# IMPLEMENTACIÓN DE UN PROTOTIPO PARA EL RECONOCIMIENTO DE GESTOS DE UNA MANO UTILIZANDO UNA RED NEURONAL ARTIFICIAL

## *IMPLEMENTATION OF A PROTOTYPE FOR THE RECOGNITION OF HAND GESTURES USING AN ARTIFICIAL NEURAL NETWORK*

***Luis Javier Ramírez Vidal***
Universidad Autónoma del Carmen, México
*161377@mail.unacar.mx*

***Rafael Alejandro Santamaría Gómez***
Universidad Autónoma del Carmen, México
*161333@mail.unacar.mx*

***José Luis Vázquez Ávila***
Universidad Autónoma del Carmen, México
*jvazquez@pampano.unacar.mx*

***Jorge Gabriel Pacheco Richard***
Universidad Autónoma del Carmen, México
*jpacheco@pampano.unacar.mx*

***Rafael Sánchez Lara***
Universidad Autónoma del Carmen, México
*rsanchez@pampano.unacar.mx*

***Manuel May Alarcón***
Universidad Autónoma del Carmen, México
*mmay@pampano.unacar.mx*

**Recepción:** 29/noviembre/2022 **Aceptación:** 13/marzo/2023

## Resumen

El objetivo de este trabajo es la implementación de un prototipo para el reconocimiento de gestos de una mano utilizando el entrenamiento de una red neuronal artificial secuencial sencilla. Los datos de entrada se obtienen en tiempo real de un sensor acelerómetro-giroscopio MPU-6050 colocado en un guante en dicha extremidad y procesado en una tarjeta Arduino Nano. Se utilizan herramientas como el lenguaje Python y del Arduino, así como de sus librerías tales como, Tensorflow, PySerial, entre otras. La red resultante consta de una sola capa oculta

de 4 neuronas, lo cual implica un costo computacional extremadamente bajo, que le permite ser programada en un microprocesador de bajo costo. En el entrenamiento de la red neuronal artificial se ingresaron 600 datos de entrada por clase, y se obtuvo una exactitud del 100%.

**Palabras Clave:** Arduino, inteligencia artificial, machine learning, Python, redes neuronales artificiales.

## *Abstract*

*The objective of this work is the implementation of a prototype for the recognition of hand gestures using the training of a simple sequential artificial neural network. The input data is obtained in real time from an MPU-6050 accelerometer-gyroscope sensor placed in a glove on the extremity and processed on an Arduino Nano board. Tools such as Python and Arduino languages are used, as well as their libraries such as Tensorflow, PySerial, among others. The resulting network consists of a single hidden layer of 4 neurons, which implies an extremely low computational cost, which allows it to be programmed in a low-cost microprocessor. In the training of the artificial neural network, 600 input data per class were entered, and an accuracy of 100% was obtained.*

***Keywords*****:** *Arduino, artificial intelligence, artificial neural networks, machine learning, Python.*

## 1. Introducción

Durante los últimos años se ha producido un gran avance en la investigación y el desarrollo de aplicaciones relacionadas con las Redes Neuronales Artificiales (RNA) para el control y óptimo aprendizaje de robots. La capacidad que tiene el cerebro humano de pensar, recordar y resolver problemas ha inspirado a muchos científicos a intentar modelar en las computadoras su funcionamiento.

Las Redes Neuronales Artificiales (RNA) están claramente inspiradas en las redes neuronales biológicas del cerebro humano. Estas neuronas artificiales se comportan de una forma muy parecida a las neuronas biológicas en sus habilidades más básicas, por lo que poseen una estructura similar.

La propiedad más importante de las redes neuronales artificiales es su capacidad de aprender a partir de un conjunto de patrones de entrenamientos, es decir, es capaz de encontrar un modelo que ajuste los datos, es por esto por lo que las etapas de una red neuronal artificial son entrenamiento, validación y predicción o inferencia [Salas, 2004], [Paluszek, 2020].

