TEC | Tecnológico de Costa Rica

Área Académica de Ingeniería Mecatrónica

MT 8008 – Inteligencia Artificial

Tarea: Redes Neuronales Densas

Elaborado por:

Diana Cerdas Vargas
2020023718

Jeaustin Calderón Quesada
2020027413

Juan Luis Crespo Mariño, Dr. Ing.

Setiembre 2023

Contenidos

Introducción y exposición del problema general

Una neurona artificial emula la estructura y el funcionamiento de una neurona biológica. Cada neurona artificial recibe y procesa señales del entorno o de otras neuronas artificiales. Cuando se activa, emite una señal a todas las conectadas. La activación y la intensidad de la señal de salida se rigen por una función denominada función de activación. La neurona artificial calcula una señal de entrada neta a partir de los pesos asociados a las conexiones y usa esta señal como entrada para la función de activación, la cual determina la salida [1]. .

Una estructura compuesta de neuronas artificiales en capas define lo que conocemos como una red neuronal artificial. Estas redes pueden incluir una capa de entrada, capas ocultas y una capa de salida. Las neuronas artificiales dentro de una capa están interconectadas, total o parcialmente, con las de la capa siguiente. Un tipo de red neuronal artificial es el perceptrón multicapa, cuya principal característica es que todas las neuronas de una capa se conectan con cada una de las neuronas de la siguiente capa [1].

Para el problema se brindan datos de fuerza y torsión medidos en un robot para los tres ejes clásicos y con un intervalo de tiempo definido, a partir de esto se indica el tipo de error que aconteció o si hay comportamiento normal. Los requerimientos de solución se dividen en dos partes, la primera corresponde a detectar a partir de las fuerzas y torques si existe un error o no. La segunda parte solicita detectar si hay o no error, y en caso de que haya se debe clasificar el tipo de error.

Estrategia para la solución de la tarea

Se van a utilizar los siguientes pasos predefinidos [2] para completar exitosamente el desarrollo de una red neuronal.

Preprocesado: corresponde a la primera etapa del proceso y se refiere a manipular los datos propios del problema. Al finalizar esta etapa es necesario que los datos estén en un formato que sea compatible con la implementación de la red neuronal, es importante verificar que el conjunto de datos sea significativo (en este caso se asume que así es). Además, los datos se dividen de manera que se tengan datos para entrenamiento, prueba y validación, por lo general se toma un 70%, 20% y 10% del conjunto de datos original, respectivamente. Asimismo, se revisa que no existan datos incompletos o atípicos que entorpezcan alguno de los procesos.

Aprendizaje: consiste en plantear un modelo para la red neuronal artificial, este modelo debe tomar en cuenta la naturaleza del problema y lo que se espera como resultado. Es necesario también escoger hiperparámetros que permitan el desarrollo más certero y eficiente posible para la red, para esto se realiza un estudio de hiperparámetros. Durante el aprendizaje se utiliza una función de pérdida que le permite al sistema ir realizando las correcciones correspondientes, esta es exclusiva al tipo de problema que se tenga.

Evaluación: se toma el conjunto de datos destinado para este fin y se pone al modelo escogido a predecir, tomando entonces los resultados obtenidos y mediante herramientas como la matriz de confusión (caso de clasificación) es posible evaluar el sistema.

Predicción y revisión: con el modelo de la red ya definido y evaluado, es necesario utilizar otro grupo de datos que no haya sido expuesto al modelo anteriormente, esto para poder verificar si es necesario plantear un reentreno.

Problema de clasificación binaria de falla de un robot

En esta sección se brindan una serie de datos que corresponden a fuerzas y torsiones que experimenta un robot antes de fallar o bajo condiciones donde el equipo está trabajando normalmente. Estos datos están etiquetados de acuerdo con el tipo de falla que presentó el robot o bien con la etiqueta de normal, si no hubo ningún problema. A partir de esto, se requiere diseñar una red neuronal densa que sea capaz de clasificar los tipos de falla que puede presentar el robot. Además, es necesario realizar diferentes pruebas para evaluar el desempeño del modelo obtenido.

A continuación, se hace referencia a recursos que se consultaron para la implementación de la red neuronal artificial.

- Los datos con los que se trabajaron están disponibles en [3].
- El preprocesado de los datos se realizó por medio de la librería de numpy y pandas [4][5].
- El diseño del modelo de entrenamiento de la red neuronal, la normalización de los datos, la visualización de las curvas de pérdida y la predicción de resultados se hizo con ayuda de la guía 1 desarrollada en clase [6].
- Para la creación de la matriz de confusión se tomó como referencia la guía de uso de scikit-learn y seaborn [7][8].
- El reporte de clasificación se realizó con la documentación de scikit-learn [9]

Preparación de los datos

Los datos brindados corresponden a 5 archivos con extensión .DATA, estos se componen por una serie de experimentos en los que se miden seis parámetros diferente, estos son mediciones de torque y fuerza en cada uno de los ejes ortogonales. Estos parámetros se miden 15 veces con intervalos de tiempo regulados entre ellos, esto entonces corresponde a 90 datos de entrada por cada uno de los resultados. Es necesario asignar un valor numérico al resultado, como en este caso se desea únicamente definir si hay error o no en el robot, se asigna un 1 al caso en que hay error y 0.001 en el caso que hay funcionamiento normal. En la tabla 1 se muestra cómo se organizaron los datos para usarlos como entrada en la red neuronal artificial.

Tabla 1. Tabla con los datos de entrada etiquetados y organizados

	Resultado	Fx1	Fy1	Fz1	Tx1	...	Fy15	Fz15	Tx15	Ty15	Tz15
0	0.001	0.576310	0.278689	0.940694	0.393486	...	0.349862	0.999380	0.601208	0.580036	0.396648
1	0.001	0.576310	0.278689	0.940694	0.394366	...	0.347107	0.996278	0.600201	0.577338	0.396648
2	0.001	0.576310	0.281967	0.936909	0.391725	...	0.349862	0.993176	0.599194	0.578237	0.396648
3	0.001	0.578588	0.278689	0.938170	0.394366	...	0.352617	0.998759	0.595166	0.579137	0.396648
4	0.001	0.578588	0.275410	0.941956	0.392606	...	0.352617	0.995037	0.594159	0.577338	0.391061
..	...	...	...	...	...	...	...	...	...	...	...
458	1.000	0.258799	0.695652	0.898441	0.358176	...	0.588034	0.969508	0.449585	0.699721	0.543210
459	1.000	0.343685	0.660079	0.914027	0.479422	...	0.584615	0.969836	0.453144	0.703911	0.537037
460	1.000	0.244306	0.701581	0.910005	0.449388	...	0.589744	0.971803	0.451957	0.701117	0.537037
461	1.000	0.184265	0.709486	0.907994	0.444939	...	0.591453	0.970492	0.451957	0.694134	0.543210
462	1.000	0.287785	0.648221	0.906486	0.489433	...	0.588034	0.970164	0.451957	0.695531	0.543210

En el anexo 1 se muestra el código utilizado para reacomodar los datos según se necesita. Para esto se utilizaron las librerías numpy y pandas.

Estudio de hiperparámetros

Para el desarrollo del estudio de hiperparámetros, se realizaron pruebas con 48 posibles combinaciones, en las que se estudió el comportamiento de dos optimizadores distintos, cuatro posibles cantidades de neuronas en las capas ocultas, dos posibilidades de número de capas ocultas y tres tasas de aprendizaje distintas. Cabe destacar que para todos los casos se utilizó un valor de 1500 iteraciones. En el En la tabla 2 se muestran los resultados de pérdidas al probar cada una de estas posibilidades.

Tabla 2. Valores de pérdida para las distintas combinaciones de hiperparámetros.

		Número de capas											
		1						2					
		Tasa de aprendizaje											
		0.01		0.03		0.001		0.01		0.03		0.001	
Optimizador	Cantidad neuronas	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
ADAM	20	0.3408	0.3769	0.3761	0.5031	0.3533	0.4916	0.1554	0.2176	0.1457	0.2547	0.1754	0.2894
	40	0.2492	0.2965	0.2698	0.3097	0.2823	0.3666	0.1552	0.8782	0.0580	0.4514	0.1835	0.4189
	60	0.2288	0.4102	0.2669	0.4106	0.2393	0.3579	0.1260	0.2680	0.0945	0.2164	0.0906	0.1703
	80	0.1975	0.3053	0.2228	0.2750	0.2500	0.2733	0.1853	0.5530	0.1659	0.3595	0.0973	0.3810
DG	20	0.5419	0.5691	0.5552	0.5771	0.5269	0.5958	0.5479	0.4908	0.5255	0.5134	0.5217	0.6219
	40	0.5588	0.4906	0.5368	0.5604	0.5482	0.5218	0.4905	0.5492	0.5219	0.5970	0.5342	0.5776
	60	0.5070	0.5805	0.5073	0.6224	0.5270	0.6000	0.5111	0.4941	0.5041	0.5504	0.4977	0.5874
	80	0.5435	0.4882	0.5137	0.5662	0.5369	0.5313	0.5135	0.4912	0.5169	0.5690	0.5170	0.5051

Al observar los resultados, es posible realizar algunas observaciones basadas en las generalidades de algunos casos. Uno de los resultados más obvios y esperados es que al utilizar el optimizador con estimación adaptativa del momentum (ADAM) obtuvo resultados con valores de pérdida mucho menores que el descenso por gradiente sencillo (SGD). Por otro lado, se nota una leve mejoría al utilizar dos capas ocultas en lugar de una sola. Para tomar una decisión sobre los hiperparámetros exactos que se van a utilizar, se van a escoger los casos con los que se obtuvo una menor pérdida tanto en el entrenamiento como en la prueba y se van a analizar de manera individual, tomando en cuenta otros indicadores importantes, tales como la matriz de confusión, la precisión y el *recall*.

- **Primer caso**

En la tabla 3 se muestran los hiperparámetros utilizados en el primer caso.

Tabla 3. Valores de los parámetros de la red para la primera iteración.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.03	20	2	1500

Seguidamente, en la figura 1 se muestran las curvas de pérdida y la matriz de confusión para el caso uno.

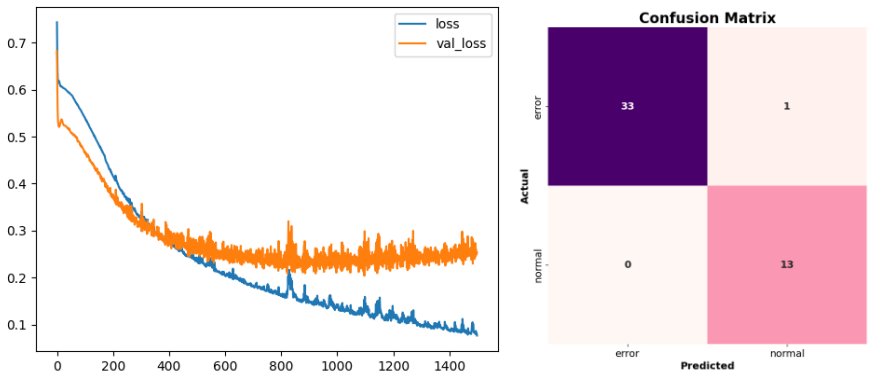

Figura 1. Curvas de pérdida de entrenamiento y prueba de la red para el primer caso de combinación de hiperparámetros, así como la matriz de confusión obtenida.

