Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación



# Generación de una plataforma que asista en la creación de redes neuronales para su simulación en Matlab.

Una disertación presentada en cumplimiento parcial de los requisitos para obtener el grado de

**Doctorado en ingeniería del lenguaje y del conocimiento**

Presenta:

M.C. Daniel Marcelo González Arriaga

Director: Dra. María Aurora Diozcora Vargas Treviño

Co-director: Dra. Josefina Guerrero García

Julio 2021

## Resumen

Este artículo presenta una plataforma de software (SP) que ayuda a generar redes neuronales convolucionales y feedforward y se propone un algoritmo de identificación paramétrico para un robot cartesiano utilizando redes neuronales convolucionales (CNN) generadas y entrenadas con el SP propuesto. La interfaz de usuario es amigable y ayuda en el desarrollo de redes neuronales convolucionales y de avance. El SP está hecho en LabVIEW®. El usuario no necesita el conocimiento completo de las matemáticas utilizadas para realizar el entrenamiento o la ejecución de la red o las habilidades de programación para realizar el código que realiza dichas tareas. El SP está diseñado para reducir el tiempo de despliegue de las redes neuronales, proporcionando al usuario una interfaz gráfica que guía al usuario a través de la formación y desarrollo de la red neuronal, esta es una de las principales contribuciones. La interfaz del SP se describe en este artículo, mostrando todas las opciones que se ofrecen. Se presentan tres aplicaciones de redes neuronales generadas son el SP. Red neuronal para la identificación de números en imágenes. Se busca comparar diferentes estructuras de redes neuronales con la finalidad de identificar números en imágenes de $28 \times 28$ pixeles, la tasa de error mas baja fue de 8.78% Red neuronal para la identificación de 10 parámetros del modelo dinámico de un robot de 2 grados de libertad, obteniendo porcentajes de similitud de 95.32%. Comparación de identificación de 6 y 10 parámetros del modelo dinámico de un robot de 3 grados de libertad. Se utilizan los modelos dinámicos del robot cartesiano; uno tiene seis parámetros y el otro diez parámetros. La máxima similitud alcanzada por el algoritmo de identificación con el modelo dinámico de seis parámetros fue del 99,15% y del 99,92% para el modelo de diez parámetros; ambos son análisis numéricos de torque. Se identificó un robot cartesiano real y se reconstruyeron cinco pares experimentales para cada eje; la máxima similitud fue del 97,87%. El algoritmo propuesto se compara con mínimos cuadrados, los cuales obtienen un 95,81%. Por lo tanto, los resultados muestran que el método propuesto genera una mejor identificación paramétrica.

# Contenido

# Índice de tablas

# Índice de figuras

# Capitulo 1. Introducción

La IA (inteligencia artificial) se está volviendo omnipresente muy rápidamente debido a su robusta aplicabilidad en los problemas, particularmente aquellos que no pueden ser resueltos bien por humanos, por ejemplo, en la medicina donde los algoritmos se utilizan para identificar sujetos con antecedentes familiares de una enfermedad hereditaria o un riesgo aumentado de una enfermedad crónica o en la evaluación de cambios en el desempeño humano en tales situaciones como rehabilitación (Hamet & Tremblay, 2017).

Mantener un enfoque riguroso del ciclo de vida y la evolución del aprendizaje automático puede ser difícil, especialmente cuando se trata de diseñar un nuevo mecanismo o proceso de aprendizaje automático. El acceso a herramientas y recursos de prueba adecuados es esencial para los investigadores que operan en este campo. Como desarrollador de redes neuronales, probablemente todos pasamos horas personalizando un entorno de trabajo al menos una vez. Reconocimos que las herramientas y técnicas son las mismas. Por lo tanto, mantener un entorno personalizado manteniendo todo el software actualizado puede ser una tarea monótona. Este trabajo presenta nuestros puntos de vista sobre el desarrollo de una estación de trabajo comunitaria con IA en mente. Sin embargo, todavía se necesita una plataforma local para investigadores y científicos que quieran experimentar y desarrollar esta tecnología.

## 1.1 Planteamiento del problema

Hay un tipo particular de red neuronal artificial que marca una diferencia en la práctica, que es precisamente la que corresponde a las redes utilizadas para procesar señales: las redes convolucionales. Su éxito en la resolución de problemas de visión por computadora que, hasta hace poco, se consideraban casi insolubles, ha servido para reafirmar las redes neuronales en la IA [2]. Los recientes avances en las CNN (redes neuronales convolucionales) han dado lugar a grandes mejoras en la precisión de los sistemas auditivos y de visión. Caracterizadas por estructuras profundas y muchos parámetros, las CNN profundas desafían el rendimiento de las computadoras actuales. La mayor parte del trabajo se centra en la implementación y mejora de estas redes.

En los últimos años, se han hecho muchos avances con redes neuronales profundas, ofreciendo resultados de vanguardia en tareas de aprendizaje automático como la clasificación de imágenes y documentos. Las empresas hacen uso de los últimos avances en el aprendizaje profundo gracias a un entorno de software en rápida evolución que comprende diferentes marcos de aprendizaje profundo.

La industria 4.0 denomina una hipotética cuarta mega etapa de la evolución técnico-económica de la humanidad, contando a partir de la Primera Revolución Industrial. Esta cuarta etapa habría comenzado recientemente y su desarrollo estaría proyectado hacia la tercera década del siglo XXI. La IA es señalada como elemento central de esta transformación, íntimamente relacionada a la acumulación creciente de

grandes cantidades de datos, el uso de algoritmos para procesarlos, y la interconexión masiva de sistemas y dispositivos digitales.

En el sector industrial, se ha observado cómo la mecanización permite aumentar espectacularmente la productividad de una fábrica reemplazando operadores humanos por máquinas y robots cada día más eficientes y versátiles. Ese aumento de productividad no se ha producido en el sector cuaternario: la productividad de un médico o de un profesor no ha cambiado tan radicalmente (aunque sí puedan ofrecer un mejor servicio gracias a determinados avances tecnológicos).

Hay quien prefiere ver estas herramientas de redes neuronales como cajas negras que se pueden incorporar, como un módulo más, a un proyecto ya en marcha para mejorarlo. Es la estrategia de los que se limitan a utilizar soluciones suministradas por terceros. En el caso del aprendizaje profundo existen herramientas como Keras, TensorFlow, MSNet, Microsoft Cognitive Toolkit, Apache SINGA, Torch, Caffe (referencias). Sin embargo, cuando todas las soluciones son adquiridas a proveedores externos, la velocidad de adopción se convierte en el único factor diferencial. Aquellas empresas que se proponen tener un crecimiento sostenido a más largo plazo, no obstante, tendrán que ser capaces de desarrollar su propia tecnología.

### 1.1.1 Definición del problema

Existen diversas herramientas para la implementación de redes neuronales, en la tabla 1-1 se muestran algunas de ellas.

Tabla 1-1 : Descripción de herramientas para redes neuronales.

| | |
|---|---|
| TensorFlow (Abadi et al., 2016) | Empezó en el 2011 como un proyecto interno de Google llamad “Google Brain” y que se hizo público en el 2017 como un sistema de código abierto de aprendizaje profundo, es decir, de una red neuronal, la cual puede correr en múltiples CPUs y GPUs. Se usa para entrenar redes neuronales que puedan detectar y descifrar patrones y correlaciones análogas a las que vemos en el aprendizaje y razonamiento humano. |
| Caffe (Jia et al., 2014) | Esta herramienta a fue creada por BAIR (Berkeley Artificial Intelligence Research), en el 2014, y se hizo popular en la investigación académica. En un marco de trabajo de aprendizaje profundo usando redes convolucionales. |
| ONNX (Bai et al., 2019) | Esta herramienta significa Open Neural Network exchanged y se anunció apenas en septiembre del 2017. Es un esfuerzo conjunto de Microsoft y Facebook. ONNX es un formato pensado para hacer fácil el intercambio de modelos de aprendizaje profundo entre entornos de esta naturaleza. La iniciativa busca hacer más fácil para los desarrolladores usar múltiples entornos de programación de redes neuronales. |
| Matlab | Cuenta con toolboxes especializadas para trabajar con machine learning, redes neuronales, aprendizaje profundo, visión artificial y conducción autónoma.<br>Puede utilizar MATLAB Coder a fin de generar código C y C++ para su red entrenada, lo cual permite simular una red entrenada en hardware de PCs y, posteriormente, desplegar la red en sistemas embebidos. |
| Weka (Frank et al., 2016) | Es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato.<br>Es usada principalmente para hacer clasificadores y solo pueden utilizar redes neuronales de tipo perceptrón multicapa |

Las herramientas mencionadas sirven para intercambiar modelos de aprendizaje como ONNX, o para realizar el entrenamiento de alguna red neuronal como TensorFlow. Matlab es una herramienta interesante ya que cuenta con diferentes redes neuronales ya implementadas, el principal inconveniente es que no es

de código abierto, esto además de generar un costo elevado, no se tiene acceso total a lo que las herramientas del software hacen, y si hay alguna función que no se encuentre implementada no se puede hacer nada.

No se ha encontrado reportado en la bibliografía revisada hasta este momento una herramienta de uso libre que asista en la creación de diversos tipos de redes neuronales, reduciendo el tiempo de creación y que permita implementar dichas redes creadas redes en un sistema embebido listo para su implementación.

La investigación por realizar es del tipo cuantitativa debido a la naturaleza del problema, el resultado esperado es la obtención de una plataforma experimental lista para trabajar con redes neuronales. Dado que se busca un resultado único se contempla una investigación descriptiva, esto nos permite llegar a un único resultado, y es posible medir la exactitud de este, se definen los objetivos a alcanzar y la metodología para llegar a ellos.

## 1.2 Objetivos y preguntas de investigación

Objetivo general:

- Generar una plataforma que asista en la creación de redes neuronales para su simulación en Matlab.

Objetivos específicos:

1. Identificar la estructura de una red neuronal y los procesos repetitivos que se puedan simplificar, con el fin de asistir al usuario.
2. Proponer una estrategia que permita la simplificación de los datos de entrada para generar una red neuronal dada.
3. Diseñar la interfaz de usuario para el uso de la plataforma considerando la experiencia de usuario y usabilidad.
4. Demostrar la viabilidad del sistema, validando su desempeño con base en las métricas identificadas en la literatura.

Preguntas de investigación:

1. ¿Cuáles son las operaciones y representaciones matemáticas necesarias para trabajar con redes neuronales?
2. ¿Cuáles son los procesos repetitivos en el desarrollo de redes neuronales convolucionales y multicapa?
3. ¿Cuáles serían los componentes de una plataforma que asista al diseño de redes neuronales?
4. ¿Cuáles son las métricas existentes para medir el desempeño de una red neuronal?

## 1.3 Justificación

Las computadoras han demostrado, dada su potencia de cálculo, ser capaces de resolver fácilmente problemas que para nosotros resultan muy difíciles, desde problemas de cálculo numérico hasta el análisis de situaciones complejas que requieran evaluar millones de escenarios alternativos. Sin embargo, existen aún algunas tareas que a nosotros los humanos nos resultan sencillas pero que, sin embargo, para las computadoras siguen siendo extremadamente difíciles. Sin duda, una posible causa de estas diferencias se debe a la evolución en sentidos opuestos que han experimentado los humanos y animales. Millones de años de evolución dotaron a los animales de capacidades sensoriales y motoras antes de que sólo los animales más evolucionados fuesen capaces de prever las consecuencias de sus acciones, razonar, comunicarse, utilizar herramientas o desarrollar el cálculo. En apenas unas décadas, las computadoras, que comenzaron siendo herramientas de cálculo, se utilizaron luego para controlar herramientas, transmitir datos, razonar de forma simbólica y planificar. Sólo recientemente se ha conseguido dotar a las máquinas de capacidades sensoriales similares a las de animales y métodos, aún muy rudimentarios, de procesar el lenguaje natural mediante el que nos comunicamos los humanos gracias al uso de redes neuronales.

Como se ha descrito, las aplicaciones de las redes neuronales son muchas y en muy diversas áreas. El área de visión computacional es una de las aplicaciones con más auge, la cual se apoya en redes neuronales, esta puede ser muy útil en sectores como el industrial, por ejemplo, en la identificación de elementos que una maquina puede recolectar, soldar, acomodar, ordenar, transportar, etcétera (Gharbi et al., 2016; Kakani et al., 2020; Meng et al., 2022; Yin et al., 2022). La visión en vehículos autónomos es una parte importante para la toma de decisiones.

Dado que los diferentes tipos de redes neuronales pueden mejorar muchos procesos, e incluso realizar algunos otros que los seres humanos no pueden, es importante tener plataformas versátiles de bajo costo, en las que se puedan desarrollar redes neuronales funcionales. La mayoría de las veces se utilizan computadoras con gran capacidad de procesamiento para desarrollar y entrenar redes neuronales, pero en algunos casos se necesitan sistemas más pequeños que sean capaces de mejorar el desempeño de máquinas o robots ya existentes y que se puedan acoplar sin representar un problema de tamaño o incluso de costo energético.

## 1.4 Contenido de la tesis

La tesis se estructura de la siguiente manera:

- El capítulo dos integra el marco teórico donde se explicarán los conceptos principales necesarios para la investigación
- El capítulo tres muestra el análisis del estado del arte haciendo énfasis en las investigaciones más recientes respecto a las redes neuronales en sistemas embebidos, así como en los lenguajes de dominio especifico.
- En el capítulo cuatro se describe le metodología propuesta para la solución del problema.
- En el capítulo cinco se muestra el análisis a una red neuronal convolucional y la identificación de los elementos que el asistente tiene que realizar para simular la red neuronal.
- En el capítulo seis se muestra el desarrollo realizado para generar la plataforma, la interfaz de usuario, el funcionamiento interno y como se genera la estructura de la red neuronal con los datos de entrada.
- En el capítulo cinco está conformado por los resultados obtenidos en cada uno de los experimentos realizados.
- En el capítulo seis se muestran las conclusiones obtenidas del proceso de investigación.

# Capitulo 2. Marco teórico

La definición de IA varia de autor en autor, dado que es prácticamente imposible alcanzar una definición de IA con la que todo el mundo esté de acuerdo, Keith Downing (Downing, 2015) propone que nos conformemos con una definición mucho más ambigua y menos ambiciosa: hacer lo correcto en el momento adecuado, desde el punto de vista de un observador humano externo. Otro punto de vista es el de Elaine Rich (Berzal, 2018), de la Universidad de Texas en Austin: el estudio de cómo hacer que las computadoras hagan cosas que, por ahora, los humanos hacemos mejor. Ésta es una definición móvil de la IA, ya que, en cuanto un problema abordado por técnicas de IA se resuelve satisfactoriamente, pasa a quedar automáticamente fuera de su ámbito de actuación. Por otro lado, mantiene el interés de la IA centrando nuestra atención en aquellos problemas que, actualmente, los seres humanos somos capaces de resolver mejor que las máquinas.

## 2.1 Redes neuronales

El aprendizaje automático, o machine learning, proporciona mecanismos mediante los cuales la computadora es capaz de aprender por sí mismo a resolver un problema. En estos casos, el programador se encarga de diseñar un algoritmo de aprendizaje que resulte adecuado para el problema que se pretende resolver, pero es la computadora el que resuelve el problema, aprovechando para ello los datos a los que tenga acceso y las heurísticas de aprendizaje incorporadas en el algoritmo de aprendizaje creado por el programador. En cierto modo, la computadora es capaz de programarse a sí mismo.

En IA, el aprendizaje se entiende como un proceso por el cual una computadora es capaz de mejorar su habilidad en la resolución de un problema a través de la adquisición de conocimiento, conocimiento que obtiene a través de la experiencia (Berzal, 2018) figura 2-1.

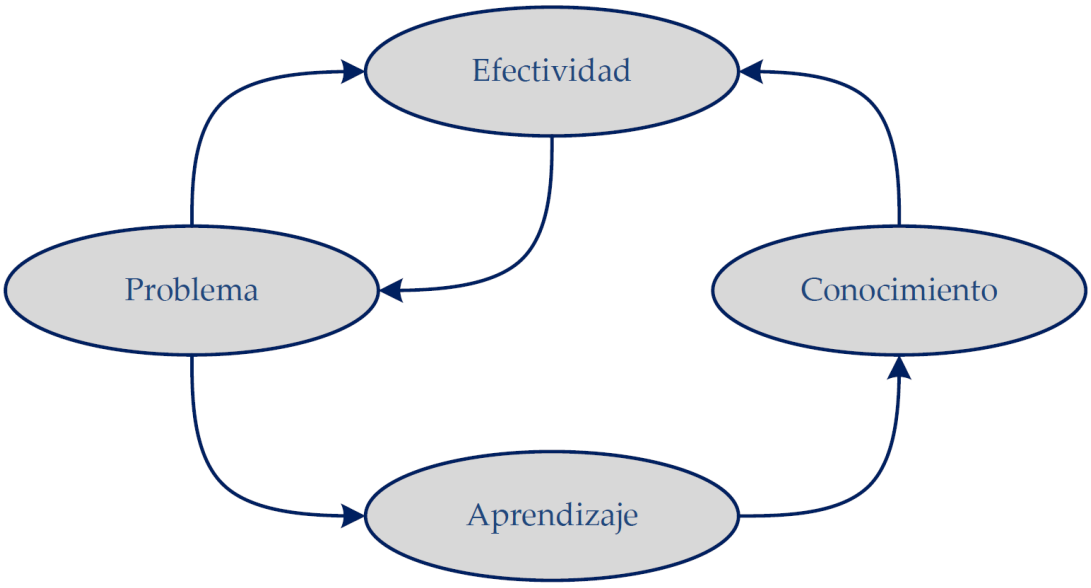



Figura 2-1 Interacciones entre el problema, problema, el proceso de aprendizaje, el conocimiento adquirido y la efectividad observada en la resolución del problema.

Tal como lo definió Herbert Simon (Berzal, 2018), en general, el aprendizaje denota cambios en un sistema que son adaptativos en el sentido de que permiten al sistema hacer la misma tarea, a partir de la misma posición, de un modo más efectivo.

Una primera clasificación de las técnicas de aprendizaje automático puede realizarse atendiendo a la filosofía utilizada en el proceso de adquisición del conocimiento:

- En el aprendizaje supervisado (o aprendizaje a partir de ejemplos, con profesor), los ejemplos de entrenamiento van acompañados de la salida correcta que el sistema debería ser capaz de reproducir. El entrenamiento de un modelo de aprendizaje supervisado consiste en ajustar sus parámetros para que sea capaz de reproducir una salida lo más parecida posible a la deseada. Una vez entrenado el modelo, lo verdaderamente importante es que sea capaz de generalizar correctamente. Esa capacidad de generalización consiste en que el modelo proporcione salidas adecuadas para datos de entrada diferentes a los datos utilizados durante su entrenamiento. La familia de técnicas de aprendizaje supervisado engloba al aprendizaje memorístico [rote learning], a los modelos de aprendizaje por ajuste de parámetros y a una amplia gama de métodos de construcción de distintos tipos de modelos de clasificación, desde árboles de decisión y listas de decisión hasta máquinas de vectores de soporte SVM [Support Vector Machines] o redes neuronales utilizadas para clasificación. Tendremos ocasión de analizarlas con detalle más adelante.
- En el aprendizaje no supervisado (o aprendizaje por observación, sin profesor) se construyen descripciones, hipótesis o teorías a partir de un conjunto de hechos u observaciones, sin que exista información adicional acerca de cómo deberían clasificarse los ejemplos del conjunto de entrenamiento. Será el método de aprendizaje no supervisado el que decida cómo han de agruparse los datos del conjunto de entrenamiento (en los métodos de agrupamiento o clustering) o qué tipo de patrones son más interesantes dentro del conjunto de entrenamiento (en las técnicas de extracción de reglas de asociación).

### 2.1.1 Redes neuronales multicapa

Las capas de una red multicapa se dividen en dos categorías: capas visibles y capas ocultas. Las capas visibles son aquellas capas (parcialmente) observables desde el exterior de la red. En particular, la primera y la última: las capas de entrada y de salida de la red neuronal. Todas las capas intermedias diferentes a las capas de entrada y de salida reciben el nombre de capas ocultas, por no ser observables directamente desde el exterior.

En una red neuronal simple, tanto la capa de entrada como la capa de salida son visibles desde el exterior de la red. Si añadimos nuevas capas intermedias, estas capas ya no serán visibles desde el exterior, lo que nos obligará a utilizar algoritmos como backpropagation para ajustar sus parámetros internos, en la figura 2-2 se muestra una red neuronal multicapa con una única capa oculta.

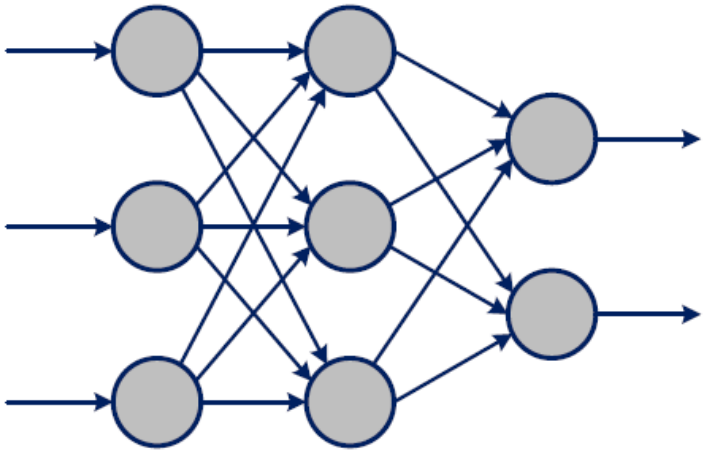


Figura 2-2 Red neuronal multicapa, con una única capa oculta (Berzal, 2018).

La presencia de una única capa oculta ya dota a la red multicapa de su capacidad de aproximador universal (informalmente, su capacidad de aprender cualquier cosa que se pueda aprender). Este tipo de redes fue muy popular desde los años 80 hasta finales del siglo XX.

Las redes multicapa de tipo feed-forward en ocasiones se denominan backpropagation networks porque el algoritmo de entrenamiento que se suele utilizar con ellas está basado en la propagación hacia atrás del error; esto es, en el uso de backpropagation. Las redes multicapa usadas en deep learning tienen una capa de entrada, múltiples capas ocultas y una capa de salida. Desde un punto de vista formal, podemos verlas como una función matemática f que, a partir de un vector de entrada x, obtiene un vector de salida $y = f(x)$. Dado un conjunto de entrenamiento en forma de pares $(x, y)$, el objetivo del algoritmo de entrenamiento es ser capaz de aproximar la función f de forma que para cada posible entrada x se obtenga una salida $\hat{y} = f(x)$ lo más similar posible a la observada en el conjunto de entrenamiento.

### 2.1.2 Redes neuronales convolucionales

Una señal es, en principio, cualquier cantidad física detectable mediante la que se pueden transmitir mensajes. Puede ser una señal de tráfico, la luz de un semáforo, el piloto parpadeante de un coche que va a cambiar de dirección o el gesto de un árbitro señalando el punto de penalty entre las protestas del público en un partido de fútbol. En general, podemos interpretar una señal como una variable que contiene información relevante sobre algún sistema de interés para nosotros. Dicha señal puede ser una función de otras variables. En el caso más sencillo, una señal x puede representar una cantidad que varía a lo largo del tiempo, por lo que podemos representarla mediante la función x(t). Es lo que sucede, por ejemplo, cuando

utilizamos un micrófono para captar un sonido. También podemos trabajar con señales que dependen de varias dimensiones. El ejemplo más habitual, una imagen bidimensional que podemos representar mediante una función $f(x, y)$, donde x e y corresponden a las coordenadas espaciales que nos permiten consultar el valor de la imagen en un punto concreto. Las señales, definidas como funciones, pueden ser continuas o discretas. Las señales continuas pueden resultar convenientes desde el punto de vista matemático al tratarse de funciones de variable real. Sin embargo, dado que trabajamos con computadoras digitales, los sensores utilizados para captar las señales las discretizan, por lo que siempre trabajaremos con señales discretizadas. Una señal discreta está formada por una serie de muestras de la señal.

**La operación de convolución.**

Las redes convolucionales son, simplemente, redes multicapa en las que algunas capas realizan una operación de convolución en lugar de la tradicional multiplicación matricial de entradas por pesos. Matemáticamente, la convolución es una operación matemática que se realiza sobre dos funciones para producir una tercera que se suele interpretar como una versión modificada (filtrada) de una de las funciones originales (Berzal, 2018). La convolución de las funciones f y g, que se suele denotar mediante un asterisco $*$ o una estrella ★, se define como la integral del producto de dos funciones después de que una de ellas se refleje y se desplace:

$$(f * g) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau \tag{2-1}$$

En procesamiento digital de señales, cuando utilizamos señales discretas, la integral anterior se convierte en una sumatoria:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m] = \sum_{m=-\infty}^{\infty} f[n - m]g[m] \tag{2-2}$$

Normalmente, uno de los operandos de la convolución es la señal que deseamos procesar, $x[n]$, y el otro corresponde al filtro, $h[n]$, con el que procesamos la señal. Cuando el filtro es finito y se define sólo sobre el dominio $\{0, 1, \ldots, K - 1\}$, la operación de convolución consiste en, para cada valor de la señal, realizar K multiplicaciones y $K - 1$ sumas:

$$(x * k)[n] = \sum_{k=0}^{K-1} h[k]x[n - k] \tag{2-3}$$

La operación de convolución, que hasta ahora hemos definido sobre funciones de una variable, se puede extender fácilmente al caso multidimensional. En el caso de señales discretas definidas sobre dos variables a las que se aplica un filtro de tamaño $K1 \times K2$, la convolución se calcula utilizando la siguiente expresión:

$$(x * h)[n_1, n_2] = \sum_{k_1=0}^{K_1-} \sum_{k_2}^{K_2-1} h[k_1, k_2]x[n_1 - k_1, n_2 - k_2] \tag{2-4}$$

En procesamiento digital de imágenes, en las que las variables $[n_1, n_2]$ corresponden a las coordenadas $[x, y]$ de los píxeles de una imagen, el signo menos que aparece en la definición de la operación de convolución se suele sustituir por un signo más, de forma que la convolución se calcula como:

$$(x * h)[x, y] = \sum_{k_1=0}^{K_1-} \sum_{k_2}^{K_2-1} h[k_1, k_2] x[x + k_1, y - k_2] \tag{2-5}$$

Técnicamente, no se trata de una convolución, sino de una operación similar denominada correlación cruzada. En el caso discreto para señales reales, la única diferencia es que el filtro utilizado $h[x, y]$ aparece reflejado con respecto a la definición formal de convolución. Dado que la diferencia es mínima, en muchas ocasiones se habla de convolución cuando, realmente, se está calculando una correlación cruzada. Para comprender mejor en qué consiste la operación de convolución, utilicemos un ejemplo sencillo. Supongamos que tenemos una imagen en blanco y negro captada por una cámara. Dicha imagen, de 7×7 píxeles, se puede representar mediante una matriz binaria:

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2-6}$$

Centrado en la imagen aparece un pequeño objeto cuadrado. Veamos qué sucede cuando, a la imagen anterior, le aplicamos un filtro que definimos utilizando la siguiente máscara:

$$H = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{2-7}$$

Sólo tenemos que ir situando dicho filtro sobre diferentes posiciones de la imagen para obtener la imagen filtrada. Por ejemplo, si situamos la máscara del filtro centrado sobre el píxel que corresponde a la esquina superior izquierda del objeto representado en la imagen, sólo tenemos que fijarnos en los píxeles alrededor de esa posición para calcular el resultado de la convolución sobre ese píxel:

$$\begin{bmatrix} - & - & - & - & - & - & - \\ - & 0 & 0 & 0 & - & - & - \\ - & 0 & 1 & 1 & - & - & - \\ - & 0 & 1 & 1 & - & - & - \\ - & - & - & - & - & - & - \\ - & - & - & - & - & - & - \\ - & - & - & - & - & - & - \end{bmatrix} \tag{2-8}$$

Sólo tenemos que ir multiplicando elemento a elemento y sumando el resultado: $0 * 0 - 1 * 0 + 0 * 0 - 1 * 0 + 4 * 1 - 1 * 4 + 0 * 0 - 1 * 1 + 0 * 1 = 4 - 1 - 1 = 2$. En ese cálculo, para el píxel de la esquina superior izquierda del objeto, todos los demás píxeles de la imagen son irrelevantes. Si repetimos el proceso para las todas las posiciones en las que se puede situar la máscara sobre la imagen, obtenemos el resultado de la convolución:

$$I * H = \begin{bmatrix} 0 & -1 & -1 & -1 & 0 \\ -1 & 2 & 1 & 2 & -1 \\ -1 & 1 & 0 & 1 & -1 \\ -1 & 2 & 1 & 2 & -1 \\ 0 & -1 & -1 & -1 & 0 \end{bmatrix} \quad (2\text{-}9)$$

Mediante la convolución hemos conseguido identificar los píxeles de la imagen que corresponden a las esquinas y a las fronteras del objeto que aparecía en la imagen original. Observe que, tal como hemos realizado la convolución, la imagen obtenida como resultado es de tamaño 5×5, más pequeña que la imagen original de 7 × 7. Si, por algún motivo, deseamos obtener una imagen del mismo tamaño que la original, una suposición habitual es asumir que los píxeles de la imagen están rodeados por ceros, tantos como nos hagan falta para poder obtener una imagen de las mismas dimensiones que la original. En nuestro caso, la imagen resultante sería de la forma:

$$I *_{zp} H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 & 0 & 0 \\ 0 & -1 & 2 & 1 & 2 & -1 & 0 \\ 0 & -1 & 1 & 0 & 1 & -1 & 0 \\ 0 & -1 & 2 & 1 & 2 & -1 & 0 \\ 0 & 0 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2\text{-}10)$$

La operación de convolución resulta particularmente interesante en procesamiento digital de señales porque permite construir la salida de un sistema para cualquier señal de entrada arbitraria si conocemos la respuesta impulsiva del sistema (la salida del sistema cuando se introduce un impulso como entrada).

En procesamiento de imágenes, muchas de las operaciones utilizadas en programas de retoque fotográfico se pueden representar de forma compacta definiendo la máscara o kernel del filtro que se aplica sobre la imagen realizando una convolución.

La figura 2-3 se utiliza para ilustrar las operaciones de procesamiento de imágenes que se pueden realizar con una simple convolución.

Figura 2-3 Altgeld Hall, sede del Departamento de Matemáticas de la Universidad de Illinois en Urbana-Champaign(Berzal, 2018).

En la figura 2-4 se muestra la detección de fronteras de la figura 2-3, para los operadores que distinguen entre fronteras horizontales y verticales, la fila superior muestra la detección de fronteras horizontales y la fila inferior reproduce las fronteras verticales detectadas.

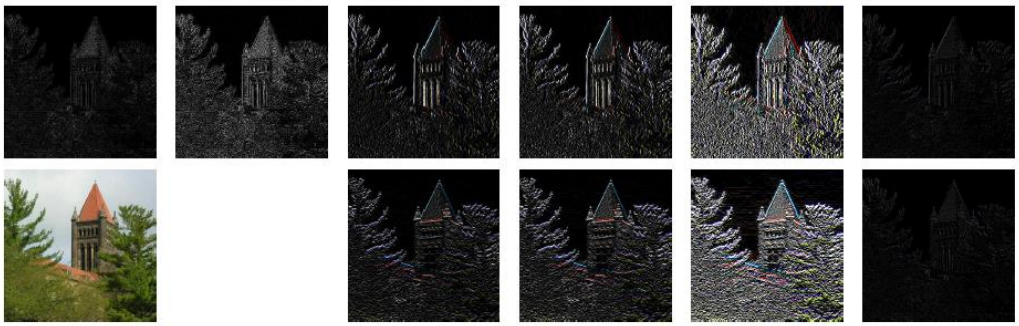


Figura 2-4 Detección de fronteras: Los detectores de fronteras H4, H8, P, SF, S y R (Berzal, 2018).