En la literatura existen diversos trabajos relacionados con el reconocimiento de patrones o clasificación de gestos efectuados con la mano. Podemos ver interesantes trabajos relacionados con las RNA, tales como: el reconocimiento de gestos en un espacio tridimensional [Naosekpam, 2009], control de prótesis ortopédicas [Arvetti, 2007], videojuegos [Nacke, 2011] y robots [Artemiadis, 2010]. Por otro lado, en [Nope, 2008] se estudian varias técnicas para el reconocimiento de gestos de la mano visuales mediante una cámara WEB, entre las técnicas utilizadas están las redes neuronales artificiales y los clasificadores bayesianos obteniendo resultados de precisión entre 87 y 97%. En [Ortega, 2017] se propone un sistema de reconocimiento de gestos mediante el uso de visión artificial y redes neuronales convolucionales implementado en MATLAB. En [Vargas, 2010] se presenta un sistema para el reconocimiento de lenguaje de señas, mediante el uso de visión artificial y redes neuronales artificiales logrando una precisión cercana al 99%. Los trabajos anteriormente mencionados hacen uso de técnicas de visión artificial, lo que implica el uso de cámaras e imágenes. El procesamiento de las imágenes requiere de cierto poder computacional por lo que es necesario el uso de una computadora. Por supuesto, esos trabajos están orientados a reconocimientos de los gestos como tal y posiblemente usan las herramientas que resultan eficientes para el trabajo a realizar. Por otro lado, existe varios trabajos relacionados con la implementación de sistemas de reconocimiento de gestos mediante el uso de señales electromiográficas extraídas de los músculos principalmente del brazo y técnicas de machine learning. En [Lee, 2021] y [Hristov, 2022] se presenta el estudio de detección de gestos por señales extraídas de los nervios que activan los dedos. En [Fatayerji, 2022] y [Ozdemir, 2022] se emplean redes neuronales artificiales y redes neuronales convolucionales para la clasificación de gestos a través de señales extraídas de músculos de los brazos. Estos últimos trabajos hacen uso de

sistemas de adquisición de datos de alto costo económico y los algoritmos resultantes requieren de un alto costo computacional. En este trabajo se presenta una propuesta para la generación y clasificación de gestos, económica, para su aplicación en diferentes ámbitos del control o manipulación de sistemas mecatrónicos. El sistema está basado en la adquisición de datos a través de un Arduino Nano y un acelerómetro. Los datos adquiridos son utilizados para entrenar una red neuronal, la cual posteriormente es implementada en un Arduino Nano para su uso.

## 2. Métodos

Para poder resolver la problemática se ha empleado un método de aprendizaje automático (machine learning), más específicamente usando una red neuronal artificial. El proceso se ha dividido en dos fases, la primera está constituida por la creación del dataset o base de datos que servirá como entrada para nutrir la red, estos datos son las señales recogidas por el sensor acelerómetro-giroscopio MPU-6050 colocado en la mano; la segunda fase comprende el entrenamiento de la RNA, aplicando los modelos de la librería Tensorflow sobre Python, donde se crea el modelo de la red multicapa de tipo secuencial.

### Red Neuronal Artificial (RNA)

La forma de representar un RNA es mediante el perceptrón multicapa, como el que se muestra en la figura 1. El elemento fundamental de una red neuronal es la neurona, también denominada “nodo”. La red neuronal artificial se compone por:

- Entradas (x): Es la información que recibe la red.

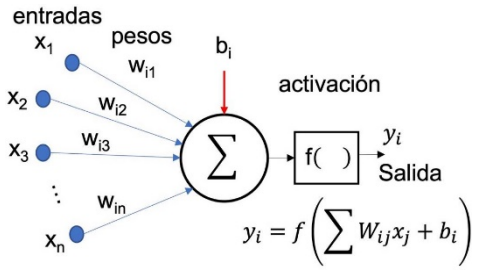


Figura 1 Estructura de una neurona artificial simple, también llamado Perceptrón.

- Pesos (w): son los valores numéricos que se encargan de establecer la influencia de una entrada en la salida deseada.
- Bias (b): Es parecido al peso solo que este está asociado a una entrada constante. Lo que hará al modelo más flexible.
- Función de activación o transferencia f(): Las funciones de activación se utilizan para introducir la no linealidad en las RNA [Nacelle, 2009], [Paluszek, 2020].

**Creación de la base de datos**

De acuerdo con lo mostrado en figura 1, las RNA necesitan conjuntos de datos de entrada para que con base en ellos puedan aprender, es por ello que lo primero que se realiza es la producción de una base de datos asociados al problema a resolver, en este caso los diferentes movimientos de una mano. Para ello se hace uso de lo siguiente:

- Arduino Nano: es el microcontrolador que alimentará al sensor y en el cual se procesarán las señales recibidas.
- Sensor MPU-6050: este dispositivo cuenta con todo lo necesario para medir movimientos en seis grados de libertad, combinando un giroscopio de tres ejes y un acelerómetro de tres ejes en un mismo chip.
- Python: es el lenguaje de programación donde se harán los diferentes programas para poder procesar todos los datos obtenidos y posteriormente el entrenamiento de la red neuronal artificial mediante el uso de TensorFlow.
- Numpy: es una librería de Python que proporciona un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y una variedad de rutinas para operaciones rápidas en matrices, que incluyen manipulación matemática, lógica, de formas, clasificación, selección, transformadas discretas de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

La figura 2 muestra la interconexión de la placa Arduino con el acelerómetro. Una vez conectado correctamente el sensor al microcontrolador, se carga un programa en dicha placa para poder leer las señales recibidas.