Los valores de pérdida al correr nuevamente esta combinación de parámetros fueron 0.0766 para el entrenamiento y 0.2536 para las pruebas. Finalmente, en la tabla 4 se muestran los resultados de clasificación.

Tabla 4. Resultados de clasificación para el primer caso.

```
-----------------------------------------------------
Classification report:

              precision    recall  f1-score   support

       error     1.0000    0.9706    0.9851        34
      normal     0.9286    1.0000    0.9630        13

    accuracy                         0.9787        47
   macro avg     0.9643    0.9853    0.9740        47
weighted avg     0.9802    0.9787    0.9790        47

-----------------------------------------------------
```

- **Segundo caso**

En la tabla 5 se muestran los hiperparámetros utilizados en el segundo caso.

Tabla 5. Valores de los parámetros de la red para la primera iteración.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.001	20	2	1500

Seguidamente, en la figura 2 se muestran las curvas de pérdida y la matriz de confusión para el caso dos.

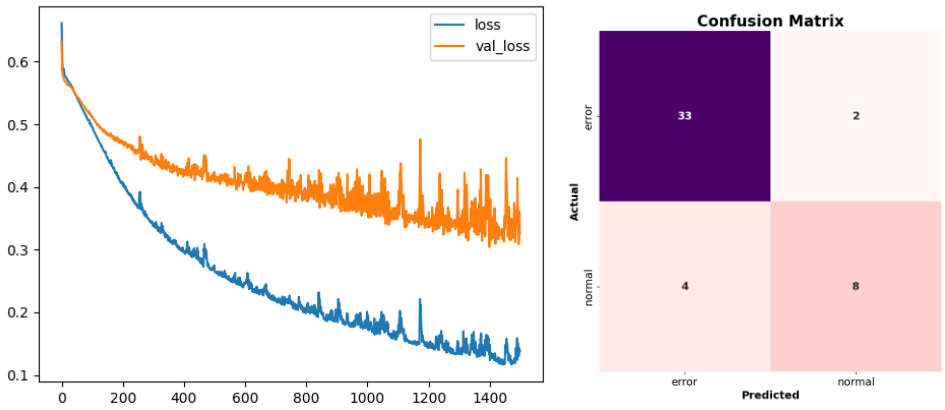

Figura 2. Curvas de pérdida de entrenamiento y prueba de la red para el segundo caso de combinación de hiperparámetros, así como la matriz de confusión obtenida.

Los valores de pérdida al correr nuevamente esta combinación de parámetros fueron 0.1378 para el entrenamiento y 0.3251 para las pruebas. Finalmente, en la tabla 6 se muestran los resultados de clasificación.

Tabla 6. Resultados de clasificación para el segundo caso.

```
------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

       error     0.8919    0.9429    0.9167        35
      normal     0.8000    0.6667    0.7273        12

    accuracy                         0.8723        47
   macro avg     0.8459    0.8048    0.8220        47
weighted avg     0.8684    0.8723    0.8683        47

------------------------------------------------------
```

- **<u>Tercer caso</u>**

En la tabla 7 se muestran los hiperparámetros utilizados en el tercer caso.

Tabla 7. Valores de los parámetros de la red para la primera iteración.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.03	60	2	1500

Seguidamente, en la figura 3 se muestran las curvas de pérdida y la matriz de confusión para el caso tres.

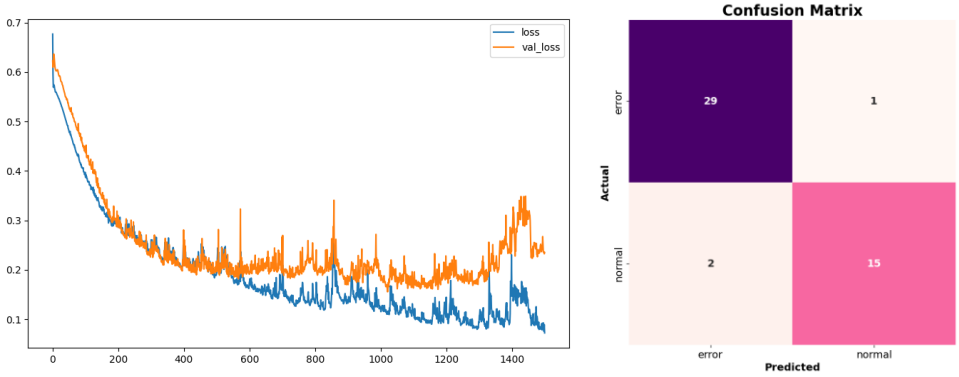

Figura 3. Curvas de pérdida de entrenamiento y prueba de la red para el tercer caso de combinación de hiperparámetros, así como la matriz de confusión obtenida.

Los valores de pérdida al correr nuevamente esta combinación de parámetros fueron 0.0725 para el entrenamiento y 0.2352 para las pruebas. Finalmente, en la tabla 8 se muestran los resultados de clasificación.

Tabla 8. Resultados de clasificación para el tercer caso.

```
------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

       error     0.9355    0.9667    0.9508        30
      normal     0.9375    0.8824    0.9091        17

    accuracy                         0.9362        47
   macro avg     0.9365    0.9245    0.9300        47
weighted avg     0.9362    0.9362    0.9357        47

------------------------------------------------------
```

- **Cuarto caso**

En la tabla 9 se muestran los hiperparámetros utilizados en el cuarto caso.

Tabla 9. Valores de los parámetros de la red para la primera iteración.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.001	60	2	1500

Seguidamente, en la figura 4 se muestran las curvas de pérdida y la matriz de confusión para el caso uno.

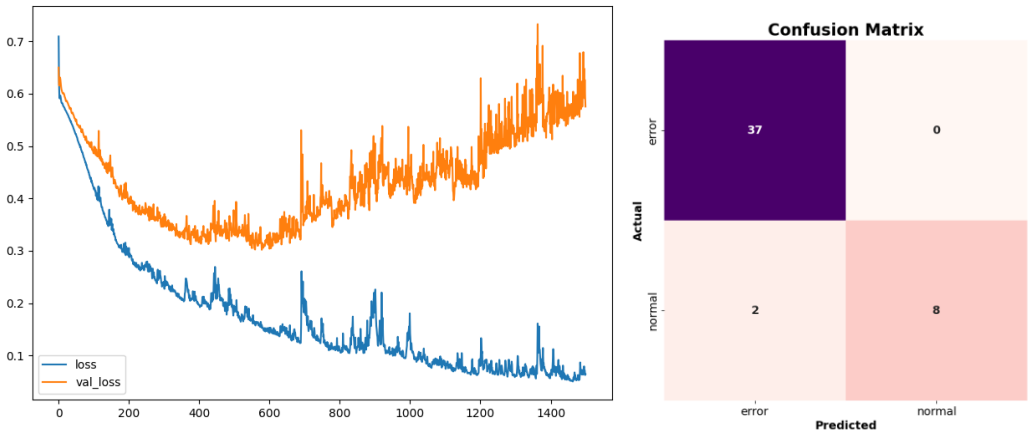

Figura 4. Curvas de pérdida de entrenamiento y prueba de la red para el cuarto caso de combinación de hiperparámetros, así como la matriz de confusión obtenida.

Los valores de pérdida al correr nuevamente esta combinación de parámetros fueron 0.0644 para el entrenamiento y 0.5756 para las pruebas. Finalmente, en la tabla 10 se muestran los resultados de clasificación.

Tabla 10. Resultados de clasificación para el cuarto caso.

```
------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

       error     0.9487    1.0000    0.9737        37
      normal     1.0000    0.8000    0.8889        10

    accuracy                         0.9574        47
   macro avg     0.9744    0.9000    0.9313        47
weighted avg     0.9596    0.9547    0.9556        47

------------------------------------------------------
```

Corrigiendo a 1100 iteraciones para evitar el over-fitting:

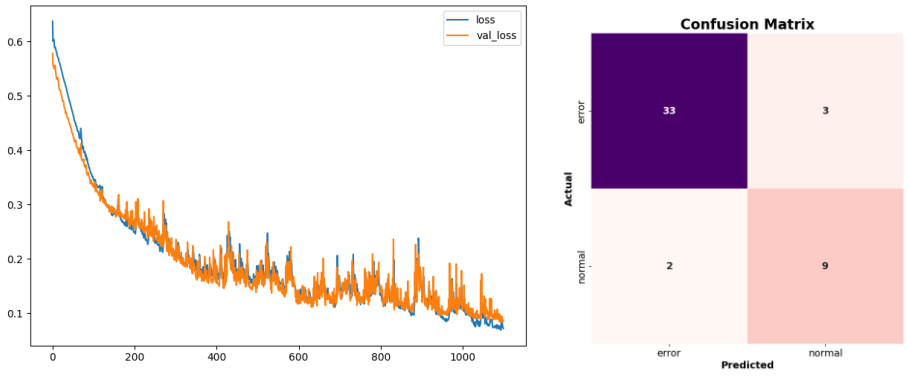

Figura 5. Curvas de pérdida de entrenamiento y prueba de la red para el cuarto caso de combinación de hiperparámetros, con menos iteraciones, así como la matriz de confusión obtenida.

Los valores de pérdida al correr nuevamente esta combinación de parámetros , con menos iteraciones, fueron 0.0722 para el entrenamiento y 0. 0849 para las pruebas. Finalmente, en la tabla 11 se muestran los resultados de clasificación.

Tabla 11. Resultados de clasificación para el cuarto caso con sus iteraciones variadas.

```
------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

       error     0.9429    0.9167    0.9296        36
      normal     0.7500    0.8182    0.7826        11

    accuracy                         0.8936        47
   macro avg     0.8464    0.8674    0.8561        47
weighted avg     0.8977    0.8936    0.8952        47

------------------------------------------------------
```

Análisis de la red neuronal obtenida

Se presentaron decisiones que se tomaron sin necesidad de iterar, esto como resultado de experiencia de otras personas que se encuentra documentada y respaldo teórico del funcionamiento de las redes neuronales. Dos parámetros como los definidos anteriormente son las funciones de activación y la función de pérdida. La función de activación escogida fue la sigmoide, esto porque el problema consiste en una clasificación binaria [10]. Finalmente, la función de pérdida escogida fue la de entropía cruzada binaria, esto porque es de las pocas funciones de clasificación binaria disponibles en la librería escogida [11].

Para el caso 1 se obtuvieron resultados considerablemente buenos , de los 47 casos analizados únicamente se obtuvo un falso negativo en el que se predijo un funcionamiento normal en lugar de un error. De las curvas se puede destacar un comportamiento no ideal en el que la curva de prueba empieza a levantarse antes de que la curva de entrenamiento llegue a su asíntota. Los valores de pérdida son aceptables, tanto el de entrenamiento como el de prueba. Por otro lado, del reporte de clasificación se puede resaltar el *accuracy* en el que se obtuvo un valor de 0.9787, valor que representa que tan certero es el modelo. Se sabe que el *f1-score* es un valor que permite condensar la precisión y el *recall* en uno solo y en este caso se encuentran valores considerablemente altos que se tomarán en cuenta.