## 2.2 Interacción humano computadora

Los sistemas humano-máquina se refieren al sistema compuesto por humanos y máquinas y que cumplen algunas funciones a través de la interacción entre humanos y máquinas. Los humanos se refieren a los operadores y gerentes de máquinas y las máquinas se refieren a maquinarias, equipos, herramientas y entornos de trabajo. Todos los sistemas humano-máquina consisten en humanos, máquinas, interfaces entre humanos y máquinas y entornos en los que se ubican los sistemas. Hoy en día, las formas de interacción entre humanos y máquinas en los sistemas humano-máquina se han convertido en las comunicaciones y los diálogos entre los dos "sistemas inteligentes": usuarios y computadoras, que se han desarrollado desde los primeros dispositivos mutuos, incluidos paneles simples y unidades de visualización, hasta la actualidad. dispositivos mutuos con muchos tipos de capacidades de percepción, como la tecnología de seguimiento, la tecnología de identificación por voz, la retroalimentación inteligente, etc. (Chao, 2009), y el portador real de estos dispositivos mutuos es lo que llamamos interfaz humano-computadora, es decir, la interfaz de usuario. La interfaz de usuario se basa en la tecnología de la información, que, a través del modelado, descripción de formalización, aritmética de conformidad, métodos de evaluación y marco de software de los sistemas de información para eventualmente realizar y aplicar la teoría de la interacción humano-computadora. Por tanto, la plataforma de uso de los sistemas humano-máquina, es decir, la investigación sobre el diseño de la interfaz humano-computadora, adquiere mucha más importancia.

**El concepto de interfaz humano-computadora.** Como parte importante de los sistemas humano-máquina, la interfaz humano-computadora es un punto clave de la actividad de diseño de los sistemas, cuya existencia no es la tarea en sí, sino en la situación real de interacción humano-computadora para lograr la comunicación mutua. y establecer la plataforma mutua entre usuarios y máquinas, mediante la cual realizar la operación en las máquinas, es decir, diseñar formas de entrada y salida de información, para lograr así una función sólida de interacción humano-computadora. De ahí que la clave del diseño de la interfaz esté en cómo lograr una perfecta y armoniosa interacción humano-computadora; sólo así se podrá reducir fundamentalmente la carga cognitiva de las personas y potenciar las capacidades a perceptivas y operativas de los usuarios. Para los usuarios, la interfaz es el sistema. Simplemente hablando, la interfaz se compone de un panel de control que consta de pantalla y controlador, pantalla táctil incorporada para controlar y mostrar, interfaz de software y muchos tipos de interfaces que pueden usarse en algunos productos o sistemas complicados.

# Capitulo 3. Estado del arte

Dado que los diferentes tipos de redes neuronales pueden mejorar muchos procesos, e incluso realizar algunos otros que los seres humanos no pueden, es importante tener plataformas versátiles de bajo costo, en las que se puedan desarrollar redes neuronales funcionales. La mayoría de las veces se utilizan computadoras con gran capacidad de procesamiento para desarrollar y entrenar redes neuronales, pero en algunos casos se necesitan sistemas más pequeños que sean capaces de mejorar el desempeño de máquinas o robots ya existentes y que se puedan acoplar sin representar un problema de tamaño o incluso de costo energético.

El propósito de la revisión es el análisis del progreso en esta línea de investigación, que sistemas embebidos existen, que limitantes tienen, que tipo de redes neuronales son capaces de trabajar, en que lenguaje de programan, si es un lenguaje abierto o con derechos de autor, si cuentan con un lenguaje específico de dominio (DSL) y como lo diseñaron, como miden su porcentaje de eficiencia, que proceso siguen para crear el sistema multipropósito.

Un DSL es un lenguaje informático especializado en un dominio de aplicación particular. La línea divisoria entre lenguajes de uso general y lenguajes de dominio específico no siempre es nítida, ya que un lenguaje puede tener características especializadas para un dominio en particular, pero ser aplicable de manera más amplia, o, por el contrario, puede en principio ser capaz de una amplia aplicación, pero en la práctica usado principalmente un dominio específico.

A continuación, se mencionan las etapas consideradas a seguir para asegurar una correcta revisión sistemática del estado del arte:

1. Búsqueda y selección de congresos y revistas que publiquen trabajos en los campos relacionados con el tema de investigación.
2. Búsqueda de artículos utilizando palabras clave.
3. Revisión y clasificación de los artículos.
4. Extracción de información de los artículos.
5. Integración de la información comparando las bases de datos, modelos, métricas de evaluación y resultados.

Se realizo una búsqueda de foros de publicación con la finalidad de seleccionar congresos y revistar que aborde los problemas relacionados con redes neuronales en sistemas embebidos.

Para la búsqueda de artículos se seleccionaron palabras clave que permita encontrar artículos de interés relacionados con el tema de investigación, las palabras clave seleccionadas son las siguientes: FPGA, neural networks, embedded system, multipurpose, multipurpose architecture, multicontext, adaptation, reconfigurable, domain specific language, custom processor.

Para la búsqueda de artículos se seleccionaron tres bases de datos, Web Of Science (Journal Citation Reports) y Elsevier (Scopus) ya que son las dos bases de datos más importantes de la comunidad científicas, y IEEE Xplore ya que en esa base de datos se publica sobre problemas que relacionan la electrónica y la computación como es el caso del tema de investigación.

## 3.1 Redes neuronales

La IA es una de las áreas clave de investigación en informática. Con su rápido avance tecnológico y su vasta área de aplicación, la IA se está volviendo omnipresente muy rápidamente debido a su robusta aplicabilidad en los problemas, particularmente que no pueden ser resueltos bien por humanos, así como por las estructuras informáticas tradicionales.

### 3.1.1 Redes convolucionales

Si existe un tipo particular de red neuronal artificial que marca la diferencia en la práctica, ése es precisamente el correspondiente a las redes que se utilizan para procesar señales: las redes convolucionales. Su éxito en la resolución de problemas de visión artificial que, hasta hace poco, se consideraban casi intratables, sirvió para volver a poner de moda las redes neuronales en IA (Berzal, 2018). Los recientes avances en las CNN profundas han llevado a grandes mejoras en la precisión de los sistemas auditivos y de visión. Caracterizados por sus estructuras profundas y una gran cantidad de parámetros, las CNN profundas desafían el rendimiento computacional de hoy. La mayoría de los trabajos encontrados se enfocan em implementar y mejorar este tipo de red particular.

En las redes convolucionales la mayor parte de la carga de trabajo computacional se puede convertir en multiplicaciones matriciales, en (Qiao et al., 2017) adoptan una arquitectura aceleradora basada en multiplicadores matriciales. Las unidades dedicadas están diseñadas para eliminar la sobrecarga de conversión. También diseñaron un sistema de memoria personalizado de acuerdo con el patrón de acceso a la memoria de las CNN utilizando un FPGA Xilinx Zynq-zq7045. Los resultados experimentales muestran que, para una aplicación típica de CNN, clasificación de imágenes, se logra un promedio de 77.8 GFLOPS, mientras que la eficiencia energética es 4.7x mejor que la GPGPU Nvidia K20.

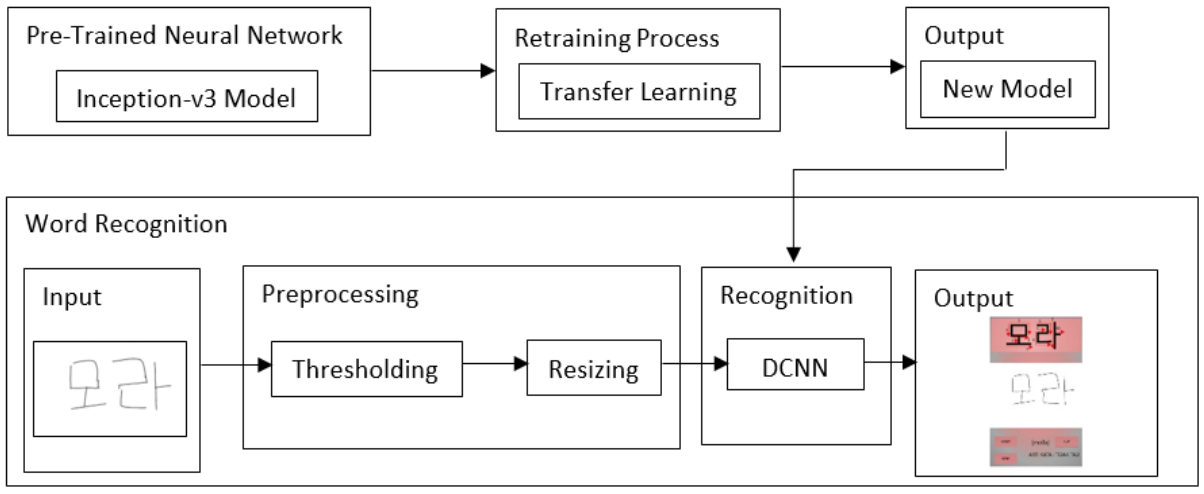



Figura 3-1 Arquitectura general (Purnamawati et al., 2018).

Las redes convolucionales no solo se pueden implementar en un FPGA, en (Purnamawati et al., 2018) la implementan en el sistema operativo para smartphones Android. Para adquirir el idioma coreano, cada alumno debe ser capaz de comprender el carácter coreano no latino. Es necesario llevar a cabo un enfoque digital para facilitar el proceso de aprendizaje de Corea. Este estudio se realiza utilizando la CNN profundas. CNN profundas realizan el proceso de reconocimiento de la imagen en función del modelo que se ha entrenado. Posteriormente, se lleva a cabo un proceso de reentrenamiento que utiliza la técnica de aprendizaje de transferencia con el valor entrenado y reentrenado del modelo para desarrollar un nuevo modelo con un mejor rendimiento sin errores sistémicos específicos figura 3-1. La precisión de las pruebas de esta investigación resulta en 86,9%.

Otras plataformas donde se ha implementado las redes convolucionales son en una Raspberry Pi en (Lu et al., 2019) donde se propone un método de diagnóstico de falla del motor insitu implementando un modelo CNN mejorado en un sistema embebido diseñado que consiste en un Raspberry Pi y un circuito de adquisición y procesamiento de señal. El SBC (computadora de placa única) se comunica con la MCU y la computadora portátil a través del bus transmisor receptor asíncrono síncrono universal (USART). Aquí, la computadora portátil se utiliza para recopilar los resultados de la inferencia de CNN para las estadísticas, y la computadora portátil no es necesaria en el reconocimiento in situ de fallas del motor.

Es posible implementar redes neuronales en CPU en sistemas embebido como en (Hu et al., 2017) que utilizan un CPU basada en Cortex-A53 ARMv8 de 1 GHz, este artículo se concentra en la optimización de CNN en plataformas integradas con un estudio de caso de detección de peatones en ADAS. El modelo CNN ha sido entrenado con GPU localmente y luego transformado en una implementación eficiente en plataformas integradas. La implementación eficiente utiliza una escala de red dramáticamente pequeña y se obtiene una CNN liviana. Específicamente, Los parámetros de la red se comprimen mediante la adopción de pesos enteros para reducir la complejidad computacional. Mientras tanto, también se han propuesto otras optimizaciones para adaptar la arquitectura general del procesador ARM.

### 3.1.2 Función de activación

Una parte importante de las redes neuronales con las funciones de activación en (Hajduk, 2017) se presenta brevemente un método de implementación de FPGA de las funciones de activación tangente hiperbólica y sigmoidea para redes neuronales artificiales. Se propone una especie de implementación directa de las funciones. Los resultados de la implementación muestran que la precisión obtenida del método es relativamente alta en comparación con otras soluciones publicadas.

### 3.1.3 Redes neuronales de perceptrón multicapa

Las redes multicapa usadas en deep learning tienen una capa de entrada, múltiples capas ocultas y una capa de salida. Desde un punto de vista formal, podemos verlas como una función matemática f que, a partir de un vector de entrada x, obtiene un vector de salida y = f(x).

Este tipo de red neuronal tiene muchas aplicaciones para resolver problemas que no son tan complejos en comparación con problemas de visión, un buen ejemplo de las aplicaciones de estas redes se presenta en (Baehr, S. Skambraks, S. Neuhaus, S. Kiesling, C. Becker, 2017) .El fondo que se origina de eventos fuera del punto de interacción jugará un papel importante en el próximo experimento Belle II. Para reducir este fondo, se utiliza un activador de seguimiento basado en la reconstrucción de la posición z de un evento en los FPGA. Este artículo presenta la arquitectura e implementación de redes neuronales y el preprocesamiento de soporte que se utilizará en la próxima prueba de rayos cósmicos de Belle II. Al utilizar la información de éxito de las simulaciones, se presentan cifras de mérito como latencia y demanda de

recursos. En (Szadkowski, Zbigniew Pytel, 2015) se presenta un disparador basado en una red neuronal artificial canalizada implementada en un gran FPGA que, después del aprendizaje, puede reconocer diferentes tipos de formas de onda de los detectores de superficie Pierre Auger. La estructura de un algoritmo de red neuronal artificial que se está desarrollando en una plataforma MATLAB se ha implementado en la lógica rápida de la FPGA Cyclone V. En (Guellal et al., 2015) se propone un nuevo algoritmo PWM en línea basado en la teoría de la Red Neural Artificial (ANN) en combinación con el PWM SHE. Se presenta una implementación FPGA del algoritmo propuesto para generar los ángulos de conmutación para validar este algoritmo en una aplicación en tiempo real. Los resultados obtenidos muestran que el algoritmo ANNSHE PWM propuesto controla el voltaje fundamental y elimina de manera eficiente los armónicos deseados en tiempo real y en todo el rango de variación de velocidad. En la tomografía por emisión de positrones (PET) de animales pequeños, las mejoras en la resolución espacial dependen de la minimización del tamaño del detector y, a menudo, se reducen la fracción fotoeléctrica del detector. En (Geoffroy et al., 2015) se demuestra la capacidad de procesar tripletas en tiempo real usando un ANN implementado en el campo de compuerta de puerta programable (FPGA). La arquitectura canalizada ANN puede procesar más de 1 millón de trillizos / segundo utilizando menos de 6000 cortes FPGA. El procesamiento en tiempo real en el escáner LabPET I produjo un aumento general del 39.7% en la eficiencia de detección en relación con la configuración tradicional de alta resolución.

En los artículos anteriores se desarrolló el firmare para cada FPGA de manera específica para cada caso, en (Hajduk, 2018) se presenta dos implementaciones de redes neuronales artificiales de retroalimentación en FPGA. Las implementaciones difieren en los requisitos de recursos FPGA y la velocidad de los cálculos. Ambas implementaciones ejercen aritmética de punto flotante, aplican la realización de funciones de activación de muy alta precisión y permiten una fácil alteración de la estructura de la red neuronal sin la necesidad de una reimplementación de todo el proyecto FPGA.

## 3.2 Lenguajes de dominio específico

Un lenguaje de dominio específico (DSL) normalmente es menos complejo que un lenguaje de propósito general, como Java o C. Generalmente, los DSL se desarrollan en estrecha coordinación con los expertos del campo para el que se diseña. En muchos casos, los DSL están destinados no para ser usados por expertos en software, sino por no programadores que son versados en el dominio de aplicación del DSL.

Para la evaluación cualitativa de los lenguajes específicos de dominio nos basamos en (Johanson & Hasselbring, 2017; Kahraman & Bilgen, 2015; Slivnik, 2016), se presenta la evaluación de los artículos consultados a fin de compararlos y tener una noción del estado de este tema, y se presenta un resumen de cada artículo en caso de que el lector tenga interés en algún trabajo.

A continuación, se enlistan las características básicas que debería tener un DSL:

1. Idoneidad funcional: la idoneidad funcional se refiere al grado en que un DSL está completamente desarrollado. Esto significa que toda la funcionalidad necesaria está presente en el DSL. Por otro lado, DSL no debería incluir funcionalidad que no esté en el dominio. Usamos esta característica para cubrir la corrección, integridad, falta de comprensión del dominio, incompletitud y adecuación del dominio.
2. Usabilidad: La usabilidad de un DSL es el grado en que usuarios específicos pueden utilizar un DSL para lograr objetivos específicos. Un DSL debe ser lo más simple posible para expresar los conceptos del dominio y apoyar a sus usuarios. Debe evitarse el uso de símbolos demasiado simples o similares o poco atractivos . Consideramos comprensibilidad, transparencia semántica, ajuste cognitivo, manejo de la complejidad, expresividad visual, comprensibilidad, adecuación, capacidad de aprendizaje, esfuerzo de adopción, esfuerzo requerido para construir modelos, transparencia,

economía espacial, facilidad de escritura y legibilidad y simplicidad, todo bajo el título de usabilidad.

3. Fiabilidad: La fiabilidad de un lenguaje se define como la propiedad de un lenguaje que ayuda a producir programas fiables. El soporte de DSL para la prevención de errores y la verificación de modelos es señalado como una cualidad significativa.
4. Mantenibilidad: el grado en el que un idioma es fácil de mantener. Los DSL se pueden modificar y se pueden agregar nuevos conceptos y extensiones de conceptos. La mantenibilidad cubre la comprensibilidad y la modificabilidad en este estudio. La modificabilidad se puede describir como la cantidad de esfuerzo requerido para modificar el DSL para proporcionar una funcionalidad diferente o adicional. Consideramos el modularidad también bajo esta característica.
5. Productividad: la productividad de un DSL se refiere al grado en que un lenguaje promueve la productividad de la programación. La productividad es una característica relacionada con la cantidad de recursos gastados por el usuario para lograr objetivos específicos.
6. Extensibilidad: el grado en que un lenguaje tiene mecanismos generales para que los usuarios agreguen características. La escalabilidad es otra sub-característica que se maneja en la extensibilidad.
7. Compatibilidad: el grado en el que un DSL es compatible con el dominio y el proceso de desarrollo. Consideramos la compatibilidad de procesos bajo el título de compatibilidad. Es el grado de un DSL para encajar en un proceso, ya que DSL se usa como parte de un proceso de desarrollo con fases y roles.
8. Expresividad: el grado en que una estrategia de resolución de problemas se puede mapear en un programa de forma natural. En otras palabras, la expresividad es la relación entre el programa y lo que el programador tiene en mente. La expresividad se señala como una de las principales características de las DSL. La unicidad es el principio que puede definirse como la sub-característica del lenguaje que proporciona una y sólo una buena manera de expresar cada concepto de interés. Debe haber una correspondencia uno a uno entre los conceptos y su representación en el lenguaje. Se debe evitar duplicar los conceptos y la semántica de los lenguajes de programación tradicionales, elegir el paradigma de representación incorrecto y utilizar bibliotecas como lenguaje y seleccionar el nivel de abstracción adecuado para no utilizar conceptos demasiado genéricos o específicos. Usamos esta característica para implicar también ortogonalidad, lo que significa que cada construcción del lenguaje se usa para representar exactamente un concepto distinto en el dominio. Economía gráfica, conformidad y consistencia son otras características que se manejan en expresividad.
9. Reutilización: el grado en el que una construcción de lenguaje se puede utilizar en más de un idioma. La reutilización se refiere a qué partes de un DSL se reutilizan desde o por otros DSL.
10. Integralidad: grado en el que un idioma se puede integrar con otros idiomas. DSL se puede integrar con otros lenguajes utilizados en el proceso de desarrollo. En (Amatriain & Arumi, 2011) se muestra un caso de estudio de un metamodelo para el dominio multimedia, el enfoque propuesto en este flujo de trabajo se basa en la aplicación de las mejores prácticas conocidas para el desarrollo de marcos. En particular, basan su modelo de proceso en los siguientes principios:
    - pequeña inversión inicial
    - desarrollo iterativo e incremental, diseño de marco
    - impulsado por aplicaciones
    - luchar por un comportamiento de caja negra

En (Voelter et al., 2019) se busca demostrar que los DSL pueden ser usados en situaciones críticas, se definen dominios de seguridad, estándares y herramientas, bancos de trabajo de los idiomas y verificación y validación de los lenguajes, asegurar la exactitud.

En (Nazir et al., 2017) se crea un DSL para los sistemas de gestión de eventos e información de seguridad, se trabajó en la especificación formal del lenguaje, con un modelo meta-pila de 3 niveles definidos, se define una sintaxis formal, se evalúan las reglas y su exactitud de aplicación.

En (Contreras et al., 2016) se crea un lenguaje específico de dominio (DSL) para sistemas ubicuos, este DSL se ha desarrollado utilizando un enfoque basado en modelos. Como en cualquier otro lenguaje, ya sea de propósito general o específico de un dominio, U-DSL comparte una serie de características cuya comprensión es básica para su desarrollo:

- Sintaxis abstracta
- Sintaxis concreta
- Semántica

En (Tran Tan et al., 2016) se busca proporcionar una generación automática de código de dominio especifico de alto rendimiento, se centra en tres esqueletos orientados a datos:

- transformar que aplica una operación arbitraria a cada (o cierto) elemento (s) de una entrada y almacena el resultado en una salida.
- doblez que aplica una reducción parcial de los elementos de una entrada mesa a un dado dimensión de la tabla y almacena el resultado en una salida.
- escanear que aplica un escaneo de prefijo de los elementos de una entrada mesa a una mesa dada dimensión y almacena el resultado en una salida mesa.

En (Wang & Li, 2016) se crea un DSL para aplicaciones basadas en cuadrículas paralelas, el programa GridFOR consta de dos partes: subrutinas y transformaciones. Una subrutina GridFOR se construye sobre una estructura de cuadrícula básica declarada. Una estructura de cuadrícula representa un finito norte-espacio discreto dimensional y está parametrizado por el tamaño de cada dimensión.

En (Menotti et al., 2012) se crea un lenguaje para programar motores de aceleración personalizados basados en FPGA, El lenguaje permite el uso de directivas (en forma de construcciones específicas) para guiar la sincronización y la generación de la canalización. El compilador LALP usa estas directivas para determinar, siempre que sea posible, el número de ciclos de reloj y otros parámetros que asegurarán la exactitud del código que se está ejecutando. Cuando se omiten las directivas, el compilador intenta inferir esos parámetros. Sin embargo, tenga en cuenta que siempre que el compilador LALP no puede determinar los ciclos de reloj que garantizarían la corrección, advierte al usuario sobre eso.

En (Delaval & Rutten, 2007) se crea un lenguaje de programación simple, específico para el dominio de los sistemas de control multitarea en tiempo real, como los sistemas robóticos, automotrices o de aviónica, la capa de computación realiza transformación de datos de algoritmos por ejemplo cálculos numéricos en un bucle, puede ser empezado, detenido, suspendido y notificar. La capa de control gestiona los inicios de los cálculos y se detiene encapsulándolos en tareas, luego estas tareas pueden convertirse en aplicaciones utilizando estructuras como bucles, secuencias o enunciados paralelos.

En (Tôn & Truong, 2016) el objetivo es construir un DSL para el desarrollo de juegos que integre lenguajes similares a secuencias de comandos en modelos conceptuales, que es análogo a la forma en que Object Constraint Language (OCL) pone más semántica en los modelos UML (pero el suyo es esencialmente en un nivel más alto de abstracción que OCL / UML).

En (Borelli et al., 2020) proponen un lenguaje llamado BIoTA (Buildout IoT Application Language), un DSL cuyo objetivo es ayudar y agilizar la construcción de arquitecturas de software para IoT. La

especificación e implementación del lenguaje BIoTA involucran una gramática y un compilador, responsables del análisis sintáctico y semántico, así como de la generación de código.

# Capitulo 4. Diseño de la metodología de investigación

La metodología utilizada en esta investigación es hibrida ya que la investigación por realizar es cuantitativa y descriptiva. La investigación por realizar es del tipo cuantitativa debido a la naturaleza del problema, el resultado esperado es la obtención de una plataforma capaz de asistir en la creación de redes neuronales, por lo tanto, se puede medir su desempeño con métricas establecidas. Dado que se busca un resultado único se contempla una investigación descriptiva, esto nos permite llegar a un único resultado, y es posible medir la exactitud de este, se definen los objetivos a alcanzar y la metodología para llegar a ellos.

Tanto la investigación, como la experimentación se llevará a cabo en el laboratorio de robótica de la maestría en Ciencias de la Electrónica, debido a que en este laboratorio se cuenta con diferentes robots con los cuales se puede poner a prueba la plataforma a desarrollar.

## 4.1 Metodología de investigación

Las redes neuronales consideradas en este trabajo son: red neuronal multicapa y red neuronal convolucional. esta selección a partir de los trabajos reportados en el estado del arte se encontró que son las redes neuronales que más se utilizan en un sistema embebido.

Los resultados que se obtengan servirán para ajustar la metodología propuesta y así obtener un mejor rendimiento en el desempeño de las redes neuronales y de las estrategias que utiliza la plataforma.

Para resolver este problema se propone una metodología que consta de 5 pasos mostrados en el diagrama de la figura 4-1.

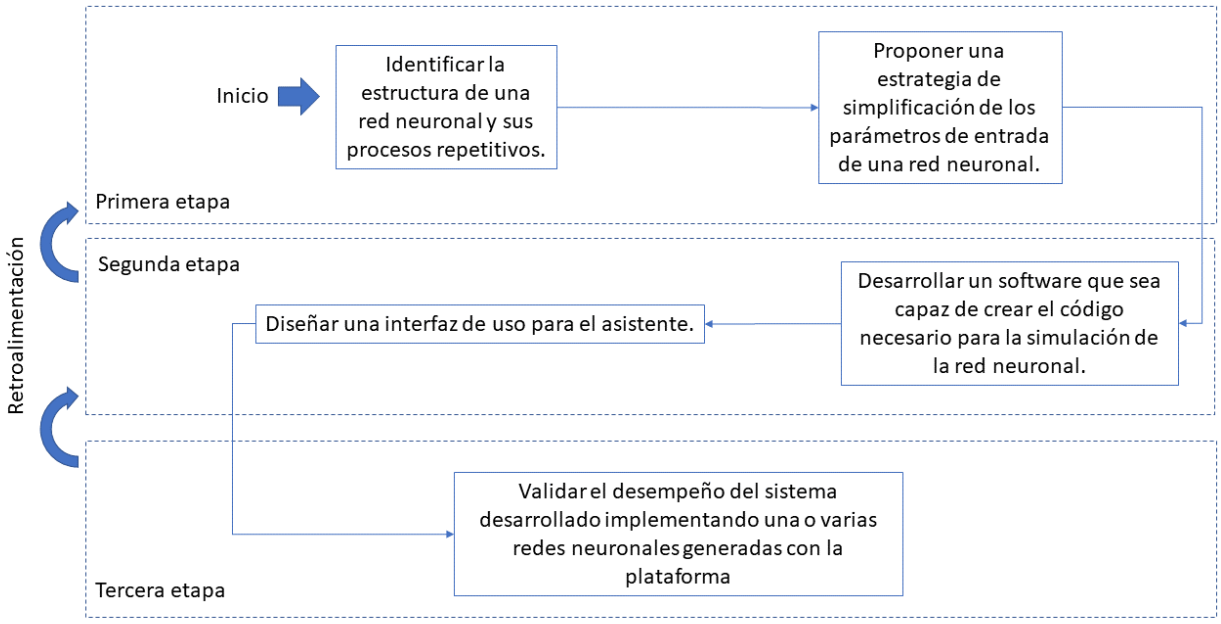



Figura 4-1 Diagrama metodológico.

### 4.1.1 Identificación de la estructura de las redes neuronales

Se analiza la estructura de las redes neuronales que la plataforma será capaz de asistir (red neuronal multicapa y red neuronal convolucional) para identificar los procesos necesarios en su desarrollo, esto permite conocer los procesos repetitivos y desarrollar estrategias para el asistente.

Las unidades de procesamiento se organizan en capas. Hay tres partes normalmente en una red neuronal que se presentan en la figura 4-2: una capa de entrada, con unidades que representan los campos de entrada; una o varias capas ocultas; y una capa de salida, con una unidad o unidades que representa el campo o los campos de destino. Las unidades se conectan con fuerzas de conexión variables (o ponderaciones). Los datos de entrada se presentan en la primera capa, y los valores se propagan desde cada neurona hasta cada neurona de la capa siguiente. al final, se envía un resultado desde la capa de salida.

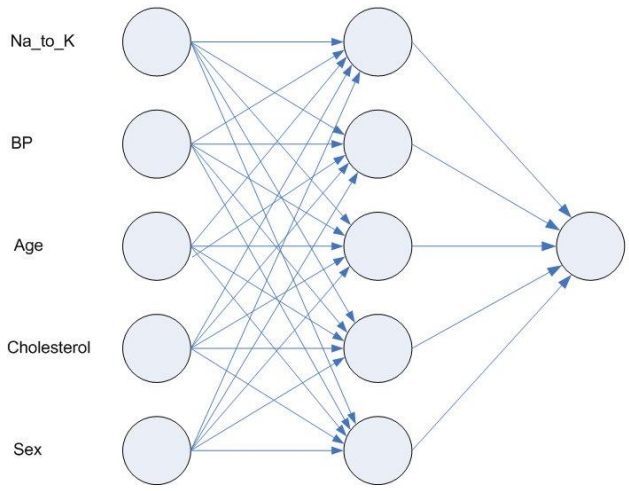



Figura 4-2 Estructura de una red neuronal.

### 4.1.2 Estrategia de simplificación

Se desarrollará una estrategia para la simplificación de los datos de entrada que proporciona el usuario, para facilitar el uso de la plataforma, así como la selección de los parámetros y variables de cada red neuronal.

Las variables con las que se va a trabajar son: número de entradas de la red neuronal, numero de capas ocultas, tipo de datos, en el caso de la red convolucional, numero de convoluciones a realizar a las imágenes, filtro o kernel a aplicar a la imagen.

La estrategia de centra en permitir modificar las variables con las que se va a trabajar cada red neuronal, para que el ajuste en la misma no requiera modificar toda la estructura de la red neuronal. Si se quiere agregar una capa o desconectar alguna conexión entre alguna neurona en específico el software lo realice de manera automática.

### 4.1.3 Diseño de interfaz de uso de la plataforma

Al diseñar la interfaz de usuario se tiene que considerar la experiencia de usuario y la usabilidad de la plataforma.

Función: En el hardware, las señales electrónicas activan diferentes situaciones. Los datos se escriben, se leen, se envían, se reciben, se comprueban errores, etc.

En el software, las instrucciones activan el hardware a través de protocolos de enlace de datos, métodos de acceso, etc.

Fácil de usar: Si el producto carece de usabilidad nadie lo deseará. La facilidad con la que alguien utiliza el producto es lo que logrará el objetivo deseado.

Se debe considerar la usabilidad inherente de las interfaces para poder comprender y usar el sistema subyacente. La usabilidad debe ser sencilla si se desea que las personas lo usen ampliamente.

Fácil de aprender: Toda interfaz se debe diseñar para que sea intuitiva y familiar, ya que los usuarios después de usar un producto no recordarán realmente todas las funciones. Para reducir la complejidad, la interfaz debe tener consistencia, además de ser previsible.

Retroalimentación y tiempo de respuesta: La retroalimentación es clave para el diseño de la interfaz. El producto debe comunicarse con los usuarios brindando una retroalimentación cuando se realice la tarea deseada y sobre qué se debe hacer a continuación.