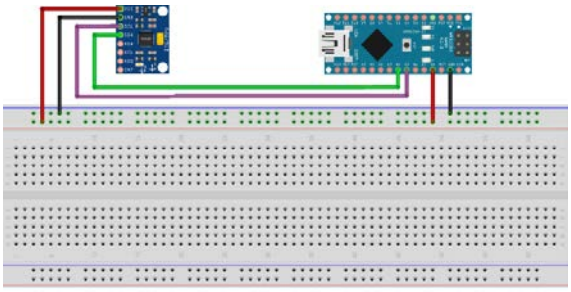

Figura 2 Diagrama de conexión entre la placa Arduino Nano y el sensor MPU-6050.

Para fines de este trabajo solamente se considera la información del acelerómetro, del cual se usan los tres ejes. El tiempo de muestreo se consideró de 100 milisegundos. El código en Arduino para esta etapa se muestra en la figura 3.

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

double accelX = 0, accelY =0, accelZ;

unsigned long lastTime = 0, sampleTime = 100;

void setup() {
  Serial.begin(9600);

  while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
  {
    delay(500);
  }
}

void loop() {

  if(millis()-lastTime>=sampleTime)
  {
    lastTime = millis();

    Vector normAccel = mpu.readNormalizeAccel();

    accelX = normAccel.XAxis;
    accelY = normAccel.YAxis;
    accelZ = normAccel.ZAxis;

    Serial.println(accelX);
    Serial.println(accelY);
    Serial.println(accelZ);
  }
}
```

Figura 3 Programa en Arduino para leer los datos del sensor MPU-6050.

La adquisición de los datos se puede llevar a cabo ya sea a través del uso de Matlab o Python. En este caso se usa Python y la librería pySerial para comunicar los datos de acelerómetro a través del Arduino, dichos datos son almacenados en vectores de numpy, los cuales serán usados posteriormente. La figura 4 muestra un pequeño script en Python para realizar esta tarea.

```
import serial
import time
import collections
import matplotlib.pyplot as plt
import matplotlib.animation as animation

import numpy as np

maxSample = 100
sampleTime = 100 # Tiempo de muestreo 100ms
sizeData = 3
legendLabel = ['ax', 'ay', 'az']
xLimit = [(0, maxSample)]*sizeData
yLimit = [(-20, 20), (-20, 20), (-20, 20)]
style = ['r-', 'g-', 'b-']
anim = []


