Tarea Sistemas Conexionistas Clasificador de fallas en operación de robots

Mauricio García Aguilar, Lorenzo Sancho Fallas
maurici oga117@estudiantec.cr, lorenzo7sancho@estudiantec.cr
Área Académica de Ingeniería Mecatrónica
Tecnológico de Costa Rica

Resumen

En la presente memoria escrita se detalla como se realizó la primera tarea del curso de Inteligencia Artificial. Dicha tarea consta de un clasificador de fallas de un robot, el cual puede presentar distintos comportamientos en diversos experimentos, partir del paradigma de sistemas conexionistas de la Inteligencia Artificial. Se detallará cuál fue el proceso para seleccionar las clases, las pruebas realizadas a los mismos y se discutirá acerca de los resultados de las pruebas.

Palabras clave— Red neuronal, neurona artificial, dropout, regularización, curva de Aprendizaje, curva de precisión, matriz de confusión, sensibilidad de entradas, hiperparámetros, procesamiento de datos

I. Presentación del problema

El problema en cuestión describe distintos tipos de falla en un robot que realiza su propio movimiento para el transporte de piezas. Los datos de falla se capturaron en 5 experimentos, cada uno de ellos con reportes distintos de falla, colisión, obstrucción o éxito. La información sobre este conjunto de datos se puede apreciar en el Cuadro I.

Cuadro I
Información sobre los datos de fallas en el robot

Conjunto de Datos	Descripción	Número de muestras	Distribución
LP1	Failures in approach to grasp position	88	24 % normal 19 % collision 18 % front collision 39 % obstruction
LP2	Failures in transfer of a part	47	43 % normal 13 % front collision 15 % back collision 11 % collision to the right 19 % collision to the left
LP3	Position of part after a transfer failure	47	43 % ok 19 % slightly moved 32 % moved 6 % lost
LP4	Failures in approach to ungrasp position	117	21 % normal 62 % collision 18 % obstruction
LP5	Failures in motion with part	164	27 % normal 16 % bottom collision 13 % bottom obstruction 29 % collision in part 16 % collision in tool

Cada conjunto de datos se centra en un aspecto específico del proceso de movimiento del robot. Este análisis de datos busca agrupar las fallas en clases coherentes que permitan una mejor comprensión de los problemas que enfrenta el sistema. Los datos, además de registrar fallos, también incluyen situaciones exitosas, como se observa en los casos etiquetados como “normal” o “ok”. Los errores relacionados con colisiones y obstrucciones, que constituyen una parte considerable de las muestras, son clave para distribuir las fallas en subclases.

La idea es, para el presente problema, crear una red neuronal que sea capaz de clasificar los datos en distintas categorías según los escenarios descritos en el conjunto de datos. De esta forma, la naturaleza del problema exige un paradigma de Inteligencia Artificial que resuelva un problema de clasificación para mejorar la eficiencia del robot en su tarea de manipulación de piezas.

II. Desarrollo del modelo neuronal #1: Clasificación en función de "Error" vs "No Error"

II-A. Planteamiento de la estrategia

Se desarrolló un programa en Python que procesa los cinco conjuntos de datos experimentales, separándolos en dos clases: "Error" y "No Error". La estrategia utilizada consistió en agrupar las mediciones de fuerza y torque de los ejes (Fx, Fy, Fz, Tx, Ty, Tz) en secuencias de 15 mediciones consecutivas, generando un total de 90 entradas por fila. Estas 90 entradas fueron seleccionadas para capturar suficiente contexto del comportamiento del sistema en diferentes momentos, brindando una visión clara de los patrones asociados a los fallos o al funcionamiento correcto del robot.

Además, se empleó un esquema one-hot encoding para las salidas, donde se añadieron dos columnas adicionales: una correspondiente a la clase "No Error" y otra a la clase "Error". Cada fila de datos se etiquetó con un valor de 1 en la columna correspondiente a su clase, y un 0 en la otra, asegurando así una correcta representación de las clases para el entrenamiento supervisado de la red neuronal.

En el Cuadro II se tabula cómo los datos fueron clasificados en las dos clases. Consultar el **Anexo 1** para encontrar el código que genera el preprocesamiento de estos datos.

Cuadro II
Agrupamiento en dos clases: No Error y Error

Conjunto de Datos	No Error	Error
LP1	Normal	Collision, Obstruction Front collision, Back collision Right collision, Left collision
LP2	Normal	Front collision, Back collision Collision to the right, Collision to the left
LP3	Ok	Slightly moved Moved, Lost
LP4	Normal	Collision, Obstruction
LP5	Normal	Bottom collision, Bottom obstruction Collision in part, Collision in tool

II-B. Descripción de la solución

Para la red neuronal clásica, se generó un modelo de clasificación de 90 entradas y 2 neuronas de salida. Aquí se encuentra la importancia de realizar el preprocesado del conjunto de datos, ya que en este código de la implementación, la red únicamente debe realizar una normalización de los datos.

Para el estudio de hiperparámetros, se tienen los valores tabulados en el Cuadro III.

- Porcentaje de entrenamiento del 70 %, un valor por defecto que no presentó problemas para la validación.
- La arquitectura se deja en 2 capas ocultas con 8 neuronas cada una, se estudió que esta configuración es suficiente para capturar patrones complejos sin sobre ajustar el modelo. Con 6 neuronas el modelo no logra acertar con precisión y con 10 comienza a sobre entrenar.
- La función de activación sigmoide, pues es adecuada para el problema de clasificación.
- Tasa de aprendizaje (0.002): Esta tasa baja genera un ajuste lento de los pesos, pero evita cambios bruscos y asegura una convergencia estable.
- Función de pérdida (Entropía cruzada categórica): Por ser problema de clasificación.
- Tamaño del batch (20): Se estudió que valores menores a este no mejoran la precisión en la convergencia.
- Número de épocas (800): El estudio aseguró que este valor en el que se alcanza a visualizar la estabilidad por completo del entrenamiento.

Cuadro III
Parámetros del primer modelo

Parámetro	Valor
Porcentaje de entrenamiento	70 %
Número de capas	2
Unidades por capa	8
Función de activación	Sigmoid
Tasa de aprendizaje	0.002
Función de pérdida	Entropía cruzada categórica
Tamaño del batch	20
Número de épocas	800

II-C. Análisis y Resultados Obtenidos

Al realizar el entrenamiento con los hiperparámetros mostrados en el Cuadro III, se obtuvo las curvas de la **Figura 1** como resultado. Al analizar la gráfica de pérdida en el entrenamiento, se puede apreciar que el modelo logró mejorar considerablemente el error basal con un entrenamiento exitoso, alcanzando un valor de pérdida que converge a 0. Sin embargo, la gráfica de validación indica que la red presenta sobre-entrenamiento y requiere revisión de su configuración. Al analizar el error basal y parte de la gráfica, se podría determinar que lo mejor es detener el entrenamiento alrededor de las 100 épocas, que es donde se alcanza el mejor rendimiento del modelo. Este rendimiento ofrece una pérdida de 0.05.

La respuesta del equipo de trabajo, en lugar de corregir hiperparámetros, fue aplicar 2 capas de los métodos vistos en clase de corrección del sobre ajuste. Esto con el fin de aplicar diversos conocimientos y no únicamente una variación de prueba y error en los hiperparámetros. Los métodos en cuestión fueron los de penalización a pesos altos por regularización L2 y la desconexión aleatoria de nodos por "dropout". Al aplicar estos métodos se obtuvo una curva como la de la **Figura 2**.