El tiempo de respuesta en la retroalimentación también es un factor clave. Debe ser en tiempo real y de respuesta inmediata, dentro del rango entre 0,1 segundos y 5 segundos.

### 4.1.4 Desarrollar un software que sea capaz de crear el código necesario para la simulación de la red neuronal

Desarrollando un software que produzca el código para Matlab permite tener el control al momento de desarrollar la red neuronal, esto siguiendo los parámetros del usuario.

Es necesario tener especial cuidado en la sintaxis del código creado.

Al desarrollar el código se le da la posibilidad al usuario de revisarlo y modificarlo de ser necesario.

### 4.1.5 Validar el desempeño del sistema desarrollado implementando una o varias redes neuronales generadas con la plataforma

Generar diferentes redes neuronales utilizando la plataforma para generar el código de ejecución y de entrenamiento permite validad el desempeño de esta.

Al proponer varias estructuras de redes neuronales y hacer una comparación de su desempeño se reafirma la utilidad de la plataforma ya que el usuario no tiene que hacer el cambio en el código de ejecución y entrenamiento de cada estructura, esto permite al usuario probar diferentes estructuras de redes neuronales y elegir la mejor.

## 4.2 Modelado en lenguaje UML

Un modelo es una representación en un cierto medio de algo en el mismo u otro medio. El modelo capta los aspectos importantes de lo que se está modelando, desde un cierto punto de vista, y simplifica u omite el resto. La ingeniería, la arquitectura y muchos otros campos creativos utilizan modelos.

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual de propósito general que se utiliza para especificar, visualizar, construir y documentar los artefactos de un sistema software. Captura decisiones y conocimiento sobre sistemas que deben ser construidos. Se usa para comprender, diseñar, ojear, configurar, mantener y controlar la información sobre tales sistemas (Booch et al., 2006).

UML también contiene construcciones organizativas para agrupar los modelos en paquetes, lo que permite a los equipos de software dividir grandes sistemas en piezas con las que se pueda trabajar, comprender y controlar las dependencias entre paquetes y gestionar las versiones de las unidades del modelo, en un entorno de desarrollo complejo.

Ante todo, UML no es un lenguaje de programación. Puede ser utilizado para escribir programas, pero carece de las facilidades sintácticas y semánticas que proporcionan la mayoría de los lenguajes de programación para facilitar la tarea de programar. Las herramientas pueden proporcionar generadores de código para UML sobre diversos lenguajes de programación, así como construir modelos de ingeniería inversa a partir de programas existentes. UML no es un lenguaje altamente formal pensado para demostrar teoremas.

En la figura 4-3 se presenta el diagrama de componentes que muestra los componentes de un sistema, es decir, las unidades software con las que se construye la aplicación, así como las dependencias entre componentes

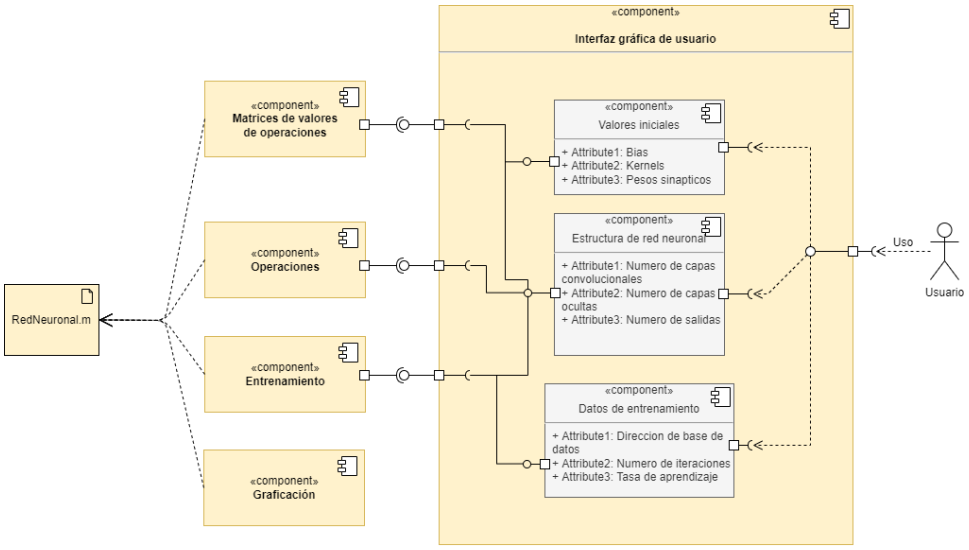



Figura 4-3 Diagrama de componente.

En la figura 4-4 se muestra el diagrama de secuencia, el cual muestra una interacción como un gráfico de dos dimensiones. La dimensión vertical es el eje de tiempo, que avanza hacia el final de la página. La dimensión temporal muestra los roles que representan objetos individuales en la colaboración.

En la figura 4-5 se presenta el diagrama de casos de uso, este representa la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso) , además de la forma en como un usuario (Actor) opera con el sistema en desarrollo.

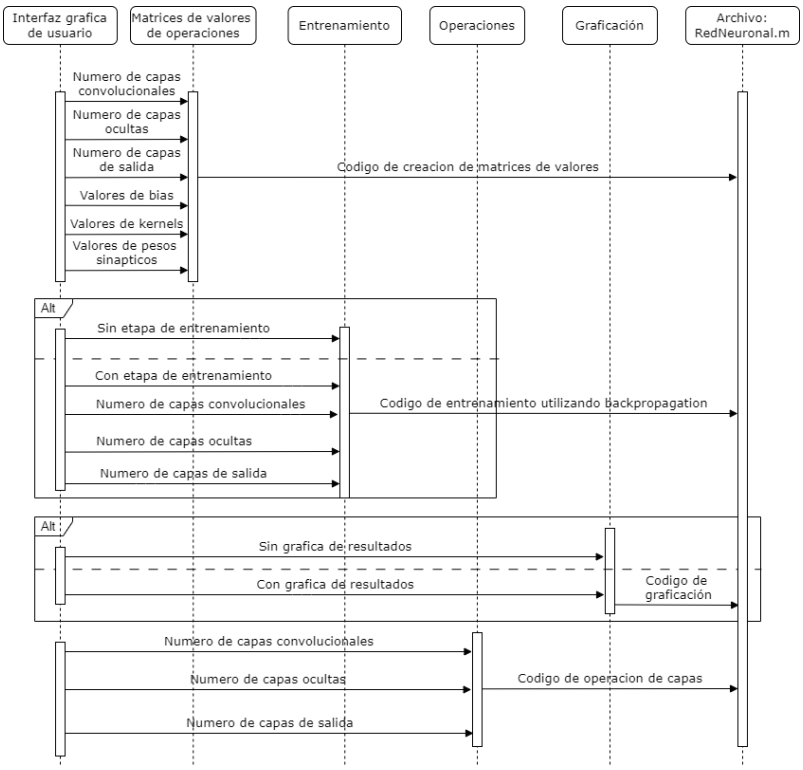



Figura 4-4 Diagrama de secuencia.

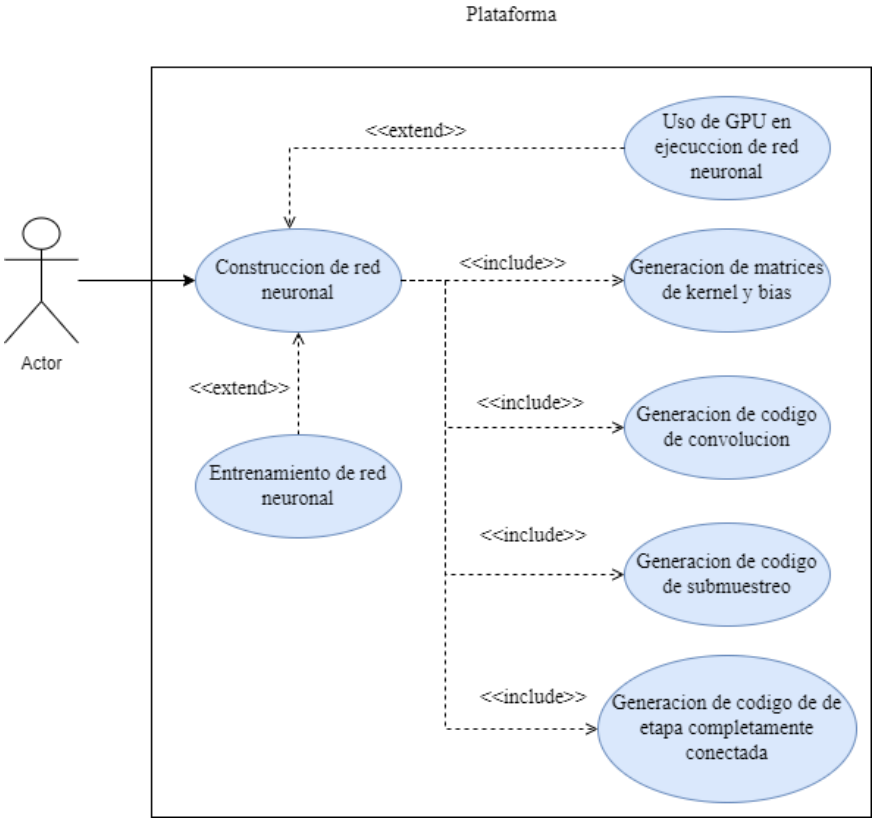



Figura 4-5 Diagrama de casos de uso.

# Capitulo 5. Estructura de las redes neuronales

En este capítulo se analizan las redes neuronales que el asistente será capaz de trabajar, se analiza su estructura en busca de oportunidades para reducir el trabajo del diseñador de la red por medio de la asistencia en las diferentes etapas del desarrollo de la red neuronal.

## 5.1 Identificación de las partes de una red neuronal convolucional

Para el análisis de la red neuronal convolucional se utilizó una red desarrollada para realizar la identificación paramétrica de un robot, de un trabajo de doctorado en desarrollo, esta red cuenta con 6 capas, 3 convolucionales y 3 completamente conectadas, el diagrama general de la red neuronal se muestra en la figura 5-1.

Se busca analiza este caso particular de una red convolucional y llevarlo a al caso más general posible para que el asistente sea capaz de ayudar en el desarrollo de cualquier red neuronal convolucional.

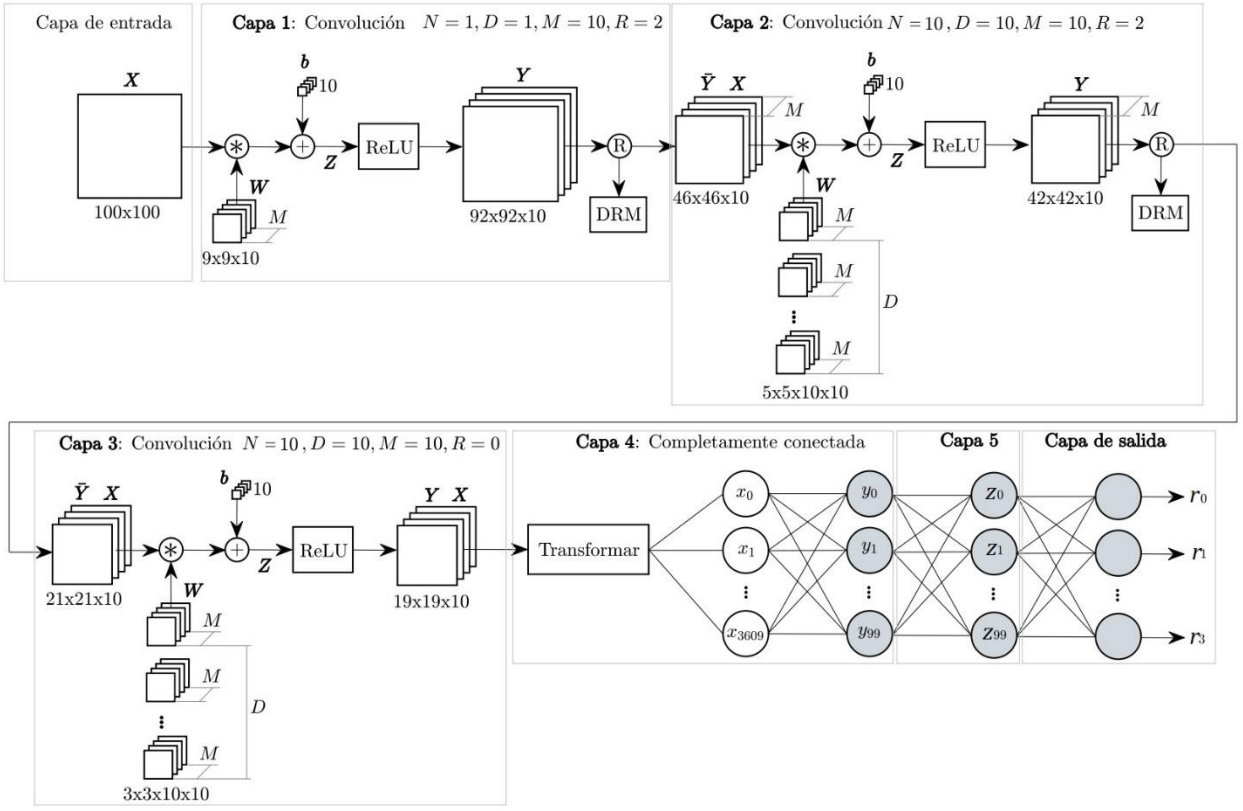



Figura 5-1 Diagrama general de red neuronal convolucional.

La capa de entrada importa la información de una imagen, en este caso de 100 x 100. Para el asistente se propone que sea capaz de tener n imágenes de entrada o en caso de tener una imagen para ingresar a la red neuronal, proporcionar una manera de convertir los datos que el usuario tenga en una imagen por medio de una multiplicación de vectores y así tener una matriz que sería la imagen de entrada, en este caso en particular donde se busca la identificación paramétrica los vectores de entrada para la construcción de la imagen tendrían implícito el modelo dinámico del robot, pero en el caso general del asistente puede contener cualquier tipo de datos con los que el usuario quiera trabajar.

La primera operación que se realiza en capa 1 es una convolución con una matriz W que contiene diez filtros de 9x9, el asistente debe ser capaz de permitirle al usuario realizar la convolución con n cantidad de filtros de tamaño axb. En la figura 5-2 se ilustra la operación de convolución, se toma una ventana de la imagen de entrada X y se multiplica por la matriz que contiene el filtro W, se suman los valores obtenidos y se obtiene el elemento de la matriz C que contiene los elementos ya convolucionados.

Estas operaciones son repetitivas ya que se toman las n imágenes de entradas y se va realizando la convolución una por una con los n filtros, en esta operación el usuario solo deberá indicar la cantidad de filtros, el tamaño de los filtros y el contenido de cada filtro.

Se busca que el asistente facilite lo máximo posible el desarrollo de las redes neuronales, para tal objetivo se debe diseñar una interfaz que asista en la selección de los filtros, ya que en este paso se pueden llegar a tener que seleccionar una gran cantidad de filtros con redes neuronales grandes, por lo tanto, se debe de tener la mayor cantidad posible de filtros predeterminados y de conjuntos de filtros.

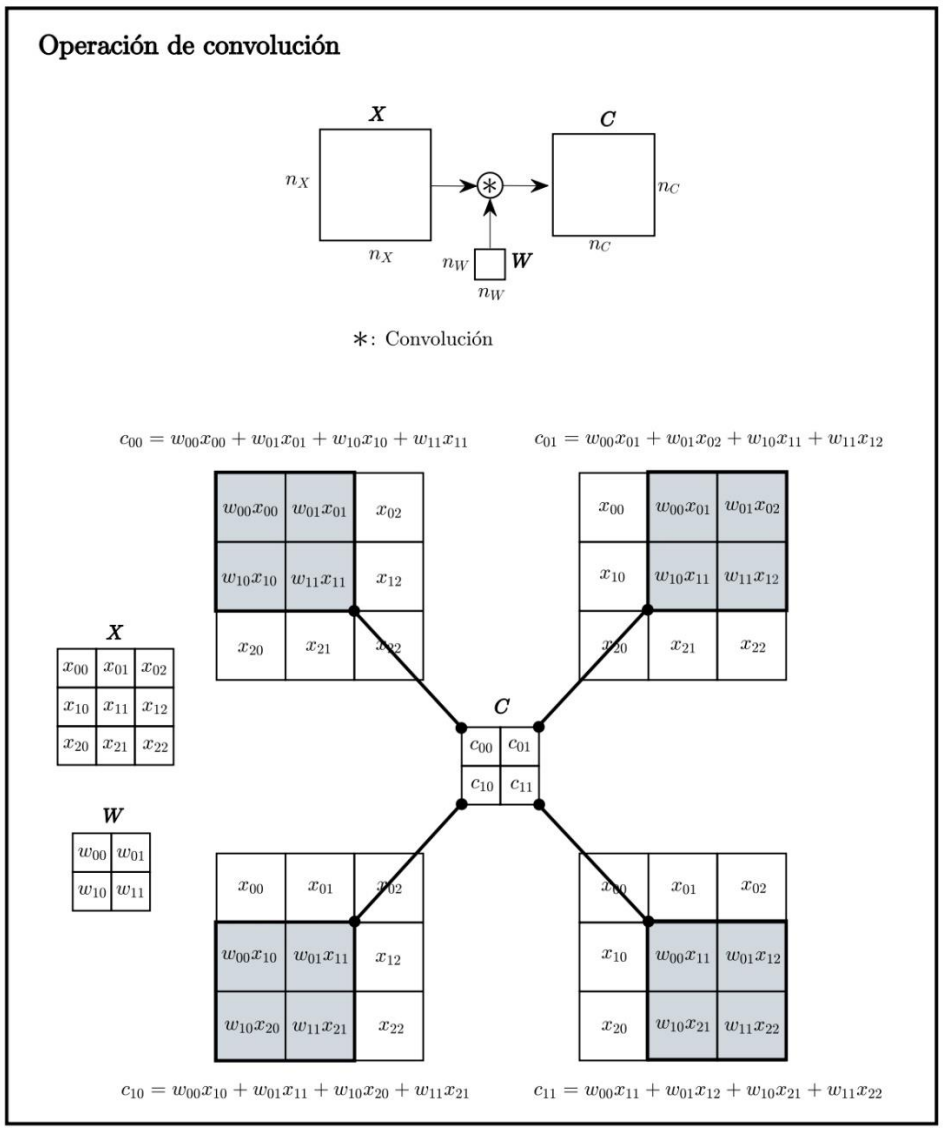



Figura 5-2 Operación de convolución.

Después de obtener la matriz C se suma con la matriz b, que debe ser de la misma dimensión que C, esta contiene los valores de bias como se ilustra en la figura 5-3, estos valores controlan qué tan predispuesta está la neurona a disparar un 1 o un 0 independiente de los pesos.

Este proceso debe ser automático en el asistente, y la interfaz deber permitir al usuario importar la matriz b, ingresarla manualmente, que se seleccione aleatoriamente o tener valores ya prediseñados para diferentes fines.

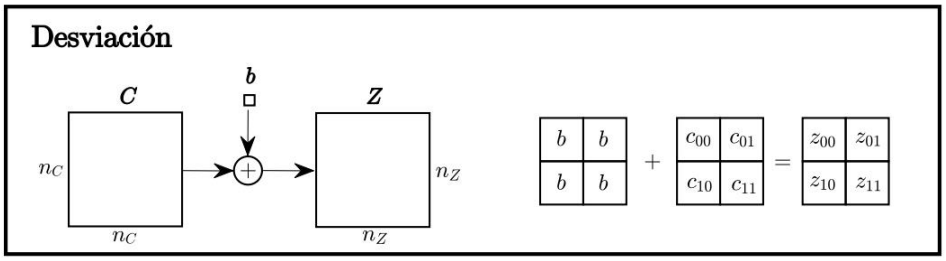



Figura 5-3 Operación de desviación.

El resultado de sumar la matriz C y b se pasa por una función de activación no lineal que se muestra en la figura 5-4. Existen diferentes funciones de activación que el usuario puede seleccionar, el asistente deberá permitir la selección de diferentes funciones o ingresarla de manera manual.

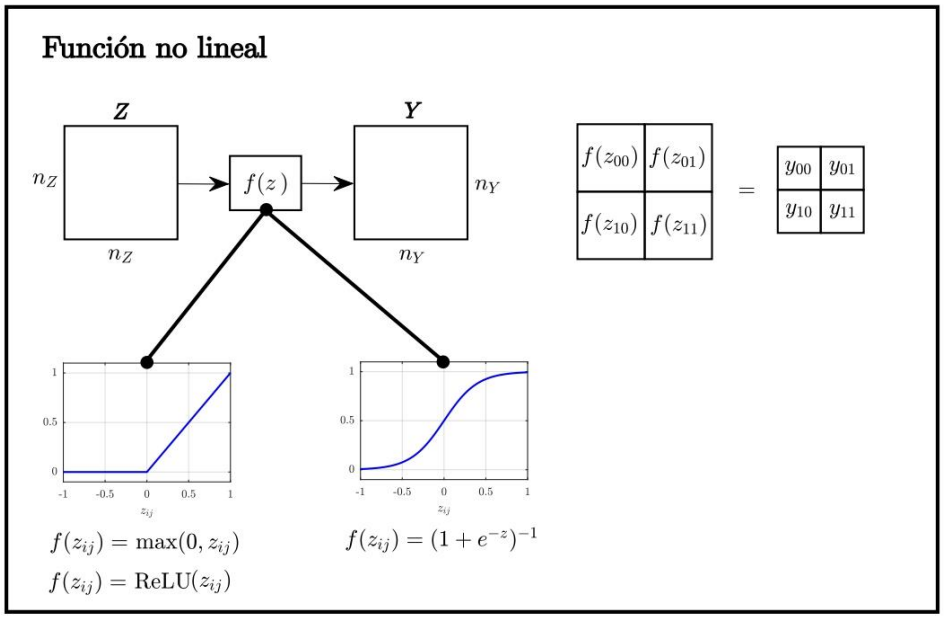



Figura 5-4 Función de activación no lineal.

Lo último que se realiza en la capa es la reducción o submuestreo de la matriz de salida de la función de activación, en este caso esta reducción se realiza por valor máximo como se ilustra en la figura 5-5, es decir se toma una ventana de 2x2 y se toma el valor máximo, el resultado de esta reducción para una matriz de nxm es de $\frac{n}{2} x \frac{m}{2}$.

En el asistente se debe permitir realizar la reducción no solo con la técnica de valor máximo, también existe la técnica de valor promedio, entre otras, se debe de permitir la opción de seleccionar diferentes tamaños de ventana, al modificar el tamaño de ventana se modifica el resultado en términos de dimensión de la reducción, se debe permitir al usuario seleccionar el tamaño de salto de ventana, es decir en cada selección de la ventana cuantos espacios se saltan, este valor puede hacer que las ventanas se traslapen o no dependiendo de lo que el usuario busque al realizar la reducción.

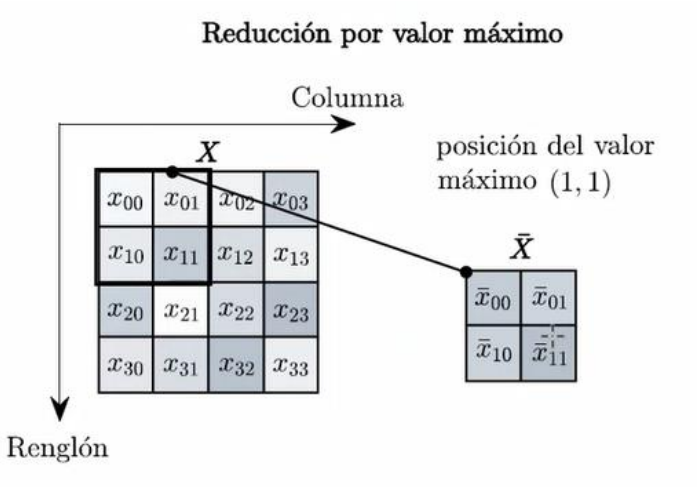



Figura 5-5 Reducción o submuestreo de matriz por valor máximo.

Es importante que el asistente permita activar o desactivar las diferentes partes de la capa, es decir, si el usuario no quiere una reducción en la salida de la capa o si el usuario quiere modificar el orden del proceso el asistente debe ser capaz de permitirle y asistirle al usuario la personalización completa de la red neuronal.

En este caso en particular, en la segunda capa que se muestra en la figura 5-6 se realiza el mismo proceso, pero utilizando de entrada la salida de la capa 1, es decir ya no se tiene una matriz de entrada de 100x100, ahora se tienen 10 matrices de 46x46, por lo tanto, la matriz de convolución ahora es de 10 conjuntos de 10 matrices cada uno de 5x5. En esta capa se realizan los mismos pasos de la capa anterior, convolución, bias, función de activación y submuestreo.

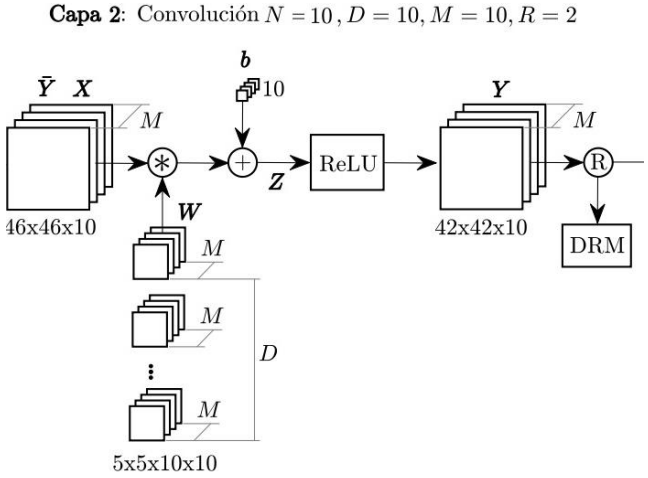



Figura 5-6 Capa 2 de convolución.

En la tercera capa que se muestra en la figura 5-7, se tiene como entrada la salida de la capa anterior, y se realiza la convolución, la suma con la matriz de bias y el paso por la función ReLU, en este caso en particular en esta capa no se realiza una reducción en la salida.

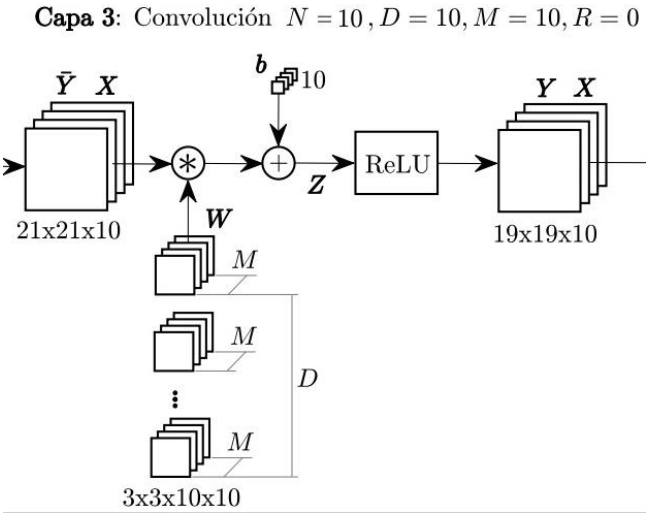



Figura 5-7 Capa 3 de convolución.

Estudiando este caso en particular se observa las partes repetitivas al crear una red neuronal y lo grande que puede llegar a crecer el código de una red neuronal. La capa 1, 2 y 3 tienen una estructura muy parecida en la que solo cambian las constantes que configuran las diferentes convoluciones, los valores de bias, puede ser o no igual la función de activación y puede o no tener reducción en la salida.

Esto da pie a que el asistente desarrolle estas capas, en el caso en particular estudiado solo se tienen tres capas, pero en el caso de que sean 10 o más la modificación de parámetros en cada una de las capas en un código desarrollado donde no se cuente con una interfaz en la que sea fácil identificar donde se modifica cada parámetro de la red puede llegar a ser un reto.

La capa 4, 5 y 6, son capas completamente conectadas, al principio de la capa 4 se hace una transformación del mapa de 19x19x10 a un vector de dimensión 3610x1, en la capa 4 se tienen 100 neuronas de salida, en la capa 5 también se tienen 100 neuronas de salida, y por último en la capa de salida se tienen 4 neuronas de salida como se observa en la figura 5-8.

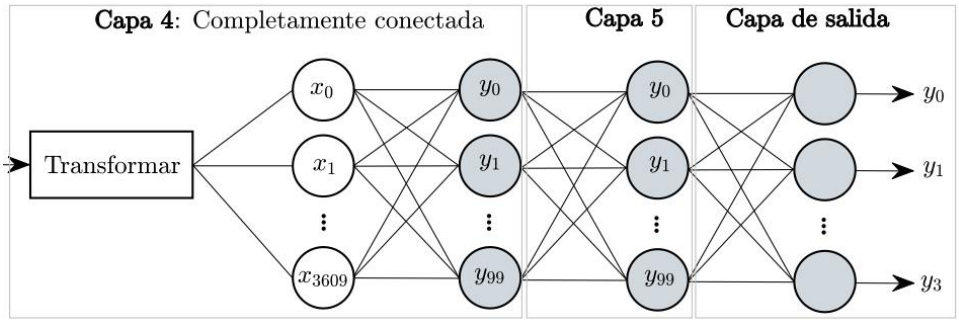



Figura 5-8 Capas 4, 5 y 6 completamente conectadas.

**Identificación preliminar de los procesos que puede realizar el asiste en el proceso de creación de una red neuronal convolucional:**

1. Diseño de un método para crear imágenes a partir de datos que sirvan de entrada en la red neuronal convolucional.
2. Soporte para el ingreso de n imágenes de entrada a la red neuronal.
3. Asistencia en la creación de filtros de convolución.
4. Realizar operación de convolución con n conjuntos de arreglos de m filtros de a x b tamaño.
5. Asistencia en la creación de loa valores bias y su posterior integración en la matriz de datos.
6. Asistencia en la selección de una función de activación que evalué cada valor de la matriz.
7. Realizar operación de submuestreo, teniendo la posibilidad de selección el tipo, tamaño de ventana y tamaño de salto de ventana.
8. Asistir en la selección de cuales de los procesos antes mencionados se quieren integrar o no en la capa.
9. Asistir en la transformación de la salida de la última capa de convolución para transformar los datos a un vector lineal para que sea usado como entrada de una red neuronal multicapa.
10. Asistir en la creación de las capas totalmente conectadas y sus salidas.

# Capitulo 6. Desarrollo del asistente

En este capítulo se integran los resultados experimentales del trabajo. Se inicia con la presentación de la interfaz de usuario de la plataforma, donde se describe sus componentes y la interacción con el usuario. Posteriormente, se especifican los procesos necesarios para que la red neuronal desarrollada pueda ser implementada en un sistema embebido. A continuación, se presenta la estructura del lenguaje de dominio especifico desarrollado, sus funciones y su alcance. Finalmente se realiza la validación de la plataforma.

El asistente propuesto está diseñado para ser utilizado por alguien que cuente con los conocimientos básicos de redes neuronales, esto con el fin de que el usuario entienda la terminología de los parámetros necesarios para configurar cada capa de la red neuronal y comprenda cada parte que la compone.

## 6.1 Interfaz de usuario

En la figura 6-1 se muestra la GUI de la plataforma donde el usuario podrá configurar la estructura, los datos de entrada y salida y los valores de entrenamiento de la red neuronal.