port = '/dev/cu.usbserial-1420'
baudRate = 9600
arduino = realTimePlot(port, baudRate, maxSample, sizeData )
```

Figura 4 Script para la comunicación con Arduino y el almacenamiento de los datos.

Para este trabajo se desea reconocer cinco tipos de gestos con la mano, los cuales son: estático o neutral, adelante, atrás, izquierda y derecha. Los datos de información de los gestos se adquieren por separado, por cada gesto. El tiempo de recolección de señales de cada movimiento fue de 60 segundos, con un tiempo de muestreo de 100 milisegundos, dando como resultado una base de datos con aproximadamente 600 muestras por clase. En la figura 5 se pueden observar claramente los cinco movimientos diferentes.

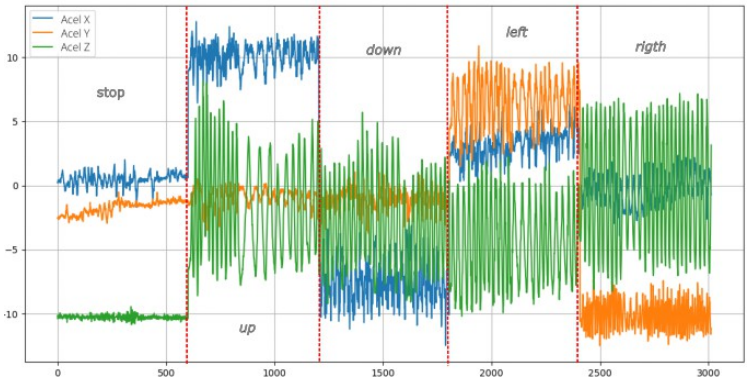


Figura 5 Gráfica de las cinco clases de movimientos diferentes de la mano a reconocer.

**Procesamiento de la base de datos**

Previo a la etapa de entrenamiento se realiza un preprocesamiento de los datos, cuyo objetivo principal es facilitar la capacitación de la red. El preprocesamiento de datos consta de diferentes métodos, tales como [Paluszek, 2000]:

- Normalización.
- Transformaciones no lineales.
- Extracción de características.
- Codificación de entradas y salidas.
- Manejo de datos faltantes.

En este caso se utilizará la normalización, ya que se recomienda normalizar los datos que usan escalas e intervalos diferentes. El modelo con datos sin normalizar podría adaptarse automáticamente a datos tan heterogéneos, pero pueden ralentizar o dificultar el aprendizaje. Para este caso se usará la normalización normal estándar. La normalización normal estándar requiere de la media y la desviación estándar del conjunto de datos por clase. El resultado de la normalización produce otro conjunto de datos con media 0 y desviación estándar 1 [Paluszek, 2000].

**Codificación**

Cuando se realiza una clasificación multiclase en RNA, es necesario representar los datos objetivos a través de una codificación que pueda representar a cada clase. En este caso se utilizará la codificación One Hot Encoding. Se llama One-Hot porque sólo un bit es “hot” o 1. La idea es asignar 0 a todas las clases, excepto para la clase a la que, sí pertenecen los datos, ya que en este caso se le asigna 1. La representación de one hot encoding para este trabajo se muestra en la tabla 1.

Tabla 1 Codificación One-Hot.

| Clase | neutral | adelante | atrás | izquierda | derecha |
|---|---|---|---|---|---|
| **neutral** | 1 | 0 | 0 | 0 | 0 |
| **adelante** | 0 | 1 | 0 | 0 | 0 |
| **atrás** | 0 | 0 | 1 | 0 | 0 |
| **Izquierda** | 0 | 0 | 0 | 1 | 0 |
| **derecha** | 0 | 0 | 0 | 0 | 1 |

**División del conjunto de datos**

Un modelo de aprendizaje automático tiene como objetivo realizar buenas predicciones sobre datos nunca vistos. Una forma de hacer esto es dividir el conjunto de datos en dos subconjuntos:

- Conjunto de entrenamiento: un subconjunto para entrenar un modelo.
- Conjunto de prueba: un subconjunto para probar el modelo.
- Por lo general se dividen en 80-20 y se toman muestras aleatorias, lo cual es justo lo que se hizo.

**Creación del modelo de Red Neuronal Artificial**

Las redes neuronales artificiales se pueden definir en TensorFlow y su librería Keras como una secuencia de capas [Campesato, 2019]. El contenedor de estas capas es la clase Sequential. Como primer paso se creó una instancia de la clase Sequential, luego se agregaron la capa de entrada, las ocultas y la de salida. La primera capa de la red debe definir el número de entradas que se esperan, en este caso son tres, que corresponden a los tres ejes del acelerómetro (x, y, z). La red que se define en este trabajo consta de una simple capa oculta con cuatro neuronas y está definida con la función de activación “relu”. La capa de salida está compuesta de cinco neuronas que definen las 5 clases de movimientos a reconocer, dicha capa tiene la función de activación “softmax”, ésta es la utilizada para problemas multiclase. Se puede apreciar el modelado de la RNA definida en TensorFlow/Python en la figura 6, donde hiddenNodes = 4.

**Compilación del modelo de Red Neuronal Artificial**

En la compilación se especificó el algoritmo de optimización que se utilizará para entrenar la red, el cual es Adam (por sus siglas en inglés, Adaptive Moment estimation); y la función de pérdida utilizada es la “entropía cruzada categórica”, ya que es la función de pérdida predeterminada para problemas de clasificación de varias clases [Campesato, 2019], figura 6. Dicha función requiere que la capa de salida esté configurada con *n* neuronas (una para cada clase), y una función de activación “softmax” para predecir la probabilidad de cada clase.

