Universidad Autónoma de San Luis Potosí
Facultad de ingeniería
Inteligencia Artificial Aplicada
Práctica: 6
Nombre Práctica: Implementación de una Red Neuronal en Sistemas Embebidos
Nombre del Alumno: Manuel Ramírez Galván

Fecha: 02/04/2025

Procedimiento

6.1.- Sigue las instrucciones del archivo “practica_6_training” para desarrollar y entrenar un modelo de red neuronal de clasificación.

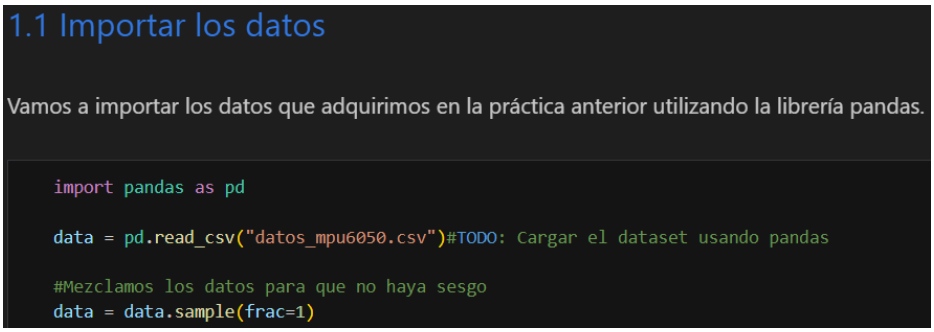

Imagen 1.- Código para Importar Datos

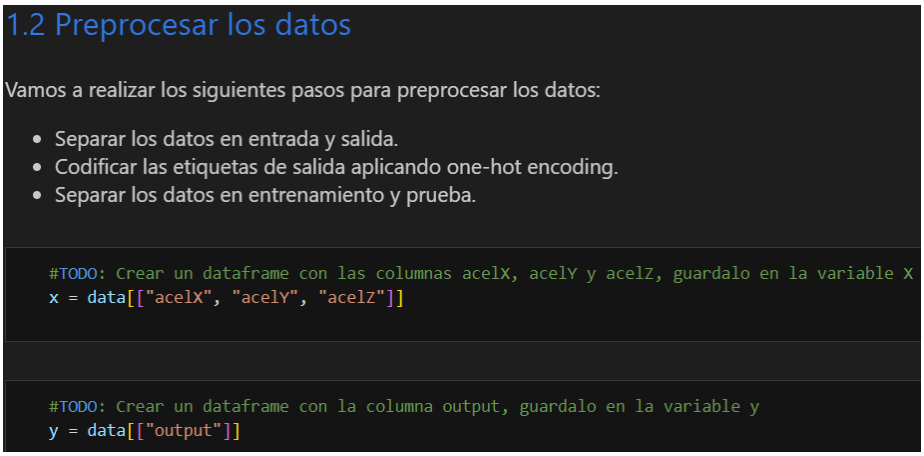

Imagen 2.- Separar Datos de Entrada y Salida

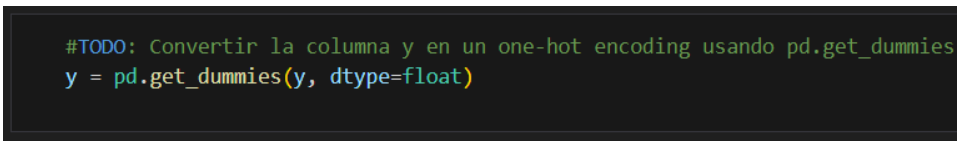

Imagen 3.- Codificar etiquetas de salida

```python
#TODO: Dividir los datos en entrenamiento y prueba usando X.sample y y.sample
# Guarda los datos de entrenamiento en las variables X_train y y_train
# Guarda los datos de prueba en las variables X_test y y_test

X_train = x.sample(frac=0.8, random_state=0)
y_train = y.sample(frac=0.8, random_state=0)

X_test = x.drop(X_train.index)
y_test = y.drop(y_train.index)
```