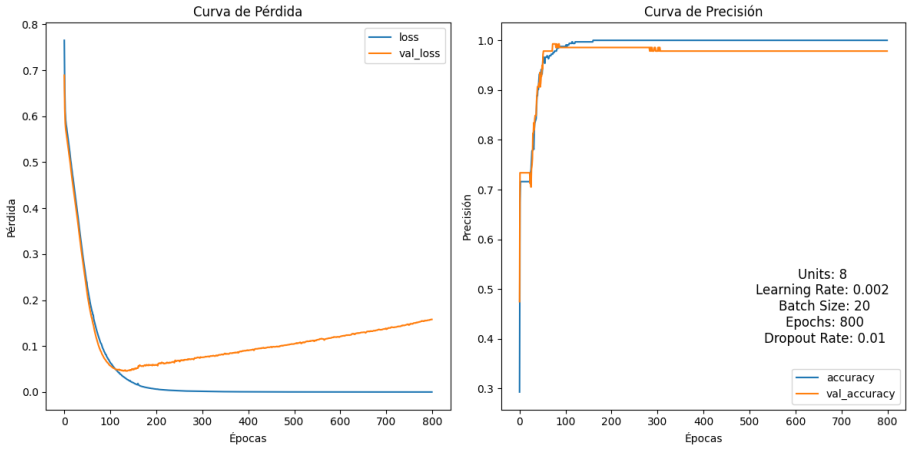

Figura 1. Curvas de aprendizaje y precisión obtenidas con el Modelo 1 para la clasificación entre Error y no Error – Anexo **3**. (Comentar líneas 63 y 64)

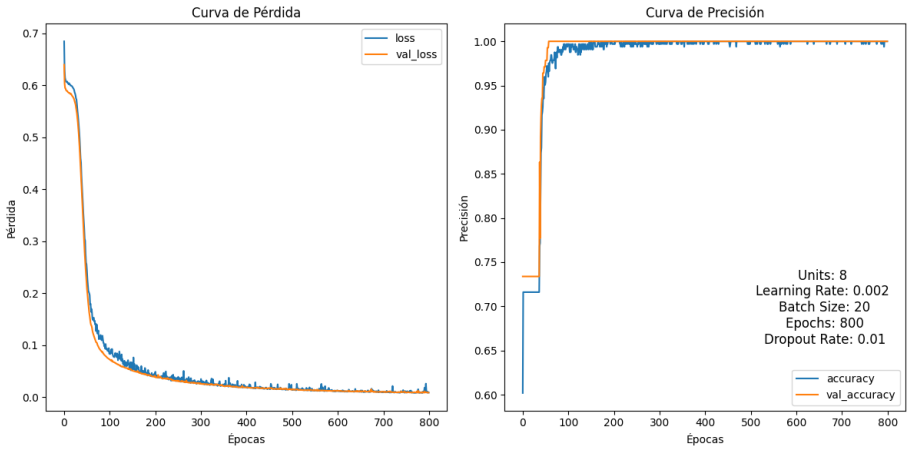

Figura 2. Curvas de aprendizaje con implementación de regularización L2 y "dropout" en el modelo – Anexo **3**.

El hecho de que las curvas de validación (tanto de pérdida como de precisión) sigan de cerca las curvas de entrenamiento respalda la conclusión de que el modelo ha logrado aprender las características generales del problema y no se ha ajustado solo a las particularidades del conjunto de entrenamiento gracias a los dos métodos aplicados.

Esto resulta en que el entrenamiento puede seguir hasta las 600 - 800 épocas y la curva de validación sigue mejorando. Tanto es el caso que la precisión de validación alcanza un valor de 1 (diferente del 0.96 obtenido sin estos 2 métodos). El único percance de aplicar estos métodos es que la curva de entrenamiento presenta ruido al tener que ajustarse continuamente debido a las perturbaciones introducidas en L2 y "dropout".

III. Desarrollo del modelo neuronal #2: Clasificación en función de "No Error" y 3 categorías de "Error"

III-A. Planteamiento de la estrategia

Para llevar a cabo la clasificación en este nuevo conjunto de clases, se realiza el preprocesamiento de los datos de la misma forma en la que se desarrolló para el primer modelo. En el Cuadro IV se tabula cómo los datos fueron clasificados en las cuatro clases. Consultar el **Anexo 2** para encontrar el código que genera el preprocesamiento de estos datos.

Cuadro IV
Agrupamiento en cuatro clases: No Error, Obstruction, Severe Collision y Mild Collision

Conjunto de Datos	No Error	Obstruction	Severe Collision	Mild Collision
LP1	Normal	Obstruction	Collision, Front collision, Back collision Right collision, Left collision	-
LP2	Normal	-	Front collision, Back collision Collision to the right, Collision to the left	-
LP3	Ok	-	Lost	Moved, Slightly moved
LP4	Normal	Obstruction	Collision	-
LP5	Normal	Bottom obstruction	Bottom collision, Collision in part, Collision in tool	-

III-B. Descripción de la solución

El estudio de hiperparámetros indicó que, a pesar del aumento en las categorías por clasificar, los hiperparámetros del primer modelo (Cuadro III) funcionaron correctamente para este segundo modelo. Sin embargo, el número de épocas sí necesitó elevarse, puesto que se necesita un entrenamiento más extenso por la cantidad de categorías a clasificar. Por otro lado, la relación entre número de neuronas y la tasa de "dropout" son dos hiperparámetros que tienen una relación interesante para el rendimiento de la red.

Otros hiperparámetros interesantes que se detallarán en el análisis son los siguientes:

- Número de neuronas
- Tasa de "Dropout"

Cuadro V
Parámetros del segundo modelo

Parámetro	Valor
Porcentaje de entrenamiento	70 %
Número de capas	2
Unidades por capa	40
Función de activación	Sigmoid
Tasa de aprendizaje	0.002
Función de pérdida	Entropía cruzada categórica
Tamaño del batch	20
Número de épocas	1000
Dropout	8 %

III-C. Análisis y Resultados Obtenidos

Al realizar el entrenamiento con los hiperparámetros mostrados en el Cuadro V, se obtuvo las curvas de la **Figura 3** como resultado. Al analizar la gráfica de pérdida en el entrenamiento, se puede apreciar que el modelo logró mejorar considerablemente el error basal con un entrenamiento exitoso, alcanzando un valor de pérdida que converge aproximadamente al 1.

Sin embargo, la gráfica de validación indica que la red presenta sobre-entrenamiento y requiere revisión de su configuración. Al analizar el error basal y parte de la gráfica, se podría determinar que lo mejor es detener el entrenamiento alrededor de las 80 épocas, que es donde se alcanza el mejor rendimiento del modelo, aunque este modelo no es adecuado para su uso debido a que ofrece una pérdida de 0.8.

La respuesta del equipo de trabajo fue reducir el número de neuronas en cada capa oculta. Esto se debe a que al tener una gran cantidad de neuronas, estas tienden a aprenderse los valores del conjunto de entrenamiento. Otra de las alternativas era aumentar los valores de la regularización L2 y del "dropout", sin embargo, esto resulta ineficiente debido a que el problema no requiere tanto poder computacional para cumplir con la categorización. Al aplicar este cambio se obtuvo una curva como la de la **Figura 4**.

