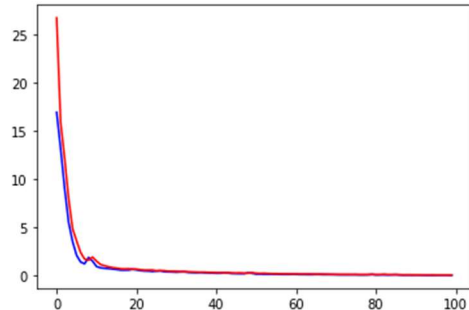


Figura 27. Grafica de Val-Loss

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	6
2	1.00	1.00	1.00	6
3	1.00	1.00	1.00	6
4	1.00	1.00	1.00	6
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Figura 28. resultados evaluación modelo

## ENTRENAMIENTO CON EDGE IMPULSE

Realizando el mismo procedimiento capturando los datos desde Edge impulse y realizando el entrenamiento desde la herramienta se obtiene los siguientes resultados:

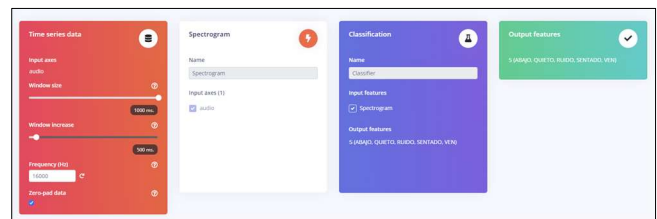




Figura 30. Utilizando espectrograma


<https://studio.edgeimpulse.com/public/160496/latest/create-impulse>

Model

Model version:  Quantized (int8)

Last training performance (validation set)

 ACCURACY  
85.7%

 LOSS  
0,50

Confusion matrix (validation set)

	ABAJO	QUIETO	RUIDO	SENTADO	VEN
ABAJO	100%	0%	0%	0%	0%
QUIETO	0%	0%	0%	0%	100%
RUIDO	0%	0%	100%	0%	0%
SENTADO	0%	0%	0%	100%	0%
VEN	0%	0%	0%	0%	100%
F1 SCORE	1.00	0.00	1.00	1.00	0.67

Figura 31. Entrenamiento edge impulse

## Los resultados para la parte de test

Model testing results

%

ACCURACY

90.00%

	ABAJO	QUIETO	RUIDO	SENTADO	VEN	UNCERTAIN
ABAJO	100%	0%	0%	0%	0%	0%
QUIETO	0%	66.7%	0%	0%	0%	33.3%
RUIDO	0%	0%	100%	0%	0%	0%
SENTADO	0%	16.7%	0%	83.3%	0%	0%
VEN	0%	0%	0%	0%	100%	0%
F1 SCORE	1.00	0.73	1.00	0.91	1.00	

Figura 32. Test edge impulse

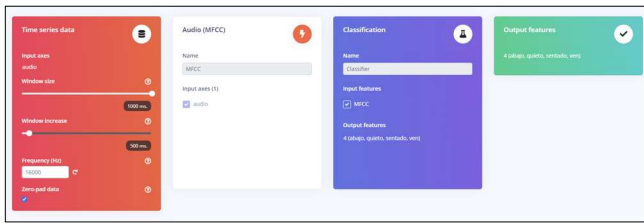


Figura 33. Utilizando el MFCC

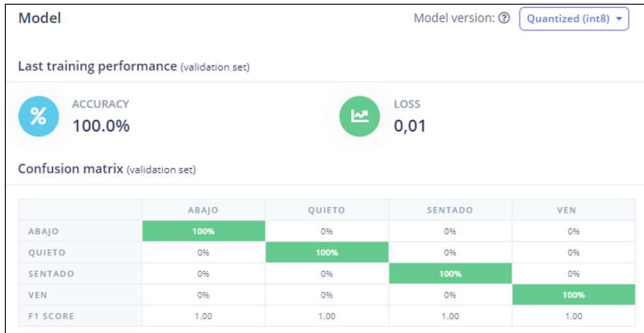


Figura 34. Entrenamiento con edge impulse

<https://studio.edgeimpulse.com/public/162412/latest/learning/keras/5>

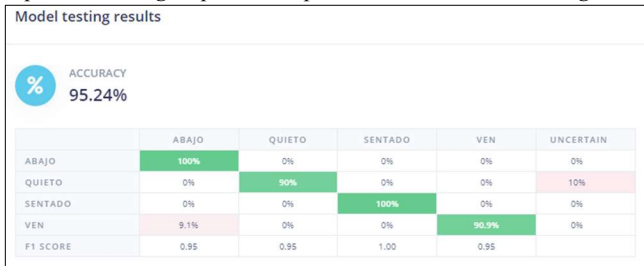


Figura 35. Test con el edge impulse

<https://studio.edgeimpulse.com/public/162412/latest/validation>

#### IV. DESPLIEGUE DE MODELO EN ARDUINO

Para el desarrollo de la aplicación de ML para despliegue en el Arduino se escogió el modelo de clasificación de movimiento.

A partir de modelo ya entrenado de construye la versión del modelo para TensorFlow Lite[4], para despliegue en el Arduino

#La siguiente celda crea una matriz de bytes constantes que contiene el modelo TF Lite.