```
#Dividir el conjuntos de datos en conjuntos de entrenamiento y prueba.

from sklearn.model_selection import train_test_split

P_train, P_test, T_train, T_test = train_test_split(P,one_hot_labels, test_size=0.20, random_state=42)

#Establecer hiperparámetros de algoritmo (tasa de aprendizaje, épocas).
epochs=1000
hiddenNodes = 4

#Definir la arquitectura de la red neuronal.
model = Sequential()
model.add(Dense(hiddenNodes, activation='relu', input_dim=sizeInput)) # puede ser tanh

model.add(Dense(5, activation='softmax')) # capa de salida
model.summary()

#Declara las funcion de pérdida.
loss='categorical_crossentropy'

#Optimizador.
optimizer = tf.keras.optimizers.Adam()

model.compile(loss=loss,
              optimizer=optimizer,
              metrics=['accuracy'])
```

Figura 6 Estructura de la RNA.

### Entrenamiento del modelo de la Red Neuronal Artificial

Una vez compilado el modelo de la red, ésta se entrena sólo ajustando el número de épocas, ya que al utilizar el optimizador “Adam”, ésta combina las bondades de otros optimizadores como AdaGrad y RMSProp [Páez, 2020].

Los pesos se actualizan después de cada época, el entrenamiento se ejecutó en 300 épocas y con sólo una capa de entrada u oculta con 4 neuronas. Estas configuraciones se eligieron mediante ensayo y error. La evaluación del modelo se ejecuta mediante dos métricas, la primera es el valor de pérdida y la segunda será la exactitud de la red.

## 3. Resultados

### Adquisición de las señales y entrenamiento de la red

La figura 7 muestra varias etapas de la implementación del sistema. La adquisición de los datos mediante el uso de Python y Arduino. La figura 7 también muestra algunas posiciones de la mano como la posición estática y hacia atrás.

Se desarrolló un modelo de Red Neuronal Artificial de tipo secuencial basado en el modelo Keras de Python, este modelo se ejecuta en la librería TensorFlow, la cual cuenta con la arquitectura adecuada para el buen manejo de redes neuronales y machine learning, así como también garantiza un comportamiento fiable cuando se utilizan conjuntos de datos grandes. En figura 8 se muestra un resumen del modelo de la red desarrollada.

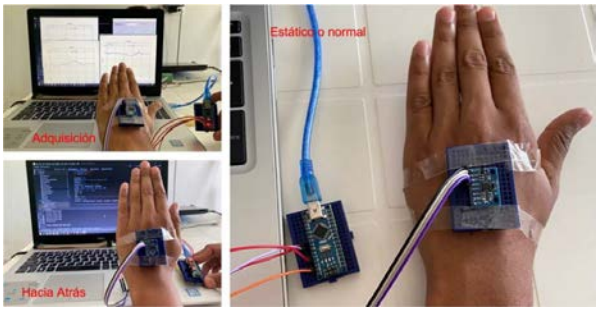


Figura 7 Implementación física.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4)                 16

 dense_1 (Dense)             (None, 5)                 25

=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
```

Figura 8 Resumen de la estructura de RNA creada.

En la figura 9 se puede visualizar una característica muy interesante de este modelo: a medida que aumenta el número de iteraciones (épocas), el error porcentual disminuye, éste indica que el modelo de la red evoluciona de buena manera; a partir de la época 30 la exactitud tiende a 1, por lo que, tanto el optimizador como la normalización de los datos y la estructura del modelo de la RNA son acertados para poder resolver la problemática planteada.

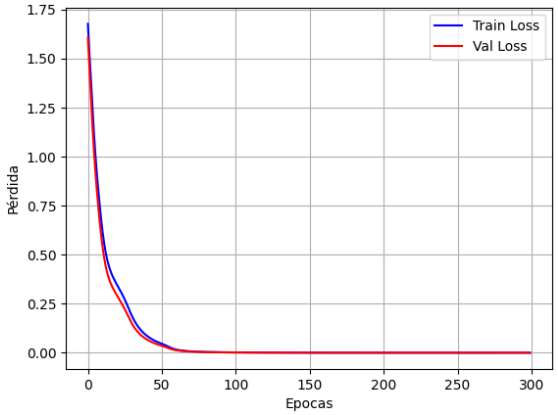


Figura 9 Tendencia de los valores de pérdida.

La figura 10 muestra el resultado de la corrida con una precisión de 100% tanto en el entrenamiento como en la validación. Este resultado nos indica que la red es capaz de distinguir completamente entre los 5 diferentes gestos.