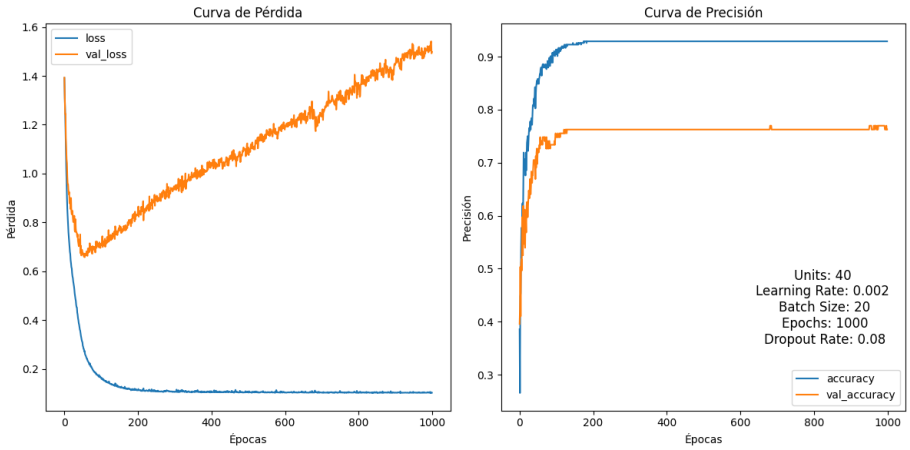

Figura 3. Curvas del Modelo 2 - Primera Configuración: Con un **Dropout de 0**, un **Número de Neuronas de 40** e implementación de **Pesos a las clases** Anexo **4**. (Comentar líneas 54 y 55)

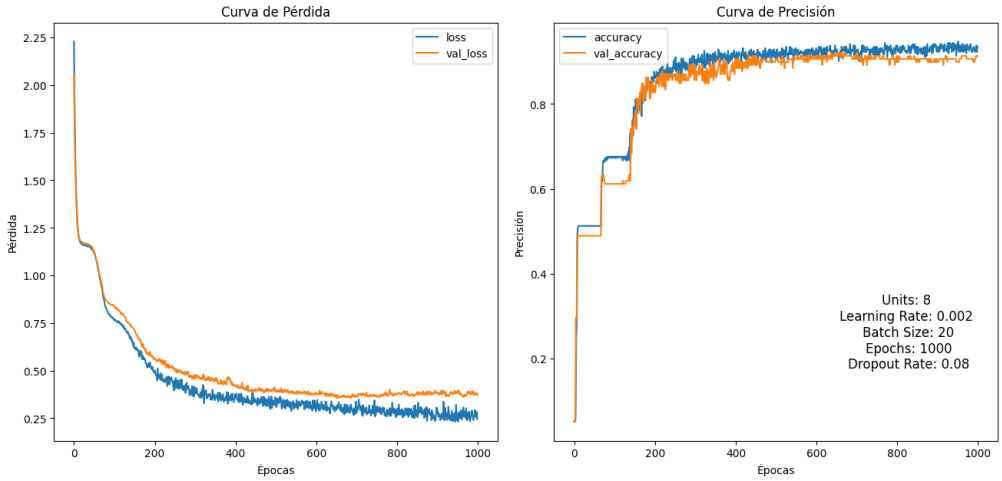

Figura 4. Curvas del Modelo 2 - Segunda Configuración: Con un **Dropout de 0.08** y un **Número de Neuronas de 8** – Anexo **4**.

El hecho de que las curvas de validación (tanto de pérdida como de precisión) sigan de cerca las curvas de entrenamiento respalda la conclusión de que el modelo ha logrado aprender gran parte de las características generales del problema, aunque no logró aprender todas las características como en el modelo 1, ya que el valor de perdida converge en 0.25 mientras que la precisión en un 0.90

IV. Estudio de la sensibilidad de entradas

El presente estudio de sensibilidad de entradas tiene como objetivo determinar y analizar la respuesta del modelo ante las variaciones en una única entrada de las 90 que conforman la red. Dado que el problema es de clasificación, lo que se quiere encontrar es en la cuales entradas se puede conseguir un cambio en la clase predicha ante la variación porcentual de esta.

La forma en que se puede visualizar el cambio es mediante una gráfica donde en un eje se muestre el porcentaje de variación de la entrada en cuestión y en otro eje la probabilidad que la red asigna a cada clase. Esta variación se realizó con la metodología “ceteris paribus”, por lo que las 89 entradas restantes se mantienen constantes.

A continuación, se presentan los resultados obtenidos en las gráficas generadas en Python. Con el fin de no colocar las 90 entradas, se muestran únicamente las respuestas generadas por las entradas más y menos significativas para cada modelo.

IV-A. Entrada más significativa para el modelo neuronal #1

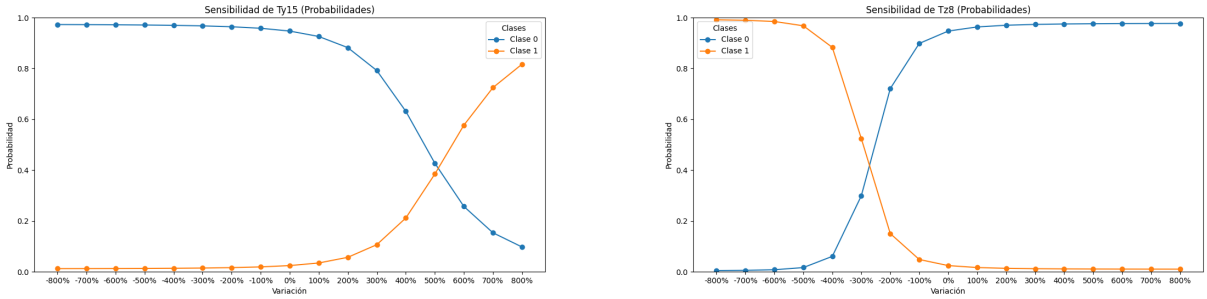

Figura 5. Gráfica de probabilidad calculada por el modelo ante la variación de la entrada Ty15 y Tz8 – Anexo **5**.

La Figura 5 presenta que **Ty15** y **Tz8** son las entradas cuya variación fue **más** significativa, pues presentaron mayor cambio en la respuesta de la red. Sin embargo, dichas entradas no son igual de significativas durante todo el espacio de variación, sino que se encuentran rangos de valores donde presentan sensibilidad.

Ty15 por su parte, presenta un rango sensible que va de 0 % a 800 %. Fuera de este, la variación de esta entrada no provoca cambio alguno en la probabilidad asignada a las clases. **Tz8**, por otro lado, presenta un rango sensible mucho más pequeño que va de -500 % a -100 %. Lo cual es interesante, pues se ubica en magnitudes negativas de variación. De igual forma, fuera del rango, la variación de esta entrada es insignificante.

IV-B. Entrada menos significativa para el modelo neuronal #1

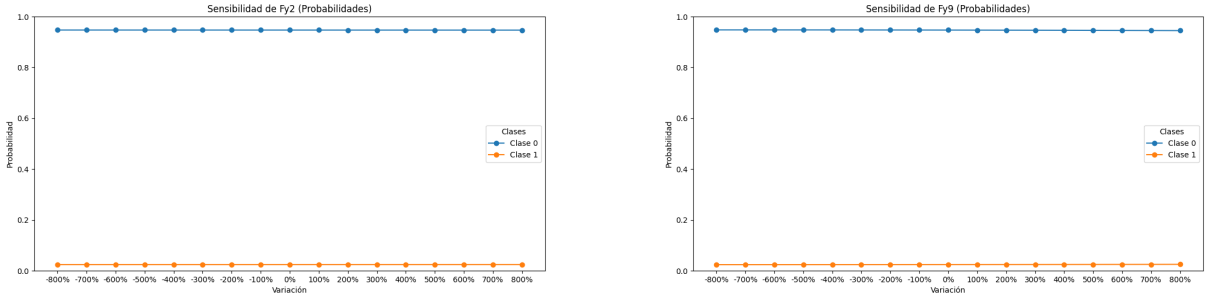

Figura 6. Gráfica de probabilidad calculada por el modelo ante la variación de la entrada Fy2 y Fy9 – Anexo **5**.

Para la menos significativa, hubo muchas entradas que se generaron gráficas muy estables, sin embargo, durante todas las 15 mediciones, la entrada cuya variación fue **menos** significativa fue **Fy**. En la Figura 6 se aprecian Fy2 y Fy9 como ejemplo.