Imagen 4.- Separar datos de Entrenamiento y Prueba

```python
mean = X_train.mean(axis = 0)
std = X_train.std(axis=0)

print(mean)
print(std)
```

```
[-1.54082997e-17  2.86154138e-17 -1.10059284e-16]
[0.99969016 0.99969016 0.99969016]
```

Imagen 5.- Media y Desviación Estándar de los Datos

```python
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
```

Imagen 6.- Normalización de los Datos

```python
#TODO: Convertir los datos de entrenamiento y prueba
X_train = X_train.values
y_train = y_train.values

X_test = X_test.values
y_test = y_test.values
```

Imagen 7.- Convertir a un Arreglo de Numpy

```python
from tensorflow.keras.models import Sequential    Import "tensorflow.keras.mod
from tensorflow.keras.layers import Dense, Input    Import "tensorflow.keras.l

#TODO: Crear un modelo secuencial
model = Sequential()

#TODO: Agregar una capa de entrada con 3 neuronas, una para cada columna de X
model.add(Input(shape=(3,)))

#TODO: Agregar una capa densa con 8 neuronas y activación relu
model.add(Dense(8, activation='relu'))

#TODO: Agregar una capa densa con 5 neuronas y activación softmax, 5 neuronas
model.add(Dense(5, activation='softmax'))

model.summary()
```

Imagen 8.- Crear el Modelo

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

Imagen 9.- Compilar el Modelo

```
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test))
```

```
Epoch 1/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.3531 - loss: 1.4507 - val_accuracy: 0.5173 - val_loss: 1.2587
Epoch 2/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.6090 - loss: 1.2298 - val_accuracy: 0.8416 - val_loss: 1.0601
Epoch 3/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8820 - loss: 1.0298 - val_accuracy: 0.8936 - val_loss: 0.8789
Epoch 4/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9275 - loss: 0.8507 - val_accuracy: 0.9455 - val_loss: 0.7145
Epoch 5/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9637 - loss: 0.6705 - val_accuracy: 0.9728 - val_loss: 0.5689
Epoch 6/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9801 - loss: 0.5482 - val_accuracy: 0.9901 - val_loss: 0.4471
Epoch 7/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9908 - loss: 0.4264 - val_accuracy: 0.9926 - val_loss: 0.3486
Epoch 8/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9980 - loss: 0.3267 - val_accuracy: 0.9950 - val_loss: 0.2718
Epoch 9/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9975 - loss: 0.2648 - val_accuracy: 1.0000 - val_loss: 0.2137
Epoch 10/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.2029 - val_accuracy: 1.0000 - val_loss: 0.1706
Epoch 11/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.1637 - val_accuracy: 1.0000 - val_loss: 0.1380
Epoch 12/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 0.1316 - val_accuracy: 1.0000 - val_loss: 0.1132
Epoch 13/20
...
Epoch 19/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0432 - val_accuracy: 1.0000 - val_loss: 0.0406
Epoch 20/20
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0381 - val_accuracy: 1.0000 - val_loss: 0.0364
```

Imagen 10.- Entrenar el Modelo

```
from tensorflow import lite as tflite

model_name = "mpu6050_model" #Nombre del archivo donde se guardará el modelo

converter = tflite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert() #Convertimos el modelo a un modelo tflite

with open(f"{model_name}.tflite", 'wb') as f: #Abrimos un archivo en modo escritura binaria
    f.write(tflite_model) #Guardamos el modelo en un archivo llamado model.tflite
```