Seguidamente, para el segundo caso los resultados son menos acertados. Se tiene un valor de *accuracy* igual a 0.8723, sin embargo, al analizar el reporte de clasificación se puede observar que el sistema se ve sesgado especialmente con falsos negativos, es decir, predice errores cuando no es así. Las curvas de pérdidas se ven como se podría esperar teóricamente, con la curva de pérdida de prueba se encuentra siempre por encima de la curva de pérdida de aprendizaje. Los valores finales de pérdida son relativamente bajos, pero no tan bajos como en el caso uno. Al analizar los valores del *f1-score* se puede observar que para la categoría “error” es considerablemente más alto que para la categoría “normal”, es decir, como se mencionó anteriormente, existe un sesgo en el que se tiende a fallar las categorías que deberían corresponder a normal.

Para el tercer caso se encuentran resultados que se podrían tomar como intermedios con respecto a los dos casos anteriores. Se tiene un *accuracy* menor al caso uno, pero mayor al caso dos. Únicamente se dieron 3 errores en toda la predicción y el valor de *f1-score* es considerablemente bueno, lo que nos indica que tanto para “error” o “normal” se tiene un comportamiento dentro de lo esperado y sin sesgos o al menos con sesgos despreciables. Las

curvas de pérdidas, de igual manera, son esperadas y lo único resaltable es que se pueden notar una curva llena de saltos y variaciones abruptas, situación esperada al utilizar una tasa de aprendizaje relativamente alta.

Finalmente, se procede a analizar la segunda parte de la cuarta iteración, en la que se cambió la cantidad de iteraciones. La curva se observa de manera correcta, con una tendencia a estabilizarse conforme llega al final de las iteraciones. Se tuvieron 5 desaciertos en total, sin embargo, respaldado por el reporte de clasificación, se puede observar un sesgo considerable que se puede observar en el *f1-score*. Dicho valor es aceptable para la categoría "error", pero para la categoría "normal" se tiene un valor considerablemente menor. Asimismo, la *accuracy* es de las más bajas de los casos analizados.

En conclusión, al analizar los casos anteriores con sus parámetros, curvas de comportamiento y resultados de validación, se escoge el caso uno como el modelo que mejor predice el resultado ante los conjuntos de datos brindados.

Estudio de sensibilidad de las entradas

En la sección anterior se definió el modelo a utilizar, con esto es posible analizar todo el conjunto de datos y con esto se puede predecir cuantos falsos "errores" o cuantos falsos "normales". Para el modelo propuesto se obtuvieron 11 en total, 8 por falsos "errores" y 3 por falsos "normales". Para realizar el estudio de sensibilidad de las entradas se implementó una estrategia en la que se varía porcentualmente una columna del conjunto de datos y se pone a predecir el modelo con estos datos modificados, con dicha predicción se estima la cantidad de errores totales y se compara con los errores obtenidos con el conjunto de datos original. En la tabla 12 se muestran los resultados de cuál fue la diferencia de aciertos entre el set de datos original y el modificado en cierto porcentaje. En el anexo 3 se muestra el código desarrollado en el lenguaje Python para poder determinar los resultados correspondientes.

Tabla 12. Tabla de diferencia de aciertos entre los datos de entrada originales y los datos con una columna modificada en un cierto porcentaje.

	5%	10%	15%	20%
Original	0.0	NaN	NaN	NaN
Fx1	-3.0	-13.0	-23.0	-38.0
Fy1	-1.0	-19.0	-48.0	-55.0
Fz1	-2.0	-2.0	-11.0	-15.0
Tx1	-1.0	-3.0	-9.0	-24.0
...	...	...	...	...
Fy15	-7.0	-30.0	-53.0	-64.0
Fz15	-2.0	-8.0	-13.0	-20.0
Tx15	-1.0	-1.0	-2.0	-8.0
Ty15	-20.0	-47.0	-64.0	-70.0
Tz15	-1.0	-4.0	-11.0	-22.0

Para interpretar estos datos de mejor manera, se redujo la cantidad de datos a únicamente los cinco más y menos influyentes. En la tabla 13 se condensa esta información.

Tabla 13. Tabla con los datos de entrada etiquetados y organizados

	Variación							
	5%		10%		15%		20%	
	Menos influyente	Más influyente	Menos influyente	Más influyente	Menos influyente	Más influyente	Menos influyente	Más influyente
1	Fx13	Ty14	Tx3	Ty14	Tx3	Ty5	Fx9	Ty14
2	Tz6	Ty5	Tx4	Ty5	Tx4	Ty14	Tx3	Ty5
3	Fx11	Ty4	Fx8	Fx15	Tz5	Fy8	Tz5	Fy8
4	Tz4	Ty15	Fx9	Ty4	Fx9	Ty2	Fx11	Fy6
5	Fz6	Ty2	Tz9	Ty2	Fx11	Ty4	Tx4	Fx15

Con la información condensada es más fácil sacar conclusiones. Para esto, se puede observar cual parámetro se repite en varias de las columnas como más influyente o menos influyente. Al analizar la tabla 13 podemos sacar varias conclusiones:

- El torque y la fuerza en el eje y corresponden a parámetros de suma relevancia, esto porque de en 18 de los 20 parámetros más relevantes están ocupados por componentes que se ejercen a lo largo del eje mencionado, además, no se presentan en ninguna de las posiciones de parámetros menos influyentes.

- La fuerza Fx15 corresponde a una excepción en la que una fuerza ejercida en el eje x tiene gran relevancia con respecto al resultado de la predicción de la red neuronal.
- Los parámetros de más relevancia corresponden a Ty14 y Ty5, estos se encuentran en los dos puestos de más relevancia para todas las variaciones realizadas.
- Las entradas Tx3, Tx4 y Fx11 corresponden a los tres parámetros con menor incidencia en el comportamiento del robot, esto porque se repiten en los puestos de menor incidencia al realizar variaciones porcentuales a sus valores.

Problema de clasificación del tipo de falla de un robot

En esta sección se brindan una serie de datos que corresponden a fuerzas y torsiones que experimenta un robot antes de fallar o bajo condiciones donde el equipo está trabajando normalmente. Estos datos están etiquetados de acuerdo con el tipo de falla que presentó el robot o bien con la etiqueta de normal, si no hubo ningún problema. A partir de esto, se requiere diseñar una red neuronal densa que sea capaz de clasificar los tipos de falla que puede presentar el robot. Además, es necesario realizar diferentes pruebas para evaluar el desempeño del modelo obtenido.

Para la solución de este problema se utilizaron las siguientes referencias:

- Los datos con los que se trabajaron están disponibles en [3].
- El preprocesado de los datos de prueba y entrenamiento se realizó por medio de la librería de pandas [4][5].
- El diseño del modelo de entrenamiento de la red neuronal, la normalización de los datos, la visualización de las curvas de pérdida y la predicción de resultados se hizo con ayuda de la librería tensorFlow, tomando como base la guía [6].
- Para la creación de la matriz de confusión y el reporte de clasificación se tomó como referencia [7][8] respectivamente.

A continuación, se explican unos conceptos que se utilizaron para la validación de la red, las cuales fueron tomadas de [12]:

La precisión es una medida de la capacidad del modelo para clasificar todos los datos correctamente. Se define como la cantidad de verdaderos positivos entre el número total de muestras analizadas.

La exhaustividad (recall) es la capacidad del modelo para identificar todas las muestras pertenecientes a la clase de interés. Se define como el cociente entre el número de verdaderos positivos y la cantidad total de elementos de la clase de interés que hay en toda la población.

El puntaje de F1, también conocido como F-balanceado, se puede interpretar como la media armónica de la precisión y la exhaustividad. Un puntaje de F1 igual a 1 indica una clasificación totalmente acertada, mientras que un puntaje de cero significa lo contrario.

Preparación de los datos

De manera simular al problema de clasificación anterior, los datos que se deben utilizar se componen por una serie torques y fuerzas en cada uno de los ejes ortogonales, que al final resultan en un total de 90 datos de entrada por cada uno de los resultados. En este caso se desea hacer una clasificación del tipo de falla que presenta el sistema, y como cada set presenta un bajo número de datos, fue necesario combinar los documentos Lp1.data y Lp4.data. No obstante, aún habían categorías que estaban mal representadas, como la de colisión frontal, que apenas tenía un total de 16 datos. Para solucionar esto, se duplicaron manualmente todos los datos de dicha categoría, de modo que al final se tenían un total de 36 datos. En la tabla 14 se muestra cómo se organizaron los datos para usarlos como entrada en la red neuronal artificial.

Tabla 14. Tabla con los datos de entrada etiquetados y organizados

	Resultado	Fx1	Fy1	Fz1	Tx1	Ty1	Tz1	Fx2	...	Ty14	Tz14	Fx15	Fy15	Fz15	Tx15	Ty15	Tz15
0	1.0	0.576310	0.278689	0.940694	0.393486	0.312500	0.582979	0.554054	...	0.562852	0.384146	0.430233	0.349862	0.999380	0.601208	0.580036	0.396648
1	1.0	0.576310	0.278689	0.940694	0.394366	0.312500	0.582979	0.551802	...	0.560038	0.378049	0.430233	0.347107	0.996278	0.600201	0.577338	0.396648
2	1.0	0.576310	0.281967	0.936909	0.391725	0.309722	0.582979	0.554054	...	0.560976	0.378049	0.430233	0.349862	0.993176	0.599194	0.578237	0.396648
3	1.0	0.578588	0.278689	0.938170	0.394366	0.312500	0.578723	0.554054	...	0.562852	0.384146	0.430233	0.352617	0.998759	0.595166	0.579137	0.396648
4	1.0	0.578588	0.275410	0.941956	0.392606	0.311111	0.582979	0.551802	...	0.559099	0.378049	0.430233	0.352617	0.995037	0.594159	0.577338	0.391061
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
216	3.0	0.236674	0.721311	0.955615	0.116386	0.472648	0.386047	0.253829	...	0.675386	0.777778	0.609337	0.588034	0.987972	0.449585	0.699721	0.543210
217	3.0	0.324094	0.684426	0.972193	0.283308	0.695842	0.297674	0.317287	...	0.678954	0.777778	0.613022	0.584615	0.988306	0.453144	0.703911	0.537037
218	3.0	0.221748	0.727459	0.967914	0.241960	0.557987	0.288372	0.236324	...	0.677765	0.774603	0.611794	0.589744	0.990311	0.451957	0.701117	0.537037
219	3.0	0.159915	0.735656	0.965775	0.235835	0.450766	0.288372	0.192560	...	0.670630	0.777778	0.608108	0.591453	0.988974	0.451957	0.694134	0.543210
220	3.0	0.266525	0.672131	0.964171	0.297090	0.595186	0.288372	0.273523	...	0.675386	0.777778	0.609337	0.588034	0.988640	0.451957	0.695531	0.543210

En el anexo 4 se muestra el código utilizado para reacomodar los datos según se necesita. Para esto se utilizaron las librerías numpy y pandas.