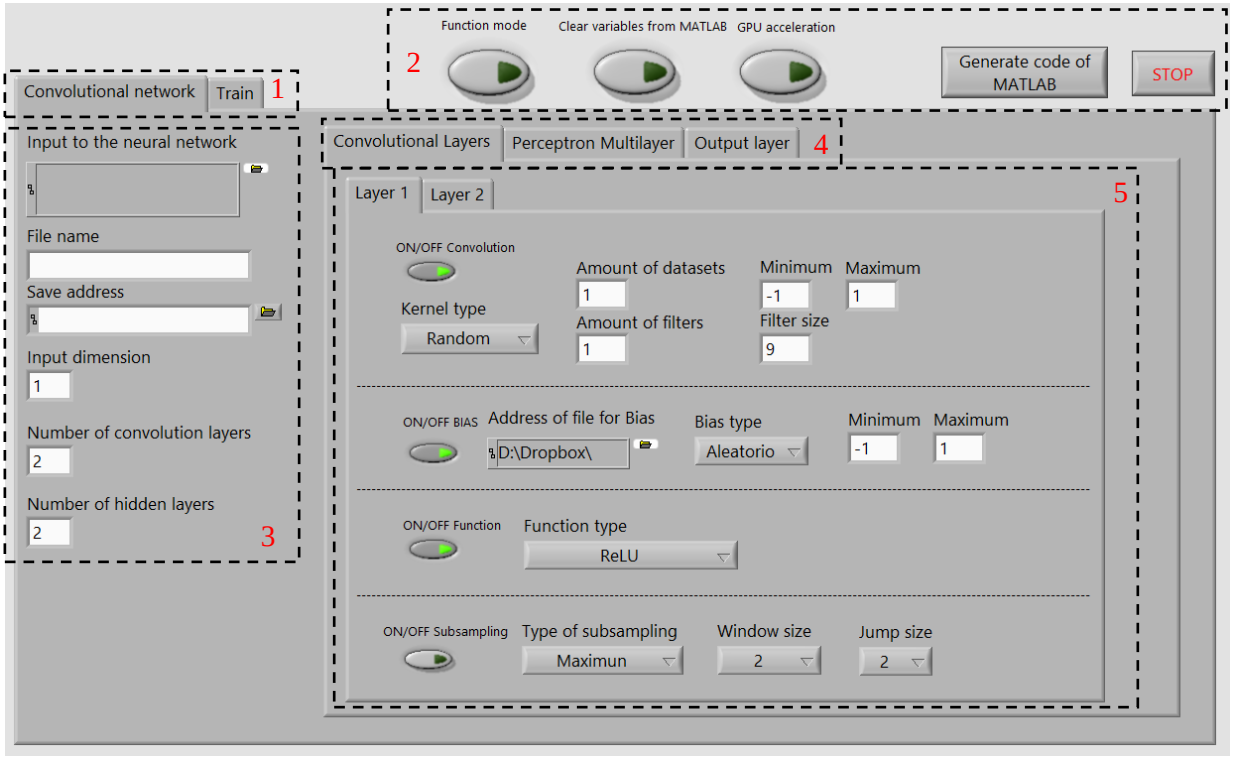



Figura 6-1 GUI de la plataforma.

En la sección 1 de la interfaz mostrada en la figura 6-1 tenemos la selección de la pestaña principal, las opciones son "Convolutional network", es donde se configura la estructura que tendrá la red neuronal, y "Train" es donde se configuran los parámetros de entrenamiento.

En la sección 2, primero tenemos el botón "Modo de función" al activar esta función, el archivo de red neuronal generado no será un archivo ejecutable, será una función que se puede llamar desde cualquier otro programa MATLAB, Esto es muy útil cuando se desea validar una red neuronal y llamar a la función con cada valor de entrada para probar, es importante tener mucho cuidado al realizar la llamada para ingresar

correctamente los valores solicitados por la función. Seguido es el botón "Borrar variables de MATLAB" que activa el botón genera un comando en el archivo de ejecución en la red neuronal que limpia las variables en el espacio de trabajo MATLAB. El tercer botón "aceleración GPU" añade al código de ejecución el uso de la GPU del equipo para tomar ventaja de la paralelización ofrecida por este tipo de hardware y hacer más rápida la ejecución del programa, es importante mencionar que no todos los equipos son compatibles, solo aquellos que tienen una tarjeta gráfica dedicada. El botón "Generar código de MATLAB" genera el archivo de salida de la red neuronal a ejecutar.

En la sección 3, la entrada se configura en el caso de que la red neuronal se genere para una sola ejecución, se le pide al usuario el nombre y la dirección del archivo de salida que contiene la red neuronal generada, los parámetros que determinan el tamaño de la red neuronal están configurados, dimensión de entrada, número de capas convolucionales y ocultas.

La sección 4 selecciona la interfaz que desea mostrar para ser configurado, capas convolucionales, capas ocultas o capa de salida.

En la sección 5 la interfaz parece configurar el número de capas convolucionales seleccionadas en la sección 3, a medida que cambiamos el número de capas que la interfaz ajusta automáticamente, dentro de las opciones de configuración para estas capas son: tipo de núcleo y sesgo a utilizar (aleatorio o importado de un archivo), función de activación y capa de submuestra.

La interfaz mostrada en la figura 6-2 permite al usuario configurar las capas ocultas, al igual que las capas convolucionales muestran el número de pestañas correspondientes a la configuración de la red neuronal. Las opciones de configuración son: número de neuronas, tipo de peso sináptico y sesgo (aleatorio o importado del archivo), y función de activación.

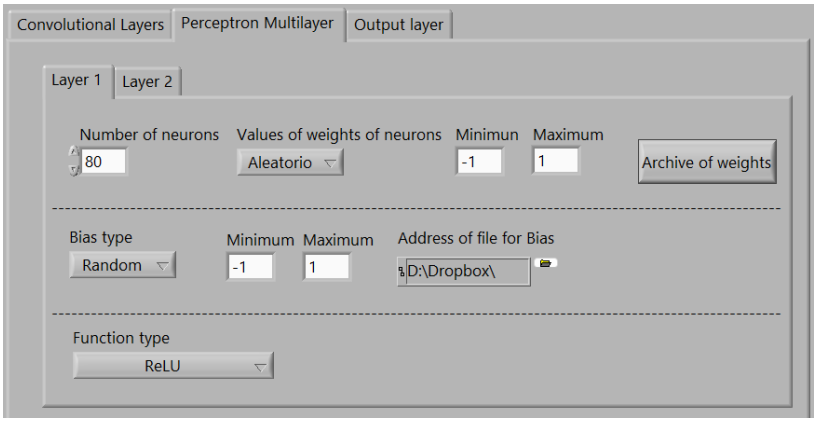



Figura 6-2 Interfaz de configuración de etapa multicapa.

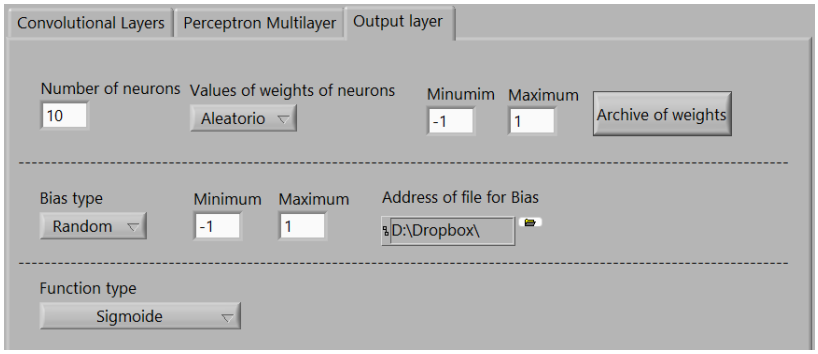



Figura 6-3 Interfaz de configuración de capa de salida.

La figura 6-3 muestra las opciones de configuración de la capa de salida: número de neuronas, tipo de peso sináptico y sesgo (aleatorio o importado del archivo) y función de activación.

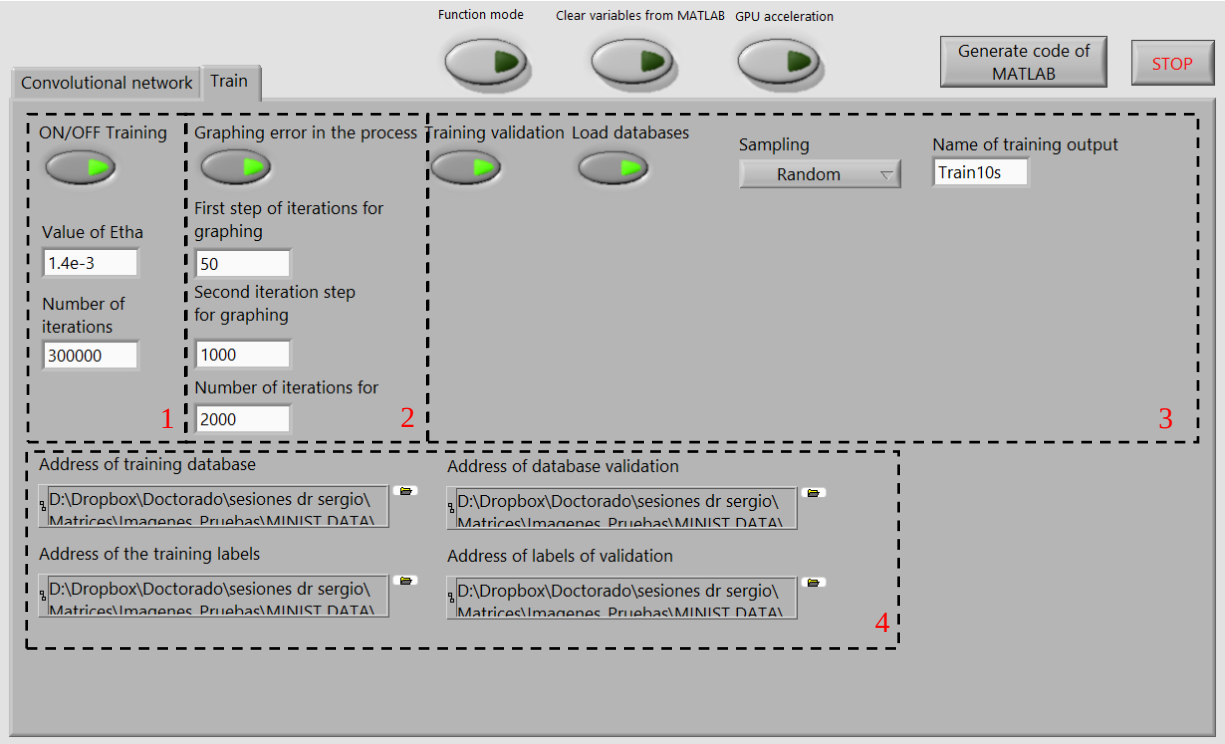



Figura 6-4 Interfaz de entrenamiento.

La pestaña de entrenamiento permite al usuario configurar los parámetros de entrenamiento de la red neuronal configurada en los pasos anteriores, la plataforma utiliza la configuración de la estructura de la red neuronal para configurar el entrenamiento, en la figura 6-4 se muestra la interfaz de entrenamiento.

En la sección 1 de la figura 6-4 se activa o desactiva el entrenamiento, se determina el valor de *Ehta* o tasa de aprendizaje y el número de iteraciones del algoritmo de entrenamiento.

En la sección 2 se activa o desactiva la opción de graficar el error de la red neuronal con los valores deseados, esto permite al usuario tener una idea del progreso del entrenamiento de la red neuronal y tener una idea de si continuar o no el entrenamiento. Se pueden configurar dos pasos distintos de iteraciones para

mandar a graficar el error, es decir en que numero de iteración se grafica al principio y pasando un numero establecido utilizar un numero diferente.

En la sección 3 se activa la validación del entrenamiento, por cada iteración de entrenamiento se hace una de validación con una base de datos completamente diferente, esto permite al usuario verificar si la red neuronal va mejorando su resultado con un conjunto desconocido, la gráfica de entrenamiento y validación deben tener un comportamiento similar.

En la sección 4 el usuario selecciona la ubicación de las bases de datos de entrenamiento y validación, estas bases de datos deben de estar en formato .mat y deben contener una matriz de 4 dimensiones con el siguiente formato:

$$n \times m \times e \times i$$

$n$ y $m$ son las dimensiones de cada imagen, $e$ es al número de imágenes de entrada de la red neuronales e $i$ es el número de imágenes en la base de datos. Las etiquetas de cada elemento de entramiento deben estar en un archivo separado con formato .mat y debe contener un vector de 2 dimensiones con el formato:

$$p \times i$$

$p$ es el numero de etiquetas por cada imagen y $i$ es el número de imágenes que debe coincidir con la matriz de la base de datos o de validación dependiendo de cuál sea el ingresado.

## 6.2 Generación de red neuronal

El software sigue un proceso para convertir la configuración seleccionada por el usuario para la red neuronal, así como los datos de entrada para generar un código, este proceso se muestra en la figura 6-5.

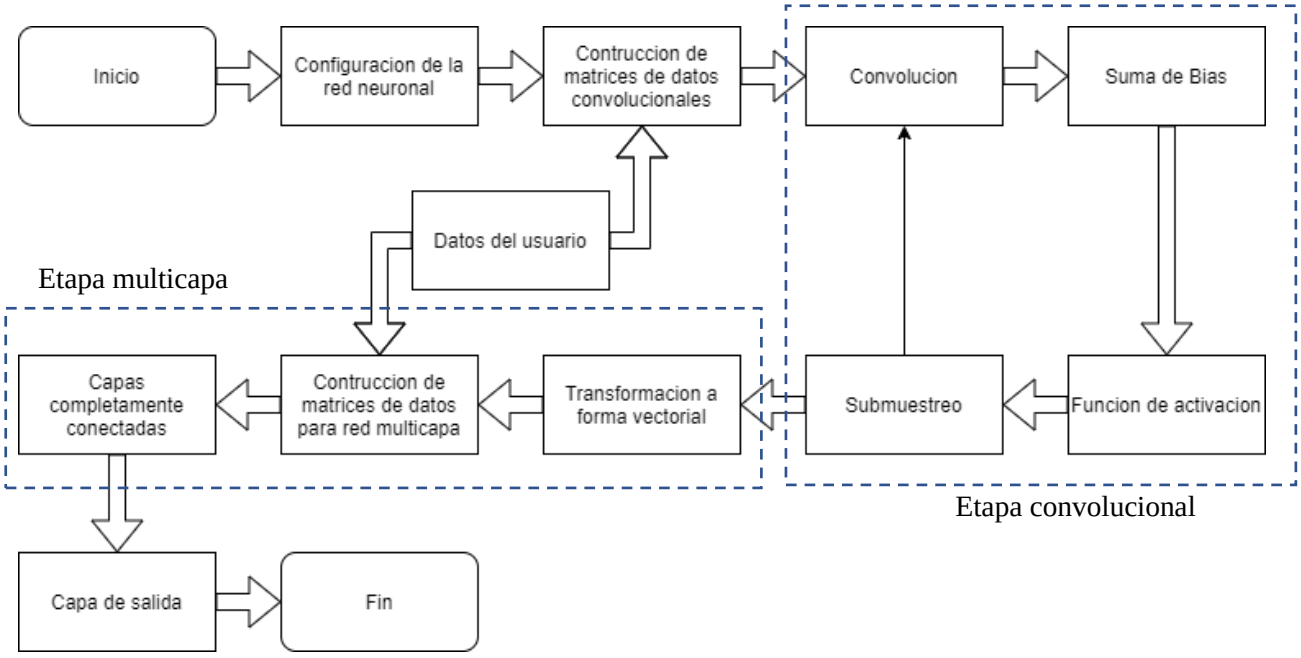



Figura 6-5 Diagrama del proceso de construcción de código de red neuronal.

En la etapa de configuración de la red neuronal se toma información proporcionada por el usuario en la interfaz, la información utilizada en esta etapa es: dirección de la imagen de entrada a la red neuronal,

numero de capas convolucionales, numero de conjuntos y numero de filtros utilizados en cada capa convolucional, elementos activos en cada capa convolucional (bias, función de activación y submuestreo), numero de capas completamente conectadas, cantidad de neuronas por cada capa, cantidad de neuronas en la capa de salida. Sabiendo la cantidad de capas convolucionales y utilizando los datos del usuario se construyen las matrices que servirán en el proceso de convolución, estas matrices se les asigna un nombre de la siguiente manera:

kernel_C_D_F

donde: C es el número de capa convolucional, D es el número de conjunto, F es el número de filtro.

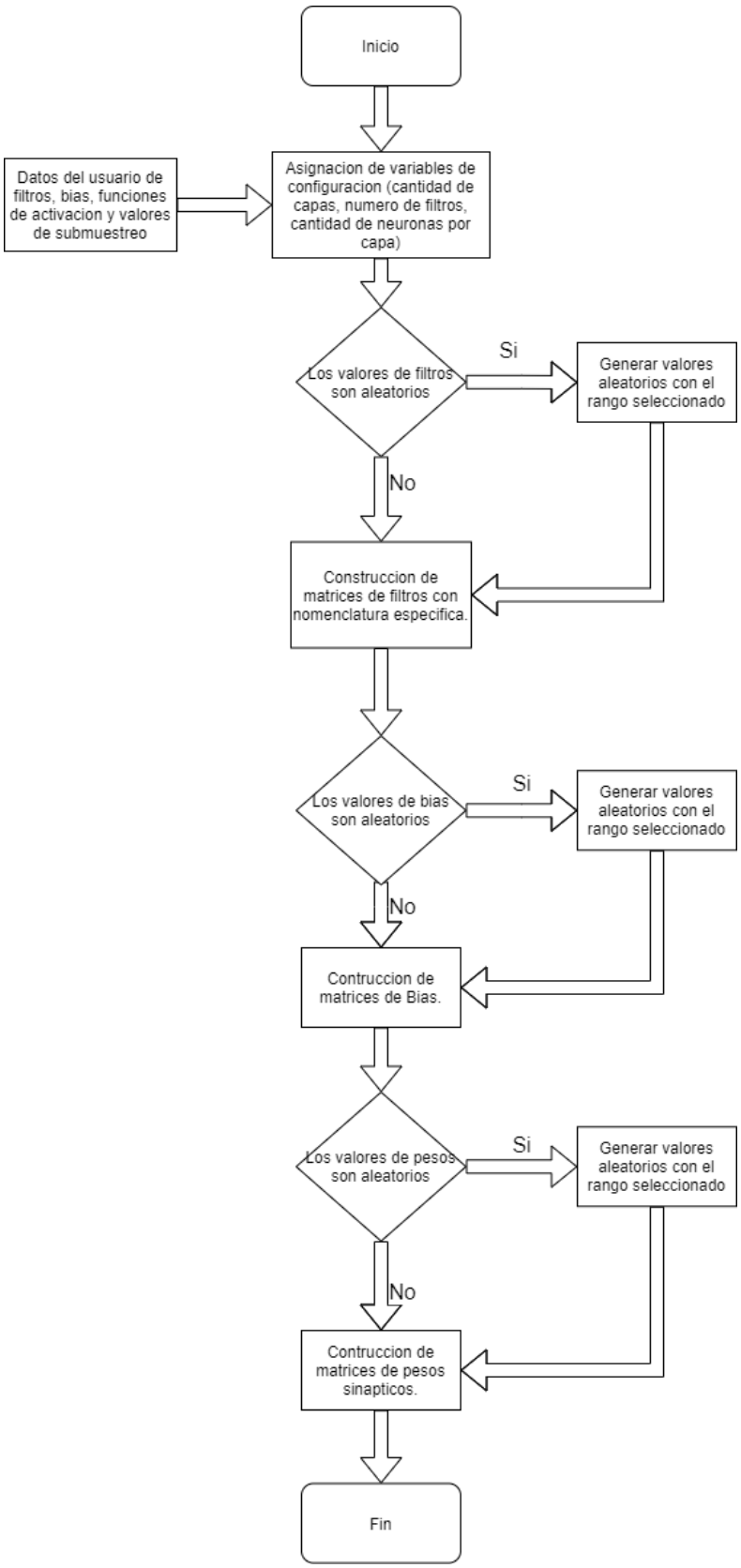



Figura 6-6 Diagrama de configuración y creación de matrices.

Para crear las matrices de valores de bias se utiliza el nombre de la capa donde se utilizarán por ejemplo “BiasCapa1”, para los pesos sinápticos de la etapa totalmente conectada se crea una matriz para cada capa con la numeración de la capa por ejemplo “PesosCapa1” y se asignan los valores proporcionados por el usuario, todo este proceso se muestra en la figura 6-6.

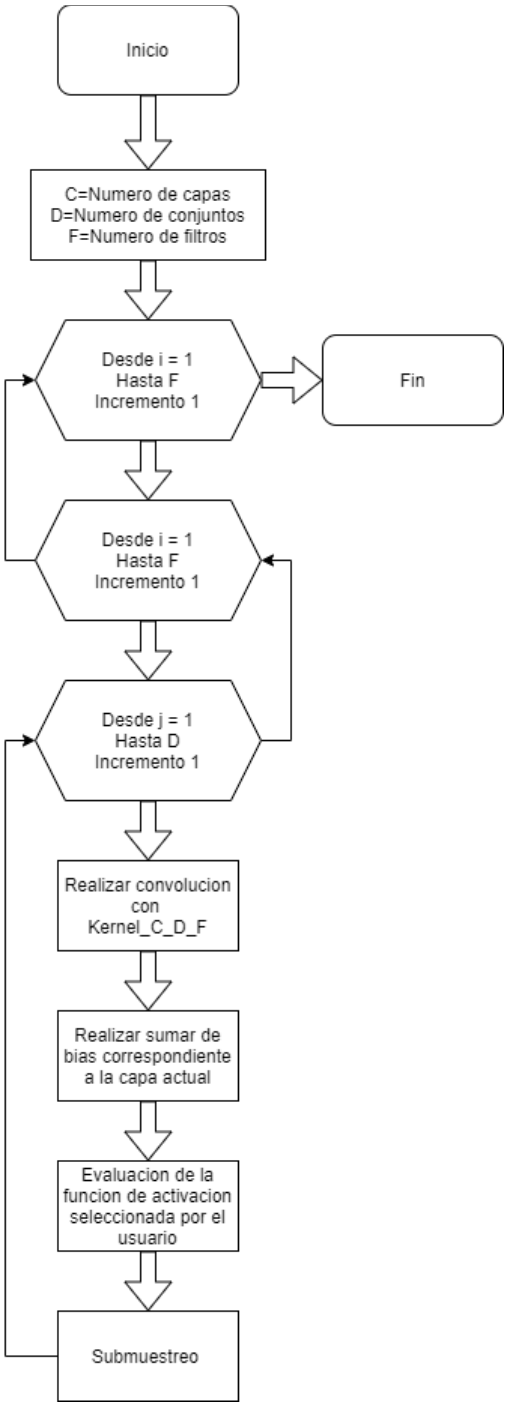



Figura 6-7 Diagrama de proceso convolucional.

La segunda etapa es la convolucional, primero se toman los valores de usuario y configuración que se usaran (cantidad de capas, conjuntos y filtros y valores de filtros y de bias), se inicializa el primer for que

controlara la cantidad de capas, después el segundo for para la cantidad de conjuntos y el ultimo for para la cantidad de los filtros, dentro de cada iteración de realiza la convolución con el kernel correspondiente,

$$Y_j = \sum_{i=1}^{C} W_i X_i \tag{3-1}$$

donde:

Y es el elemento de la matriz de salida, $W_i$ es la matriz que contiene el kernel para convolución, $X_i$ es la ventana seleccionada de la entrada, C es el tamaño del filtro. Teniendo una matriz de entrada X de 3x3, y un filtro W de 2x2. Se generan las siguientes operaciones:

$$\begin{aligned} C_{00} &= W_{00}X_{00} + W_{01}X_{01} + W_{10}X_{10} + W_{11}X_{11} \\ C_{01} &= W_{00}X_{01} + W_{01}X_{02} + W_{10}X_{11} + W_{11}X_{12} \\ C_{10} &= W_{00}X_{10} + W_{01}X_{11} + W_{10}X_{20} + W_{11}X_{21} \\ C_{11} &= W_{00}X_{11} + W_{01}X_{12} + W_{10}X_{21} + W_{11}X_{22} \end{aligned} \tag{3-2}$$

El segundo proceso que se realiza es la suma de bias,

$$Z = C + B \tag{3-3}$$

donde:

Z es la matriz de salida, C es la matriz de entrada, B es el número de bias.

el tercer proceso es la evaluación de los valores con una función de activación definida, las funciones utilizadas son:

Función ReLU:

$$f(Z_{ij}) = \max(0, Z_{ij}) \tag{3-4}$$

Función tangente hiperbólica:

$$f(Z_{ij}) = \tanh(Z_{ij}) \tag{3-5}$$

Función sigmoide:

$$f(Z_{ij}) = \frac{1}{(1 + e^{-z_{ij}})} \tag{3-6}$$

por último, se realiza el proceso de submuestreo, se toma una ventana de la matriz de entrada y se evalúa de la siguiente manera, para el submuestreo de valor máximo:

$$s_{ij} = \max(Z_{ij}, Z_{i+1\,j}, Z_{i\,j+1}, Z_{i+1\,j+1}) \tag{3-7}$$

para el submuestreo de valor medio:

$$s_{ij} = \text{promedio}\,(Z_{ij}, Z_{i+1\,j}, Z_{i\,j+1}, Z_{i+1\,j+1}) \tag{3-8}$$

realizando estas operaciones se reduce a la mitad el tamaño de la matriz de entrada, todo este proceso de muestra en la figura 6-7.

La tercera etapa es convertir la salida de la etapa convolucional a un vector que sirva como entrada a la etapa totalmente conectada, para esto se obtiene el tamaño de salida de la última capa convolucional y el número de filtros que se aplicaron en la última capa, el tamaño de vector resultante de la transformación estará dado por la multiplicación de estos valores:

$$\text{Tamaño de vector } = [1 \quad S * S * L]$$

donde:

S es el tamaño de una dimensión de la matriz de salida de la capa convolucional.

L es el número de filtros aplicado en la capa de salida de la etapa convolucional.

Esto se muestra en la figura 6-8.

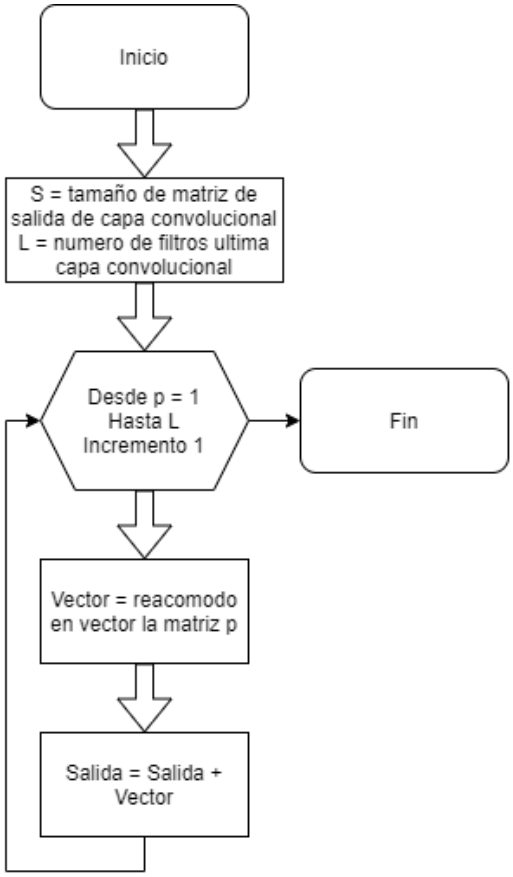



Figura 6-8 Diagrama de transformación de salida convolucional a vector.

La última etapa es realizar el recorrido por las capas ocultas, esto se genera con la configuración proporcionada por el usuario, la cantidad de neuronas por capa, la cantidad de capas y la cantidad de salidas en la última capa. En cada capa se realiza la multiplicación de la entrada por el peso simpático, se realiza

la suma de bias y la evaluación de función de activación del mismo modo que se realizó en la etapa convolucional.

$$Y_c = \sum_{i=1}^{M_c} W_{ij} X_i \tag{3-9}$$

donde:

$W_{ij}$ es cada peso sináptico.

$X_i$ es el elemento de entrada a la neurona.

Yc es el elemento resultante.

$M_c$ es el numero de neuronas en la capa.

C es la capa actual.

Se presenta el seudocódigo que genera la plataforma para simular la red neuronal analizada en el capítulo 5.

**Algoritmo 1**

**Input:** Image of 100 x 100 pixels

1. Y0 ← **conv** (X0,W0)
2. Y0 ← **bias** (Y0,B0)
3. Y0 ← **ReLU** (Y0)
4. X1 ← **reduction by maximum value** (Y0)
5. Y1 ← **conv** (X1,W1)
6. Y1 ← **bias** (Y1,B1)
7. Y1 ← **ReLU** (Y1)
8. X2 ← **reduction by maximum value** (Y1)
9. Y2 ← **conv** (X2,W2)
10. Y2 ← **bias** (Y2,B2)
11. Y2 ← **ReLU** (Y2)
12. X3 ← **vectorize** (Y2)
13. Z3 ← W3X3+B3
14. Y3 = **ReLU** (Z3)
15. X4 ← Y3
16. Z4 ← W4X4+B4
17. Y4 = **ReLU** (Z4)
18. X5 ← Y4
19. Z5 ← W5X5+B5
20. Y5 ← **sigmoide** (Z5)

**Output:** identification of four parameters.

## 6.3 Entrenamiento de la red neuronal

El entrenamiento de la red neuronal es una parte crucial ya que el entrenamiento es el que le da a la red neuronal la capacidad de procesar la entrada y obtener una salida deseada.

El entrenamiento de una red neuronal se realiza con algoritmos como backpropagation. Este algoritmo, de propagación de errores hacia atrás, utiliza el error observado en la salida de una red neuronal multicapa para ir ajustando paulatinamente los parámetros que definen la respuesta de la red ante una entrada dada, con la intención de reducir el error cometido por la red. Las redes neuronales son, pues, capaces de aprender a partir de ejemplos, lo que les permite generalizar sin tener que formalizar el conocimiento adquirido.

Aunque el método de aprendizaje consistente en combinar backpropagation con una técnica de optimización para minimizar una función de error dada es la forma más habitual de entrenar redes neuronales, no es la única que existe.

El proceso de entrenamiento que sigue la plataforma se describe en la figura 6-9, se tiene una base de datos y se extrae una imagen en cada iteración, se ingresa en la red neuronal convolución, con la salida de realiza la propagación del error y después se actualiza los pesos de toda la red con la regla delta, este proceso se repite hasta disminuir a un rango definido la función de error.