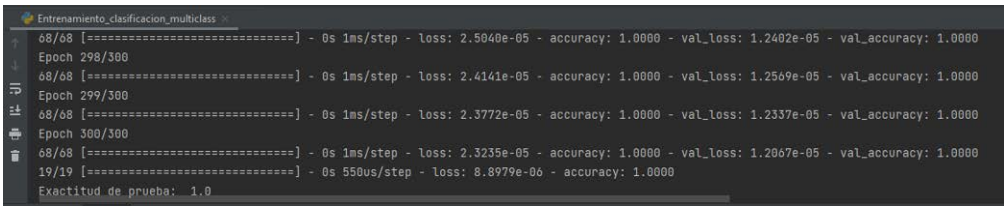


Figura 10 Resultado de la precisión de la clasificación del modelo.

## Implementación del sistema físico

Una vez entrenada la RNA se está en posición de implementarla dentro de la tarjeta Arduino, para llevar a cabo esta tarea se requirió del conocimiento de los pesos y los bias. Los resultados obtenidos de los pesos y los bias correspondientes a las conexiones de la capa oculta y las entradas, capa oculta y capa de salida, respectivamente, se muestran en las tablas 2, 3 y 4.

Tabla 2 Pesos de las conexiones de la capa oculta y las entradas.

| $W1_{ij}$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | -0.983 | -0.462 | -2.617 |
| 1 | -3.773 | -3.433 | 0.009 |
| 2 | 3.021 | -3.476 | -0.233 |
| 3 | 2.779 | 3.458 | -1.222 |

Tabla 3 Pesos de las conexiones de la capa oculta y la capa de salida.

| $W2_{ij}$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 2.223 | -0.989 | 0.844 | -0.469 |
| 1 | -2.066 | -3.993 | 2.948 | 0.677 |
| 2 | 0.486 | 2.331 | -3.97 | -4.38 |
| 3 | 0.004 | -4.315 | -2.713 | 2.469 |
| 4 | -1.71 | 0.644 | 2.485 | -3.942 |

Tabla 4 Bias correspondientes a las neuronas de la capa oculta y capa de salida.

| bi | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| b1i | 0.201 | 1.94 | 2.337 | 1.738 | N/A |
| b2i | -0.341 | -0.688 | 0.807 | 0.066 | -0.2 |

Los pesos y bias, resultados del entrenamiento de la RNA, se utilizan para la implementación de la red neuronal en Arduino. La figura 11 muestra un extracto del código para la implementación de la RNA en Arduino. Como se puede observar en la figura 11, los datos son adquiridos a través del acelerómetro y separados como acelX, acelY y acelZ, posteriormente son normalizados con el uso de la media y desviación estándar que se obtuvo de los datos del entrenamiento, estos serán las entradas para la red ya entrenada (a0).

```
// Lecturas del acelerómetro
Vector normAccel = mpu.readNormalizeAccel();
accelX = normAccel.XAxis;
accelY = normAccel.YAxis;
accelZ = normAccel.ZAxis;
// Normalización de los datos del acelerómetro
a0[0] = dataNormalized(accelX,mean[0],dstd[0]);
a0[1] = dataNormalized(accelY,mean[1],dstd[1]);
a0[2] = dataNormalized(accelZ,mean[2],dstd[2]);

////////////////////////////// Estructura Red Neuronal ////////////////////////////////
// Interconexión capa de entrada con capa (a0) oculta (a1). Función de activación ReLu
for(int i = 0 ; i<4; i++ ) {aux=0.0;for(int j = 0 ; j <3 ; j++ ) { aux=aux+W1[i][j]*a0[j];} a1[i]=relu(aux+b1[i]);}
double aux1 = 0;
// Interconexión capa oculta (a1) con capa de salida (a2). Función de activación SoftMax
for(int i = 0 ; i<3; i++ ) {aux=0.0;for(int j = 0 ; j <4 ; j++ ){ aux=aux+W2[i][j]*a1[j];} a2[i]=(aux+b2[i]);aux1=aux1+exp(a2[i]);}
double minimo = 0.0;
int classes = 0;
// Determinación de la Clasificación
for(int i = 0;  i<3; i++){a2[i] = exp(a2[i])/aux1;if(a2[i]>minimo){minimo=a2[i];classes=i;}} //
////////////////////////////////////////////////////////////////////////////////////
// Clasificación
switch (classes){
  case 0:
    Serial.println(stop);
    break;
  case 1:
    Serial.println(up);
    break;
  case 2:
    Serial.println(down);
    break;
  case 3:
    Serial.println(left);
    break;
  case 4:
    Serial.println(rigth);
    break;
}
```

Figura 11 Porción de código para la implementación de la RNA en Arduino.