Estudio de hiperparámetros

En este caso, al tratarse de un problema de clasificación de cuatro categorías, es decir, cuenta con cuatro neuronas de salida, se realizó un estudio con dos posibles optimizadores, con tres valores de tasa de aprendizaje, tres valores diferentes de neuronas en la capa oculta, dos números diferentes de capas ocultas, para un total de 36 iteraciones. Los resultados de estas iteraciones condensaron en una tabla, donde además del valor de pérdida de entrenamiento y la pérdida de prueba, se agregó el valor del parámetro recall, que como se mencionó anteriormente, brinda en forma de porcentaje cuántos verdaderos positivos hubo en la predicción red.

Tabla 15. Valores de pérdida de entrenamiento, pérdida de prueba y recall para diferentes combinaciones de hiperparámetros.

			Optimizador					
			ADAM			DG		
			Neuronas					
Número de capas ocultas	Tasa de aprendizaje	Valores	5	15	25	5	15	25
1	0,001 (1700 iteraciones)	Pérdida entrenamiento	0,6167	0,5405	0,5028	1,3109	1,2793	1,2765
		Pérdida prueba	0,7712	0,7131	0,6549	1,3374	1,2229	1,2529
		Recall	0,8333	0,9583	1	0,25	0,25	0,25
	0,01 (1000 iteraciones)	Pérdida entrenamiento	0,4102	0,3011	0,2772	1,3194	0,9587	0,954
		Pérdida prueba	0,5491	0,5743	0,6971	1,2605	1,0205	1,0298
		Recall	0,7143	1	0,9167	0,25	0,25	0,4083
	0,03 (500 iteraciones)	Pérdida entrenamiento	0,4261	0,3573	0,3772	0,9298	0,9104	0,8144
		Pérdida prueba	0,798	0,518	0,6125	0,9448	0,8015	1,0156
		Recall	0,75	0,9062	0,7083	0,625	0,7	0,6
2	0,001 (1700 iteraciones)	Pérdida entrenamiento	0,6043	0,4698	0,4527	1,3259	1,3132	1,3059
		Pérdida prueba	0,6916	0,5868	0,5996	1,3305	1,3173	1,3088
		Recall	0,7708	0,7708	0,75	0,25	0,25	0,25
	0,01 (1000 iteraciones)	Pérdida entrenamiento	0,3662	0,2916	0,322	1,2969	1,3116	1,2556
		Pérdida prueba	0,5338	0,531	0,6352	1,349	1,2689	1,4212
		Recall	0,6667	1	0,875	0,25	0,25	0,25
	0,03 (500 iteraciones)	Pérdida entrenamiento	0,3908	0,5124	0,3264	1,244	1,1634	1,0868
		Pérdida prueba	0,9015	0,7134	0,5848	1,296	1,1366	1,1464
		Recall	0,8125	0,9643	0,9583	0,25	0,1786	0,4

En la tabla anterior, se señalaron con color amarillo los casos donde se obtuvo la menor pérdida de prueba y un valor del recall cercano a uno. Luego, para cada uno de ellos, se van a mostrar los valores de pérdida, la matriz de confusión y un reporte de clasificación. Esto con el objetivo

de observar de mejor manera el impacto que tiene cada parámetro en la forma en que clasifica la red generada y para poder definir cuál es la mejor combinación de parámetros.

- **Primer caso**

Tabla 16. Valores de los parámetros de la red para el primer caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.001	15	1	1700

Resultados de la red

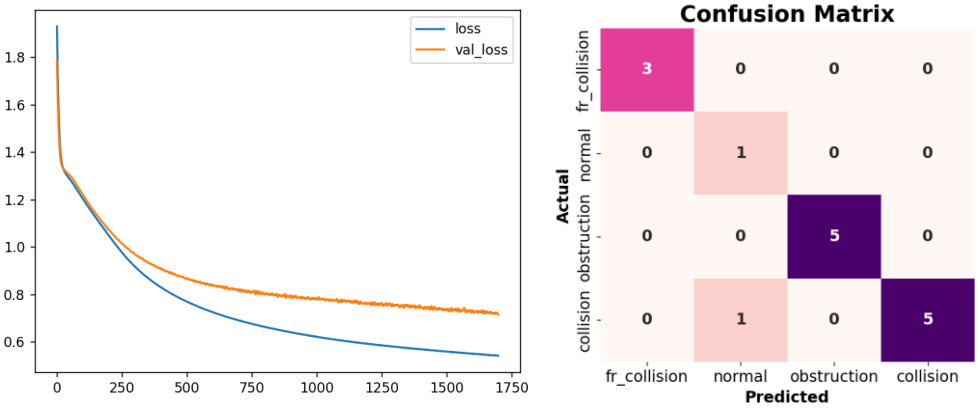

Figura 6. Curvas de pérdida de entrenamiento y prueba de la red para el primer caso, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.5405

Pérdida de prueba: 0.7131

Tabla 17. Resultados de clasificación para el primer caso.

```
--------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     1.0000    1.0000    1.0000         3
      normal     0.5000    1.0000    0.6667         1
 obstruction     1.0000    1.0000    1.0000         5
   collision     1.0000    0.8333    0.9091         6

    accuracy                         0.9333        15
   macro avg     0.8750    0.9583    0.8939        15
weighted avg     0.9667    0.9333    0.9414        15

--------------------------------------------------------
```

- **Segundo caso**

Tabla 18. Valores de los parámetros de la red para el segundo caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.001	25	1	1700

Resultados de la red

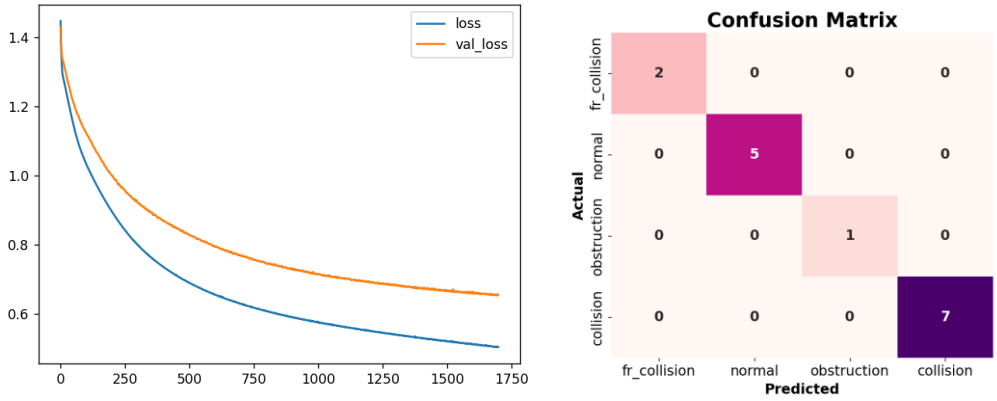

Figura 7. Curvas de pérdida de entrenamiento y prueba de la red para el segundo caso, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.5028

Pérdida de prueba: 0.6549

Tabla 19. Resultados de clasificación para el segundo caso.

```
--------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     1.0000    1.0000    1.0000         2
      normal     1.0000    1.0000    1.0000         5
 obstruction     1.0000    1.0000    1.0000         1
   collision     1.0000    1.0000    1.0000         7

    accuracy                         1.0000        15
   macro avg     1.0000    1.0000    1.0000        15
weighted avg     1.0000    1.0000    1.0000        15

--------------------------------------------------------
```

- **Tercer caso**

Tabla 20. Valores de los parámetros de la red para tercer caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.01	5	1	1000

Resultados de la red

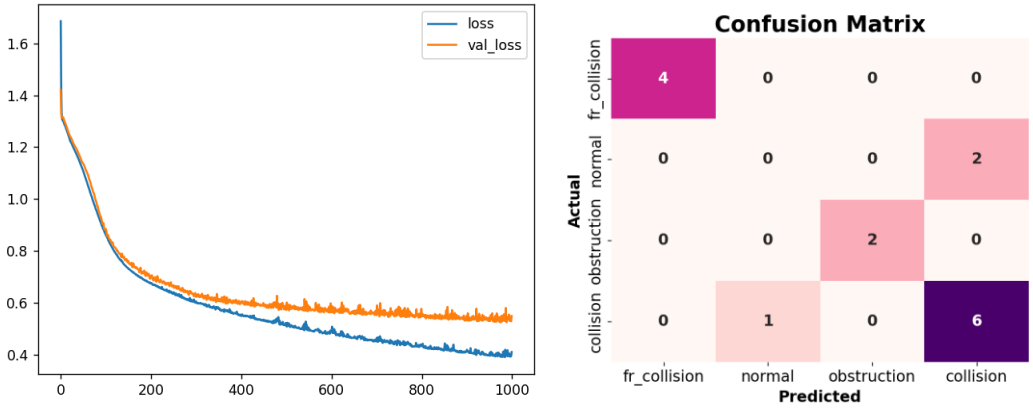

Figura 8. Curvas de pérdida de entrenamiento y prueba de la red para el tercer caso, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.4102

Pérdida de prueba: 0.5491

Tabla 21. Resultados de clasificación para el tercer caso.

```
-----------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     1.0000    1.0000    1.0000         4
      normal     0.0000    0.0000    0.0000         2
 obstruction     1.0000    1.0000    1.0000         2
   collision     0.7500    0.8571    0.8000         7

    accuracy                         0.8000        15
   macro avg     0.6875    0.7143    0.7000        15
weighted avg     0.7500    0.8000    0.7733        15

-----------------------------------------------------
```

- **Cuarto caso**

Tabla 22. Valores de los parámetros de la red el cuarto caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.01	15	1	1000

Resultados de la red

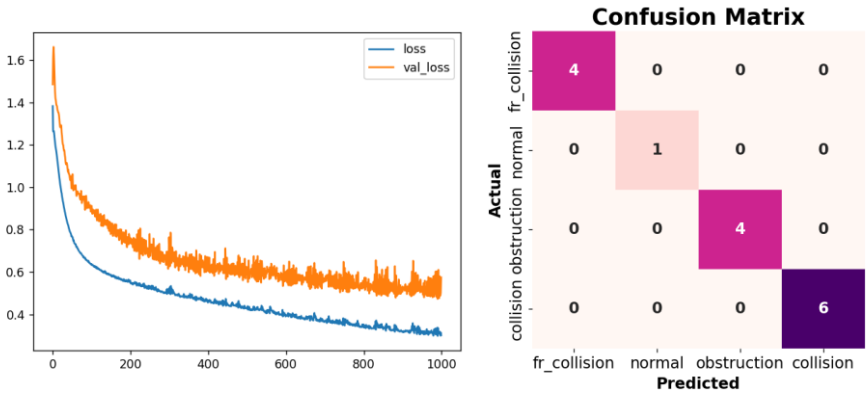

Figura 9. Curvas de pérdida de entrenamiento y prueba de la red para el cuarto caso, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.3011

Pérdida de prueba: 0.5743

Tabla 23. Resultados de clasificación para el cuarto caso.

```
--------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     1.0000    1.0000    1.0000         4
      normal     1.0000    1.0000    1.0000         1
 obstruction     1.0000    1.0000    1.0000         4
   collision     1.0000    1.0000    1.0000         6

    accuracy                         1.0000        15
   macro avg     1.0000    1.0000    1.0000        15
weighted avg     1.0000    1.0000    1.0000        15

--------------------------------------------------------
```