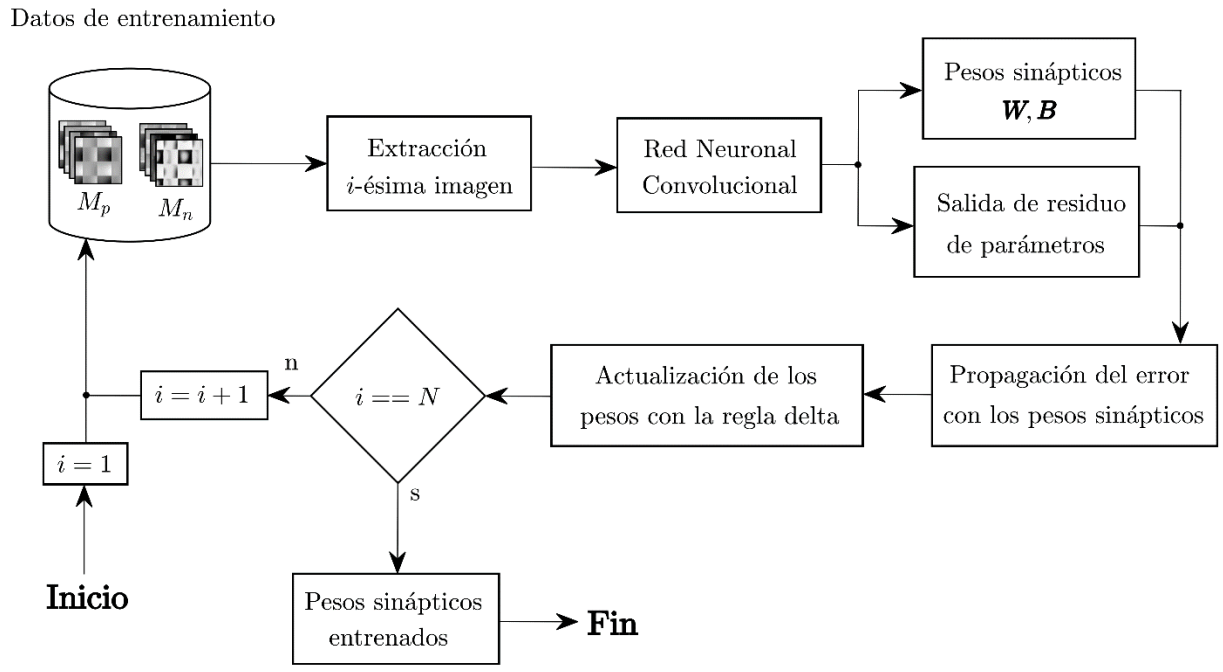



Figura 6-9 Diagrama de entrenamiento.

El entrenamiento de una neurona consiste en ajustar su peso sináptico $\omega$ para obtener una salida deseada $y_d$, se tiene una función de costo de error $E$, no se tiene una sola función de costo, una muy utilizada es:

$$E = \frac{1}{2}(y_d - y)^2 \tag{3-10}$$

Se calcula el gradiente de E con respecto a cada peso sináptico $\omega$:

$$\frac{\partial E}{\partial \omega} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w} \tag{3-11}$$

para la función de costo cuadrática el gradiente es:

$$\frac{\partial E}{\partial \omega} = -(y_d - y)x \tag{3-12}$$

la variación de los pesos sinápticos se realiza con la regla delta que utiliza un parámetro denominada tasa de aprendizaje $\eta$:

$$\omega_n = \omega_{n-1} - \eta\left(\frac{\partial E}{\partial \omega}\right) \tag{3-13}$$

donde $n$ indica el número de iteración del algoritmo de aprendizaje. Extendiendo el gradiente de $E$ a una red de $N$ entradas y $M$ salidas:

$$E_j = \frac{1}{2}\left(y_{dj} - y_j\right)^2, \ \forall\, j \in 1,2,\dots,M$$
$$\nabla E = \left[\frac{\partial E_j}{\partial w_{1j}}, \frac{\partial E_j}{\partial w_{2j}}, \dots, \frac{\partial E_j}{\partial w_{nj}}\right]^T \tag{3-14}$$

donde:

$$y_j = \sum_{i=1}^{N} \omega_{ij} x_i \tag{3-15}$$

cuando se utiliza una función de activación $f(z_j)$, el gradiente de $E$ se determina con la regla de la cadena para derivadas:

$$\nabla E_j = \left[\left(\frac{\partial E_j}{\partial y_j}\right)\left(\frac{\partial y_j}{\partial z_j}\right)\left(\frac{\partial z_j}{\partial \omega_{1j}}\right), \left(\frac{\partial E_j}{\partial y_j}\right)\left(\frac{\partial y_j}{\partial z_j}\right)\left(\frac{\partial z_j}{\partial \omega_{2j}}\right), \dots, \left(\frac{\partial E_j}{\partial y_j}\right)\left(\frac{\partial y_j}{\partial z_j}\right)\left(\frac{\partial z_j}{\partial \omega_{Nj}}\right)\right]^T \tag{3-17}$$

donde

$$\frac{\partial y_j}{\partial z_j} = \frac{\partial f(z_j)}{\partial z_j} \tag{3-18}$$

Para la capa de salida se calcula $\delta$ con la función de costo $E$:

$$\delta_j^c = \left(\frac{\partial E_j}{\partial y_j}\right) f'(z_j) \tag{3-19}$$

se determinan las variaciones de los pesos de la capa de salida:

$$\nabla w_{ij} = -\eta \delta_j x_i \tag{3-20}$$

para las capas ocultas de determina el $\delta$ con la propagación del error de la capa inmediata siguiente:

$$\delta_j = \left[\sum_{p=1}^{M} \delta_p^{C+1} \omega_{jp}^{C+1}\right] f'(z_j^C) \tag{3-21}$$

una vez que el error se propago hasta la primera capa, se actualizan los pesos sinápticos:

$$\omega_{ij}^q = \omega_{ij}^q + \nabla\omega_{ij}^q \tag{3-22}$$

## 6.4 Simulación de red neuronal

**Lectura de la imagen de entrada**

Tomando en cuenta el punto 2 en la identificación de procesos, el asistente crea el código que permite a Matlab tener la información de la imagen, toma la dirección de la imagen proporcionada por el usuario y utilizando la función "imread" se importa la información de la imagen, después convierte los valores de la imagen a doble precisión para poder trabajar con ellos y se obtiene el tamaño de la imagen y se guarda una variable con la información original en caso de que se necesite comparar con la información original, en la figura 6-10 se muestra el código obtenido. Este proceso se puede repetir para n imágenes. El programa genera un archivo de Matlab con extensión .m en la misma carpeta del sistema donde se encuentre la imagen de entrada y con el mismo nombre para su fácil identificación en el sistema.

```
1. close all
2. clear all
3. clc
4. % Lectura de imagen
5. a = imread(' C:\Users\dm_ga\Desktop\Imagen_Generada.png');
6. sal=double(a(:,:,1))./255;
7. s=size(a);
8. x0=sal;
```

Figura 6-10 Código de Matlab que hace la lectura de la imagen de entrada.

**Función generadora**

Tomando en cuenta el punto 1 en la identificación de procesos, el asistente permite crear una imagen en dado caso de que no se cuente con una como entrada de la red neuronal convolucional, esta imagen se crea con los vectores ϕ y α que el usuario proporciona para generar una imagen que puede servir de entrada para la red neuronal convolucional. En la figura 6-11 se muestra el código que se genera en Matlab para realizar dicha función.

```
1. close all
2. clear all
3. clc
4. % Funcion generadora
5. alfa = dlmread('C:\Users\dm_ga\Desktop\alpha.txt');
6. phi = dlmread('C:\Users\dm_ga\Desktop\phi.txt');
7. M = alfa * phi';
8. M =255*( (M-min(min(M)))./(max(max(M))-min(min(M))));
9. imshow(uint8(M));
10.     M = uint8(M);
```

```
11.      imwrite(M,'Imagen_Generada.png');
```

Figura 6-11 Código de Matlab que genera una imagen.

Teniendo la dirección de los datos de los vectores se leen como variables de Matlab y se crean como vectores, se realiza la multiplicación de vectores $\text{imagen} = \alpha\phi^t$, y ya se tiene la matriz de entrada a la red neuronal.

**Generación de matrices de kernels**

Tomando en cuenta el punto 4 en la identificación de procesos, en la figura 6-12 se muestra el código que sigue el programa para generar las matrices de kernel a partir de un archivo .MAT dado por el usuario y en la figura 6-13 se muestra el código generando las matrices de manera aleatoria.

El código va generando el nombre de los diferentes kernels con los índices explicados con anterioridad dependiendo del número de capa, conjunto y filtro que se esté generando, se extraen los datos del usuario y se asignan, utilizando un for anidado para cada índice.

```
1. % Kernels
2. load('C:\Users\dm_ga\Desktop\W0.mat')
3. c=1;
4. m=10;
5. d=1;
6. data=W0;
7. for i=1:d
8. for j=1:m
9. kernel =
   genvarname(['kernel_',num2str(c),'_',num2str(i),'_',num2str(j)]);
10.      eval([kernel ' = data(:,:,i,j);']);
11.      end
12.      end
13.      load('C:\Users\dm_ga\Desktop\W1.mat')
14.      c=2;
15.      m=10;
16.      d=10;
17.      data=W1;
18.      for i=1:d
19.      for j=1:m
20.      kernel =
   genvarname(['kernel_',num2str(c),'_',num2str(i),'_',num2str(j)]);
21.      eval([kernel ' = data(:,:,i,j);']);
22.      end
23.      end
24.      load('C:\Users\dm_ga\Desktop\W2.mat')
25.      c=3;
26.      m=10;
27.      d=10;
28.      data=W2;
29.      for i=1:d
30.      for j=1:m
31.      kernel =
   genvarname(['kernel_',num2str(c),'_',num2str(i),'_',num2str(j)]);
```

```
32.      eval([kernel ' = data(:,:,i,j);']);
33.      end
34.      end
```

Figura 6-12 Código que genera las matrices de los kernels desde archivo.

```
1. % Kernels
2. kernel_1_1_1=[0.605306,0.976482,0.871832;0.613929,-
   0.246550,0.838060;-0.187951,0.558160,-0.896399];
3. kernel_1_1_2=[-0.771907,0.348093,0.007212;-0.415013,-0.733958,-
   0.163132;0.794569,-0.899817,0.443486];
4. kernel_1_1_3=[0.278648,0.352096,0.429324;-0.685890,-
   0.935274,0.607883;-0.175378,-0.745203,0.691476];
5. kernel_1_1_4=[0.660035,0.233742,-0.974149;-0.295227,0.328015,-
   0.437212;0.281698,-0.326822,0.920714];
6. kernel_1_1_5=[0.358758,-0.253177,-0.359489;-0.352821,-0.956998,-
   0.732182;0.197223,-0.685236,0.595891];
7. kernel_1_1_6=[0.785945,0.235996,-0.502995;0.937184,0.582801,-
   0.881697;-0.997913,0.623610,-0.507896];
8. kernel_1_1_7=[-0.634531,-0.086006,0.237194;-
   0.901947,0.045075,0.640453;0.120093,0.114563,-0.793665];
9. kernel_1_1_8=[0.709070,-0.691533,-0.957085;-0.434939,-
   0.825332,0.873522;-0.270332,-0.007586,0.917254];
10.      kernel_1_1_9=[0.057183,0.840314,0.568171;-
   0.350860,0.492448,-0.489058;-0.681038,-0.287530,-0.726677];
11.      kernel_1_1_10=[0.445774,-0.108570,-
   0.075295;0.099348,0.011827,-0.955089;0.025343,-
   0.378205,0.694592];
```

Figura 6-13 Código que genera las matrices de los kernels de manera aleatorio.

Como se puede observar en la figura 6-13, los números aleatorios se generan en el software en LabVIEW y se declaran las matrices directas en MATLAB.

**Capas convolucionales**

Tomando en cuenta el punto 4 en la identificación de procesos, en la figura 6-14 se muestra el proceso que sigue el programa para generar el código de una capa de convolución, el código construye el nombre del kernel requerido, realiza la convolución con la matriz de datos correspondiente y sigue la siguiente matriz de datos hasta terminar.

```
1. arc=0;
2. c=1;
3. d=1;
4. f=10;
5. for j=1:f
6. kernel0=0;
7. suma=0;
8. for l=1:d
9. ker =
   genvarname(['kernel_',num2str(c),'_',num2str(l),'_',num2str(j)]);
```

```
10.      kernel=eval(ker);
11.      salida = genvarname(['y1_',num2str(j)]);
12.      entrada = genvarname(['x0_',num2str(l)]);
13.
14.      k=size(kernel);
15.      for q1 = 1:k(1)
16.      for q2 = 1:k(1)
17.      kernel0(q1,q2) = kernel(k(1)-q1+1,k(1)-q2+1);
18.      end
19.      end
20.      arc=conv2(sal,kernel0,'valid');
21.      suma=suma+arc;
22.      end
23.      eval([salida ' = suma;']);
24.      end
25.      int=0;
26.      for j=1:f
27.      salida = genvarname(['y1_',num2str(j)]);
28.      entrada = genvarname(['x0_',num2str(l)]);
29.      int=eval(salida);
30.      eval([entrada ' = int;']);
31.      end
```

Figura 6-14 Código de Matlab que realiza la convolución.

**Capa de bias**

El código que realiza la suma de bias realiza la suma directa con la matriz correspondiente, en el caso de utilizar datos de usuario se extraen en el momento como se muestra en la figura 6-15, cuando se utiliza un bias aleatorio se genera en el momento como se observa en la figura 6-16.

```
1. % Bias Capa 1 Filtro 1
2. bias = dlmread('C:\Users\dm_ga\Desktop\bias_Capa1.txt');
3. bi=bias(1);
4. y1_1=y1_1+bi;
```

Figura 6-15 Código de Matlab que realiza la suma de bias desde un archivo.

```
1. % Bias Capa 1 Filtro 1
2. a=-1.000000;
3. b=1.000000;
4. bi=((b-a).*rand(1) + a);
5. y1_1=y1_1+bi;
```

Figura 6-16 Código de Matlab que realiza la suma de bias con datos aleatorios.

**Capa de función de activación**

El código mostrado en la figura 6-17 muestra la evaluación que hace el programa utilizando la función de activación de cada filtro de cada capa.

```
% Funcion de activacion Capa 1 Filtro 1
y1_1=max(0,y1_1);
```

Figura 6-17 Código de Matlab que realiza la evaluación de la función de activación.

**Capas de pooling (submuestreo)**

Tomando en cuenta el punto 7 en la identificación de procesos, en la figura 6-18 se muestra el código generado para una capa de submuestreo, donde se puede seleccionar el tamaño de la ventana del submuestreo y del salto.

```
% Submuestreo Capa 1 Filtro 1
s=size(y1_1);
o=0;
p=0;
pol=0:0;
for i=1:2:s(1)-1
o=o+1;
p=0;
for j=1:2:s(2)-1
    p=p+1;
    ventana=y1_1(i:i+1, j:j+1);
    m=max(ventana(:));
    pol(o,p)=m;
    end
    end
```

Figura 6-18 Código de Matlab que realiza el submuestreo.

**Capa de transformación**

El código que transforma la salida matricial de la etapa convolucional en un vector de datos se muestra en la figura 6-19, el código va transformando cada matriz en un vector y lo guarda en una variable, toma la siguiente matriz, la transforma y la junta con la variable ya existente hasta transformar todas las matrices de la etapa convolucional.

```
% Transformacion a vector de entrada
s=size(x3_1);
Entrada_Capa1=[];
for p=1:l;
    Salida_conjunto = genvarname(['x3_',num2str(p)]);
    r=eval(Salida_conjunto);
    r = reshape(r,[1,(s(1)*s(2))]);
    Entrada_Capa1=[Entrada_Capa1 r];
```

```
9. end
10.      SumaEntradaRed=sum(Entrada_Capa1);
```

Figura 6-19 Código que transforma la salida de la etapa convolucional en entrada de la etapa completamente conectada.

**Capa totalmente conectada**

El código que realiza la multiplicación de los pesos sinápticos con las capas de la red neuronal previamente construye la matriz de pesos sinápticos ya sea con los datos del usuario como se muestra en la figura 6-20, o de manera aleatorio con el valor mínimo y máximo proporcionado por el usuario como se muestra en la figura 6-21.

```
1. % Valores Pesos 1
2. PesosCapa1 =
   importdata('C:\Users\dm_ga\Desktop\Pesos_Capa4.txt');
3.
4. % Capa Oculta 1
5. SalidaCapa1=PesosCapa1*Entrada_Capa1';
6. SalidaCapa1=SalidaCapa1';
```

Figura 6-20 Código que genera la matriz de pesos sinápticos a partir de datos del usuario y realiza la multiplicación con las neuronas.

```
1. % Valores Pesos 1
2. y=100;
3. s=size(Entrada_Capa);
4. PesosCapa1=[];
5. max=1.000000;
6. min=-1.000000;
7. for b=1:y;
8. for z=1:s(2);
9. PesosCapa1(b,z)=((max-min)*rand)+min;
10.      end
11.      end
12.
13.      % Capa Oculta 1
14.      SalidaCapa1=PesosCapa1*Entrada_Capa1';
15.      SalidaCapa1=SalidaCapa1';
```

Figura 6-21 Código que genera la matriz de pesos sinápticos a partir de datos del usuario y realiza la multiplicación con las neuronas.

# Capitulo 7. Resultados

En este capítulo se muestran los resultados obtenidos con la plataforma, se diseñan, entrenan y ponen a prueba diferentes redes neuronales para propósitos diferentes y se compara el desempeño de diferentes estructuras. Todas las redes neuronales asi como su entrenamiento fueron generados por la plataforma presentada.

## 7.1 Red neuronal para la identificación de números en imágenes

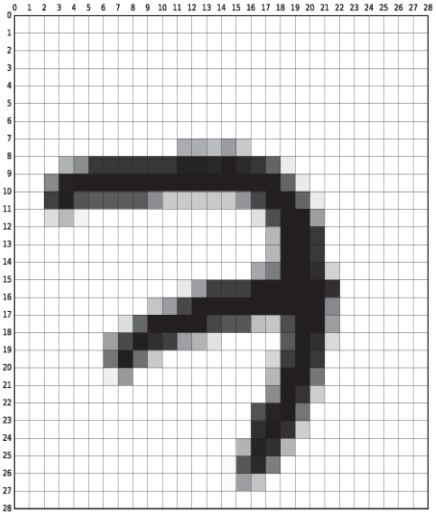
Se busca comparar diferentes estructuras de redes neuronales con la finalidad de identificar números en imágenes de $28 \times 28$ pixeles.

### 7.1.1 Base de datos MNIST

La base de datos del MNIST (Mixed National Institute of Standards and Technology) fue presentada por LeCun (*MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges*, n.d.) en 1998. Desde entonces, este conjunto de datos ha sido ampliamente utilizado como banco de pruebas para diferentes propuestas de aprendizaje automático y reconocimiento de patrones. La base de datos MNIST contiene un total de 70,000 instancias, de las cuales 60,000 son para entrenamiento y el resto para pruebas.

Las imágenes originales se sometieron a preprocesamiento. Este procedimiento involucró primero la normalización de las imágenes para que cupieran en un cuadro de $20 \times 20$ píxeles mientras se conservaba la relación de aspecto. Luego, se aplicó un filtro anti-aliasing y, como resultado, las imágenes en blanco y negro se transformaron efectivamente en escala de grises. Finalmente, se introdujo un relleno en blanco para ajustar las imágenes en un cuadro más grande de $28 \times 28$píxeles. Un ejemplo de una instancia correspondiente al dígito '7' se puede encontrar en la Figura 7-1a.

Al observar la figura 7-1, podemos ver que algunos dígitos pueden confundirse fácilmente. Dependiendo de cómo se escriban, puede ser difícil dilucidar si un determinado dígito es un '4' o un '9', por poner un ejemplo. Algunos casos de dígitos difíciles de reconocer, incluso para humanos, se pueden ver en la Figura 7-1b (ver el noveno '2' o el noveno '7').



(a) Ejemplo de instancia de digito '7'.

(b) 100 muestras del juego de entrenamiento MNIST.

Figura 7-1 Ejemplo de la base de datos MNIST.

Debido a que MNIST ha sido ampliamente utilizado para probar el comportamiento de muchas implementaciones de clasificadores, ha habido un esfuerzo por publicar algunas clasificaciones en el pasado, utilizando MNIST como punto de referencia. La mayoría de estas clasificaciones, así como la literatura establecida, utilizan la métrica de "tasa de error de prueba" cuando se refieren al rendimiento sobre MNIST. Esta métrica es el porcentaje de instancias clasificadas incorrectamente.

### 7.1.2 Estructura de redes neuronales

Se desarrollan y comparan 6 estructuras diferentes de redes neuronales, tres redes neuronales convolucionales (red neuronal 1, 2 y 3) y tres redes neuronales multicapa (red neuronal 4, 5 y 6). Las tres redes convolucionales tendrán la misma configuración para la etapa totalmente conectada, dos capas ocultas, de 80 y 60 neuronas respectivamente, la función de activación utilizada es tangente hiperbólica para dichas capas, la capa de salida cuenta con 10 neuronas con sigmoide como función de activación, lo cual nos permitirá identificar los 10 diferentes números posibles de las imágenes. La etapa convolucional ira variando entre cada red neuronal, la primera utilizará un filtro para la convolución de $9 \times 9$, la segunda añadirá a la configuración previa otra capa convolucional con un filtro de $5 \times 5$, por último, la última añadirá una capa extra con un filtro de $3 \times 3$, en la figura 7-2 se muestra las diferencias de la etapa convolucional de cada red, ninguna red cuenta con etapa de submuestreo ya que el tamaño de la entada es de $28 \times 28$. La operación de convolución esta ilustrada con un asterisco y $w$ es el kernel utilizado en esta operación, $b$ es el valor de sesgo que se suma a cada elemento de la imagen, $y_p$ es el valor de cada elemento en las capas ocultas.

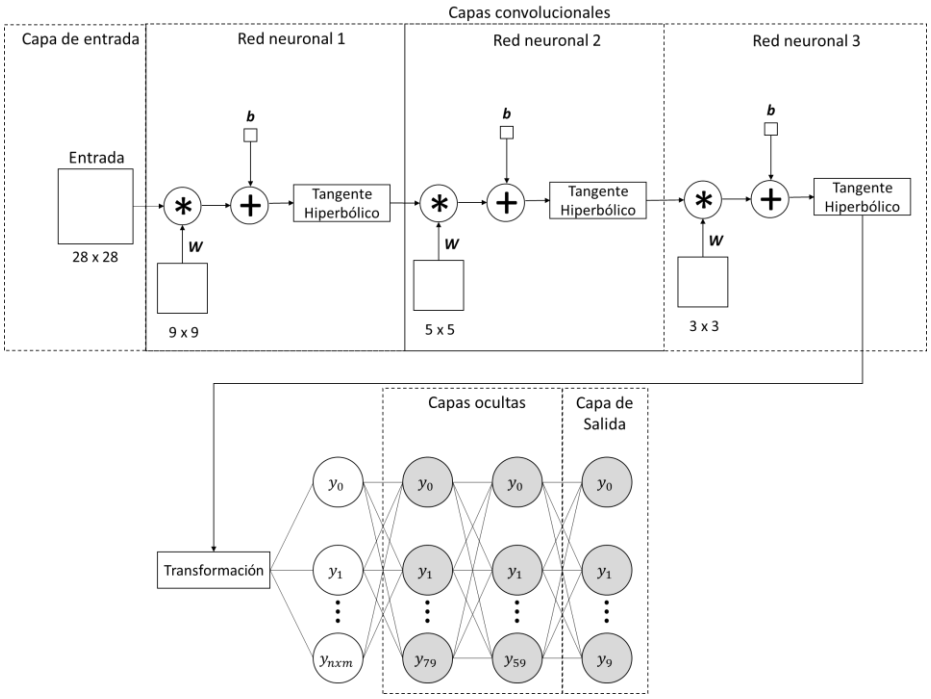



Figura 7-2 Estructuras redes neuronales convolucionales.

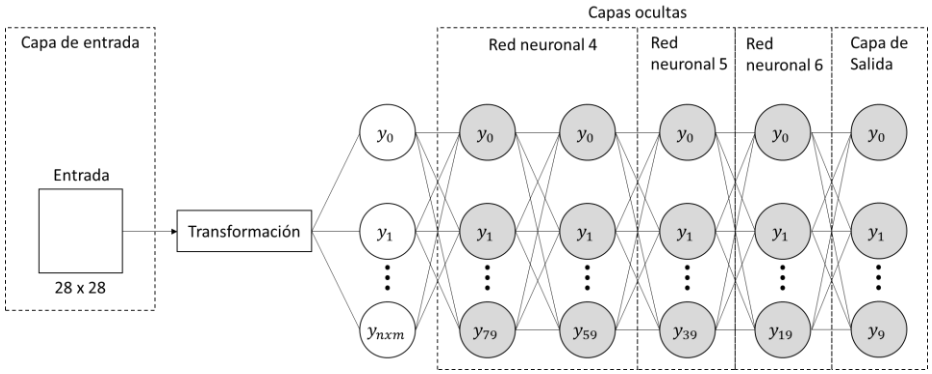



Figura 7-3 Estructuras redes neuronales multicapa.

En la etapa de transformación se realiza un reacomodo de la información para conectar la etapa convolucional con la completamente conectada, se toma la información en formato de matriz y se convierte en un vector, el número de entradas a las capas ocultas $y_n$ es de $n \times m$ donde $n$ y $m$ son las dimensiones de salida de la última capa convolucional, en el caso de las redes multicapa se hace el reacomodo a la imagen de entrada.

Para las redes neuronales multicapa, en todas se utiliza la tangente hiperbólica como función de activación a excepción de la capa de salida que utiliza función sigmoide, la primera red tiene dos capas de 80 y 60 neuronas respectivamente, la segunda red agrega una capa extra de 40 neuronas y la tercera red agrega otra capa de 20 neuronas, en la figura 7-3 se pueden observar las diferencias de cada red en la etapa de capas ocultas.

### 7.1.3 Entrenamiento

Para las seis redes se utiliza una tasa de aprendizaje de $\eta = 0.0001$ y los valores de pesos sinápticos, valores de los kernel y bias se inicializaron de manera aleatoria en un rango de -1 a 1, para todas las estructuras propuestas se realizan tres entrenamientos, de un entrenamiento de 100,000 iteraciones.

En cada iteración de entrenamiento se realiza una corrida de la red neuronal, se obtiene el error,

Para crear al código de ejecución y entrenamiento de las redes neuronales se utilizó el software propio.

Durante el entrenamiento se realiza la validación con una base de datos dedicada para este objetivo diferente a la de entrenamiento como el mostrado en la figura 7-4. Todas las redes entrenadas tuvieron un comportamiento parecido en si grafica de entrenamiento y de validación, esto significa que la red si entreno para un caso general y no solo para la base de datos con la que se entrenó.

En la figura 7-4 se muestra la diferencia de la evolución del entrenamiento de las diferentes redes neuronales. La red 2 y 3 fueron las que tuvieron el peor desempeño, la red 2 tardo mucho en empezar a disminuir el error y tuvo un rebote después de las 20,000 iteraciones, pero logro recuperarse y disminuir el error. La red 3 se estancó y ya no pudo disminuir el error. Las redes 1 y 4 siguieron un comportamiento parecido en el entrenamiento y llegaron a un error parecido que les permitió tener una tasa de error debajo del 15%. La red 6 dejo de entrenar a las 30,000 iteraciones a pesar de que empezó bien el entrenamiento. La red 5 fue la que más rápido redujo el error y mantuvo ese error a partir de las 30,00 iteraciones.

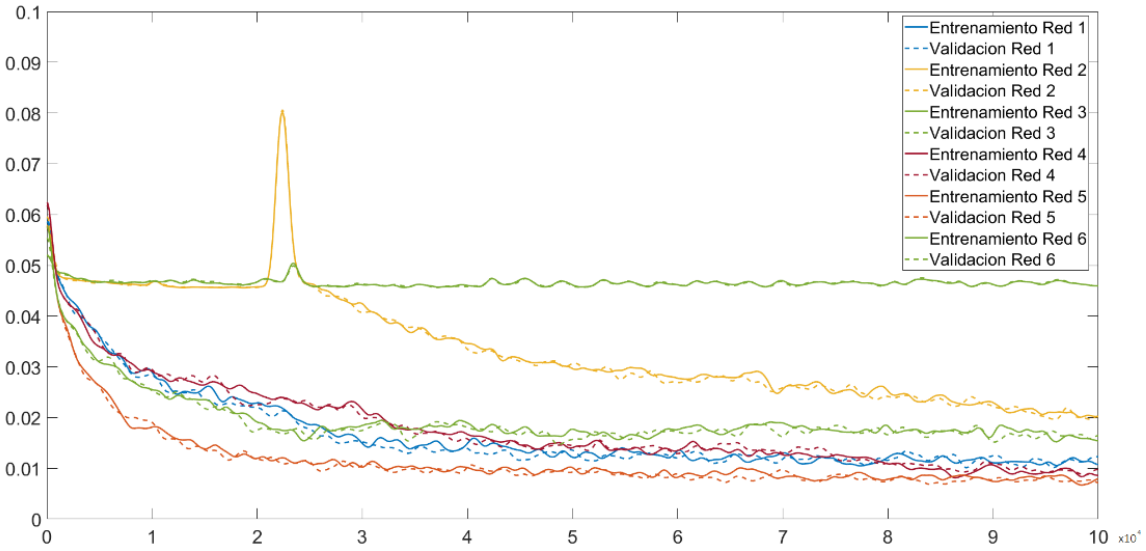



Figura 7-4 Resultado de entrenamiento de las redes neuronales.

### 7.1.4 Desempeño

Se configuro cada estructura en el software y se obtuvo el código de ejecución y de entrenamiento de cada red, en la tabla 7-1 se muestran los resultados de las redes neuronales convolucionales y en la tabla 7-2 se muestra los resultados de las redes neuronales multicapa.

En la tabla 7-2 se muestran los resultados del entrenamiento de las redes multicapa. En esta comparación no se observa una diferencia tan grande en los tiempos de entrenamiento. El mejor desempeño lo tuvo la red 4 con tres capas ocultas.

Estos resultados concuerdan con la gráfica de entrenamiento de la figura 7-4, en los resultados las peores redes fueron la 2 y la 3, y las mejores fueron 4 y 5.

Tabla 7-1 Resultados de redes neuronales convolucionales.

| Configuración de red neuronal | Numero de iteraciones de entrenamiento | Tasa de aprendizaje (Eta) | Tasa de error (%) | Tiempo de entrenamiento (S) |
|---|---|---|---|---|
| Red Neuronal 1 | 100,000 | 0.0001 | 12.12 | 93.23 |
| Red Neuronal 2 | 100,000 | 0.0001 | 16.18 | 176.74 |
| Red Neuronal 3 | 100,000 | 0.0001 | 16.19 | 249.25 |

Agregar más capas convolucionales a la red no mejora el rendimiento de la red neuronal como se observa en la tabla 7-1, el porcentaje más alto de exactitud se obtiene con la red que tiene solo una capa convolucional.

Observando los tiempos de entrenamiento en la tabla 7-2 y comparándolos con las redes que tienen capas convolucionales, se concluye que las capas convolucionales agregar tiempo de ejecución considerable al entrenamiento. Comparando las 3 redes, la red con 3 capas ocultas es la que mejor rendimiento tuvo.

Comparando las redes con y sin capa convolucional, se obtuvo un mejor rendimiento con las primeras. El tamaño de las imágenes de la base de datos utilizada influye en el bajo rendimiento de las capas convolucionales ya que son de tamaño reducido y con capas ocultas basta para extraer la información.

Tabla 7-2 Resultados de redes neuronales multicapa.

| Configuración de red neuronal | Numero de iteraciones de entrenamiento | Tasa de aprendizaje (Eta) | Tasa de error (%) | Tiempo de entrenamiento (S) |
|---|---|---|---|---|
| Red Neuronal 4 | 100,000 | 0.0001 | 10.66 | 24.68 |
| Red Neuronal 5 | 100,000 | 0.0001 | 8.78 | 25.74 |
| Red Neuronal 6 | 100,000 | 0.0001 | 19.99 | 26.42 |

## 7.2 Red neuronal para la identificación de 10 parámetros del modelo dinámico de un robot de 2 grados de libertad

El modelo dinámico de un robot manipulador contiene en su estructura matemática parámetros como centros de gravedad, masas, momentos de inercia y coeficientes de fricción. (Reyes Cortés, 2011)Estos parámetros son desconocidos; este es el caso de la mayoría de los robots comerciales donde el fabricante no proporciona sus valores nominales. Si bien existen herramientas de teoría de control, como esquemas adaptativos y controladores robustos que permiten errores en los parámetros dinámicos, el conocimiento de estos es crucial para la mayoría de los esquemas basados en el modelo dinámico de un robot.