Posteriormente hay una porción de código que muestra la manera en la que se implementa la red neuronal, donde se tienen tres ciclos *for* que realizan diferentes funciones. El primer ciclo *for* hace las operaciones relacionadas con el algoritmo de backpropagation para la interconexión de la capa de entrada con la capa oculta, las tres entradas se relacionan con las 4 neuronas de la capa oculta (para mayor visualización ver figura 1). En esta sección de la red aparecen las entradas como un vector *a0*, los pesos de las conexiones entre las entradas y la capa oculta $W1_{ij}$, correspondientes a los valores de la tabla 2, la función de activación de la capa

oculta es ReLu. El siguiente ciclo *for* hace las operaciones relacionadas con el algoritmo de backpropagation para la interconexión de la capa de oculta y con la capa salida, las 4 neuronas de la capa oculta se relacionan con las 5 neuronas de la capa de salida (para mayor visualización ver figura 1).

Los pesos de las conexiones de la capa oculta y la capa de salida $W2_{ij}$, correspondientes a la tabla 3, las activaciones de la capa oculta, *a1*, las activaciones de la capa de salida *a2* y sus respectivos bias, se encuentran relacionados en esta sección de la red. Por último, el tercer ciclo *for* realiza la determinación de la clase resultante de acuerdo con la entrada. Considerando que la función de activación de la capa de salida corresponde a *SoftMax*, se determina la salida, o neurona de salida, con la mayor probabilidad (valor entre 0 y 1) y ésta será considerada como la salida detectada. Adicionalmente, se introdujo una condicional *switch* para visualizar el tipo de dato detectado a través de la interfaz serial del IDE de Arduino, tal como se muestra en la figura 12.

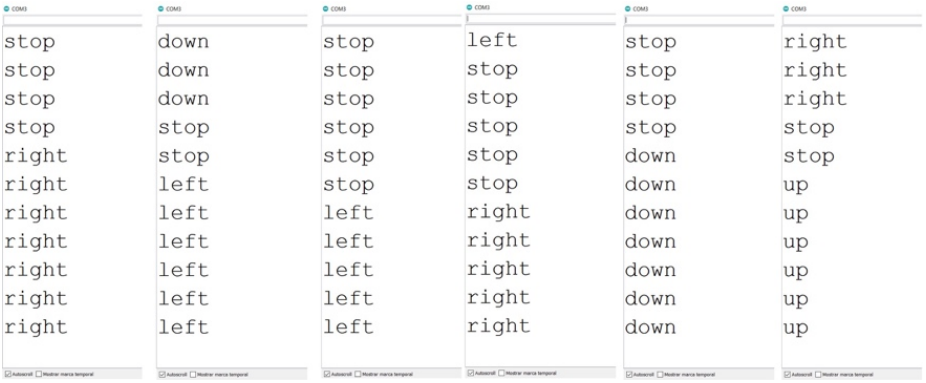


Figura 12 Resultados de las clasificaciones.

La figura 12 muestra los resultados de salida en la interfaz serial del IDE de Arduino al realizar varios movimientos de la mano. Las pruebas se realizaron usando el acelerómetro como el de la figura 7. Cada movimiento o gesto de la mano tiene un código que servirá para distinguir las señales detectadas, cuyos gestos se muestran en la figura 13. Los gestos son codificados de la siguiente manera: 0 o *stop* corresponde a mantener la mano estática; 1 o *up* corresponde a tener la mano hacia arriba; 2 o *down* corresponde a tener la mano hacia abajo; 3 o *left* corresponde a

girar la mano hacia la izquierda; 4 o *rigth* corresponde a girar la mano hacia la derecha.


Figura 13 Representación de los gestos de la mano.

## 4. Discusión

Las redes neuronales artificiales surgen como una herramienta de Machine Learning y sus aplicaciones se extienden en diferentes ámbitos de la tecnología. Una métrica muy importante de una RNA es su precisión. Para la aplicación del reconocimiento de gestos de la mano se puede observar que a medida que van aumentando las épocas en el entrenamiento, el valor de pérdida va disminuyendo rápidamente, llegando a un punto (época 80 aproximadamente) en la que éste ya no disminuye significativamente, como se puede ver en figura 9. Esto también implica que la precisión aumente hasta 1 (100%), lo que indica una distinción total entre las diferentes clases (gestos).