- **Quinto caso**

Tabla 24. Valores de los parámetros de la red para el quinto caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.01	25	1	1000

Resultados de la red

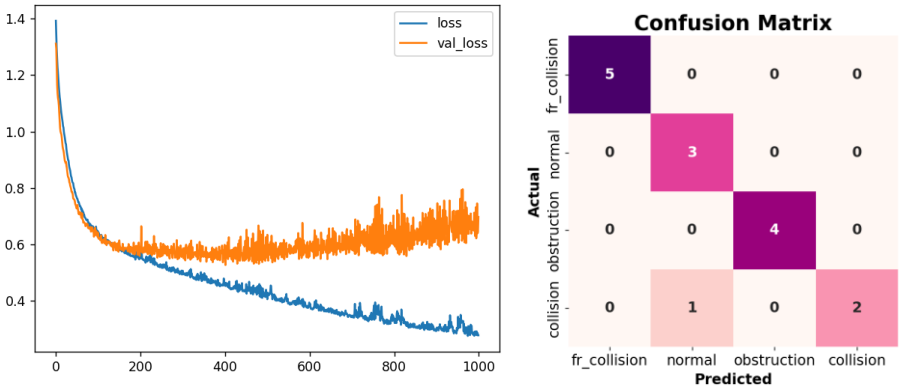

Figura 10. Curvas de pérdida de entrenamiento y prueba de la red para el quinto caso, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.2772

Pérdida de prueba: 0.6971

Tabla 25. Resultados de clasificación para el quinto caso.

```
------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     1.0000    1.0000    1.0000         5
      normal     0.7500    1.0000    0.8571         3
 obstruction     1.0000    1.0000    1.0000         4
   collision     1.0000    0.6667    0.8000         3

    accuracy                         0.9333        15
   macro avg     0.9375    0.9167    0.9143        15
weighted avg     0.9500    0.9333    0.9314        15

------------------------------------------------------
```

- **Sexto caso**

Tabla 26. Valores de los parámetros de la red para el sexto caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.01	15	2	1000

Resultados de la red

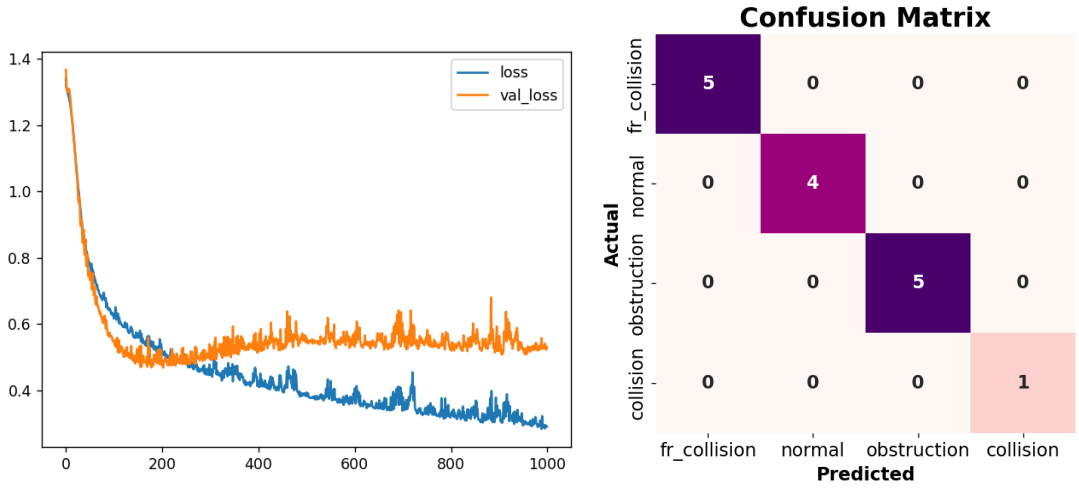

Figura 11. Curvas de pérdida de entrenamiento y prueba de la red para sexto caso, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.2916

Pérdida de prueba: 0.5310

Tabla 27. Resultados de clasificación para el sexto caso.

```
------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     1.0000    1.0000    1.0000         5
      normal     1.0000    1.0000    1.0000         4
 obstruction     1.0000    1.0000    1.0000         5
   collision     1.0000    1.0000    1.0000         1

    accuracy                         1.0000        15
   macro avg     1.0000    1.0000    1.0000        15
weighted avg     1.0000    1.0000    1.0000        15

------------------------------------------------------
```

- **Sétimo caso**

Tabla 28. Valores de los parámetros de la red para el sétimo caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.03	15	1	500

Resultados de la red

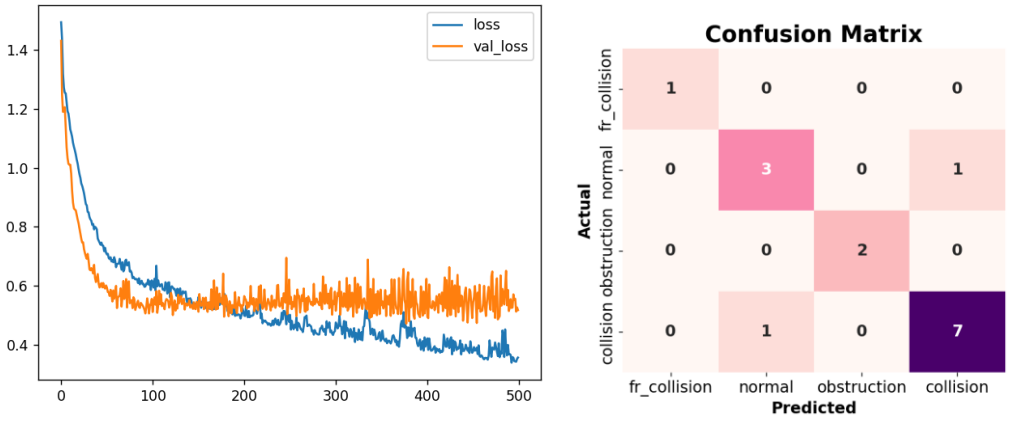

Figura 12. Curvas de pérdida de entrenamiento y prueba de la red para el sétimo caso, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.3573

Pérdida de prueba: 0.5180

Tabla 29. Resultados de clasificación para el sétimo caso.

```
--------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     1.0000    1.0000    1.0000         1
      normal     0.7500    0.7500    0.7500         4
 obstruction     1.0000    1.0000    1.0000         2
   collision     0.8750    0.8750    0.8750         8

    accuracy                         0.8667        15
   macro avg     0.9062    0.9062    0.9062        15
weighted avg     0.8667    0.8667    0.8667        15

--------------------------------------------------------
```

- **Octavo caso**

Tabla 30. Valores de los parámetros de la red para el octavo caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.03	25	1	500

Resultados de la red

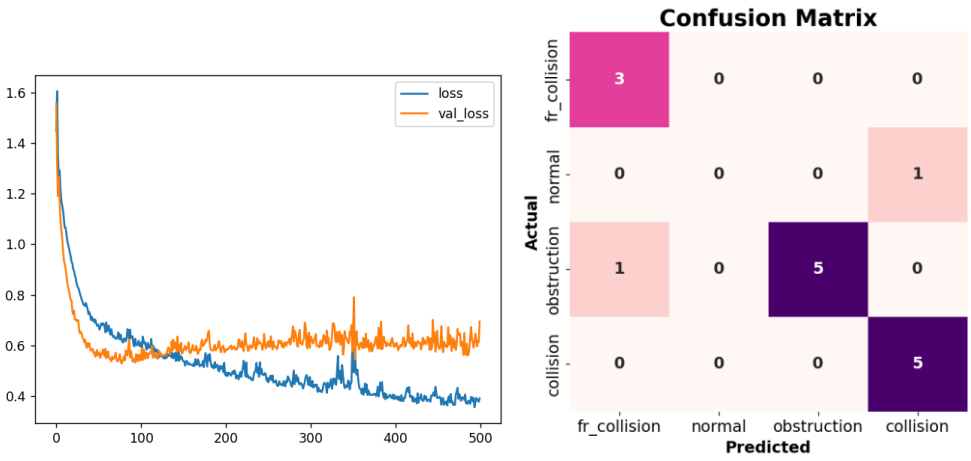

Figura 13. Curvas de pérdida de entrenamiento y prueba de la red para el octavo, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.3772

Pérdida de prueba: 0.6125

Tabla 31. Resultados de clasificación para el octavo caso.

```
------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     0.7500    1.0000    0.8571         3
      normal     0.0000    0.0000    0.0000         1
 obstruction     1.0000    0.8333    0.9091         6
   collision     0.8333    1.0000    0.9091         5

    accuracy                         0.8667        15
   macro avg     0.6458    0.7083    0.6688        15
weighted avg     0.8278    0.8667    0.8381        15

------------------------------------------------------
```

- **Noveno caso**

Tabla 32. Valores de los parámetros de la red para el noveno caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.03	15	2	500

Resultados de la red

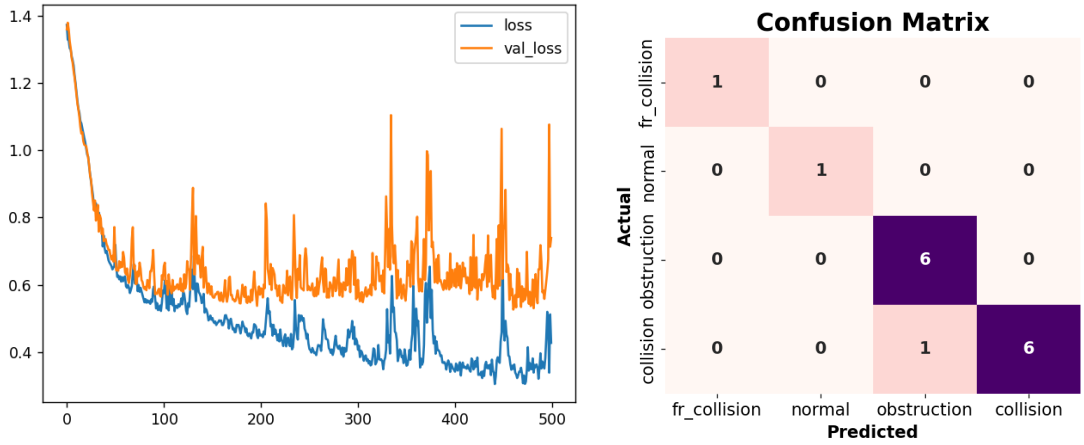

Figura 14. Curvas de pérdida de entrenamiento y prueba de la red para el noveno caso, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.5124

Pérdida de prueba: 0.7134

Tabla 33. Resultados de clasificación para el noveno caso.

```
--------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     1.0000    1.0000    1.0000         1
      normal     1.0000    1.0000    1.0000         1
 obstruction     0.8571    1.0000    0.9231         6
   collision     1.0000    0.8571    0.9231         7

    accuracy                         0.9333        15
   macro avg     0.9643    0.9643    0.9615        15
weighted avg     0.9429    0.9333    0.9333        15

--------------------------------------------------------
```

- **Décimo caso**

Tabla 34. Valores de los parámetros de la red para el décimo caso.