Imagen 11.- Convertir el Modelo

```python
def tflite_to_array(model_data, model_name):
    c_str = ""

    #Creamos las cabeceras del archivo
    c_str += f"#ifndef {model_name.upper()}_H\n"
    c_str += f"#define {model_name.upper()}_H\n\n"

    #Agregamos una variable con el tamaño del modelo
    c_str += f"const unsigned int {model_name}_len = {len(model_data)};\n\n"

    #Agregamos el modelo como un arreglo de bytes
    c_str += f"const unsigned char {model_name}[] = {{\n"

    for i, byte in enumerate(model_data):
        c_str += f"0x{byte:02X},"
        if (i + 1) % 12 == 0:
            c_str += "\n"

    c_str += "};\n\n"

    #Cerramos las cabeceras del archivo
    c_str += f"#endif // {model_name.upper()}_H\n"

    return c_str



model_array = tflite_to_array(tflite_model, model_name)

with open(f"{model_name}.h", 'w') as f:
    f.write(model_array)
```

Imagen 12.- Crear matriz de bytes

6.2.- Sigue las instrucciones del archivo “practica_6_inferencia” para implementar una red neuronal en un microcontrolador.

➔ Instalación de la librería de TensorFlow Lite para Arduino

Para instalar la librería de TensorFlow Lite para Arduino, se debe seguir los siguientes pasos:

1. Sigue las instrucciones de la documentación oficial para descargar el repositorio TensorFlow Lite para Arduino o solicite el repositorio a su profesor.
2. Descomprime el archivo descargado y copia la carpeta Arduino_TensorFlowLite en la carpeta libraries de tu instalación de Arduino.
3. Abre el IDE de Arduino y verifica que la librería de TensorFlow Lite para Arduino se haya instaladocorrectamente. Para ello, ve a File > Examples y verifica que aparezca la carpeta Arduino_TensorFlowLite.
4. Como paso adicional, crea un nuevo proyecto en el IDE de Arduino y copia dentro de la carpeta del proyecto el archivo mpu6050_model.h generado en la primer parte de la práctica.

➔ Programar el ESP32 para realizar la inferencia

Utilizar la librería de TensorFlow Lite para Arduino puede ser un poco complicado, en las siguientes secciones se explicará a grandes rasgos el código necesario para realizar la inferencia de la red neuronal en el ESP32.

- Incluir las librerías necesarias

Para realizar la inferencia de la red neuronal en el ESP32, se deben incluir las siguientes librerías:

micro_mutable_op_resolver.h proporciona las operaciones utilizadas por el intérprete para ejecutar
el modelo.

micro_interpreter.h contiene el código para cargar y ejecutar modelos.

schema_generated.h contiene el esquema para el formato de archivo TensorFlow Lite FlatBuffer.

system_setup.h contiene la configuración del sistema.

- Definir constantes simbólicas

Se deben definir las constantes necesarias para la ejecución del modelo de la red neuronal:

Los valores de **ACCEL_X_MEAN, ACCEL_X_STD, ACCEL_Y_MEAN, ACCEL_Y_STD, ACCEL_Z_MEAN** y **ACCEL_Z_STD** se obtienen de la normalización de los datos de entrada de la red neuronal en la primer parte de la práctica.

- Funciones auxiliares

Vamos a definir una función auxiliar para normalizar los datos del acelerómetro y cargarlos en un tensor de entrada para la red neuronal

También vamos a definir una función auxiliar para saber la longitud de un tensor

Por último, vamos a definir una función auxiliar para obtener la clase predicha por la red neuronal

- Configurar variables globales

Vamos a definir las variables globales necesarias para la ejecución del modelo de la red neuronal

- Función setup

En la función setup vamos a inicializar el sensor MPU6050 y cargar el modelo de la red neuronal

- Función loop

En la función loop vamos a realizar la inferencia de la red neuronal

Resultados

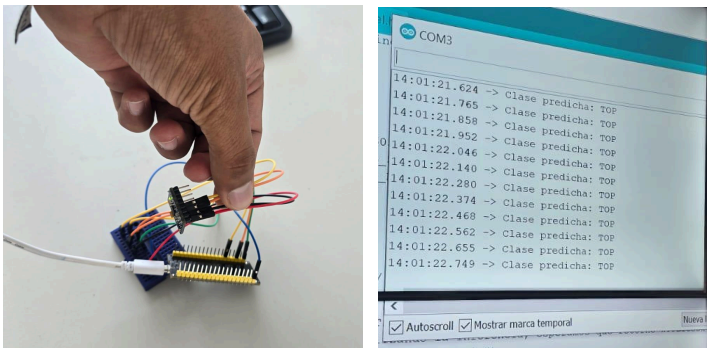

Imagen 13.- Identificar "TOP"