IV-C. Entrada más significativa para el modelo neuronal #2

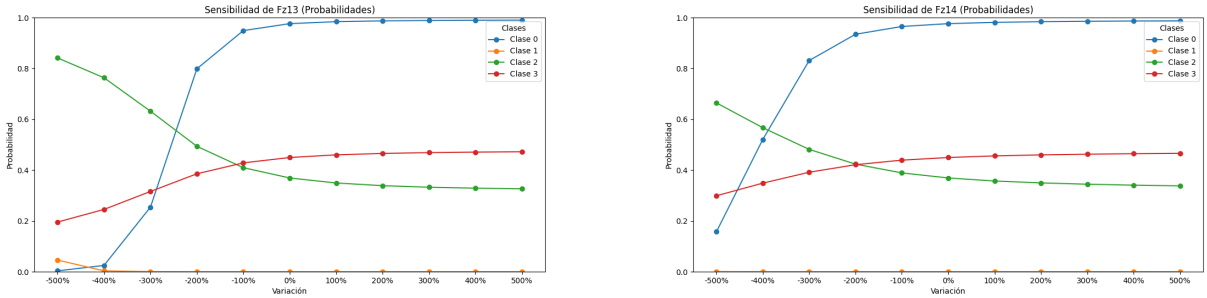

Figura 7. Gráfica de probabilidad calculada por el modelo ante la variación de la entrada Fz13 y Fz14 – Anexo **5**.

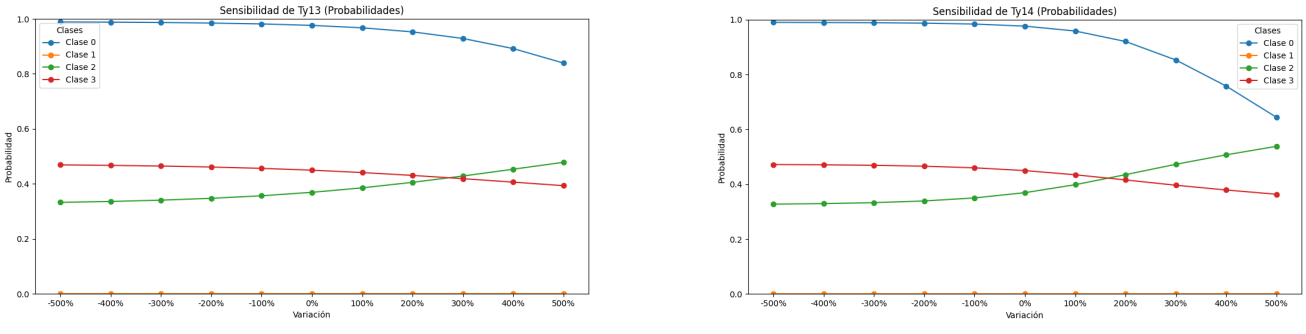

Figura 8. Gráfica de probabilidad calculada por el modelo ante la variación de la entrada Ty13 y Ty14 – Anexo **5**.

La Figuras 7 y 8 presentan que **Fz** y **Ty** son las entradas cuya variación fue más significativa, pues presentaron mayor cambio en la respuesta de la red. De igual forma, poseen rangos de valores donde tienen mayor sensibilidad. Tanto en Fz como Ty, ocurre que son las últimas mediciones (13 y 14) las más sensibles.

Fz13-14 por su parte, presentan un rango sensible que va de -500 % a 0 %. **Ty13-14**, por otro lado, tiene el rango al opuesto en magnitud, de 0 % a 500 %. De igual forma, fuera del rango, la variación de ambas entradas es insignificante.

IV-D. Entrada menos significativa para el modelo neuronal #2

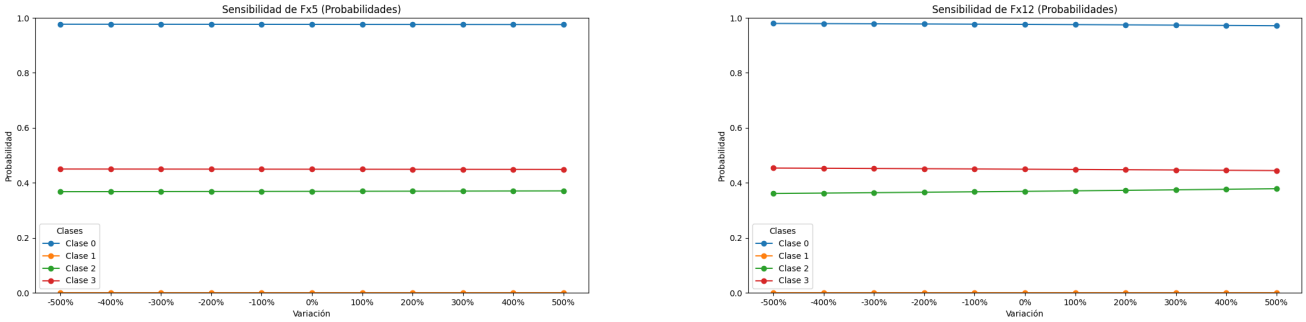

Figura 9. Gráfica de probabilidad calculada por el modelo ante la variación de la entrada Fx5 y Fx12 – Anexo **5**.

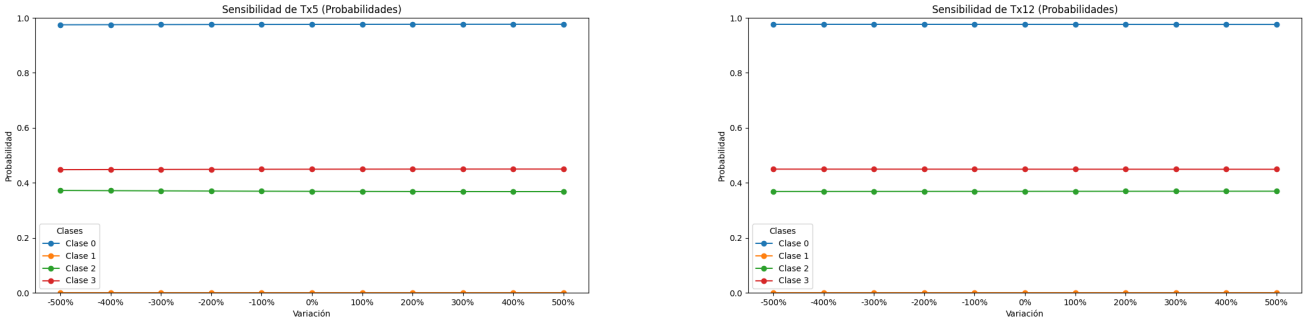

Figura 10. Gráfica de probabilidad calculada por el modelo ante la variación de la entrada Tx5 y Tx12 – Anexo **5**.

De igual forma al primer modelo, hubo muchas entradas que se generaron gráficas muy estables, sin embargo, durante todas las 15 mediciones, las entradas con variación **menos** significativas fueron **Fx y Tx**. En la Figura 6 se aprecian las mediciones 5 y 12 como ejemplo.