El problema de la identificación paramétrica ha llevado a la derivación de varios esquemas de identificación que se han convertido en una herramienta atractiva para determinar los parámetros dinámicos de la manipulación de robots. especialmente cuando es difícil medirlos directamente. Sin embargo, la naturaleza no lineal del modelo dinámico de robots manipulativos hace que la tarea de identificación paramétrica no sea trivial.

La identificación del sistema se puede definir como la caracterización de un sistema dinámico, observando su comportamiento medible. Cuando la información a priori sobre las reglas que rigen un sistema no existe o es demasiado compleja, por ejemplo, en señales electrocardiográficas, las técnicas de identificación se utilizan para construir un modelo solo observando los datos de entrada y salida. Por lo tanto, el sistema se llama caja negra porque se desconoce el comportamiento interno.

En muchos sistemas físicos, nuestro conocimiento de las leyes mecánicas, químicas o eléctricas nos permite formular un modelo, que es la herramienta fundamental para estudiar el sistema, ya sea analíticamente o a través de simulaciones. Sin embargo, no importa cuán profunda sea nuestra percepción física, los parámetros de cualquier modelo son inexactos.

Existe un área de oportunidad para introducir redes neuronales en este tema, en el uso de una red neuronal Hopfield para la estimación de parámetros, en el contexto de la identificación de un sistema dinámico. CNN podría ofrecer una solución a este problema diseñando una red neuronal que realizara la identificación paramétrica de un sistema dinámico, en un caso particular el de un robot manipulador(Atencia et al., 2005).

### 7.2.1 Descripción del robot.

Figura 7-5 muestra el diagrama de un robot cartesiano de 2 grados de libertad para realizar la identificación paramétrica, $x_T$ y $z_T$ son las longitudes totales del primer y segundo eje, $x$ y $z$ son la posición actual del robot. El modelo dinámico de un robot cartesiano de dos grados de libertad viene dado por la ecuación (7-1) (Urrea & Pascal, 2021), donde el modelo de fricción de Coulomb se ilustra en (Duan & Singh, 2006).

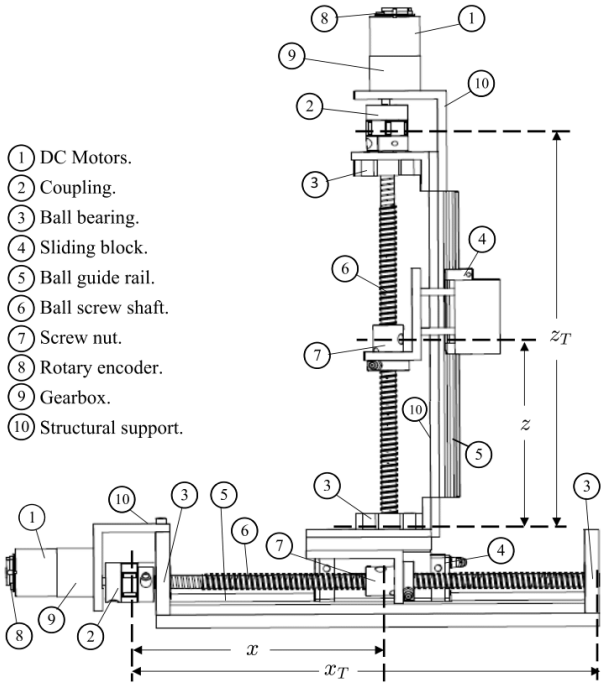



Figura 7-5 Diagrama esquemático del robot cartesiano de dos grados de libertad.

$$\begin{aligned} \tau_x &= I_x(\ddot{q})_x + \tau_{ox} + b_x\dot{q}_x + k_x \tanh(k_{sx}\dot{q}_x) \\ \tau_z &= I_z(\ddot{q})_z + \tau_{oz} + b_z\dot{q}_z + k_z \tanh(k_{sz}\dot{q}_z) \end{aligned} \tag{7-1}$$

donde:

$q = [q_x, q_z]^T$ Vector of position.

$\dot{q} = \frac{dq}{dt}$ Vector of velocity.

$\ddot{q} = \frac{d\dot{q}}{dt}$ Vector of accelerations.

$I_x, I_z$ Inertial parameters.

$\tau_{ox}$ $X$ axis offset.

$\tau_{oz}$ $Z$ axis offset and gravity torque.

$b_x, b_z$ Viscosity parameters.

$k_x, k_z$ Coulomb external coefficients.

$k_{sx}, k_{sz}$ Conditioning factor of the Coulomb friction.

El robot sigue una trayectoria predefinida utilizando un sistema de control proporcional derivado (PD). Las señales del robot (par, posición, velocidad y aceleración) se utilizan para crear la imagen de entrada a CNN. La base de datos de entrenamiento tiene 20000 imágenes etiquetadas para entrenamiento y 20000 para validación.

El modelo dinámico de la ecuación (7-1) tiene la propiedad de escalar en amplitud. Por lo tanto, la inercia ($I_x, I_z$), la viscosidad ($b_x, b_z$), las compensaciones ($\tau_{o_x}, \tau_{o_z}$) y los coeficientes externos de Coulomb ( $k_x, k_z$) puede tener valores desde cero hasta la unidad para propósitos de entrenamiento de CNN. Sin embargo, el argumento del modelo de fricción de Coulomb (Duan & Singh, 2006) ($k_s x, k_s z$) tiene valores que exceden la unidad. En este trabajo, la función de arco tangente se usa para tener valores de 0 a 1, como muestra la figura 7-6.

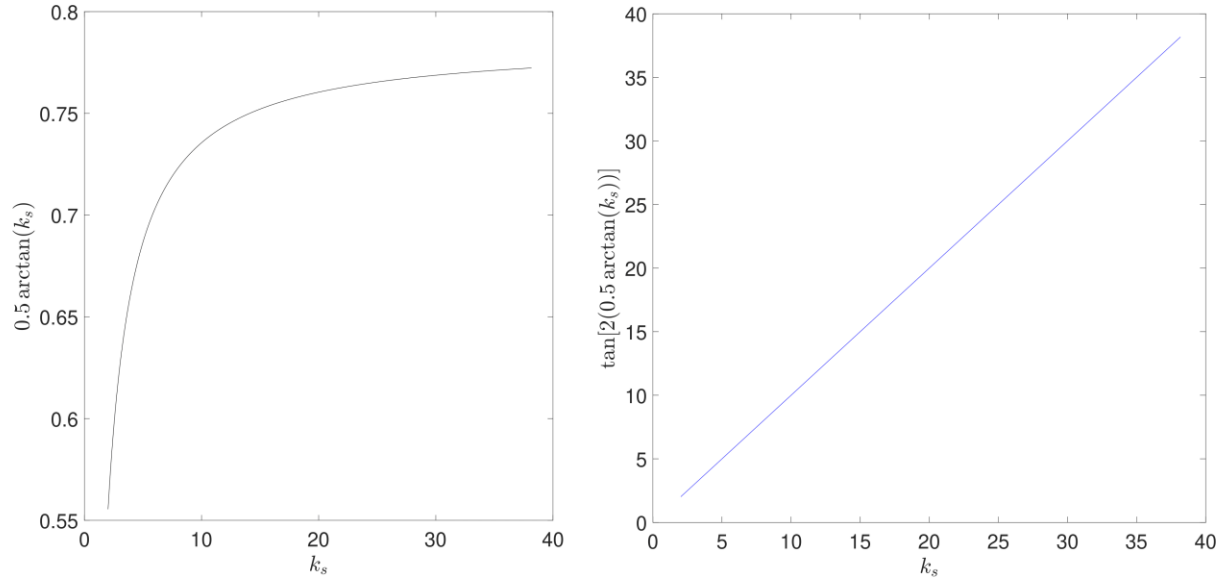



Figura 7-6 Gráfica de la función de arco tangente.

Las señales del robot y un conjunto de parámetros crean la imagen utilizando las señales sub-muestreadas y normalizadas de la ecuación (7-1). Primero, el vector α de la ecuación (7-2) contiene el par de entrada normalizado y sub-muestreado $\tau_r$. Un vector ψ contiene dos veces el par $\tau_r$ y el modelo dinámico $f$ de la ecuación (7-1) que reconstruye el par usando el conjunto de parámetros de entrada $\beta_n$: Si $\beta_n$ es el conjunto real de parámetros, ψ es aproximadamente igual a α. La primera parte de la imagen $Z1$ es estos dos vectores multiplicados en forma de transposición, como muestra ecuación (7-2).

La otra parte de la imagen $Z2$ utiliza las versiones normalizadas de α y ψ llamadas $\alpha_2$ y $\psi_2$: La función normalize varía su argumento desde 0 a 1. Las funciones logarítmicas $log$ y $log_{10}$ mejoran algunas regiones de imagen con picos en algunas partes: La red neuronal convolucional identifica rápidamente esta información de imagen debido a sus operaciones de convolución. La imagen es finalmente construida, añadiendo partes Z1 y Z2, como muestra la ecuación (7-2).

$$\begin{aligned} \alpha &= \tau_r \\ \psi &= 2\tau_r - f(q_r, \dot{q}_r, \ddot{q}_r, \beta_n) \end{aligned} \tag{7-2}$$

$$Z_1 = \psi\alpha^T$$
$$\alpha_2 = normalize(\alpha), \psi_2 = normalize(\psi)$$
$$\alpha_2 = \log(\alpha_2 + 1 \times 10^{-3})$$
$$\psi_2 = \log(\psi_2 + 1 \times 10^{-3})$$
$$Z_2 = \psi_2\alpha_2^T$$
$$Z = [Z_1/max(Z_1)] + [Z_2/max(Z_2)]$$

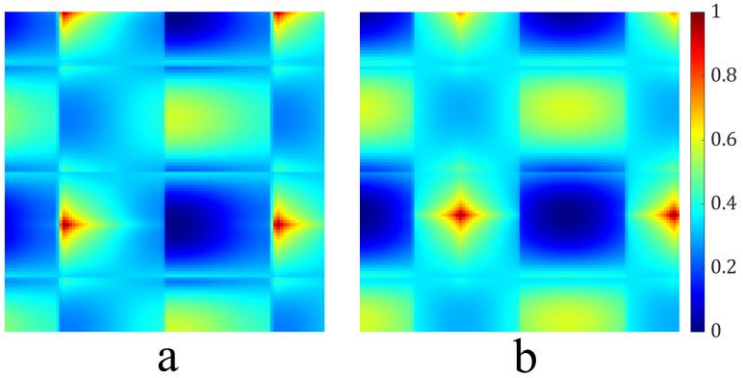



Figura 7-7 Imágenes generadas para la entrada a la red neural de identificación paramétrica. a) Par 1. b) Par 2.

### 7.2.2 Estructura de las redes neuronales

Utilizando las imágenes generadas como la figura 7-7 para el entrenamiento de la red neuronal y sabiendo cuántos parámetros se buscan identificar, se proponen 4 CNN con el mismo propósito, pero con una estructura diferente. Todas las redes neuronales tienen una entrada de 2 imágenes de $100 \times 100$ cada una, y como salida los 10 parámetros a identificar.

La primera red neuronal (figura 7-8), tiene 3 capas convolucionales, y 2 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $11 \times 11 \times 2 \times 10$, $9 \times 9 \times 10 \times 10$, y $5 \times 5 \times 10 \times 20$ respectivamente. En la primera capa, hay una capa máxima de submuestra de tamaño. Las capas totalmente conectadas tienen 50 y 100 neuronas respectivamente. Cada capa agrega un vector de sesgo. Todas las capas convolucionales están hechas con la ecuación (7-3), donde $p = [0, P-1]$ es el índice del mapa de salida, $Y_p$ es el mapa de salida, $f_p$ es la función de activación, $n = [0, N-1]$ es el mapa de entrada, $b_p$ es la desviación, y $W_n$ es el kernel.

$$Y_p = f_p\left[b_p \sum_{n=0}^{N-1} conv(X_n, W_n)\right] \tag{7-3}$$

Para todas las redes neuronales la función de activación de las capas de convolución y totalmente conectadas es la Unidad Lineal Rectificada (ReLU) ecuación (7-4). La capa de salida utiliza la función sigmoide ecuación (7-5) donde $c$ es una constante para ajustar el nivel de saturación.

$$f(z) = \max(0, z) \tag{7-4}$$

$$f(z) = [1 + \exp(-cz)]^{-1} \tag{7-5}$$

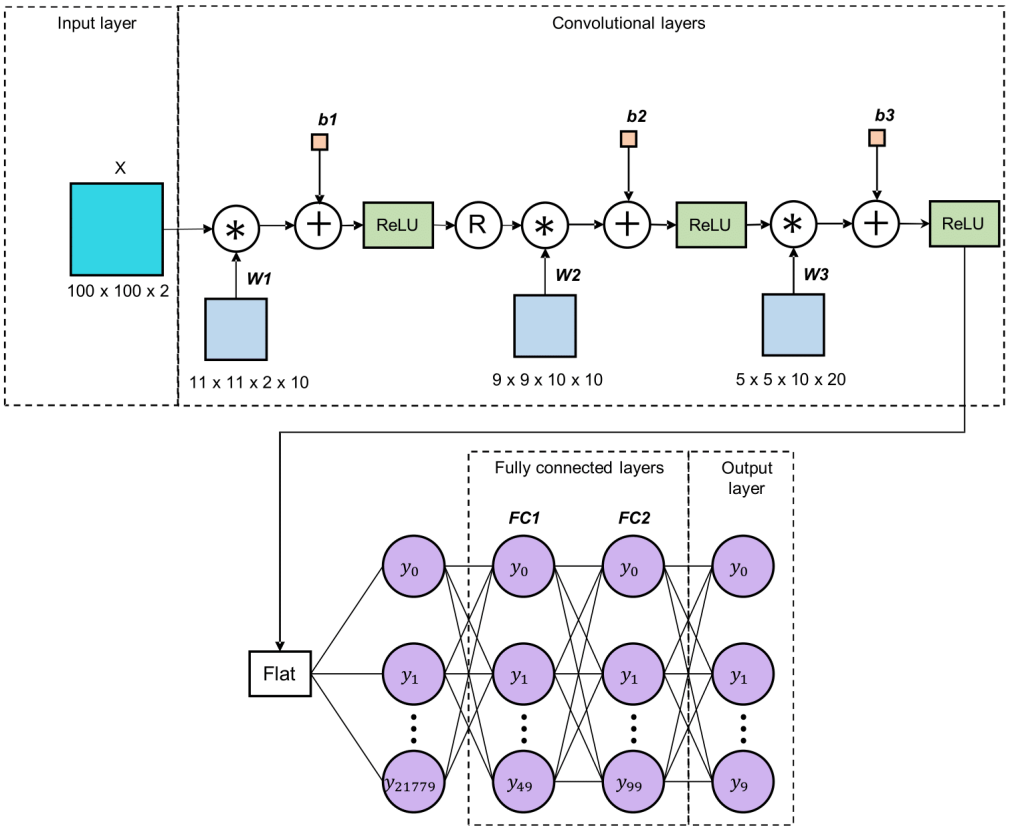



Figura 7-8 Estructura de la red neuronal uno.

La segunda red neuronal (figura 7-9), tiene 3 capas convolucionales, y 2 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $9 \times 9 \times 2 \times 10$, $5 \times 5 \times 10 \times 10$ y $3 \times 3 \times 10 \times 10$ respectivamente. En la primera y segunda capa, hay una capa máxima de submuestreo de tamaño $2 \times 2$. Las capas totalmente conectadas tienen 50 y 100 neuronas respectivamente.

La tercera red neuronal (figura 7-10), tiene 5 capas convolucionales, y 3 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $9 \times 9 \times 2 \times 10$, $5 \times 5 \times 10 \times 10$, $5 \times 5 \times 10 \times 10$, $3 \times 3 \times 10 \times 10$, y $2 \times 2 \times 10 \times 10$ respectivamente. En la primera y tercera capa, hay una capa máxima de submuestra de tamaño $2 \times 2$. Las capas totalmente conectadas tienen 100, 100 y 100 neuronas respectivamente.

La cuarta red neuronal (figura 7-11), tiene 3 capas convolucionales, y 3 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $11 \times 11 \times 2 \times 10$, $10 \times 10 \times 10 \times 10$, y $3 \times 3 \times 10 \times 10$ respectivamente. En la primera y segunda capa, hay una capa de submuestra máxima de tamaño $2 \times 2$. Las capas totalmente conectadas tienen 50, 100 y 50 neuronas respectivamente.

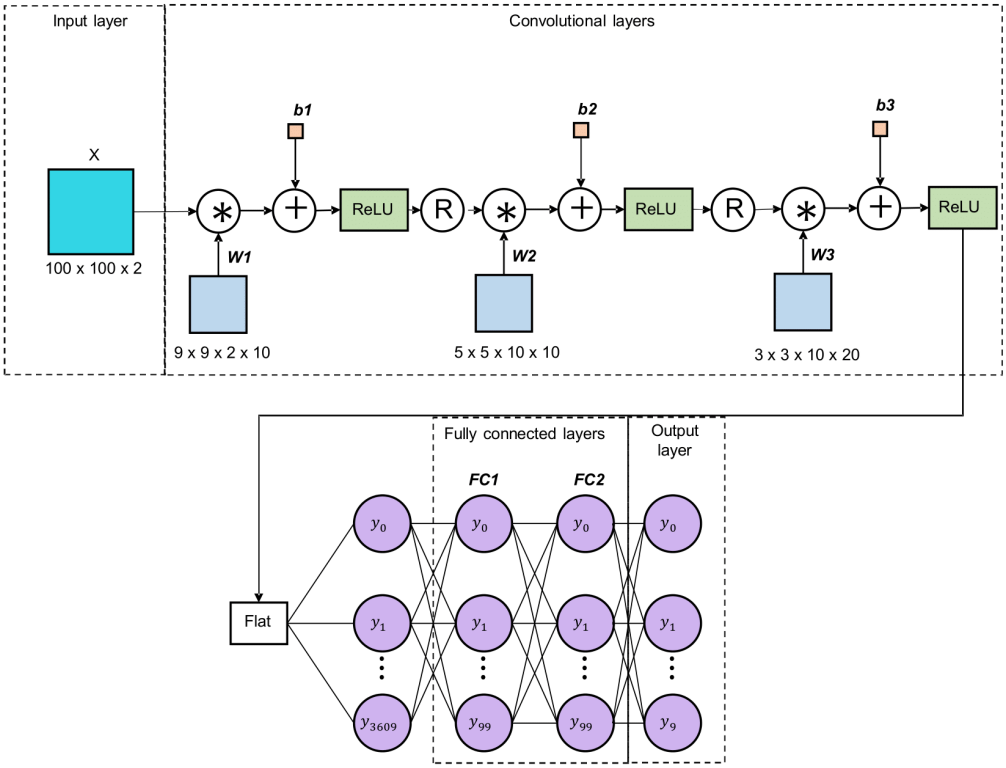



Figura 7-9 Estructura de la red neuronal dos.

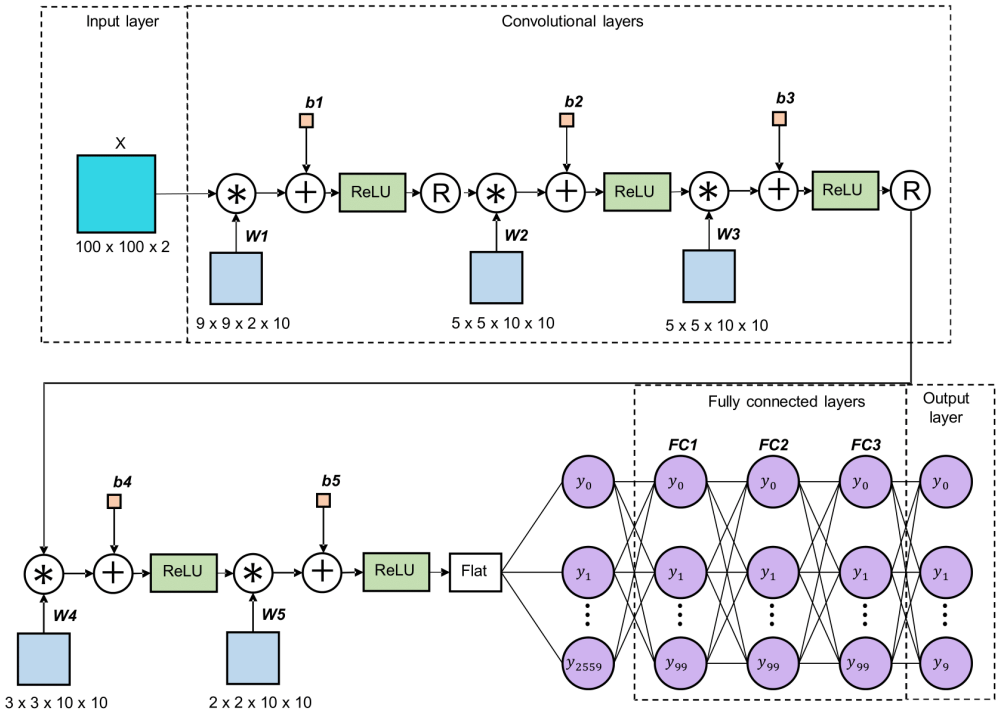



Figura 7-10 Estructura de la red neuronal tres.

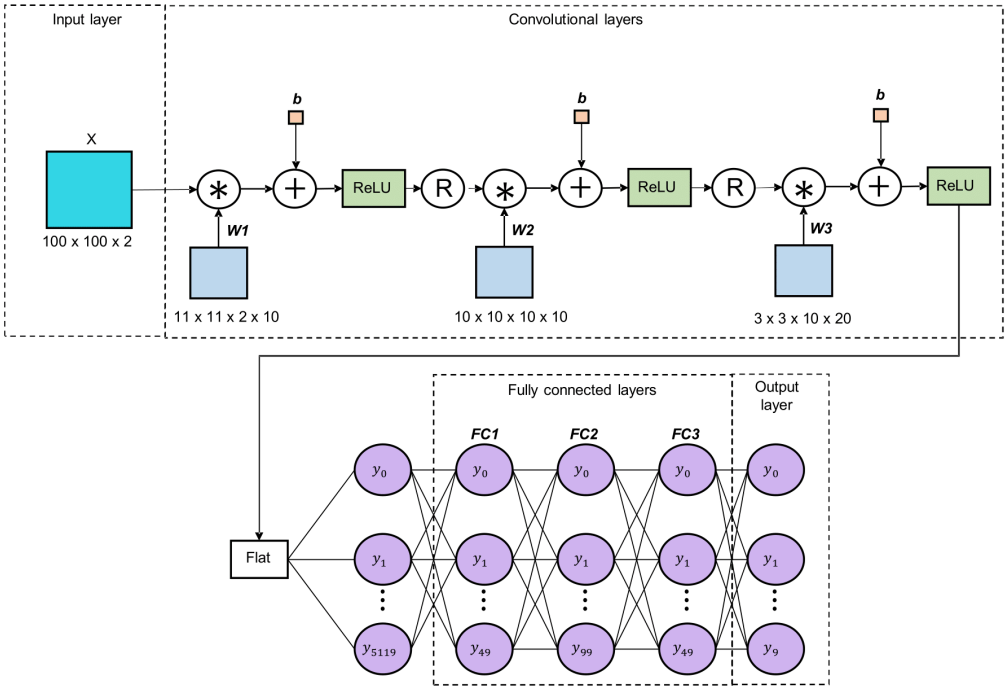



Figura 7-11Estructura de la red neuronal cuatro.

### 7.2.3 Entrenamiento

Se llevó a cabo el entrenamiento de las cuatro redes neuronales, utilizando la retropropagación con el mismo factor de aprendizaje de $1.4 \times 10^{-3}$ y 500,000 iteraciones. La figura 7-12 muestra la evolución comparativa del entrenamiento en red. A partir de las 50,000 iteraciones, se puede notar una diferencia en la evolución del entrenamiento de las redes neuronales. La segunda CNN redujo el error primero, obteniendo el error

mínimo de las cuatro. La primera, tercera y cuarta redes tuvieron un comportamiento similar. Incluso el error final de estas redes fue muy similar.

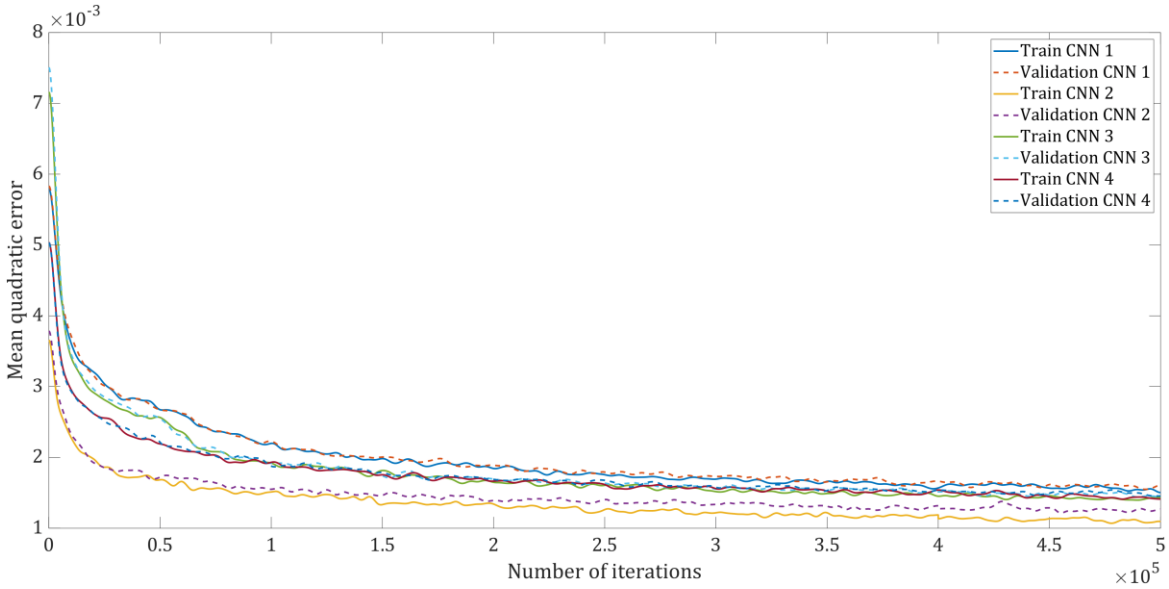



Figura 7-12 Entrenamiento y validación de las redes neuronales propuestas.

### 7.2.4 Desempeño

En este trabajo, se utiliza una métrica de similitud mostrada en el algoritmo 2 utiliza la transformación discreta del coseno (DCT) para determinar la similitud en frecuencia (línea 3): Esta métrica evaluó las señales de entrada en tiempo (línea 2) y frecuencia. Además, la métrica utiliza un factor de equilibrio de 0,4 en los resultados de salida (línea 8).

**Algoritmo 2**

**Input:** Señal 1 $x$, señal 2 $y$

1. $a \Leftarrow |\sum x - \sum y|[\sum(|x| - |y|)]^{-1}$
2. $a \Leftarrow 1 - a$
3. $\{X, Y\} \leftarrow \{DCT\}\{x, y\}$
4. $X \leftarrow X/max(X)$
5. $Y \leftarrow Y/max(Y)$
6. $b \leftarrow (\sum|X - Y|)(\sum|X| + |Y|)\text{^}\{-1\}$
7. $b \leftarrow 1 - b$
8. $z \leftarrow (ab)^{0.4}$

**Output:** Similitud $z$.

La evaluación numérica de similitud se realiza mediante la generación de entrada para la red neuronal mediante el desarrollo de dos pares a partir de la simulación del modelo dinámico de la ecuación (7-1) con parámetros definidos, entrando en la red neuronal entrenada y obteniendo los parámetros de salida. Los toques obtenidos de la red neuronal se reconstruyen, y se compara su similitud.

Se realizaron dos identificaciones paramétricas una con datos simulados y otra con datos experimentales.

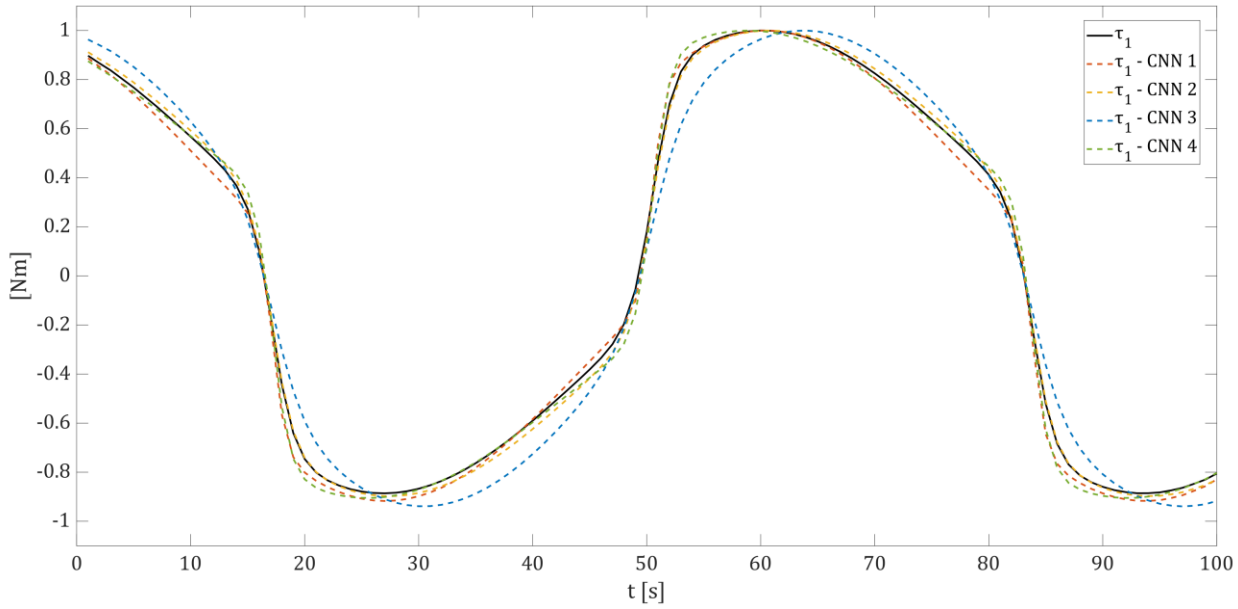



Figura 7-13 $\tau_1$ simulado y reconstruido por CNN.

Figura 7-13 y 7-14 muestran la identificación paramétrica de simulación utilizando el 1,2, 3 y 4 CNN de par 1 y 2, respectivamente. La reconstrucción del par se muestra con estos parámetros. Al identificar el par 1, el mejor tenía la red neuronal 2. La red neuronal 3 tenía un rendimiento comparable, para el par 2 era al revés. La red neuronal 1 tuvo el peor rendimiento en ambos pares. Este patrón es consistente con el observado en el entrenamiento figura 7-12.