Optimizador	Tasa de aprendizaje	Neuronas en la capa oculta	Número de capas ocultas	Número de iteraciones
ADAM	0.03	25	2	500

Resultados de la red

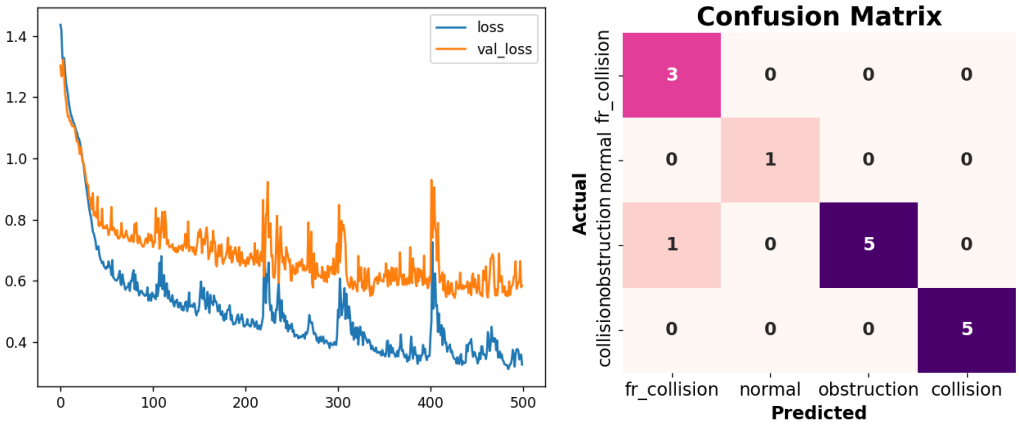

Figura 15. Curvas de pérdida de entrenamiento y prueba de la red para el décimo, así como la matriz de confusión obtenida.

Pérdida de entrenamiento: 0.3264

Pérdida de prueba: 0.5848

Tabla 35. Resultados de clasificación para el décimo caso.

```
-------------------------------------------------------
Classification report:

              precision    recall  f1-score   support

fr_collision     0.7500    1.0000    0.8571         3
      normal     1.0000    1.0000    1.0000         1
 obstruction     1.0000    0.8333    0.9091         6
   collision     1.0000    1.0000    1.0000         5

    accuracy                         0.9333        15
   macro avg     0.9375    0.9583    0.9416        15
weighted avg     0.9500    0.9333    0.9351        15

-------------------------------------------------------
```

En el anexo 5 se muestra el código utilizado para la creación, el entrenamiento y validación de la red

Análisis de la red neuronal obtenida

En esta sección se va a realizar un análisis de la red diseñada, con base en el estudio de hiperparámetros realizado, para poder definir la combinación óptima para el problema que se quiere solucionar. La red generada para la solución está compuesta de 90 neuronas de entrada y por 4 neuronas de salida, dadas por las categorías de clasificación.

Debido a la naturaleza del problema, es necesario tener una función de pérdida que permita la clasificación en múltiples categorías, en este caso, se optó por la función llamada: "CategoricalCrossentropy", la cual de acuerdo con [13] calcula la pérdida de entropía cruzada entre las etiquetas reales y las predichas por el modelo. Entonces, el resultado de la red consiste en una matriz que contiene la probabilidad de que un dato pertenezca a cada una de las etiquetas de las neuronas de entrada.

Ahora, de acuerdo con los resultados de la tabla [15], lo primero que se puede evidenciar es que al utilizar el optimizador por Descenso de Gradiente, la red presenta valores de pérdida elevados, mayormente mayores a 1. La razón de esto, puede ser que al algoritmo le está costando converger, esto tiene sentido, ya que se está trabajando en una red de tipo densa con 90 entradas, lo cual implica que se trabaja en un espacio de 90 dimensiones. Por lo que el número de pesos que se deben calcular no solo es extenso, sino que también se vuelve complejo de obtener en 90 dimensiones. Además, se puede notar que los resultados de la variable "recall" son menores al 0.7, lo cual indica que como máximo lograba predecir solo el 70% de los datos correctamente.

Por otra parte, se puede notar que el optimizador Adam presentó menos problemas con los cálculos de las pérdidas y logró un mayor porcentaje de predicción (variable recall). La razón de esto es que este es un optimizador mucho más robusto comparado con el de descenso de gradiente y está diseñado para trabajar en espacios más complejos. Es por esta razón que se selecciona como mejor opción a este optimizador.

Ahora, en cuanto a los resultados, se puede notar que en todas las iteraciones se obtuvieron resultados con diferencias puntuales. Se puede notar que al incrementar la tasa de aprendizaje se reduce significativamente el número de iteraciones necesarias para el entrenamiento, en incluso en algunos casos de los que se profundizaron (específicamente del quinto al décimo), se tienen problemas de sobreentrenamiento, aunque se redujo el número de iteraciones a menos de la mitad del valor inicial. No obstante, se puede notar también que en estos mismos casos la curva de pérdida de prueba presenta muchos saltos, lo cual implica que a la red le está costando

converger. No obstante, a pesar de estos problemas, se puede notar que la red logra porcentajes de predicción bastante altos, mayores al 90%, por lo que se puede concluir que resuelven bastante bien el problema.

Cabe resaltar que con esta red muchos de los errores de predicción pueden estar relacionados con la falta de datos en ciertas categorías, como, por ejemplo, la de colisión frontal, que solamente contaba con 16 valores.

Por último, a pesar de que se tengan múltiples resultados satisfactorios, se puede concluir que una de las combinaciones de hiperparámetros como la óptima para el problema que se deseaba solucionar y esta es la combinación del optimizador Adam, con una tasa de aprendizaje de 0.01, con 15 neuronas en una única capa oculta y un total de 1000 iteraciones de entrenamiento. La razón de esta selección es porque mantiene la red simple con una sola capa oculta, una tasa de aprendizaje intermedia, por lo que requiere menos recursos computacionales y lo más importante, tiene un alto porcentaje de predicción y precisión.

Estudio de sensibilidad de las entradas

Para este caso, se va a utilizar el modelo con optimizador ADAM, tasa de aprendizaje de 0,01, con 15 neuronas por capa oculta, con una capa oculta. A esta red ya entrenada, se le van a ingresar diferentes parámetros de validación, de modo que se pueda notar la influencia de las entradas en la predicción de la red. Para esto, se va a tomar solo una de las 90 entradas, por ejemplo, Fx1 y se le suma o resta un 10% de su valor, mientras que las 89 entradas restantes se mantienen igual. Esto se repite para cada una de las entradas y con un 15%, 20% y un 25% y en cada caso se a obtiene la diferencia entre los errores de predicción con los datos reales y los errores de predicción modificando solo una entrada. Los resultados obtenidos se muestran en la tabla 36.

Tabla 36. Tabla de diferencia de aciertos entre los datos de entrada originales y los datos con una columna modificada en un cierto porcentaje.

	10%	15%	20%	25%
Original	0.0	NaN	NaN	NaN
Fx1	0.0	0.0	1.0	2.0
Fy1	0.0	1.0	1.0	0.0
Fz1	0.0	0.0	0.0	0.0
Tx1	-1.0	0.0	-2.0	-2.0
...	...	...	...	...
Fy15	0.0	0.0	-1.0	-2.0
Fz15	-2.0	-3.0	-7.0	-10.0
Tx15	0.0	0.0	0.0	0.0
Ty15	1.0	-6.0	-14.0	-17.0
Tz15	-2.0	-3.0	-3.0	-3.0

En el anexo 6 se muestra el código implementado para el estudio de sensibilidad de las entradas para el problema de clasificación multiclase.

De manera similar al problema anterior, se obtienen los cinco componentes más y menos influyentes de cada modificación. En la tabla 37 se muestra esta información.

Tabla 37. Tabla con los datos de entrada etiquetados y organizados

	Variación							
	10%		15%		20%		25%	
	Menos influyente	Más influyente	Menos influyente	Más influyente	Menos influyente	Más influyente	Menos influyente	Más influyente
1	Ty1	Fz2	Fy1	Fz2	Ty1	Ty15	Tx7	Tz2
2	Tz1	Fz3	Ty1	Ty15	Fx4	Tz2	Fx1	Ty15
3	Fx2	Fz15	Tz1	Fz3	Tx7	Ty14	Fx4	Ty10
4	Tx2	Tz15	Fx2	Fz5	Fx1	Ty10	Ty1	Ty14
5	Fx3	Tx1	Fx3	Fz12	Fy1	Ty11	Tz1	Ty9

A partir de esta tabla, se puede obtener como parámetro más influyente el que se repite una mayor cantidad de veces en las columnas de más influyente. Esto mismo aplica para el caso de las entradas menos influyentes. Al analizar la tabla 37 se evidencia lo siguiente:

- Cuando las variaciones son pequeñas, las fuerzas tienen una mayor relevancia en la predicción, pero conforme se aumenta la variación de las entradas, los torques influyen más los resultados de la red, específicamente los torques: Tz2, Ty15, Ty10 y Ty14, pues estos se repiten en la variación de 20% y 25%.
- La entrada que tiene una menor relevancia en la red es la torsión Ty1, ya que se encuentra entre las 5 menos influyentes para todas las variaciones realizadas.
- La entrada que altera de manera más significativamente la red es Ty15, ya que se encuentra en 3 de las 4 cuatro columnas de entradas más influyentes.

Conclusiones

1. La aplicación de clasificación binaria, en la que se detecta la existencia de un error con respecto a varias mediciones de fuerzas y tensiones en distintos ejes, con la cantidad de datos presentados, se considera factible y es posible obtener resultados con porcentajes de error menores al 2%.
2. Se determinó que los parámetros que tienen más relevancia en los resultados de la red de clasificación binaria son las fuerzas y torques que se desarrollan a lo largo del eje Y, mientras que las fuerzas en el eje X o Z tienen una relevancia considerablemente menor.
3. Para la red de clasificación multiclase se logró tener un porcentaje de predicción del 100% en las validaciones, no obstante, esto no implica que la red siempre vaya a tener esa precisión, sino que esto se cumplió únicamente para los datos de validación utilizados
4. Para el modelo de clasificación binario o multiclase se recomienda utilizar un optimizador Adam debida al gran número de entradas de la red, ya que se evidenció una mejor convergencia, esto como respuesta a una mayor robustez en su planteamiento teórico-matemático.

Referencias

[1] Andries P. Engelbrecht “Computational Intelligence: An Introduction”. (2ª edición, 2007, Wiley).

[2] J. L. Crespo Mariño y F. Meza Obando, «Estructura de una aplicación en redes neuronales.,» de Tema 2: Redes Neuronales Artificiales (II.1), 2023.

[3] L. Lopes, L. Camarinha-Matos, «Robot Execution Failures,» UC Irvine, 1999. [En línea]. Available: https://archive.ics.uci.edu/dataset/138/robot+execution+failures

[4] “NumPy documentation,” NumPy documentation - NumPy v1.26 Manual. [Online]. Available: https://numpy.org/doc/stable/. [Accessed: 18-Sep-2023]

[5] “User Guide,” User Guide - pandas 2.1.0 documentation. [Online]. Available: https://pandas.pydata.org/docs/user_guide/index.html. [Accessed: 18-Sep-2023]

[6] C. C. Madrigal Sánchez, «Manual de trabajo #1: Redes neuronales simples utilizando Keras/TensorFlow», 2023.