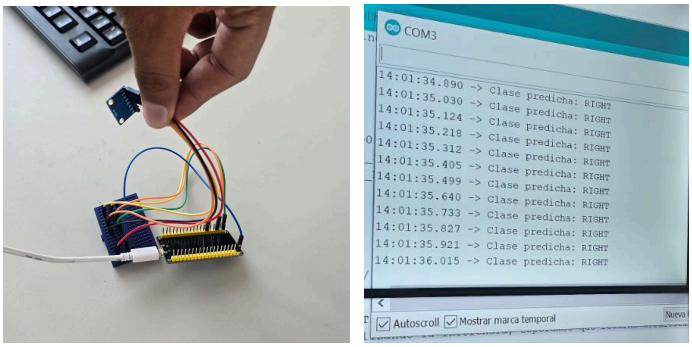

Imagen 14.- Identificar "RIGHT"

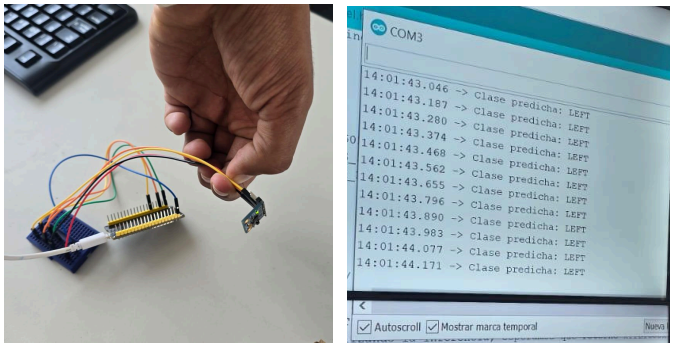

Imagen 15.- Identificar "LEFT"

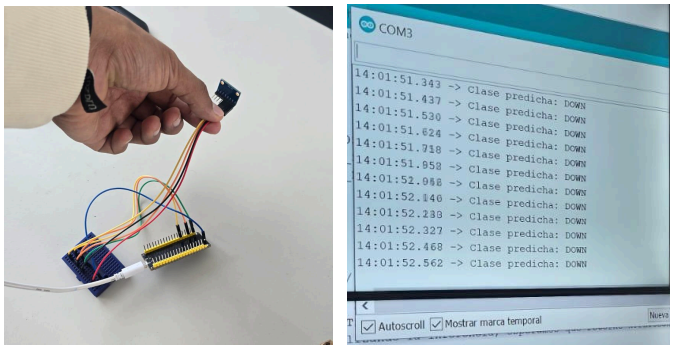

Imagen 16.- Identificar "DOWN"

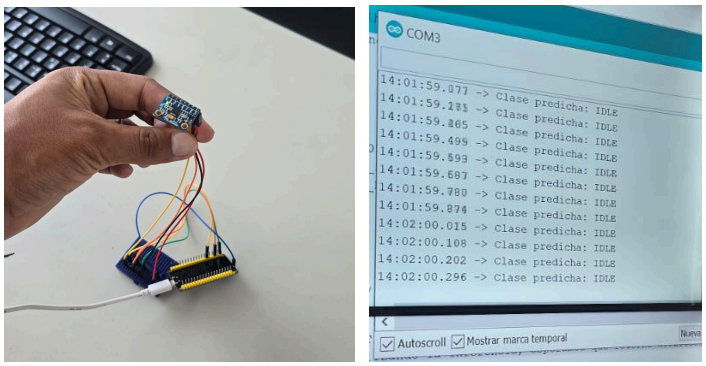

Imagen 17.- Identificar "IDLE"

Comprensión

1. **¿Qué pasos se deben seguir para entrenar una red neuronal con Keras?**

- Preparar y cargar los datos
- Preprocesar los datos
- Diseñar el modelo
- Compilar el modelo
- Entrenar el modelo
- Evaluar el modelo
- Hacer predicciones (opcional)
- Guardar el modelo

2. **¿Cuál es la función del conjunto de entrenamiento y del conjunto de prueba en el proceso de entrenamiento?**

Su función principal es reducir la dimensión espacial de los mapas de características (feature maps), manteniendo la información más importante.