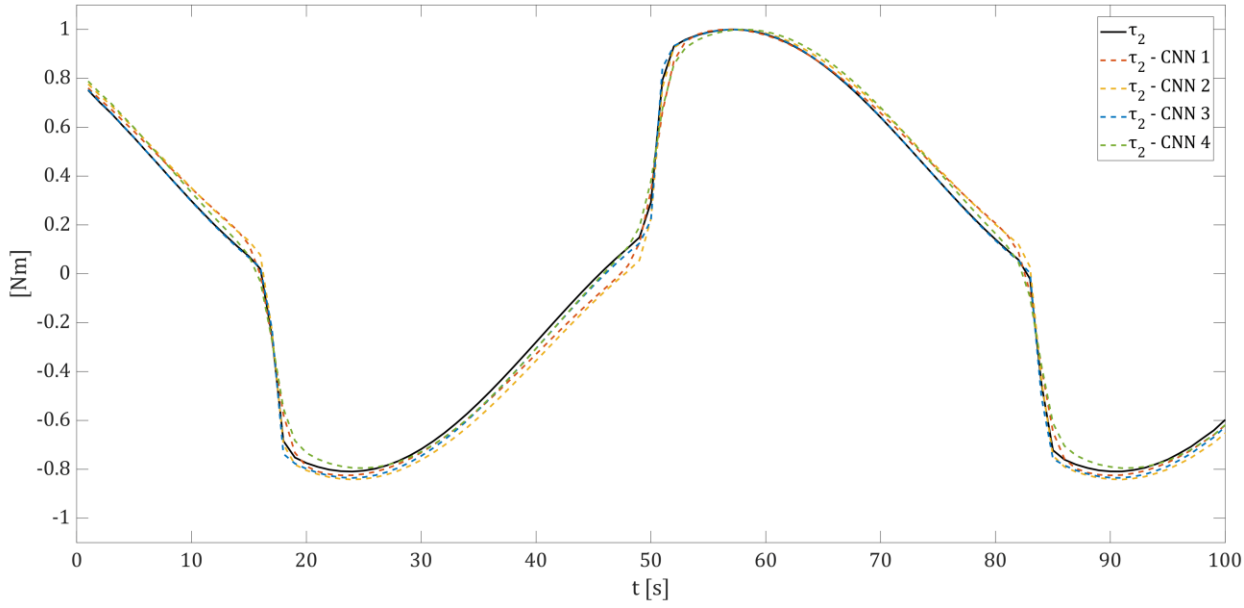



Figura 7-14 $\tau_2$ simulado y reconstruido por CNN.

La tabla 7-3 muestra el valor de los parámetros identificados en comparación con los utilizados en la ecuación (7-1) para generar el par de entrada e imágenes de entrada. Tomando como referencia los pares

reconstruidos de la figura 7-13 y 7-14, en los parámetros $K_{sx}$ y $K_{sz}$ es donde se puede encontrar la mayor diferencia numérica.

Tabla 7-3 Valor de los parámetros identificados, datos simulados.

| Parámetro | Valor numérico | Parámetros identificados CNN 1 | Parámetros identificados CNN 2 | Parámetros identificados CNN 3 | Parámetros identificados CNN 4 |
|---|---|---|---|---|---|
| $I_x$ | 0.4319 | 0.4871 | 0.4006 | 0.2850 | 0.4402 |
| $\tau_{off_x}$ | 0.0402 | 0.0152 | 0.0310 | -0.0024 | 0.0249 |
| $b_x$ | 0.5614 | 0.6565 | 0.5796 | 0.8431 | 0.4409 |
| $K_x$ | 0.8083 | 0.7239 | 0.8212 | 0.6703 | 0.9088 |
| $K_{sx}$ | 5.5693 | 9.0252 | 5.5007 | 3.6993 | 7.1746 |
| $I_z$ | 0.6991 | 0.6468 | 0.6464 | 0.7014 | 0.6644 |
| $\tau_{off_z}$ | 0.3991 | 0.3870 | 0.3741 | 0.3790 | 0.4099 |
| $b_z$ | 0.5037 | 0.4300 | 0.4896 | 0.5165 | 0.5803 |
| $K_z$ | 0.5463 | 0.6436 | 0.6221 | 0.5571 | 0.5151 |
| $K_{sz}$ | 17.2003 | 7.9548 | 15.8723 | 25.0174 | 8.8502 |

La tabla 7-4 muestra la evaluación de la similitud de la reconstrucción del par con los valores identificados por las diferentes redes neuronales propuestas. El algoritmo 2 fue usado para la evaluación. El rendimiento de cada CNN en la evaluación de similitud del par 1 difiere del par 2. Sabiendo esto, puede usar CNN con un mejor rendimiento para cada par.

Tabla 7-4 Evaluación numérica de la similitud entre pares simulados y reconstruidos con redes neuronales.

| | Similitud $\tau_1$ | Similitud $\tau_2$ |
|---|---|---|
| CNN 1 | 97.26% | 97.57% |
| CNN 2 | **99.39%** | 98.26% |
| CNN 3 | 95.27% | **98.44%** |
| CNN 4 | 98.07% | 96.95% |

Se realizó una evaluación con el par obtenido experimentalmente en el robot, y se realizó una trayectoria con un control proporcional. Los datos obtenidos se convirtieron a una imagen y entraron en las 4 redes neuronales para obtener la identificación paramétrica y completar la reconstrucción del par. La figura 7-15 muestra el par 1 y su comparación con los pares reconstruidos, la figura 7-16 lo mismo para el par 2. La forma del par experimental presenta más anomalías que el par generado numéricamente. En la figura 7-15, se observa que CNN 1 es el que se acerca más a la forma del par original, y el que tiene peor rendimiento es CNN 4. Incluso si todos siguen una trayectoria similar, algunos están más lejos en las partes curvas. Figura 7-16 muestra que CNN 2 se desvía menos en las curvas, y CNN 1 y 4 tienen peores resultados. Al igual que en la identificación paramétrica con valores numéricos, utilizando valores experimentales, el mejor resultado fue obtenido por un CNN diferente para cada par.

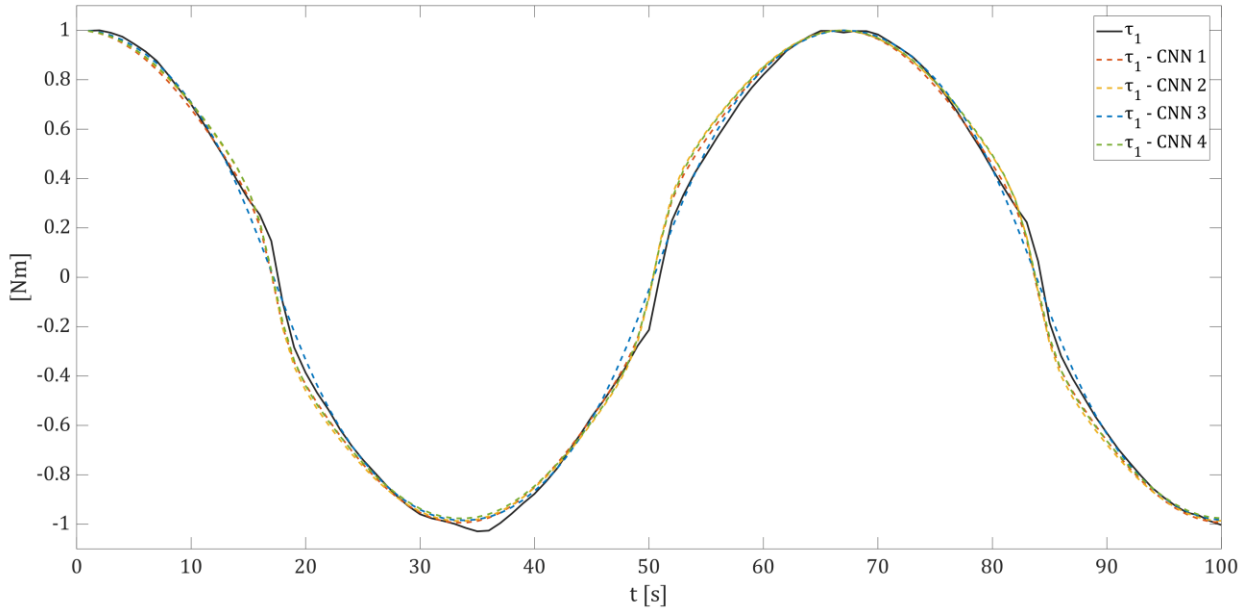



Figura 7-15 $\tau_1$ experimental y reconstruido por CNN.

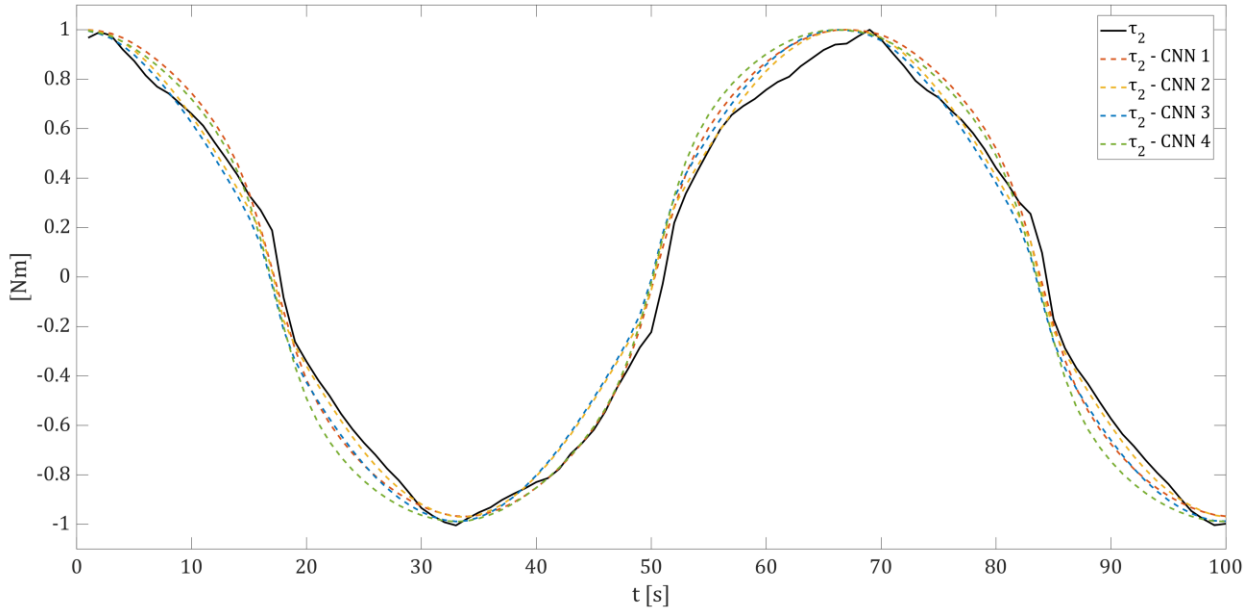



Figura 7-16 $\tau_2$ experimental y reconstruido por CNN.

La tabla 7-5 muestra el valor de los parámetros identificados por las 4 CNN; estos parámetros se utilizan para intentar recrear el par experimental 1 y 2. A diferencia de los parámetros de la tabla 7-3, los parámetros del robot experimental no están disponibles.

Tabla 7-5 Valor de los parámetros identificados, datos experimentales.

| Parámetro | Parámetros identificados CNN 1 | Parámetros identificados CNN 2 | Parámetros identificados CNN 3 | Parámetros identificados CNN 4 |
|---|---|---|---|---|
| $I_x$ | 0.0121 | 0.0141 | 0.0024 | 0.0104 |
| $\tau_o f f_x$ | 0.0059 | 0.0043 | 0.0075 | 0.0093 |
| $b_x$ | 0.3824 | 0.4151 | 0.3000 | 0.3575 |
| $K_x$ | 0.1000 | 0.0687 | 0.1917 | 0.1215 |
| $K_{sx}$ | 4.7573 | 13.7143 | 1.7734 | 6.4072 |
| $I_z$ | 0.0076 | 0.0255 | 0.0401 | 0.0318 |
| $\tau_{off_z}$ | 0.0131 | 0.0150 | 0.0113 | 0.0105 |
| $b_z$ | 0.3813 | 0.3717 | 0.4676 | 0.3284 |
| $K_z$ | 0.1642 | 0.1712 | 0.0783 | 0.2189 |
| $K_{sz}$ | 8.1397 | 5.5473 | 8.4448 | 3.3693 |

La evaluación experimental de similitud se realiza generando imágenes de entrada para la red neuronal a partir de pares de robot. Al entrar en la red neuronal entrenada y obtener los parámetros de salida, los parámetros obtenidos de la red neuronal se reconstruyen, y su similitud se compara en la tabla 7-6.

Tabla 7-6 Evaluación numérica de la similitud entre pares experimentales y reconstruidos con redes neuronales

| | Similitud $\tau_1$ | Similitud $\tau_2$ |
|---|---|---|
| CNN 1 | **95.32%** | 92.19\% |
| CNN 2 | 95.06% | 92.60\% |
| CNN 3 | 94.61% | 92.31\% |
| CNN 4 | 94.02% | **92.56\%** |

## 7.3 Comparación de identificación de 6 y 10 parámetros del modelo dinámico de un robot de 3 grados de libertad

### 7.3.1 Descripción del robot

El robot cartesiano se construye combinando movimientos ortogonales independientes que generan trayectorias complejas en un espacio tridimensional. En este análisis solo se identificaron los parámetros del eje $X$ y del eje $Y$. El robot fue diseñado principalmente para la impresión 3D, cuenta con un extrusor, la herramienta se puede reemplazar por un cortador láser, una broca de rectificado o una abrazadera para mover objetos. La distancia que puede recorrer el eje X y el eje Y es de 27,6 cm y 24,5 cm respectivamente. Los motores de inducción Z5D120-12 se implementaron para los ejes X e Y, tienen un consumo máximo de 120W y se alimentan con 12V, una velocidad máxima de 3100 rpm (revoluciones por minuto). Los motores están acoplados a unidades de engranajes 5GU9K con una relación 9:1. Se acopló un enconder E6B2-CWZ6C giratorio incremental de 1000 pulsos por revolución para medir la posición

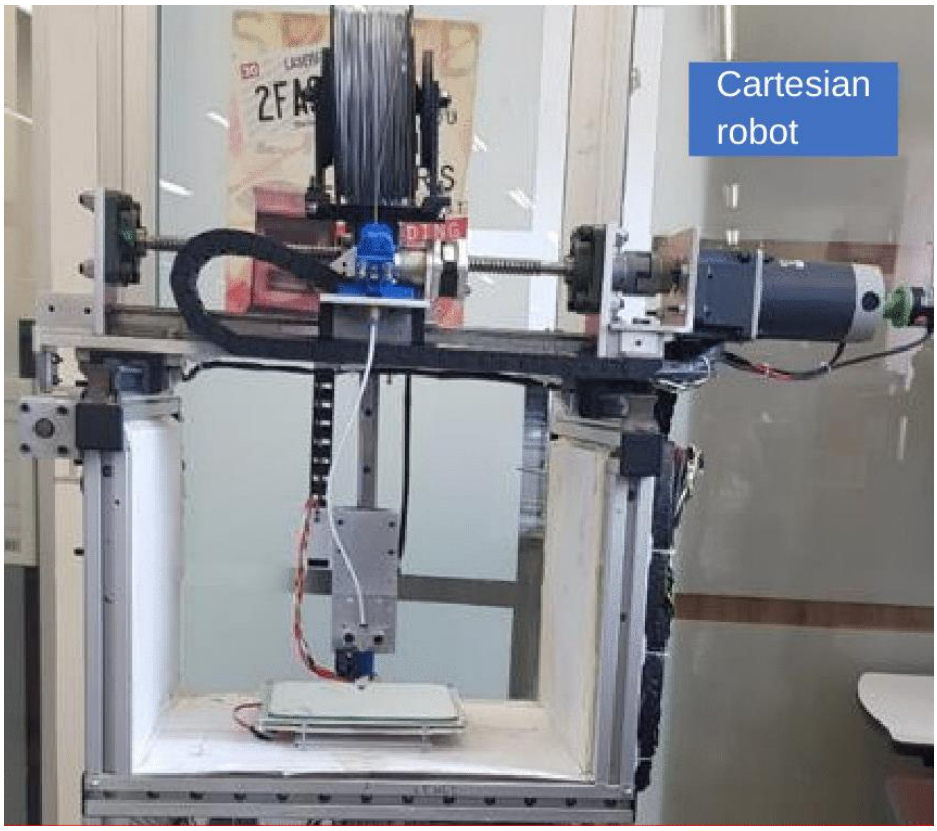



Figura 7-17 Robot cartesiano.

En la figura 7-17 se muestra una foto del robot y en la figura 7-18 se muestra el diagrama del robot cartesiano de tres grados de libertad. Este robot fue desarrollado previamente por los autores y está disponible para las pruebas experimentales necesarias. El modelo dinámico propuesto de un robot cartesiano viene dado por la ecuación 7-6, donde $i$ es reemplazado por X o Y dependiendo del eje a analizar. Usando este modelo hay seis parámetros para identificar.

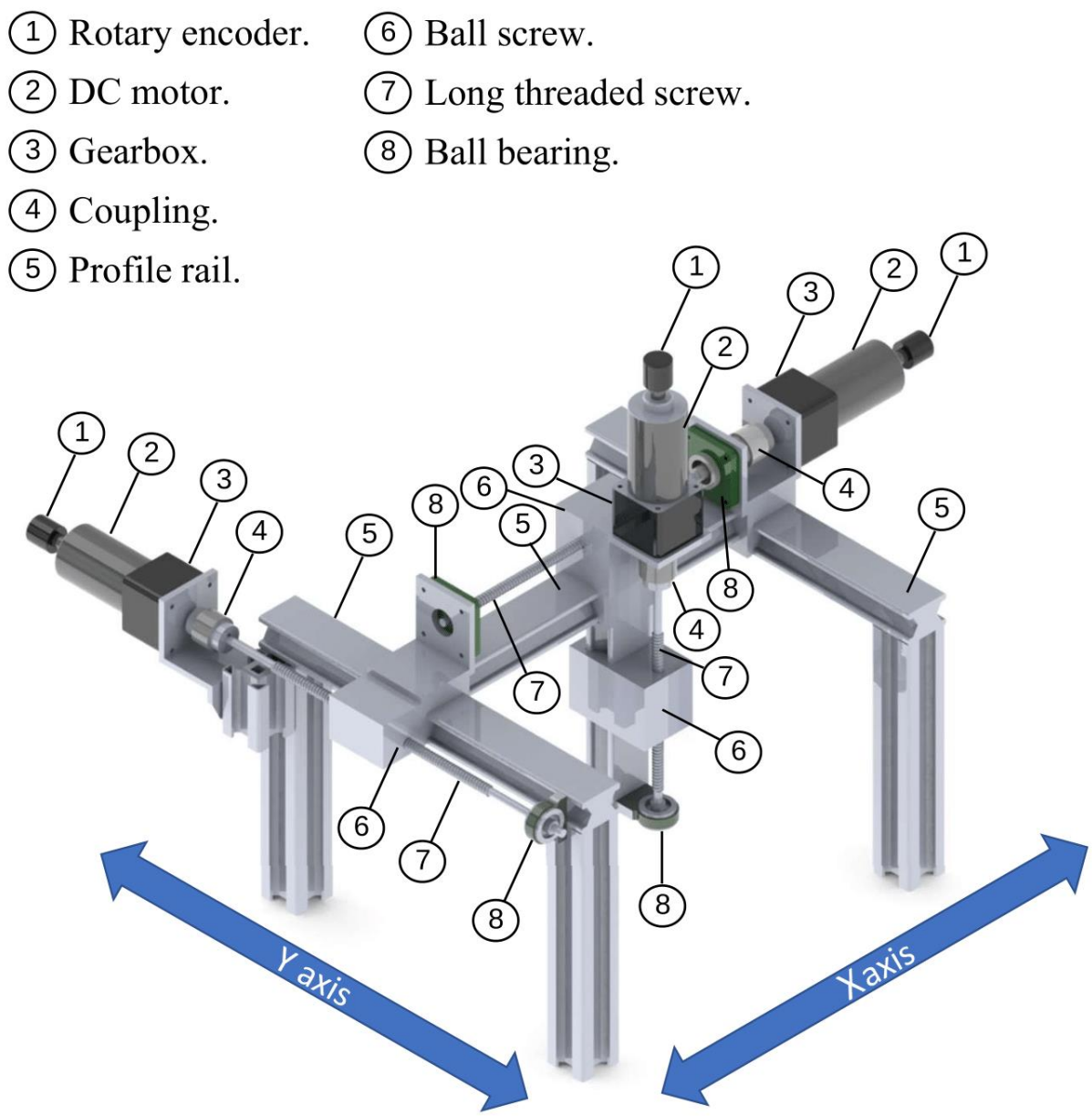



Figura 7-18 Diagrama esquemático del robot cartesiano.

$$\tau = I\ddot{q} + br_1\dot{q} + br_2 sign(\dot{q})\dot{q}^2 + br_3\dot{q}^3 + k\tanh(k_s\dot{q}) \quad (7\text{-}6)$$

donde:

$q_i$ Posición del $i$ grado de libertad

$\dot{q}_\iota = \frac{dq_i}{dt}$ Vector de velocidad

$\ddot{q}_\iota = \frac{d\dot{q}_\iota}{dt}$ Vector de aceleración

$I$ Parámetros de inercia

$br_1, br_2, br_3$ parámetros de viscosidad

$k$ Coeficientes externos de Coulomb

$k_s$ Factor de condicionamiento de la fricción de Coulomb

Para mejorar la similitud con el par real, se propone un segundo modelo dinámico del robot con datos experimentales, se intenta reproducir la fricción de transición en el robot cartesiano, se propone el modelo dinámico con diez parámetros por la ecuación 7-7, para tratar de mejorar la similitud con el par real añadiendo coeficientes de fricción y una compensación de par.

$$\tau = \mathrm{I}\ddot{\mathrm{q}} + \mathrm{br}_1\dot{\mathrm{q}} + \mathrm{br}_\mathrm{s}(\dot{\mathrm{q}})\dot{\mathrm{q}}^2 + \mathrm{br}_3\dot{\mathrm{q}} + k\tanh(k_s\dot{q}) + s_1\tanh\big((s_2+\dot{q})s_3\big)\Big(tanh\big((s_2+\dot{q})s_3\big)^2 - 1\Big)(sign(\ddot{q}-1)) + \tau_o \qquad (7\text{-}7)$$

donde:

$q_i$ Posición del $i$ grado de libertad

$\dot{q}_\iota = \frac{dq_i}{dt}$ Vector de velocidad

$\ddot{q}_\iota = \frac{d\dot{q}_\iota}{dt}$ Vector de aceleración

$I$ Parámetros de inercia

$br_1, br_2, br_3$ parámetros de viscosidad

$k$ Coeficientes externos de Coulomb

$k_s$ Factor de condicionamiento de la fricción de Coulomb

$\tau_o$ Offset de torque

$s_1$ Coeficiente de transición de fricción 1

$s_2$ Coeficiente de transición de fricción 2

$s_3$ Coeficiente de transición de fricción 3

La metodología para generar las imágenes a partir de los datos del robot es la misma que en la sección 7.2.1

### 7.3.2 Estructura de las redes neuronales

Utilizando las imágenes generadas para el entrenamiento de la red neuronal y sabiendo cuántos parámetros se buscan identificar (seis o diez), se proponen 5 CNN con el mismo propósito, pero con una estructura diferente. Todas las redes neuronales tienen como entrada una imagen de $100 \times 100$ píxeles, y la salida tiene 6 o 10 parámetros de salida. Cada red neuronal propuesta se entrenará y ejecutará tanto para identificar el modelo de seis parámetros como el de diez.

Al igual en la red neuronal de la sección 7.2 la función de activación de las capas de convolución y totalmente conectadas es la Unidad Lineal Rectificada (ReLU) ecuación (7-4). La capa de salida utiliza la función sigmoide ecuación (7-5) donde $c$ es una constante para ajustar el nivel de saturación.

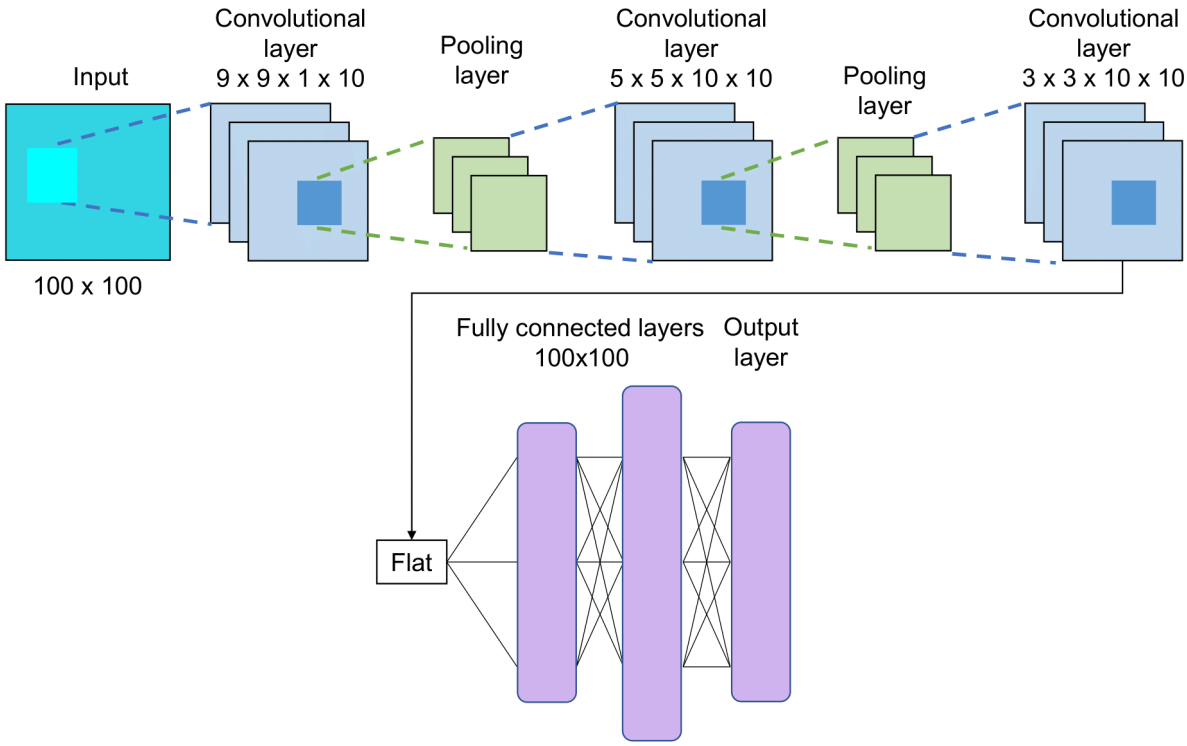



Figura 7-19 Estructura de red neuronal convolucional uno.

La primera red neuronal (figura 7-19), cuenta con 3 capas convolucionales, y 2 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $9 \times 9 \times 1 \times 10$, $5 \times 5 \times 10 \times 10$, y $3 \times 3 \times 10 \times 10$ respectivamente. En la primera capa, hay una capa máxima de submuestra de tamaño. Las capas totalmente conectadas son 50 y 100. Cada capa agrega un vector de sesgo.

La segunda red neuronal (figura 7-20), cuenta con 3 capas convolucionales, y 2 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $9 \times 9 \times 1 \times 10$, $5 \times 5 \times 10 \times 10$ y $3 \times 3 \times 10 \times 10$ respectivamente. En la primera capa, hay una capa máxima de submuestreo de tamaño $2 \times 2$. Las capas totalmente conectadas son tienen 100 y 100 neuronas respectivamente.

La tercera red neuronal (figura 7-21), cuenta con 5 capas convolucionales, y 3 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $9 \times 9 \times 1 \times 10$, $5 \times 5 \times 10 \times 10$, $5 \times 5 \times 10 \times 10$, $3 \times 3 \times 10 \times 10$ y $2 \times 2 \times 10 \times 10$ respectivamente. En la primera y segunda capa, hay una capa máxima de submuestreo de tamaño $2 \times 2$. Las capas totalmente conectadas son tienen 100, 100 y 50 neuronas respectivamente.

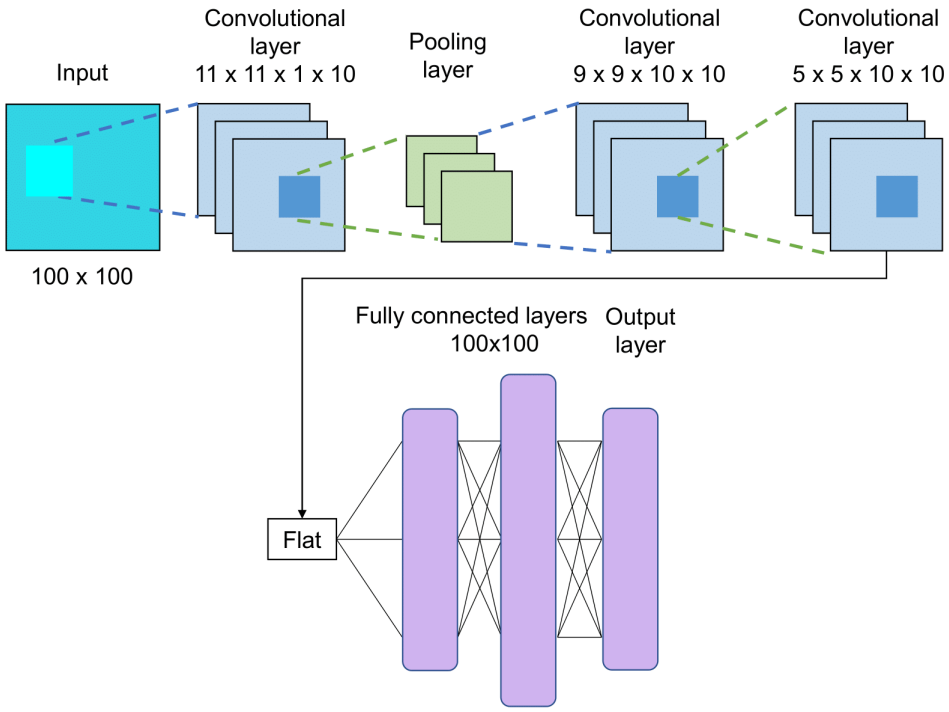



Figura 7-20 Estructura de red neuronal convolucional dos.

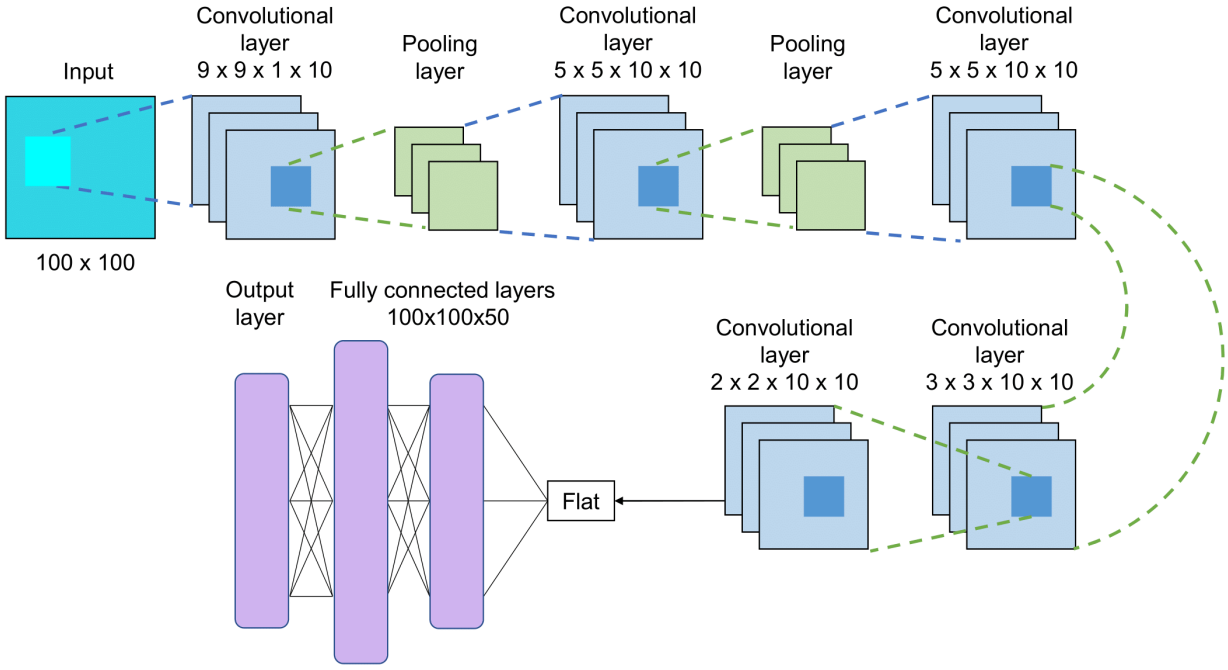



Figura 7-21 Estructura de red neuronal convolucional tres.