IV-E. Conclusiones en función de la naturaleza física del problema

A modo de resumen, la significancia de cada parámetro y su relación con el entorno físico de fallas en robots se puede encontrar en el Cuadro VI

Cuadro VI
SENSIBILIDAD PARA GENERAR UNA FALLA Y RELACIÓN A LA NATURALEZA FÍSICA DEL PROBLEMA

Parámetro	Sensibilidad para generar una falla	Relación respecto a la naturaleza de las entradas
Fx, Fy y Tx	Insignificante	La aplicación de fuerza en este eje no es relevante para generar una falla. Puede deberse a que la fuerza en este eje lleva a cabo tareas no riesgosas.
Fz y Tz	Significativa en valores negativos	La aplicación de fuerza o torque en el **eje Z** precisamente son relevantes para generar una falla. De esto, se puede inferir que la fuerza y torque en este eje están relacionados con el control del traslado y rotación, respectivamente, del propio robot o de las piezas. En estas acciones, el robot parece experimentar magnitudes que alteran su estabilidad. El hecho de que solo en valores negativos sea sensible, quiere decir que solo hay una dirección crítica de este eje en el que se vuelve inestable.
Ty	Significativa en valores positivos	La aplicación de **torque** en el **eje Y** también resulta relevante para generar una falla. Esto sugiere que el torque en este eje está vinculado principalmente con el control de la rotación lateral del robot o de las piezas durante el transporte. En estas acciones, el robot parece experimentar variaciones que afectan su estabilidad lateral. El hecho de que sea sensible únicamente en valores positivos indica que hay una dirección específica en la cual el sistema se vuelve inestable, probablemente

V. ESTUDIO DE ERROR

V-A. Estudio de error modelo neuronal #1 ("Error" vs "No Error")

Para realizar el estudio de distribución de error es necesario visualizar la distribución de los datos y como estos fueron predicho por el mejor modelo neuronal #1. En la **Figura 11** se observa la matriz de confusión correspondiente al modelo neuronal #1, en el cual la diagonal principal es la única que posee valores diferentes a 0, lo que corresponde a una matriz perfecta. Esto nos permite determinar que la distribución de error es uniforme y que la red neuronal aprendió correctamente las tendencias.

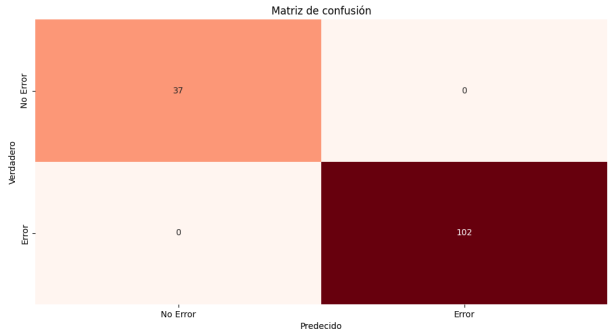

Figura 11. Matriz de confusión obtenida para el modelo 1 – Anexo **3**.

V-B. Estudio de error modelo neuronal #2 ("No Error" vs 3 tipos de "Error")

Para el estudio de error del modelo neuronal #2 se generó la **Figura 12** y **13**. En la **Figura 13** se observa que la matriz de confusión es imperfecta, teniendo los siguientes errores:

- Clasificar 1 error tipo "obstruction" como "severe collision".
- Clasificar 3 errores tipo "severe collision" como "no error".
- Clasificar 1 error tipo "severe collision" como "obstruction".
- Clasificar 7 errores tipo "severe collision" como "mild collision".

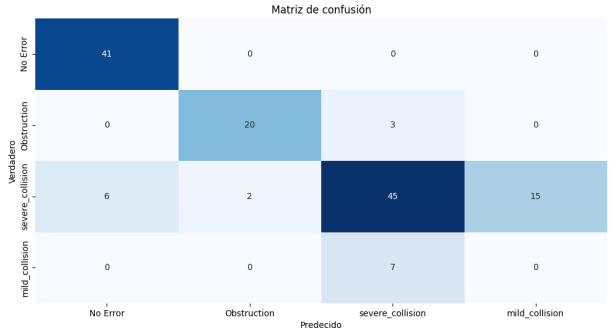

Figura 12. Matriz de confusión obtenida para el modelo 2 con un **Dropout de 0**, un **Número de Neuronas de 40** e implementación de **Pesos a las clases** Anexo **4**.

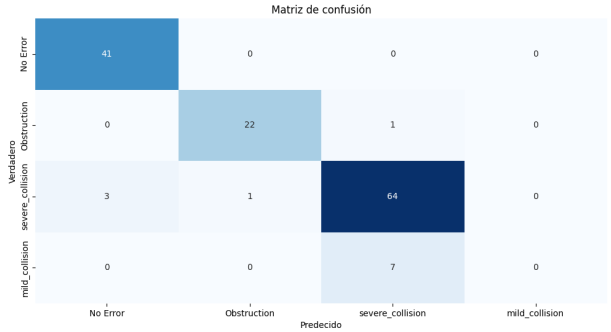

Figura 13. Matriz de confusión obtenida para el modelo 2 con un **Dropout de 0.08** y un **Número de Neuronas de 8** – Anexo **4**.

De las 12 clasificaciones incorrectas, 8 fueron determinadas como “severe collision”, lo que representa un 66.67 %, por lo tanto, la distribución del error no es uniforme.

Se tienen 3 hipótesis para la causa de la distribución no uniforme del error. La hipótesis #1 consiste en que se tiene muy pocos datos en el conjunto del error tipo “mild collision”, por lo que el modelo neuronal #2 no es capaz de sustraer las características de dicha categoría, esto se ve respaldado por el porcentaje que representa el error tipo “mild collision” que es un 5.2 % de los datos totales, mientras que el error tipo “severe collision” representa un 54.4 % de los datos totales causando que el modelo neuronal pueda favorecer el aprendizaje de las características de dicha categoría. Esto causa que los valores de los pesos prioricen la adaptación hacia el error tipo “severe collision” debido a que cuando los valores de los pesos se adaptan al error tipo “mild collision” este no se mantiene por mucho tiempo, ya que llegan unos 10 valores del error tipo “severe collision” que cambian los valores de los pesos a su favor.

La hipótesis #2 consiste en que los errores tipo “mild collision” y “severe collision” comparten varías características, causando que el modelo los confunda, lo cual se observa en la **Figura 12** donde 15 errores del tipo “severe collision” fueron clasificados como “mild collision” y los 7 errores del tipo “mild collision” fueron clasificados como “severe collision”, además, se debe considerar que esta matriz de confusión es de la red neuronal sobre entrenada de la **Figura 3**. La razón por la que ambos errores pueden compartir varias características se debe a que en el error tipo “mild collision” están los datos de posición después de un error en el transporte (LP3) y todos los datos de errores en el transporte (LP2) se encuentran en el error tipo “severe collision”, esto se observa en los Cuadros I y IV.

La hipótesis #3 consiste en la combinación de la hipótesis #1 y #2, siendo las causas de la distribución no uniforme del error, la falta de datos para el error tipo “mild collision” y la similitud entre los errores tipo “severe collision” y “mild collision”.

VI. Conclusiones

El objetivo principal de esta tarea fue desarrollar dos modelos neuronales capaces de clasificar los datos en función de “error vs no error” y en función de “tres tipos de error y ausencia de error”.

El modelo neuronal #1 es capaz de detectar si existe o no un error con una alta tasa de precisión, mientras que el modelo #2 es capaz de clasificar con alta precisión la ausencia de error y dos de los tres tipos de error, esto se debe a la optimización de los hiperparámetros y el uso de técnicas de corrección de sobre ajuste, debido a que estas últimas le permiten a la red neuronal tener más iteraciones para aprender las tendencias y no valores específicos.

El desempeño de la red neuronal #1 en el estudio de sensibilidad en las entradas, permite asegurar que la entrada más significativa es el valor final (15) del torque, principalmente las componentes y y z, mientras que la entrada menos significativa es la componente y de la fuerza. Esto implica que lo más relevante para mejorar el desempeño del robot y prevenir las fallas es la regulación del torque que este genera.

Por otra parte, el desempeño de la red neuronal #2 en el estudio de sensibilidad en las entradas, permite asegurar que la entrada más significativa son los valores finales de la componente z de la fuerza y la componente y del torque, mientras que las entradas menos significas son las componentes x de la fuerza y el torque. Esto implica que para prevenir errores específicos se debe regular el torque, principalmente su componente y.