#Importarlo como una pestaña con el siguiente código.

```
!echo "const unsigned char model[] = {" > /content/model.h
!cat gesture_model.tflite | xxd -i >> /content/model.h
!echo "};" >> /content/model.h
```

```
import os
model_h_size = os.path.getsize("model.h")
print(f"Header file, model.h, is {model_h_size:,} bytes.")
print("\nAbre el panel lateral (refrecar si es necesario). Doble click en model.h para descargar el archivo.")
```

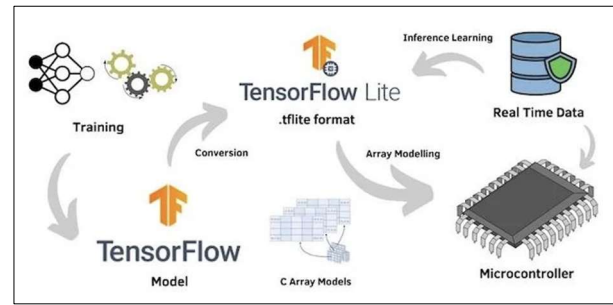


Figura 36. Descripción proceso de despliegue en Arduino[1]

#### OBJETIVO DE LA APLICACIÓN

Con un modelo ya entrenado, y el despliegue de modelos en sistemas embebidos, facilita la interacción del usuario final con la aplicación, en este caso un modelo de clasificación de movimientos en tiempo real.

#### DESCRIPCIÓN DEL PROYECTO

El proyecto tiene la finalidad de generar interacción con las personas que están la mayor parte del tiempo sentadas y requieren de una pausa activa para el estiramiento de los músculos del cuerpo



Figura 37. Movimientos de la aplicación

#### ARQUITECTURA DE LA SOLUCIÓN

En la parte de hardware se cuenta con el Arduino Nano 33 BLE, el cual tiene la inferencia del modelo desplegado. La salida de la inferencia se comunica con la aplicación desarrollada en NodeJS que cuenta con las librerías de serialport io [5] y socket io [6], cada vez que se realiza un movimiento se despliega en pantalla la inferencia del modelo.

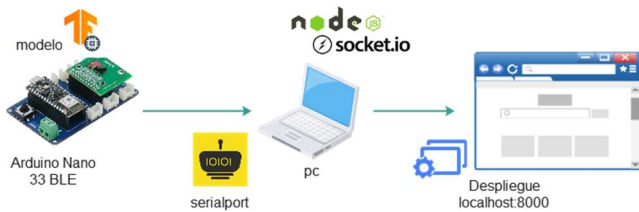


Figura 38. Arquitectura aplicación

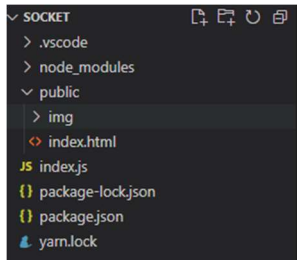


Figura 39. Proyecto nodejs

#### OBSTÁCULOS PRESENTADOS Y SUS SOLUCIONES

Vale la pena mencionar los obstáculos que se tuvieron para el desarrollo de la aplicación, de los mas representativo fue capturar los datos de movimiento cuando se realizaban las repeticiones correspondientes, ya que la construcción de dataset es totalmente manual y como mitigación se realizó esta actividad muchas veces, hasta quedar conforme con las muestras tomadas.

Otro obstáculo que se nos presentó fue como desplegar la interfaz del hardware con una aplicación web en tiempo real, ya que no se tenían la experiencia en programación para este tipo de proyectos, se realizo bastantes consultas e investigación respecto a proyectos con soluciones similares y adquirir las herramientas mediante tutoriales en línea de como manejar las herramientas de despliegue con nodejs

#### VI. CONCLUSIONES

El avance en el campo de Tiny ML, ha permitido que los desarrolladores puedan explorar hardware y lograr una mayor interactividad en tiempo real de las aplicaciones, en este caso de la detección y clasificación de movimientos para un caso de uso real.

Los movimientos como el audio son señales que su procesamiento se desarrollo en el tiempo, y con las herramientas de captura como los dispositivos móviles o embebidos de bajo coste, se pueden desarrollar aplicaciones que interaccionen con su entorno en tiempo real en Edge computing.

Las herramientas como Edge Impulse, permiten crear de manera didáctica y muy sencilla modelos utilizando dispositivos de borde, y con una conexión a internet y su ejecución en nube se logra realizar proyectos muy sencillos en el ML desplegados en hardware

#### REFERENCIAS

- [1] A. about Circuits, “¿Qué es TinyML?” <https://www.allaboutcircuits.com/technical-articles/what-is-tinyml/>
- [2] J. A. López, “Clasificación de Movimiento,” pp. 1–44.
- [3] “EDGE IMPULSE.” <https://docs.edgeimpulse.com/docs>
- [4] J. A. López, “Introducción al TensorFlow Lite”.
- [5] F. Gulotta, “Node SerialPort.” <https://serialport.io/>
- [6] Automattic, “Sockert IO.” <https://socket.io/docs/v3/client-installation/>