[7] sklearn.metrics.confusion_matrix, scikit. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. [Accessed: 18-Sep-2023]

[8] seaborn.heatmap, seaborn.heatmap - seaborn 0.12.2 documentation. [Online]. Available: https://seaborn.pydata.org/generated/seaborn.heatmap.html. [Accessed: 18-Sep-2023]

[9] “sklearn.metrics.classification_report,” scikit. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html. [Accessed: 18-Sep-2023]

[10] “Cómo elegir una función de activación para el aprendizaje profundo,” Top Big Data, 18-Jan-2021. [Online]. Available: https://topbigdata.es/como-elegir-una-funcion-de-activacion-para-el-aprendizaje-profundo/. [Accessed: 18-Sep-2023]

[11] [1]D. Godoy, “Understanding binary cross-entropy / log loss: a visual explanation,” Medium, 10-Jul-2022. [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. [Accessed: 18-Sep-2023]

[12] Scikit-learn User Guide, Metrics and scoring: quantifying the quality of predictions, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html

[13] Tensorflow API Documentation, Module: tf.keras.losses, 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/losses

Anexos

Anexo 1

Código para darle el formato a la tabla que se usa como entrada de datos etiquetados en la primera red neuronal.

```
import numpy as np
import pandas as pd

def normalizing(df):
    max_val = df.max(axis=0)  # Se obtiene el máximo de cada columna
    min_val = df.min(axis=0)  # Se obtiene el mínimo de cada columna
    range = max_val - min_val  # Se obtiene la diferencia de los dos
    df = (df - min_val)/(range)  # Y se utiliza para normalizarlas
    df = df.astype(float)  # Se asegura que los datos sean tipo float
    return df


def formatting(df):
    df = df.reset_index(drop=False)
    df.pop("index")
    # Crea las columnas necesarias para la entrada
    for i in range(14, 0, -1):
        df.insert(7, f"Tz{i+1}", " ")
        df.insert(7, f"Ty{i+1}", " ")
        df.insert(7, f"Tx{i+1}", " ")
        df.insert(7, f"Fz{i+1}", " ")
        df.insert(7, f"Fy{i+1}", " ")
        df.insert(7, f"Fx{i+1}", " ")

    n_datasets = len(df.index)/16  # Saca cuantos datos hay en el documento
para repetir el acomodo
    for k in range(0, int(n_datasets)):
        for j in range(1, 16):
            df.loc[k*16, df.columns[1+(6*(j-1))]:df.columns[6+(6*(j-1))]] =
df.loc[j+k*16, df.columns[1]:df.columns[6]].to_numpy()
            df = df.drop(index=j+16*k)
    df = df.rename(columns={'Tipo': 'Resultado'})
    arreglo = df["Resultado"].unique()  # Hace un arreglo con todos los
posibles resultados
    arreglo = np.delete(arreglo, np.where(arreglo == "normal"))  # Elimina del
arreglo el valor "normal" o "ok"
    arreglo = np.delete(arreglo, np.where(arreglo == "ok"))
    df = df.replace(arreglo, 1)  # Se remplazan todos los valores del arreglo
por un 1
    df = df.replace(to_replace="normal", value=0.01)  # Se remplazan los
valores de normal u ok por 0.01
    df = df.replace(to_replace="ok", value=0.01)
    df = normalizing(df)  # Se normalizan los datos por columnas
```

```
    df = df.replace(0.0, 0.001)  # Elimina los ceros para que no hayan
problemas con la correción
    return df


df1 = pd.read_csv("lp1.data", sep='\t', lineterminator='\n')
df1 = formatting(df1)

df2 = pd.read_csv("lp3.data", sep='\t', lineterminator='\n')
df2 = formatting(df2)

df3 = pd.read_csv("lp3.data", sep='\t', lineterminator='\n')
df3 = formatting(df3)

df4 = pd.read_csv("lp4.data", sep='\t', lineterminator='\n')
df4 = formatting(df4)

df5 = pd.read_csv("lp5.data", sep='\t', lineterminator='\n')
df5 = formatting(df5)

df = pd.concat([df1, df2, df3, df4, df5], axis=0)
df = df.reset_index() # Resetea el index para que quede en orden después de
concatenar
df.pop("index") # Elimina una columna innecesaria
df.to_csv("All_data.csv") # Guarda el dataframe en excel
```

Anexo 2

Código utilizado para el entrenamiento de la red, así como para definir los hiperparámetros.

```
import pandas as pd
import tensorflow as tf
from keras import models
from keras import layers
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report

num_neurons = 60
n_layers = 2
tasa = 0.03
train_percentage = 0.70
test_percentage = 0.20
iteraciones = 1500

df = pd.read_csv("All_data.csv")  # Lee el archivo de excel
df.pop("Unnamed: 0")  # Elimina una columna innecesaria
print(df)
# Distribución de los datos
trainset = df.sample(frac=train_percentage)  # Se extraen datos
# para el entrenamiento
tempset = df.drop(trainset.index)  # Y se le quitan esos mismos
# al dataset para crear los datos de prueba
testset = tempset.sample(frac=3.3*test_percentage)
verificationtest = tempset.drop(testset.index)

network = models.Sequential()

# Declaración de la capa de entrada y la primera oculta
network.add(tf.keras.layers.Dense(
    units=num_neurons,
    activation="relu",
    input_shape=(90,)))

# Ciclo de capas de neuronas intermedias
i = 0
while i < n_layers-1:
    network.add(layers.Dense(
        units=num_neurons,
        activation="relu"))
    i += 1

# Declaración de la capa de salidas
network.add(layers.Dense(
    units=1,
```

```
    activation="sigmoid"))

network.compile(
    optimizer="ADAM",
    loss=tf.keras.losses.BinaryCrossentropy())
tf.keras.optimizers.Adam(
    learning_rate=tasa)
losses = network.fit(
    x=trainset[trainset.columns[1:]],
    y=trainset["Resultado"],
    validation_data=(
        testset[testset.columns[1:]],
        testset['Resultado']),
    batch_size=200,
    epochs=iteraciones)
# network.save("red_neuronal_1.keras")  # Se encarga de guardar el modelo para
no tener que entrenarlo nuevamente después
# Se extrae el historial de error contra iteraciones de la clase
loss_df = pd.DataFrame(losses.history)
# Se grafica
loss_df.loc[:, ['loss', 'val_loss']].plot()
# Y se pide que se muestre la ventana en que se graficó
# plt.show()

true_labels = verificationtest["Resultado"]
datoPrueba = verificationtest.drop(columns=["Resultado"])
result = network.predict(datoPrueba)
print("Dato ingresado:")
print(verificationtest)
verificationtest["Resultado"] = result

print("Estimación: ")
print(verificationtest)

result[(result < 0.25)] = 0
result[(result >= 0.25)] = 1
result = np.where(result == 0, "normal", "error")

true_labels = np.where(true_labels == 1.000, "error", "normal")

# Muestra la matriz de confusión
plt.figure(figsize=(8, 8))
plt.title('Confusion Matrix', size=20, weight='bold')
sns.heatmap(
 confusion_matrix(true_labels, result),
 annot=True,
 annot_kws={'size': 14, 'weight': 'bold'},
 fmt='d',
```

```
 cbar=False,
 cmap='RdPu',
 xticklabels=['error', 'normal'],
 yticklabels=['error', 'normal'])
plt.tick_params(axis='both', labelsize=14)
plt.ylabel('Actual', size=14, weight='bold')
plt.xlabel('Predicted', size=14, weight='bold')
plt.show()

# Muestra el reporte de clasificación
print("\n-------------------------------------------------------")
print("Classification report:\n")
print(classification_report(true_labels, result, digits=4,
                            target_names=['error', 'normal']))
print("-------------------------------------------------------")
```

Anexo 3

Código usado para el estudio de sensibilidad de la red neuronal de clasificación binaria.

```
import pandas as pd
from keras import models
import numpy as np
from sklearn.metrics import confusion_matrix

# Cargo el modelo ya definido anteriormente
network = models.load_model("red_neuronal_1.keras")

df = pd.read_csv("All_data.csv")  # Lee el archivo de excel original
df.pop("Unnamed: 0")  # Elimina una columna innecesaria

true_labels = df["Resultado"]  # Guarda los resultados correctos
datoPrueba = df.drop(columns=["Resultado"])  # Genera un df solo con entradas

result = network.predict(datoPrueba) # Predice

result[(result < 0.25)] = 0
result[(result >= 0.25)] = 1
result = np.where(result == 0, "normal", "error")

true_labels = np.where(true_labels == 1.000, "error", "normal")

# Muestra la matriz de confusión
true_error, false_normal, false_error, true_normal =
confusion_matrix(true_labels, result).ravel()
errores = false_error + false_normal

df2 = pd.DataFrame([0], columns=["5%"], index=["Original"])
col_names = ["5%", "10%", "15%", "20%"]
functions_list = [lambda x:x*1.05, lambda x:x*1.10, lambda x:x*1.15, lambda
x:x*1.20]
for j in range(len(functions_list)):
    for i in range(1, len(df.columns)):
        df_aux = df.copy()  # Copia el dataframe original
        # Las próximas tres lineas cambian la columna correspondiente a los
valores modificados
        col = df[df.columns[i]]
        change_col = col.apply(functions_list[j])
        df_aux[df.columns[i]] = change_col

        entrada = df_aux.drop(columns=["Resultado"])  # Genera los valores de
entrada a la red
        result_aux = network.predict(entrada) # Hace la predicción
        # Pasa a binario lo que predijo la red
        result_aux[(result_aux < 0.25)] = 0
        result_aux[(result_aux >= 0.25)] = 1
```

```
        result_aux = np.where(result_aux == 0, "normal", "error")

        # Genera los valores correctos o de error
        true_error_aux, false_normal_aux, false_error_aux, true_normal_aux =
confusion_matrix(true_labels, result_aux).ravel()

        # Calcula la cantidad total de errores
        errores_aux = false_error_aux + false_normal_aux

        # Saca la diferencia de errores entre los valores originales y con la
tabla modificada
        diferencia = errores - errores_aux # Si es positivo quiere decir que
mejoró la red

        # Guarda el valor en el dataframe
        df2.at[df.columns[i], col_names[j]] = diferencia
    print(df2)
    print(f"Rubros con mayor cantidad de errores para una variación de
{col_names[j]}:")
    print(df2[col_names[j]].nsmallest(5))
    print(f"Rubros con menor cantidad de errores para una variación de
{col_names[j]}:")
    print(df2[col_names[j]].nlargest(5))
```