3. **¿Qué se entiende por función de pérdida, optimizador y métricas en el contexto del entrenamiento de una red neuronal?**

Conjunto de entrenamiento (Training set): Es el conjunto de datos con el que el modelo aprende. Se usa durante el proceso de entrenamiento para ajustar los pesos internos de la red neuronal mediante retropropagación.
Conjunto de prueba (Test set): Se utiliza después del entrenamiento para evaluar el rendimiento real del modelo sobre datos nunca antes vistos. Permite comprobar qué tan bien generaliza el modelo.

4. **¿Qué tipo de problemas se pueden resolver utilizando una red neuronal entrenada con Keras?**

Al ser una API de alto nivel para construir y entrenar redes neuronales, permite resolver una amplia variedad de problemas de aprendizaje automático. Todo depende del tipo de datos y del enfoque del modelo. Por ejemplo:

- Clasificación de imágenes
- Detección y segmentación de objetos
- Reconocimiento de voz y procesamiento de audio
- Procesamiento de texto y lenguaje natural
- Series de tiempo y predicciones
- Regresión (predicción de valores continuos)
- Detección de anomalías

5. **¿Qué es IA on the Edge?**

Es la implementación de modelos de inteligencia artificial directamente en dispositivos locales, como cámaras, sensores, teléfonos móviles, drones, microcontroladores o placas como Raspberry Pi, ESP32, Jetson Nano, etc., sin necesidad de enviar los datos a la nube para su procesamiento.

Ejemplos:

- Un celular que detecta rostros o reconoce voz sin conexión.
- Una cámara que identifica matrículas o personas en tiempo real.
- Un auto que reconoce señales de tránsito sin conectarse a servidores.
- Un dron que clasifica terrenos o cultivos mientras vuela.

6. **¿Qué ventajas tiene IA on the Edge en los sistemas embebidos?**

- **Menor latencia:** Las decisiones se toman en tiempo real, sin esperar a la nube. Ideal para aplicaciones críticas como vehículos autónomos o detección en tiempo real.
- **Privacidad de los datos:** Los datos no se envían a internet, lo que mejora la seguridad y privacidad.
- **Ahorro de ancho de banda:** Se reduce el tráfico de red, ya que no es necesario enviar todos los datos.
- **Funciona sin conexión:** Puede operar incluso si no hay acceso a internet.

7. ¿Qué es Tensorflow Lite?

TFLite es una versión ligera y optimizada de TensorFlow, diseñada específicamente para ejecutar modelos de inteligencia artificial en dispositivos con recursos limitados. Sirve para llevar modelos de aprendizaje automático al “borde” (Edge), directamente en dispositivos móviles o embebidos, permitiendo crear soluciones inteligentes rápidas, autónomas y eficientes.

Conclusiones

La implementación de una red neuronal en un sistema embebido, como el ESP32, permite a la inteligencia artificial interactuar con el mundo, ejecutando modelos ligeros de manera autónoma, rápida y sin conexión a internet.

Un ejemplo práctico de esta integración es el uso de un acelerómetro conectado al ESP32 para reconocer las posiciones “UP, DOWN, LEFT, RIGHT, IDLE” en tiempo real.

Utilizando técnicas de entrenamiento de redes neuronales en TensorFlow, y luego convirtiendo el modelo a TensorFlow Lite, es posible ejecutar este modelo directamente en el ESP32 para clasificar los movimientos según los datos del acelerómetro.