La matriz de confusión obtenida por la red neuronal #1 clasifica de forma precisa la presencio y ausencia de error, sin embargo, la matriz de confusión obtenida por la red neuronal #2 indica que la ausencia de error, fallas por obstrucción y por colisión son clasificas adecuadamente, pero la posición final después de un fallo por colisión. Esto se debe a la poca cantidad de datos para esa categoría y la similitud en la fuerza y torque con la falla por colisión.

VII. Anexos

VII-A. Anexo 1: Código para el preprocesado de datos del modelo 1

```
import pandas as pd

column_names = ['Fx', 'Fy', 'Fz', 'Tx', 'Ty', 'Tz']  # Nombres base de las columnas
input_files = ['lp1.txt', 'lp2.txt', 'lp3.txt', 'lp4.txt', 'lp5.txt']  # Archivos de entrada
output_excel_path = 'Data_two_classes.xlsx'  # Archivo Excel de salida
all_data = []  # Lista para almacenar todos los DataFrames
num_groups = 15  # Numero de grupos de mediciones por fila

# Procesar cada archivo de entrada
for input_file_path in input_files:
    # Inicializar variables para almacenar los datos
    data = []
    labels = []
    current_label = None
    current_row = []

    # Leer y procesar el archivo de texto
    with open(input_file_path, 'r') as file:
        for line in file:
            line = line.strip()
            # Verificar si la linea contiene una etiqueta de clase
            if line in ['normal', 'ok']:
                current_label = 'No Error'
            elif line in ['obstruction', 'bottom_obstruction', 'fr_collision',
                          'front_col', 'right_col', 'left_col', 'back_col',
                          'collision_in_part', 'collision_in_tool', 'collision',
                          'bottom_collision', 'slightly_moved','moved', 'lost']:
                current_label = 'Error'
            else:
                # Intentar convertir la linea en valores numericos
                if current_label:
                    try:
                        values = list(map(float, line.split()))
                        if len(values) == 6:
                            current_row.extend(values)  # Agregar las mediciones a la fila
                                actual
                            if len(current_row) == 6 * num_groups:  # Verificar si se han
                                alcanzado las 15 mediciones
                                current_row.append(1 if current_label == 'No Error' else 0)  #
                                    Agregar columna 'No Error'
                                current_row.append(1 if current_label == 'Error' else 0)  #
                                    Agregar columna 'Error'
                                data.append(current_row)
                                current_row = []  # Reiniciar la fila
                    except ValueError:
                        # Ignorar lineas que no contienen datos numericos validos
                        continue

    # Crear los nombres de las columnas dinamicamente
    dynamic_columns = []
    for i in range(1, num_groups + 1):
        for col in column_names:
            dynamic_columns.append(f'{col}{i}')


    dynamic_columns.extend(['No Error', 'Error'])   # Agregar las columnas No Error y Error
    df = pd.DataFrame(data, columns=dynamic_columns) # Crear un DataFrame con los datos
    all_data.append(df) # Agregar el DataFrame a la lista

combined_data = pd.concat(all_data, ignore_index=True) # Concatenar todos los DataFrames
combined_data.to_excel(output_excel_path, index=False) # Guardar en archivo Excel
print(f"El archivo procesado ha sido guardado como '{output_excel_path}'.")
```

VII-B. Anexo 2: Código para el preprocesado de datos del modelo 2

```
import pandas as pd  # Importa la libreria pandas

column_names = ['Fx', 'Fy', 'Fz', 'Tx', 'Ty', 'Tz']  # Nombres base de las columnas
input_files = ['lp1.txt', 'lp2.txt', 'lp3.txt', 'lp4.txt', 'lp5.txt']  # Archivos de entrada
output_excel_path = 'Data_four_classes.xlsx'  # Archivo Excel de salida
all_data = []  # Lista para almacenar todos los DataFrames
num_groups = 15  # Numero de grupos de mediciones por fila

for input_file_path in input_files:  # Procesar cada archivo de entrada
    data = []  # Lista para almacenar los datos del archivo
    current_label = None  # Etiqueta de clase actual
    current_row = []  # Lista para almacenar una fila de datos

    with open(input_file_path, 'r') as file:  # Abrir archivo de entrada
        for line in file:  # Leer linea por linea
            line = line.strip()  # Eliminar espacios en blanco alrededor de la linea
            if line in ['normal', 'ok']:  # Asignar etiqueta 'No Error'
                current_label = 'No Error'
            elif line in ['obstruction', 'bottom_obstruction']:  # Asignar etiqueta '
                obstruction'
                current_label = 'obstruction'
            elif line in ['fr_collision', 'front_col', 'right_col', 'left_col', 'lost', '
                back_col', 'collision', 'collision_in_part', 'collision_in_tool', '
                bottom_collision']:  # Asignar etiqueta 'severe_collision'
                current_label = 'severe_collision'
            elif line in ['moved', 'slightly_moved']:  # Asignar etiqueta 'mild_collision'
                current_label = 'mild_collision'
            else:
                if current_label:  # Si hay una etiqueta valida
                    try:
                        values = list(map(float, line.split()))  # Convertir linea en valores
                            numericos
                        if len(values) == 6:  # Si la linea tiene 6 valores
                            current_row.extend(values)  # Agregar valores a la fila actual
                            if len(current_row) == 6 * num_groups:  # Verificar si se han
                                alcanzado 15 mediciones
                                # Agregar las columnas:
                                current_row.append(1 if current_label == 'No Error' else 0)
                                current_row.append(1 if current_label == 'obstruction' else 0)
                                current_row.append(1 if current_label == 'severe_collision'
                                    else 0)
                                current_row.append(1 if current_label == 'mild_collision' else
                                    0)
                                data.append(current_row)  # Agregar la fila a los datos
                                current_row = []  # Reiniciar la fila actual
                    except ValueError:  # Si ocurre un error en la conversion de valores
                        continue  # Ignorar la linea no valida

    dynamic_columns = []  # Lista para nombres de columnas dinamicos
    for i in range(1, num_groups + 1):  # Crear nombres de columnas para los grupos de
        mediciones
        for col in column_names:
            dynamic_columns.append(f'{col}{i}')  # Agregar columnas como Fx1, Fy1, etc.
    dynamic_columns.extend(['No Error', 'obstruction', 'severe_collision', 'mild_collision'])
    df = pd.DataFrame(data, columns=dynamic_columns)
    all_data.append(df)  # Agregar el DataFrame a la lista de todos los datos

combined_data = pd.concat(all_data, ignore_index=True)  # Concatenar todos los DataFrames en
    uno solo

combined_data.to_excel(output_excel_path, index=False)  # Guardar el DataFrame en un archivo
    Excel

print(f"El archivo procesado ha sido guardado como '{output_excel_path}'.")  # Confirmacion de
    guardado
```