Anexo 4

Código utilizado para dar formato a los datos de entrada para el problema de clasificación multiclase.

```
# Importación de librerías
import pandas as pd
import numpy as np

# Definición de funciones

# Función para normalizar los datos en un rango de 0 a 1
def normalizing(df):
    max_val = df.max(axis=1)  # Se obtiene el máximo de cada columna
    min_val = df.min(axis=1)  # Se obtiene el mínimo de cada columna
    range = max_val - min_val  # Se obtiene la diferencia de los dos
    df = (df - min_val)/(range)  # Y se utiliza para normalizarlas
    df = df.astype(float)  # Se asegura que los datos sean tipo float
    return df

# Función utilizada para formatear y ordenar los datos en un dataframe
def formatting(df):
    df = df.reset_index(drop=False)
    df.pop("index")
    # Crea las columnas necesarias para la entrada
```

```
    for i in range(14, 0, -1):
        df.insert(7, f"Tz{i+1}", " ")
        df.insert(7, f"Ty{i+1}", " ")
        df.insert(7, f"Tx{i+1}", " ")
        df.insert(7, f"Fz{i+1}", " ")
        df.insert(7, f"Fy{i+1}", " ")
        df.insert(7, f"Fx{i+1}", " ")

    n_datasets = len(df.index)/16 # Saca cuantos datos hay en el documento
para repetir el acomodo
    for k in range(0, int(n_datasets)):
        for j in range(1, 16):
            df.loc[k*16, df.columns[1+(6*(j-1))]:df.columns[6+(6*(j-1))]] =
df.loc[j+k*16, df.columns[1]:df.columns[6]].to_numpy()
            df = df.drop(index=j+16*k)
    df = df.rename(columns={'Tipo': 'Resultado'})

    df = df.replace(to_replace="normal", value=1) # Se remplazan los valores
de normal por 0.0001
    df = df.replace(to_replace="collision", value=3) #1
    df = df.replace(to_replace="obstruction", value=2) #0.1
    df = df.replace(to_replace="fr_collision", value=0) #0.01
    df = normalizing(df) # Se normalizan los datos por columnas
    df = df.replace(0.000000, 0.00001) # Elimina los ceros para que no haya
problemas con la correción

    return df

####################### FORMATEO DE DATOS Y CREACIÓN DE ARCHIVO DE EXCEL
############################
# La siguiente sección de código se usó para formatear los datos y cargarlos
en un archivo de Excel
# Se debe ejecutar solo una vez para generar el archivo .cvs, posteriormente
se debe comentar


df1 = pd.read_csv("lp1_1.data", sep='\t', lineterminator='\n')
df1 = formatting(df1)
df2 = pd.read_csv("lp4.data", sep='\t', lineterminator='\n')
df2 = formatting(df2)
data = pd.concat([df1, df2], axis=0) # Concatena todos los grupos de datos en
un sólo DataFrame
data = data.reset_index() # Resetea el index para que quede en orden después
de concatenar
data.pop("index") # Elimina una columna innecesaria
data.to_csv("All_data.csv") # Guarda el dataframe en excel

# Se crea un excel con los resultados para determinar cuántos datos hay en
cada categoría
```

```
#arreglo = data["Resultado"]
#f = pd.DataFrame(arreglo)
#f.to_csv("tipos.csv", index=False)
```

Anexo 5

Código utilizado para el entrenamiento de la red, así como para definir los hiperparámetros para el problema de clasificación multiclase.

```
# Importación de librerías
import pandas as pd
import tensorflow as tf
from keras import models
from keras import layers
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score,
f1_score


# Definición de hiperparámetros
num_neurons = 15
n_layers = 1
train_percentage = 0.80


# Carga de datos
data = pd.read_csv("All_data_1.csv") # Se lee el archivo de excel con los
datos formateados
print(data)
data.pop("Unnamed: 0") # Elimina una columna innecesaria
print(data)

# Distribución de los datos en sets de entrenamiento y prueba
trainset = data.sample(frac=train_percentage)  # Se extraen datos para el
entrenamiento
testset = data.drop(trainset.index)  # Y se le quitan esos mismos al dataset
para crear los datos de prueba

# Conversión de los vectores de resultado a matrices de clasificación
y_train = to_categorical(trainset["Resultado"], num_classes=4)
y_test = to_categorical(testset["Resultado"], num_classes=4)

# Creación del modelo de red neuronal
network = models.Sequential()
```

```
# Declaración de la capa de entrada y la primera capa oculta
network.add(tf.keras.layers.Dense(
    units=num_neurons,      # número de neuronas en la capa oculta
    activation="sigmoid",   # función de activación
    input_shape=(90,)))     # cantidad de neuronas en la capa de entrada

# Ciclo para crear las capas de neuronas intermedias
i = 0
while i < n_layers-1:           # se repite el ciclo para completar el número
de capas deseadas
    network.add(layers.Dense(
        units=num_neurons,      # número de neuronas
        activation="sigmoid"))  # funsión de activación
    i += 1

# Declaración de la capa de salidas
network.add(layers.Dense(
    units=4,                # cantidad de neuronas de la capa de salida
    activation="sigmoid"))  # función de activación

# Compilación del modelo
network.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),  # se define el
optimizador y la tasa de aprendizaje
    loss=tf.keras.losses.CategoricalCrossentropy())          # se define la
función de pérdida

# Entrenamiento del modelo
losses = network.fit(
    x=trainset[trainset.columns[1:]],
    y=y_train,
    validation_data=(
        testset[testset.columns[1:]],
        y_test),
    batch_size=100, # cantidad de datos en un batch
    epochs=600)     # número máximo de iteraciones

# Se guarda la red creada
network.save("red_neuronal_1.keras")

# Se extrae el historial de error contra iteraciones de la clase
loss_df = pd.DataFrame(losses.history)
# Se grafican las curvas de pérdida
loss_df.loc[:, ['loss', 'val_loss']].plot()
# Y se pide que se muestre la ventana en que se graficó
plt.show()

# Se eligen 15 datos al azar
```

```
dato = data.sample(frac=15/data.shape[0])
datoPrueba = dato.drop(columns=["Resultado"])
# Y se predice el resultado, luego de revertir la normalización
result = network.predict(datoPrueba)
# Se usa la función numpy.argmax para retornar el número de categoría que fue
predicho
Prediction = np.argmax(result, axis=1)

# Se muestran los resultados en consola
print("Resultado:")
dato.insert(0, "Predicción", Prediction)
print(dato)

# Display de la matriz de confusión
true_labels = dato["Resultado"]
plt.figure(figsize=(4,4))
plt.title('Confusion Matrix', size=20, weight='bold')
sns.heatmap(
 confusion_matrix(true_labels, Prediction),
 annot=True,
 annot_kws={'size':14, 'weight':'bold'},
 fmt='d',
 cbar=False,
 cmap='RdPu',
 xticklabels=['fr_collision', 'normal', 'obstruction', 'collision'],
 yticklabels=['fr_collision', 'normal', 'obstruction', 'collision'])
plt.tick_params(axis='both', labelsize=14)
plt.ylabel('Actual', size=14, weight='bold')
plt.xlabel('Predicted', size=14, weight='bold')
plt.show()

# Display del reporte de clasificación
print("\n-----------------------------------------------------")
print("Classification report:\n")
print(classification_report(true_labels, Prediction, digits=4,
                            target_names=['fr_collision', 'normal',
                                          'obstruction', 'collision']))
print("-----------------------------------------------------")
```

Anexo 6

Código utilizado para el estudio de sensibilidad de entradas del problema de clasificación multiclase.

```
# Importación de librerías
import pandas as pd
from keras import models
import numpy as np
from sklearn.metrics import confusion_matrix
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Variables globales con el número de errores
errores = 0      # errores de la red original
errores_aux = 0  # errores de la red modificada

# Se carga el modelo ya definido anteriormente
network = models.load_model("red_neuronal_1.keras")

df = pd.read_csv("All_data.csv")  # Lee el archivo de excel original
df.pop("Unnamed: 0")  # Elimina una columna innecesaria

true_labels = df["Resultado"]  # Guarda los resultados correctos
datoPrueba = df.drop(columns=["Resultado"])  # Genera un df solo con entradas

result = network.predict(datoPrueba)    # se predicen los resultados con el
dataframe de prueba
Prediction = np.argmax(result, axis=1)  # se usa la función numpy.argmax para
extraer el número de clase predicho

# Se genera la matriz de confusión
Matrix = confusion_matrix(true_labels, Prediction).ravel()
# Descomentar el siguiente código si se quiere mostrar la matriz de confusión
"""
plt.figure(figsize=(4,4))
plt.title('Confusion Matrix', size=20, weight='bold')
sns.heatmap(
 confusion_matrix(true_labels, Prediction),
 annot=True,
 annot_kws={'size':14, 'weight':'bold'},
 fmt='d',
 cbar=False,
 cmap='RdPu',
 xticklabels=['fr_collision', 'normal', 'obstruction', 'collision'],
 yticklabels=['fr_collision', 'normal', 'obstruction', 'collision'])
plt.tick_params(axis='both', labelsize=14)
plt.ylabel('Actual', size=14, weight='bold')
plt.xlabel('Predicted', size=14, weight='bold')
plt.show()
"""

# Ciclo para contar el número de errores en la clasificación de los datos
for a in range(len(Matrix)):
      if a not in [0 , 5, 10, 15]:
          errores = errores + Matrix[a] # se suman todos los datos de la
matriz que no estén
```

```
                                        # en la diagonal (número de
clasificaciones incorrectas)


# Variables necesarias para el estudio de sensibilidad
df2 = pd.DataFrame([0], columns=["10%"], index=["Original"])  # Dataframe para
guardar el resultado
col_names = ["10%", "15%", "20%", "25%"]    # Nombres de las columnas
functions_list = [lambda x:x*1.10, lambda x:x*1.15, lambda x:x*1.20, lambda
x:x*1.25] # lista de funciones anónimas usadas para

        # escalar los datos del dataframe

# Ciclo para realizar el estudio de sensibilidad
for j in range(len(functions_list)):
    for i in range(1, len(df.columns)):
        df_aux = df.copy()  # Copia el dataframe original
        # Las próximas tres lineas cambian la columna correspondiente a los
valores modificados
        col = df[df.columns[i]]
        change_col = col.apply(functions_list[j])
        df_aux[df.columns[i]] = change_col

        entrada = df_aux.drop(columns=["Resultado"])  # Genera los valores de
entrada a la red
        result_aux = network.predict(entrada) # Hace la predicción
        # Pasa a binario lo que predijo la red
        Prediction_aux = np.argmax(result_aux, axis=1)

        # Genera los valores correctos o de error
        Matrix_aux = confusion_matrix(true_labels, Prediction_aux).ravel()

        # Calcula la cantidad total de errores
        for b in range(len(Matrix_aux)):
            if b not in [0 , 5, 10, 15]:
                errores_aux = errores_aux + Matrix_aux[b]

        # Saca la diferencia de errores entre los valores originales y con la
tabla modificada
        diferencia = errores - errores_aux # Si es positivo quiere decir que
mejoró la red
        errores_aux = 0

        # Guarda el valor en el dataframe
        df2.at[df.columns[i], col_names[j]] = diferencia

    # Se imprimen los resultados
    print(df2)
```

```
    print(f"Rubros con mayor cantidad de errores para una variación de
{col_names[j]}:")
    print(df2[col_names[j]].nsmallest(5))
    print(f"Rubros con menor cantidad de errores para una variación de
{col_names[j]}:")
    print(df2[col_names[j]].nlargest(5))
```