En las primeras 60 iteraciones el error decrece drásticamente, esto se debe al buen procesamiento de la información. El uso de la normalización de la base de datos y la aplicación de un optimizador como el optimizador Adam mejora significativamente la precisión.

## 5. Conclusiones

El desarrollo de prototipos para la clasificación de gestos ha tomado mucho auge en la actualidad. Las aplicaciones van desde el reconocimiento de patrones en los movimientos, detección de lenguajes para sordomudos, hasta el control remoto de robots. La RNA presentada en este trabajo resultó ser de muy bajo costo computacional. La red utilizada es relativamente simple y consta de tan sólo una

capa oculta con 4 neuronas y una capa de salida con 5 neuronas (una por cada clase). Con esta infraestructura, la red convergió rápidamente y en menos de 300 épocas (realizaciones) la red alcanzó un error de 0 y por tanto una precisión del 100%. Una RNA con tan pocos recursos computacionales como la presentada aquí puede ser embebida en cualquier procesador de baja capacidad, en este caso se utilizó un Arduino Nano. La salida del sistema puede ser usada en varias aplicaciones, pensando en un trabajo a futuro en el control de un robot móvil.

## 6. Bibliografía y Referencias

[1] Artemiadis, P. K., & Kyriakopoulos, K. J. (2010). An EMG-based robot control scheme robust to time-varying EMG signal features. IEEE Transactions on Information Technology in Biomedicine, 14(3), 582-588.

[2] Arvetti, M., Gini, G., & Folgheraiter, M. (2007, June). Classification of EMG signals through wavelet analysis and neural networks for controlling an active hand prosthesis. In 2007 IEEE 10th international conference on Rehabilitation Robotics, pp. 531-536. IEEE.

[3] Campesato, O. (2019). TensorFlow 2 pocket primer. Mercury Learning and Information.

[4] Fatayerji, H., Al Talib, R., Alqurashi, A., & Qaisar, S. M. (2022, March). sEMG Signal Features Extraction and Machine Learning Based Gesture Recognition for Prosthesis Hand. In 2022 Fifth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU), pp. 166-171. IEEE.

[5] Hristov, B., Nadzinski, G., Latkoska, V. O., & Zlatinov, S. (2022, June). Classification of Individual and Combined Finger Flexions Using Machine Learning Approaches. In 2022 IEEE 17th International Conference on Control & Automation (ICCA), pp. 986-991. IEEE.

[6] Lee, K. H., Min, J. Y., & Byun, S. (2021). Electromyogram-based classification of hand and finger gestures using artificial neural networks. Sensors, 22(1), 225.

[7] Nacelle, A., & Mizraji, E. (2009). Redes neuronales artificiales. Núcleo de ingeniería biomédica–Universidad de la Republica Uruguay.

[8] Nacke, L. E., Kalyn, M., Lough, C., & Mandryk, R. L. (2011, May). Biofeedback game design: using direct and indirect physiological control to enhance game interaction. In Proceedings of the SIGCHI conference on human factors in computing systems, pp. 103-112.

[9] Naosekpam, V., & Sharma, R. K. (2019). Machine learning in 3D space gesture recognition. Jurnal Kejuruteraan, 31(2), 243-248.

[10] Nope, S. E., Loaiza, H., & Caicedo, E. (2008). Estudio Comparativo de Técnicas para el Reconocimiento de gestos por Visión Artificial. Avances en Sistemas e Informática, 5(3), 127-134.

[11] Ortega Asensio, E. (2017). Sistema de reconocimiento de gestos de la mano basado en procesamiento de imagen y redes neuronales convolucionales.

[12] Ozdemir, M. A., Kisa, D. H., Guren, O., & Akan, A. (2022). Hand gesture classification using time–frequency images and transfer learning based on CNN. Biomedical Signal Processing and Control, 77, 103787.

[13] Paluszek, M., & Thomas, S. (2020). Practical Matlab deep learning. A Project-Based Approach, Michael Paluszek and Stephanie Thomas.

[14] Páez, R. F. F., & Medina, A. D. R. D. (2020). Desarrollo, entrenamiento y evaluación de modelos de machine learning para clasificación de imágenes. Revista Científica Estudios e Investigaciones, 9, 175-176.

[15] Salas, R. (2004). Redes neuronales artificiales. Universidad de Valparaıso. Departamento de Computación, 1, 1-7.

[16] Vargas, L. P., Barba Jiménez, L., & Mattos, L. (2010). Sistema de identificación de lenguaje de señas usando redes neuronales artificiales. Revista Colombiana de Física, 42(2), 5.