VII-C. Anexo 3: Código para la RNA del modelo 1

```
import winsound
import pandas
from keras import layers, models, Input
import matplotlib.pyplot as plt
from keras.src.utils.module_utils import tensorflow
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Hiperparametros
train_percentage = 0.70
n = 2
units = 8
activation = "sigmoid"
learning_rate = 0.002
loss = "categorical_crossentropy"
batch_size = 20
epochs = 800
dropout_rate = 0.01
L2_reg_factor = 0.001

# Lectura de los datos
dataset = pandas.read_excel("Data_two_classes.xlsx")
print(dataset)

# Separacion de caracteristicas (features) y etiquetas (labels)
X = dataset[["Fx1", "Fy1", "Fz1", "Tx1", "Ty1", "Tz1",
             "Fx2", "Fy2", "Fz2", "Tx2", "Ty2", "Tz2",
             "Fx3", "Fy3", "Fz3", "Tx3", "Ty3", "Tz3",
             "Fx4", "Fy4", "Fz4", "Tx4", "Ty4", "Tz4",
             "Fx5", "Fy5", "Fz5", "Tx5", "Ty5", "Tz5",
             "Fx6", "Fy6", "Fz6", "Tx6", "Ty6", "Tz6",
             "Fx7", "Fy7", "Fz7", "Tx7", "Ty7", "Tz7",
             "Fx8", "Fy8", "Fz8", "Tx8", "Ty8", "Tz8",
             "Fx9", "Fy9", "Fz9", "Tx9", "Ty9", "Tz9",
             "Fx10", "Fy10", "Fz10", "Tx10", "Ty10", "Tz10",
             "Fx11", "Fy11", "Fz11", "Tx11", "Ty11", "Tz11",
             "Fx12", "Fy12", "Fz12", "Tx12", "Ty12", "Tz12",
             "Fx13", "Fy13", "Fz13", "Tx13", "Ty13", "Tz13",
             "Fx14", "Fy14", "Fz14", "Tx14", "Ty14", "Tz14",
             "Fx15", "Fy15", "Fz15", "Tx15", "Ty15", "Tz15"]]  # Seleccionar caracteristicas (
                 variables de entrada)
Y = dataset[["No Error", "Error"]]  # Seleccionar etiqueta (variable objetivo)

scaler = StandardScaler()
X_normalized_array = scaler.fit_transform(X)

X_normalized = pandas.DataFrame(X_normalized_array, columns=X.columns)

X_train, X_test, y_train, y_test = train_test_split(X_normalized, Y, test_size=1 -
    train_percentage, random_state=23)

# Inicializacion del modelo
network = models.Sequential()

# Declaracion de la capa de entrada
network.add(Input(shape=(90,)))  # entradas

# Ciclo de capas de neuronas intermedias
for i in range(n):  # n: numero de neuronas intermedias
    network.add(layers.Dense(
            units=units,  # units: numero de neuronas por capa
            activation=activation))  # activation: funcion de activacion elegida.
    network.add(tensorflow.keras.layers.Dropout(dropout_rate))
    network.add(layers.Dense(units=units, activation=activation, kernel_regularizer=tensorflow.
        keras.regularizers.l2(L2_reg_factor)))
```

```
65
66  # Declaracion de la capa de salida
67  network.add(layers.Dense(
68          units=2,  # Una unica salida
69          activation="sigmoid"))  # Problema de regresion, por tanto salida dada por sigmoide
70
71  network.compile(
72          optimizer=tensorflow.keras.optimizers.Adam(  # optimizer: algoritmo de optimizacion
73              learning_rate=learning_rate  # learning_rate: ritmo de aprendizaje
74          ),
75          loss=loss,
76          metrics=['accuracy'])  # loss: funcion de perdida
77
78  losses = network.fit(x=X_train,  # Caracteristicas de entrada para el entrenamiento
79                       y=y_train,  # Etiquetas para el entrenamiento
80                       validation_data=(X_test, y_test),  # Datos para la validacion durante el
                             entrenamiento
81                       batch_size=batch_size,  # Numero de muestras por actualizacion de
                             gradiente
82                       epochs=epochs)  # Numero de iteraciones completas a traves de los datos de
                              entrenamiento
83
84  winsound.Beep(350, 500)  # Aviso auditivo de la finalizacion del entrenamiento.
85
86  # Se extrae el historial de error contra iteraciones de la clase
87  loss_df = pandas.DataFrame(losses.history)
88
89  # Crear la primera figura para la perdida
90  plt.figure(figsize=(12, 6))  # Tamano de la figura (ancho, alto)
91  plt.subplot(1, 2, 1)  # 1 fila, 2 columnas, primer grafico
92  loss_df.loc[:, ['loss', 'val_loss']].plot(ax=plt.gca())  # Graficar la perdida de entrenamiento
       y validacion
93  plt.title("Curva de Perdida")  # Titulo de la grafica de perdida
94  plt.xlabel("Epocas")  # Etiqueta del eje X para la grafica de perdida
95  plt.ylabel("Perdida")  # Etiqueta del eje Y para la grafica de perdida
96
97  plt.subplot(1, 2, 2)  # 1 fila, 2 columnas, segundo grafico
98  loss_df.loc[:, ['accuracy', 'val_accuracy']].plot(ax=plt.gca())  # Graficar la precision de
      entrenamiento y validacion
99  plt.title("Curva de Precision")  # Titulo de la grafica de precision
100 plt.xlabel("Epocas")  # Etiqueta del eje X para la grafica de precision
101 plt.ylabel("Precision")  # Etiqueta del eje Y para la grafica de precision
102
103 # Agregar informacion extra de la RN para saber sus hiperparametros
104 extra_info = (f"Units: {units} "
105               f"\nLearning Rate: {learning_rate} "
106               f"\nBatch Size: {batch_size}"
107               f"\nEpochs: {epochs}"
108               f"\nDropout Rate: {dropout_rate}")
109
110 plt.figtext(0.9, 0.25, extra_info, wrap=True, horizontalalignment='center', fontsize=12)
111 plt.tight_layout()  # Asegurar que las subplots no se solapen
112
113 # Matriz de confusion
114 y_pred = network.predict(X_test)
115 y_pred_classes = y_pred.argmax(axis=1)
116 y_test_classes = y_test.values.argmax(axis=1)
117 cm = confusion_matrix(y_test_classes, y_pred_classes)
118
119 # Visualizacion de la matriz de confusion
120 plt.figure(figsize=(12, 6))
121 sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', cbar=False,
122             xticklabels=["No Error", "Error"],
123             yticklabels=["No Error", "Error"])
124 plt.xlabel('Predecido')
125 plt.ylabel('Verdadero')
126 plt.title('Matriz de confusion')
127 plt.show()
```

VII-D. Anexo 4: Código para la RNA del modelo 2

```
1
2  import winsound  # Para generar un sonido al final del entrenamiento
3  import pandas  # Para manejar datos en forma de tablas
4  from keras import layers, models, Input  # Para crear la red neuronal
5  import matplotlib.pyplot as plt  # Para graficar los resultados
6  from keras.src.utils.module_utils import tensorflow  # Utilidades de Keras con TensorFlow
7  from sklearn.metrics import confusion_matrix  # Para crear la matriz de confusion
8  import seaborn as sns  # Para visualizar datos y la matriz de confusion
9  from sklearn.model_selection import train_test_split  # Para dividir datos en entrenamiento y
       prueba
10 from sklearn.preprocessing import StandardScaler  # Para normalizar los datos
11
12 train_percentage = 0.7  # Porcentaje de datos de entrenamiento
13 n = 2  # Numero de capas intermedias
14 units = 8  # Neuronas por capa
15 activation = "sigmoid"  # Funcion de activacion
16 learning_rate = 0.002  # Ritmo de aprendizaje
17 loss = "categorical_crossentropy"  # Funcion de perdida
18 batch_size = 20  # Tamano del lote
19 epochs = 1000  # Numero de iteraciones completas sobre los datos
20 dropout_rate = 0.08  # Tasa de dropout para evitar sobreajuste
21 L2_reg_rate = 0.001  # Tasa de regularizacion L2
22
23 dataset = pandas.read_excel("Data_four_classes.xlsx")  # Leer el dataset
24
25 X = dataset[["Fx1", "Fy1", "Fz1", "Tx1", "Ty1", "Tz1",  # Seleccion de columnas
26             "Fx2", "Fy2", "Fz2", "Tx2", "Ty2", "Tz2",
27             "Fx3", "Fy3", "Fz3", "Tx3", "Ty3", "Tz3",
28             "Fx4", "Fy4", "Fz4", "Tx4", "Ty4", "Tz4",
29             "Fx5", "Fy5", "Fz5", "Tx5", "Ty5", "Tz5",
30             "Fx6", "Fy6", "Fz6", "Tx6", "Ty6", "Tz6",
31             "Fx7", "Fy7", "Fz7", "Tx7", "Ty7", "Tz7",
32             "Fx8", "Fy8", "Fz8", "Tx8", "Ty8", "Tz8",
33             "Fx9", "Fy9", "Fz9", "Tx9", "Ty9", "Tz9",
34             "Fx10", "Fy10", "Fz10", "Tx10", "Ty10", "Tz10",
35             "Fx11", "Fy11", "Fz11", "Tx11", "Ty11", "Tz11",
36             "Fx12", "Fy12", "Fz12", "Tx12", "Ty12", "Tz12",
37             "Fx13", "Fy13", "Fz13", "Tx13", "Ty13", "Tz13",
38             "Fx14", "Fy14", "Fz14", "Tx14", "Ty14", "Tz14",
39             "Fx15", "Fy15", "Fz15", "Tx15", "Ty15", "Tz15"]]  # Seleccion de caracteristicas
40
41 Y = dataset[["No_Error", "obstruction", "severe_collision", "mild_collision"]]  # Etiquetas
42
43 scaler = StandardScaler()  # Normalizador de datos
44 X_normalized_array = scaler.fit_transform(X)  # Normalizacion
45 X_normalized = pandas.DataFrame(X_normalized_array, columns=X.columns)  # Conversion a
       DataFrame
46
47 X_train, X_test, y_train, y_test = (train_test_split(X_normalized, Y, test_size=1 -
       train_percentage, random_state=21))  # Division de datos
48
49 network = models.Sequential()  # Inicializacion del modelo secuencial
50 network.add(Input(shape=(90,)))  # Capa de entrada
51
52 for i in range(n):  # Anadir capas ocultas
53     network.add(layers.Dense(units=units, activation=activation))  # Capa densa
54     network.add(tensorflow.keras.layers.Dropout(dropout_rate))  # Dropout para regularizacion
55     network.add(layers.Dense(units=units, activation=activation, kernel_regularizer=tensorflow.
           keras.regularizers.l2(L2_reg_rate)))  # Regularizacion L2
56
57 network.add(layers.Dense(units=4, activation="sigmoid"))  # Capa de salida
58
59 network.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate=learning_rate), loss=
       loss, metrics=['accuracy'])  # Compilacion del modelo
60
61 losses = network.fit(x=X_train, y=y_train, validation_data=(X_test, y_test), batch_size=
       batch_size, epochs=epochs)  # Entrenamiento del modelo
```

```
62
63 winsound.Beep(350, 500)  # Aviso sonoro al final del entrenamiento
64
65 loss_df = pandas.DataFrame(losses.history)  # Creacion de DataFrame con el historial del
       entrenamiento
66
67 plt.figure(figsize=(12, 6))  # Creacion de la figura para graficar
68 plt.subplot(1, 2, 1)  # Primer grafico: perdida
69 loss_df.loc[:, ['loss', 'val_loss']].plot(ax=plt.gca())  # Grafico de perdida
70 plt.title("Curva de Perdida")  # Titulo del grafico de perdida
71 plt.xlabel("Epocas")  # Etiqueta del eje X
72 plt.ylabel("Perdida")  # Etiqueta del eje Y
73
74 plt.subplot(1, 2, 2)  # Segundo grafico: precision
75 loss_df.loc[:, ['accuracy', 'val_accuracy']].plot(ax=plt.gca())  # Grafico de precision
76 plt.title("Curva de Precision")  # Titulo del grafico de precision
77 plt.xlabel("Epocas")  # Etiqueta del eje X
78 plt.ylabel("Precision")  # Etiqueta del eje Y
79
80 extra_info = (f"Units: {units} \nLearning Rate: {learning_rate} \nBatch Size: {batch_size}\
       nEpochs: {epochs}\nDropout Rate: {dropout_rate}")  # Informacion adicional
81 plt.figtext(0.9, 0.25, extra_info, wrap=True, horizontalalignment='center', fontsize=12)  #
       Mostrar informacion adicional
82
83 plt.tight_layout()  # Ajustar el diseno para evitar solapamientos
84
85 dato = X_test.sample(n=10)  # Seleccion de 10 muestras aleatorias
86 predicciones = network.predict(dato)  # Predicciones
87 predicciones_clases = predicciones.argmax(axis=1)  # Clases predichas
88
89 y_pred = network.predict(X_test)  # Predicciones en conjunto de prueba
90 y_pred_classes = y_pred.argmax(axis=1)  # Clases predichas
91 y_test_classes = y_test.values.argmax(axis=1)  # Clases reales
92
93 cm = confusion_matrix(y_test_classes, y_pred_classes)  # Matriz de confusion
94
95 plt.figure(figsize=(12, 6))  # Figura para la matriz de confusion
96 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=["No Error", "
       Obstruction", "severe_collision", "mild_collision"], yticklabels=["No Error", "Obstruction"
       , "severe_collision", "mild_collision"])  # Visualizacion de la matriz de confusion
97 plt.xlabel('Predecido')  # Etiqueta del eje X
98 plt.ylabel('Verdadero')  # Etiqueta del eje Y
99 plt.title('Matriz de confusion')  # Titulo
100 plt.show()  # Mostrar grafico
```

VII-E. Anexo 5: Código para el estudio de sensibilidad (Agregar al final del código del modelo neuronal)

```
    # Estudio de Sensibilidad "Ceteris Paribus"
percentage_variations = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]  # Variaciones porcentuales
sensitivity_results = {}

# Generar nuevas predicciones variando cada caracteristica
for i, column in enumerate(X.columns):
    original_value = X_normalized.iloc[0, i]  # Valor original para la primera instancia
    sensitivity_results[column] = {}

    for variation in percentage_variations:
        # Crear variaciones superiores
        upper_variation = original_value * (1 + variation)

        # Copiar la instancia original y aplicar variaciones
        upper_instance = X_normalized.iloc[0].copy()

        # Aplicar variaciones
        upper_instance[i] = upper_variation

        # Realizar predicciones para las instancias variaciones
        y_upper_pred = network.predict(np.array(upper_instance).reshape(1, -1))  # Obtener
            probabilidades

        # Almacenar los resultados (almacenamos las probabilidades de la clase)
        sensitivity_results[column][f'{variation*100}%'] = y_upper_pred[0]  # Obtener las
            probabilidades

# Visualizacion de los resultados
for feature, predictions in sensitivity_results.items():
    plt.figure(figsize=(12, 6))  # Crear una figura para cada feature

    # Convertir las probabilidades en un formato adecuado para la grafica
    for class_index in range(y_upper_pred.shape[1]):  # Iterar sobre cada clase
        class_probabilities = [pred[class_index] for pred in predictions.values()]
        plt.plot(predictions.keys(), class_probabilities, marker='o', linestyle='-', label=f'
            Clase {class_index}')

    plt.title(f'Sensibilidad{feature}(Probabilidades)')
    plt.xlabel('Variacion')
    plt.ylabel('Probabilidad')
    plt.ylim(0, 1)  # Limitar el eje Y entre 0 y 1 para representar las probabilidades
    plt.legend(title="Clases")  # Agregar una leyenda para las clases

plt.show()  # Mostrar la figura
```

REFERENCIAS

[1] J. Brownlee, ”Neural Networks are Function Approximation Algorithms”, Machine Learning Mastery, 2020. [En línea]. Disponible en: https://machinelearningmastery.com/neural-networks-are-function-approximators/.

[2] A. Engelbrecht, Computational Intelligence: An Introduction, 1st ed. Wiley, 2002.

[3] L. Valesco, .ºptimizadores en redes neuronales profundas: un enfoque práctico”, Medium, 2020. [En línea]. Disponible en: https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr

[4] J. Loy, ”How to build your own Neural Network from scratch in Python”, Medium, 2018. [En línea]. Disponible en: https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6.