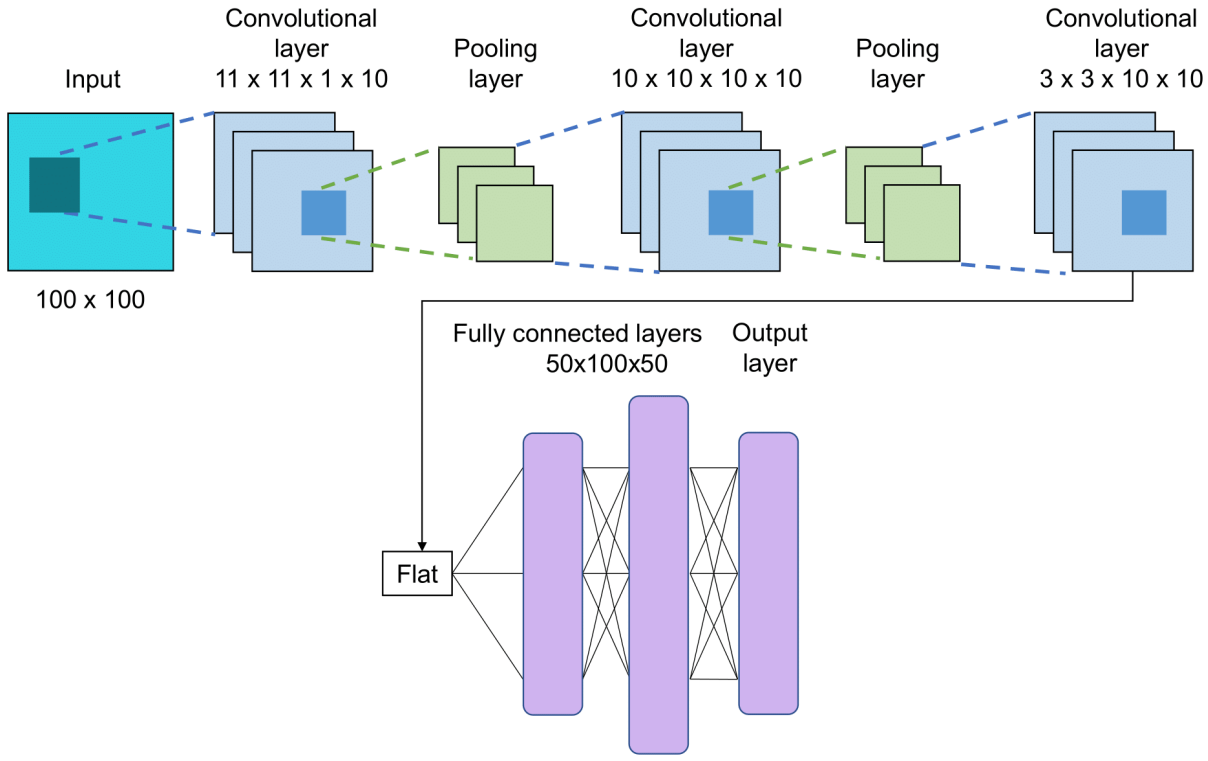



Figura 7-22 Estructura de red neuronal convolucional cuatro.

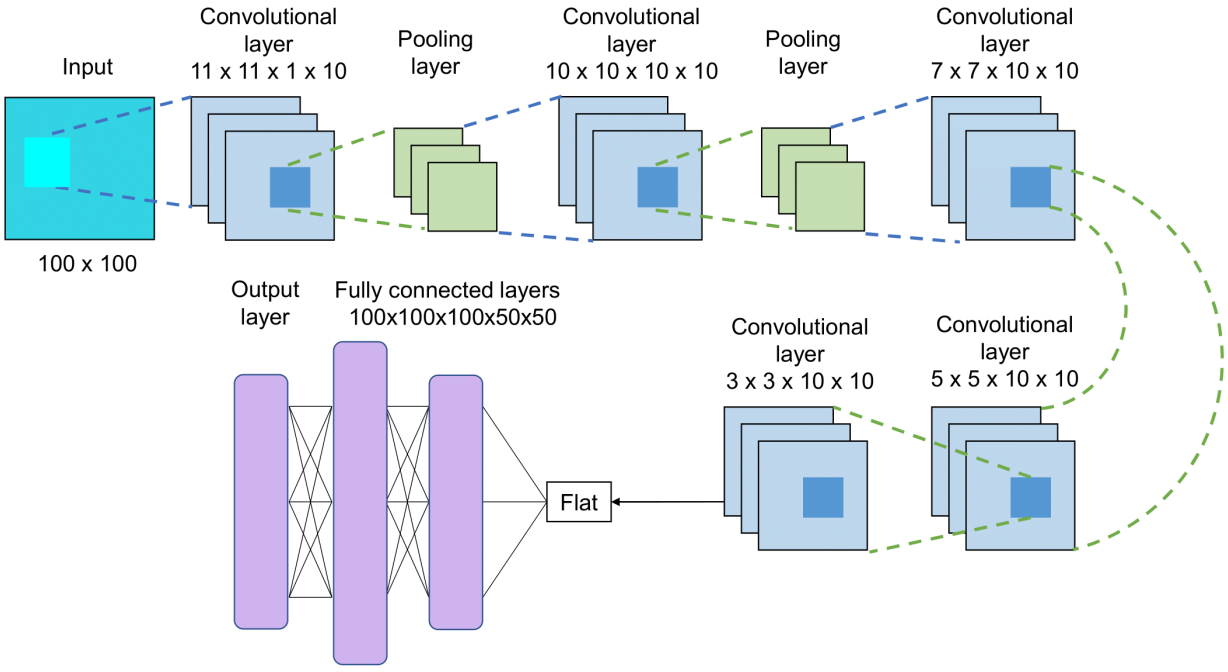



Figura 7-23 Estructura de red neuronal convolucional cinco.

La cuarta red neuronal (figura 7-22), cuenta con 3 capas convolucionales, y 3 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $11 \times 11 \times 1 \times 10$, $10 \times 10 \times 10 \times 10$, y

$3 \times 3 \times 10 \times 10$ respectivamente. En la primera y segunda capa, hay una capa máxima de submuestreo de tamaño $2 \times 2$. Las capas totalmente conectadas son tienen 50, 100 y 50 neuronas respectivamente.

La quinta red neuronal (figura7-23), cuenta con 5 capas convolucionales, y 5 completamente conectadas. Los tamaños de los núcleos de las capas convolucionales son $11 \times 11 \times 1 \times 10$, $10 \times 10 \times 10 \times 10$, $7 \times 7 \times 10 \times 10$, $5 \times 5 \times 10 \times 10$ y $3 \times 3 \times 10 \times 10$ respectivamente. En la primera y segunda capa, hay una capa máxima de submuestreo de tamaño $2 \times 2$. Las capas totalmente conectadas son tienen 100, 100, 100, 50 y 50 neuronas respectivamente.

### 7.3.3 Entrenamiento

El entrenamiento de las diez redes neuronales se llevó a cabo, utilizando backpropagation (regla delta) con el mismo factor de aprendizaje de $1.4 \times 10^{-3}$ (Berzal, 2018). Las diez CNN han sido entrenadas 800.000 iteraciones; después de cada iteración, se hizo una validación.

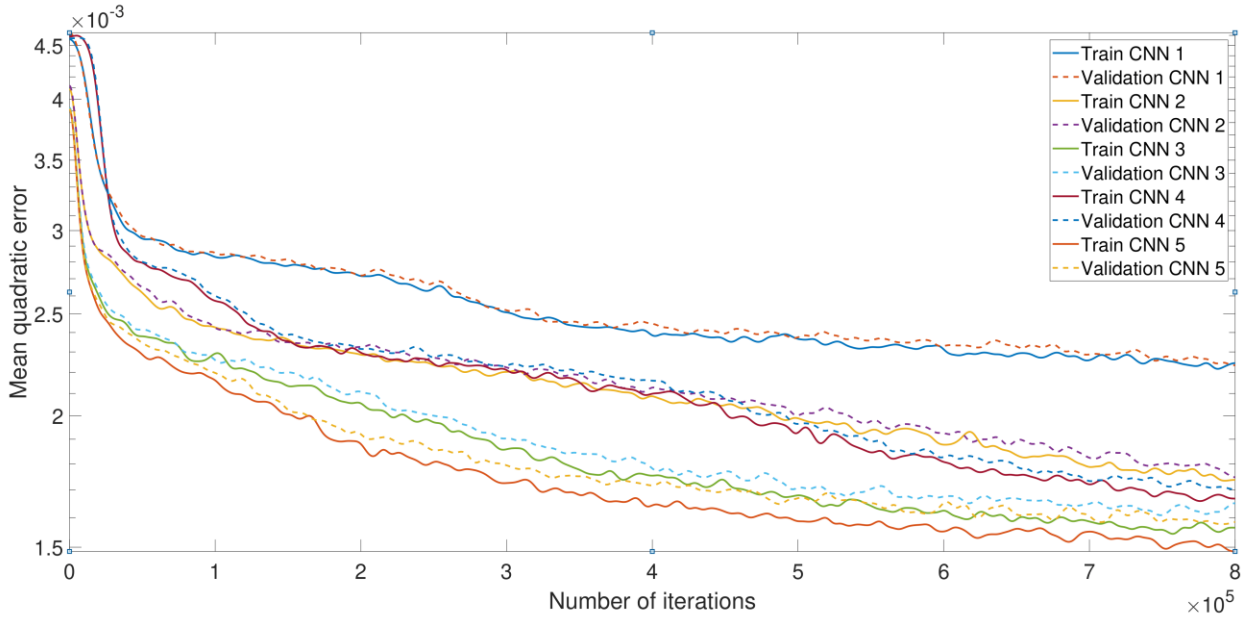



Figura 7-24 Gráfico de entrenamiento de red neuronal para modelo de seis parámetros.

Desde las 50.000 iteraciones, una diferencia en la evolución en la evolución del entrenamiento se puede notar en ambas graficas. Para el modelo de seis parámetros (figura 7-24), CNN 5 tiene el mejor rendimiento con poca diferencia de la otra CNN, a excepción de CNN 1, que fue el peor. CNN 3 obtuvo el más mínimo error en la formación sobre el modelo de diez parámetros (figura 7-25), mientras que el peor rendimiento en el modelo de diez parámetros fue CNN 1.

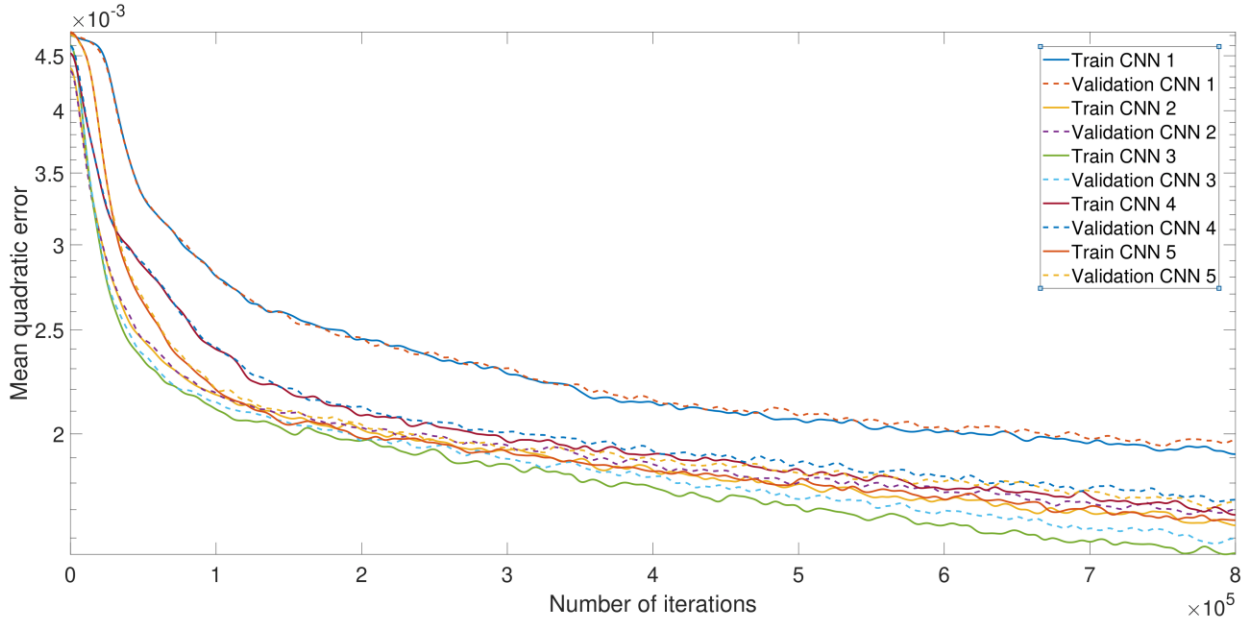



Figura 7-25 Gráfico de entrenamiento de red neuronal para modelo de diez parámetros.

La función coseno se utilizó para simular la aceleración utilizada para generar las bases de datos de entrenamiento para las redes neuronales, en la figura 7-26; se compara con la aceleración extraída de la trayectoria experimental del robot 7-26. Incluso con la notoria diferencia en la forma de la aceleración, no fue necesario entrenar la red neuronal con datos reales para obtener resultados de similitud por encima del 94%.

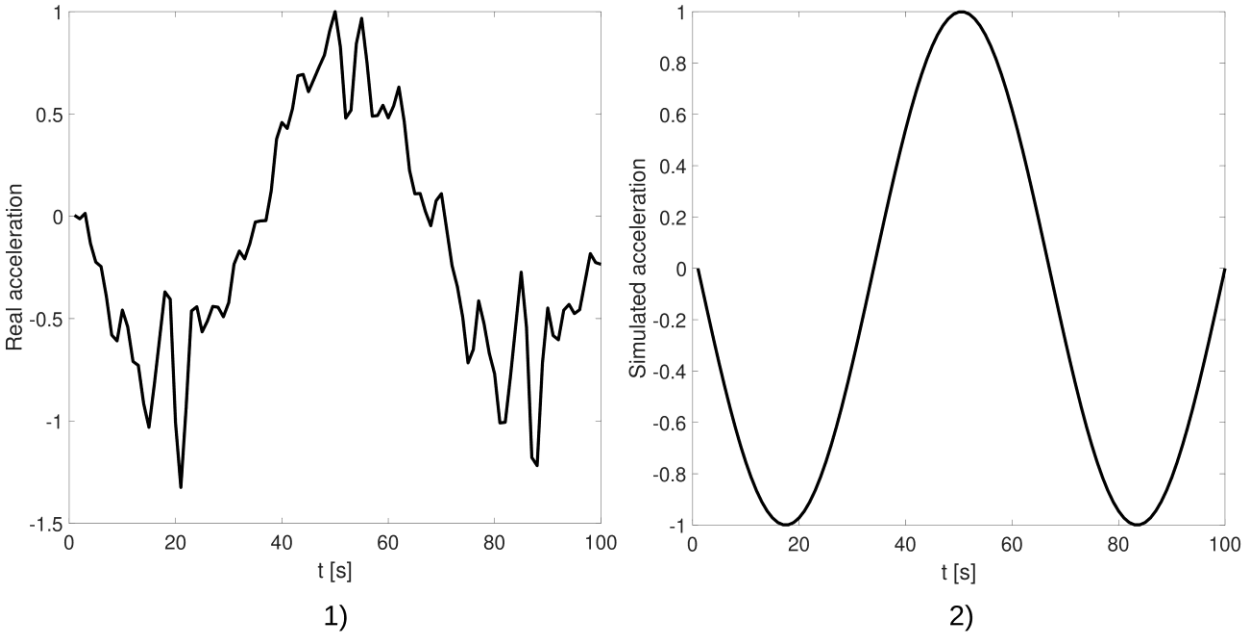



Figura 7-26 Forma de aceleración: 1) experimental, 2) numérica.

### 7.3.4 Desempeño

Se realiza la identificación paramétrica del modelo de seis y diez parámetros con datos simulados y con datos experimentales. Se utiliza el mismo algoritmo de similitud que en la red neuronal se la sección 7.2.

El algoritmo se prueba con un par creado utilizando el modelo dinámico, sustituyendo un conjunto de parámetros $\delta_1$ posición, velocidad y aceleración en la ecuación 7-6 para el modelo de seis parámetros y otro conjunto de parámetros $\delta_2$ con la ecuación 7-7 para el modelo de diez parámetros. Los parámetros seleccionados no forman parte de los datos de entrenamiento. Las tablas 7-7 y 7-8 muestran los parámetros originales $\delta_1$ y $\delta_2$ y los identificados por las cinco redes neuronales propuestas.

Tabla 7-7 Valores numéricos de los seis parámetros.

| Parámetro | $\delta_1$ | CNN 1 | CNN 2 | CNN 3 | CNN 4 | CNN 5 |
|---|---|---|---|---|---|---|
| $I$ | 0.3469 | 0.3431 | 0.3460 | 0.3460 | 0.3446 | 0.3446 |
| $br_1$ | 0.6809 | 0.6498 | 0.6892 | 0.6892 | 0.6455 | 0.6455 |
| $br_2$ | 0.1267 | 0.1474 | 0.1284 | 0.1284 | 0.1419 | 0.1419 |
| $br_3$ | 0.0044 | 0.0070 | 0.0076 | 0.0076 | 0.0188 | 0.0188 |
| $k$ | 0.7185 | 0.7288 | 0.7072 | 0.7072 | 0.7276 | 0.7276 |
| $k_s$ | 22.5515 | 22.1417 | 23.0152 | 23.0152 | 22.2705 | 22.2705 |

Tabla 7-8 Valores numéricos de los diez parámetros

| Parámetro | $\delta_2$ | CNN 1 | CNN 2 | CNN 3 | CNN 4 | CNN 5 |
|---|---|---|---|---|---|---|
| $I$ | 0.7808 | 0.7639 | 0.7532 | 0.7718 | 0.7718 | 0.7718 |
| $br_1$ | 0.6199 | 0.3450 | 0.5265 | 0.6129 | 0.6129 | 0.6129 |
| $br_2$ | 0.0674 | 0.1330 | 0.1240 | 0.1283 | 0.1283 | 0.1283 |
| $br_3$ | 0.0310 | 0.0215 | 0.0202 | 0.0321 | 0.0321 | 0.0321 |
| $k$ | 0.8071 | 0.9515 | 0.8704 | 0.7705 | 0.7705 | 0.7705 |
| $k_s$ | 8.6148 | 7.4819 | 8.3263 | 8.5172 | 8.5172 | 8.5172 |
| $\tau_o$ | 0.0779 | 0.0963 | 0.0751 | 0.0929 | 0.0929 | 0.0929 |
| $s_1$ | 0.6579 | 0.6548 | 0.7017 | 0.7008 | 0.7008 | 0.7008 |
| $s_2$ | -0.0928 | -0.0930 | -0.0854 | -0.0947 | -0.0947 | -0.0947 |
| $s_3$ | 4.5335 | 4.8719 | 4.7902 | 4.5937 | 4.5937 | 4.5937 |

La tabla 7-9 muestra la evaluación de similitud con el algoritmo 1. Las señales de par determinadas numéricamente y reconstruidas son similares, como se muestra en las figuras 7-27 y 7-28.

Tabla 7-9 Evaluación de la similitud entre pares numéricos y reconstruidos del modelo de seis parámetros del eje X (6p) y diez parámetros (10p) con redes neuronales.

| | Seis parámetros | Diez parámetros |
|---|---|---|
| CNN-1 | 99.66% | 98.84% |
| CNN-2 | 99.88% | 99.12% |
| CNN-3 | 99.88% | **99.15%** |
| CNN-4 | **99.92%** | **99.15%** |
| CNN-5 | **99.92%** | **99.15%** |

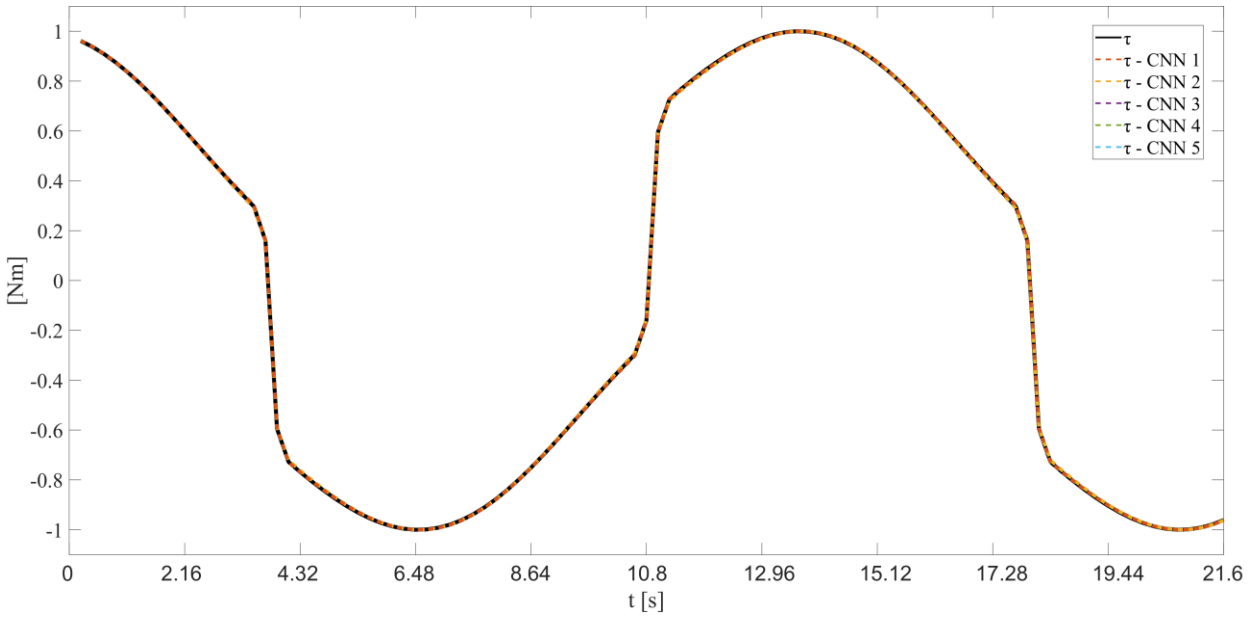



Figura 7-27 Reconstrucción de $\tau$ simulado seis parámetros.

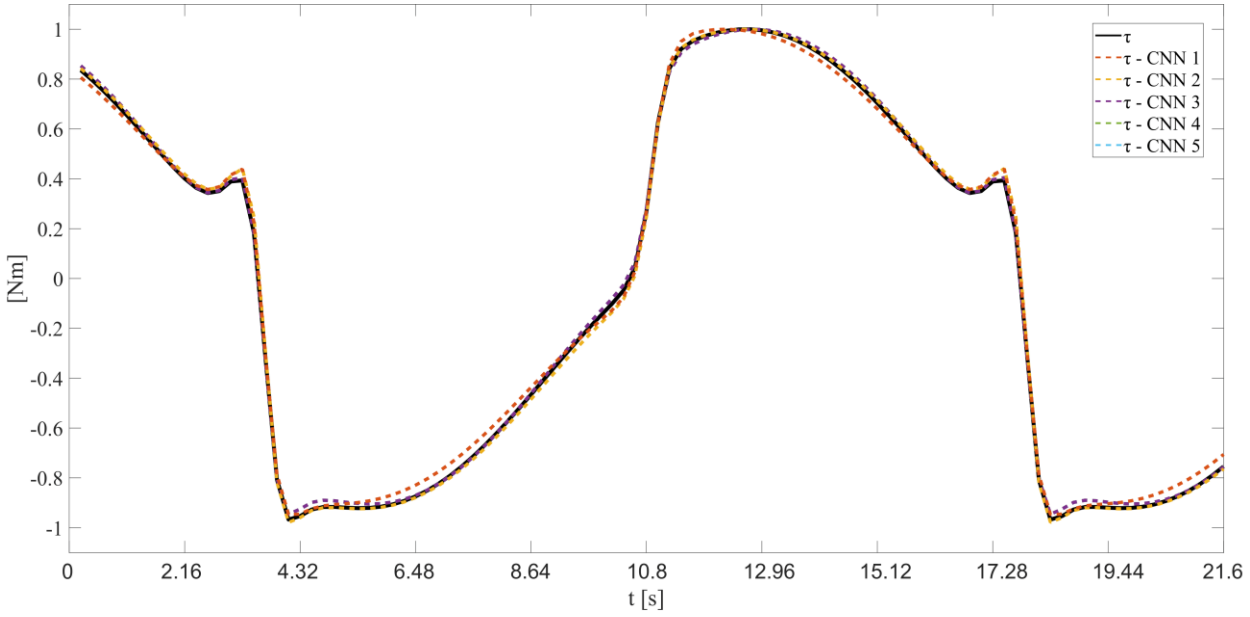



Figura 7-28 Reconstrucción de τ simulado diez parámetros

La evaluación de similitud con la métrica propuesta muestra que el par numérico construido con el conjunto de parámetros $\delta_1, \delta_2$ y el par reconstruido con los parámetros extraídos con el algoritmo están demasiado cerca, menos del 2%. La similitud alcanzada por las redes neuronales en el modelo de seis parámetros es más cercana al 100% que en el modelo de diez parámetros debido a la diferencia de complejidad entre la ecuación 7-6 y 7-7.

Se realizaron cinco trayectorias con el eje X (XP1, XP2, XP3, XP4 y XP5) y cinco con el eje Y (YP1, YP2, YP3, YP4 y YP5) obtenido experimentalmente con el robot cartesiano. Los datos de las trayectorias se convirtieron en una imagen con el mismo método usado anteriormente y se ingresaron en las diez redes

neuronales para obtener la identificación paramétrica y completar la reconstrucción del par $\tau$. Se realizo la identificación paramétrica utilizando mínimos cuadrados (LS) para tener una comparación con otro método diferente a las redes neuronales.

Las tablas 7-10 y 7-11 muestra el mejor valor de los parámetros identificados por las 5 CNN del modelo de seis parámetros, y la tabla 7-12 y 7-13 del modelo de diez parámetros; estos parámetros se utilizan para recrear el par experimental de cada eje X e Y. Se tomaron los parámetros identificados por CNN y por LS, que obtuvieron el mayor porcentaje de similitud con cada trayectoria.

Tabla 7-10 Valores de los seis parámetros identificados para el modelo dinámico del eje X.

| Parámetro | XP1 | XP2 | XP3 | XP4 | XP5 |
|---|---|---|---|---|---|
| $I$ | 0.0044 | 0.0020 | 0.0022 | 0.0023 | 0.0037 |
| $br_1$ | 0.0121 | 0.0024 | 0.0015 | 0.0037 | 0.0063 |
| $br_2$ | 5.51e-06 | 1.59e-05 | 1.05e-05 | 1.02e-05 | 1.01e-05 |
| $br_3$ | 2.98e-09 | 4.99e-09 | 3.83e-09 | 4.23e-09 | 6.01e-09 |
| $k$ | 1.3998 | 1.1915 | 0.8859 | 1.0752 | 1.6321 |
| $k_s$ | 0.0760 | 0.0593 | 0.0336 | 0.0327 | 0.0314 |

Tabla 7-11 Valores de los seis parámetros identificados para el modelo dinámico del eje Y.

| Parámetro | YP1 | YP2 | YP3 | YP4 | YP5 |
|---|---|---|---|---|---|
| $I$ | 0.0046 | 0.0038 | 0.0021 | 0.0027 | 0.0040 |
| $br_1$ | 0.0070 | 0.0114 | 0.0064 | 0.0026 | 0.0040 |
| $br_2$ | 1.04e-05 | 4.22e-06 | 9.55e-06 | 1.47e-05 | 1.35e-05 |
| $br_3$ | 7.83e-09 | 3.78e-09 | 3.51e-09 | 4.93e-09 | 7.15e-09 |
| $k$ | 1.1945 | 1.3853 | 1.1827 | 1.0721 | 1.2037 |
| $k_s$ | 0.1181 | 0.0622 | 0.0566 | 0.0328 | 0.0450 |

Tabla 7-12 Valores de los diez parámetros identificados para el modelo dinámico del eje X.

| Parámetro | XP1 | XP2 | XP3 | XP4 | XP5 |
|---|---|---|---|---|---|
| $I$ | 0.0035 | 0.0035 | 0.0059 | 0.0022 | 0.0033 |
| $br_1$ | 0.0103 | 0.0115 | 0.0072 | 0.0062 | 0.0092 |
| $br_2$ | 7.16e-06 | 3.09e-06 | 1.61e-05 | 1.12e-05 | 6.23e-06 |
| $br_3$ | 3.52e-09 | 3.05e-09 | 4.72e-09 | 3.44e-09 | 6.03e-09 |
| $k$ | 1.6797 | 1.0496 | 1.7592 | 1.0182 | 0.8528 |
| $k_s$ | 0.0406 | 0.0375 | 0.0333 | 0.0255 | 0.0325 |
| $\tau_o$ | 0.0187 | 0.0241 | 0.1027 | 0.0593 | 0.0668 |
| $s_1$ | 0.3268 | 1.2178 | 0.7987 | 0.5160 | 0.8014 |
| $s_2$ | 13.0358 | -5.6126 | -22.0237 | 0.2120 | 7.6770 |
| $s_3$ | 0.0285 | 0.0303 | 0.0205 | 0.0169 | 0.0160 |

Tabla 7-13 Valores de los diez parámetros identificados para el modelo dinámico del eje Y.

| Parámetro | YP1 | YP2 | YP3 | YP4 | YP5 |
|---|---|---|---|---|---|
| $I$ | 0.0030 | 0.0026 | 0.0018 | 0.0037 | 0.0040 |
| $br_1$ | 0.0049 | 0.0089 | 0.0096 | 0.0057 | 0.0087 |

| | | | | | |
|---|---|---|---|---|---|
| $br_2$ | 1.09e-05 | 1.23e-05 | 5.89e-06 | 1.16e-05 | 8.63e-06 |
| $br_3$ | 5.63e-09 | 1.09e-08 | 7.57e-09 | 3.35e-09 | 8.16e-09 |
| $k$ | 0.8084 | 1.6038 | 1.2194 | 1.3956 | 1.9182 |
| $k_s$ | 0.1492 | 0.0510 | 0.0665 | 0.0374 | 0.0437 |
| $\tau_o$ | -0.0048 | -0.0463 | -0.0026 | 0.0053 | -0.0339 |
| $s_1$ | 0.1665 | 0.1679 | 0.1867 | 0.0266 | 0.1258 |
| $s_2$ | -31.2242 | -10.4395 | 33.8287 | -2.5997 | 2.8666 |
| $s_3$ | 0.0324 | 0.0441 | 0.0220 | 0.0292 | 0.0259 |

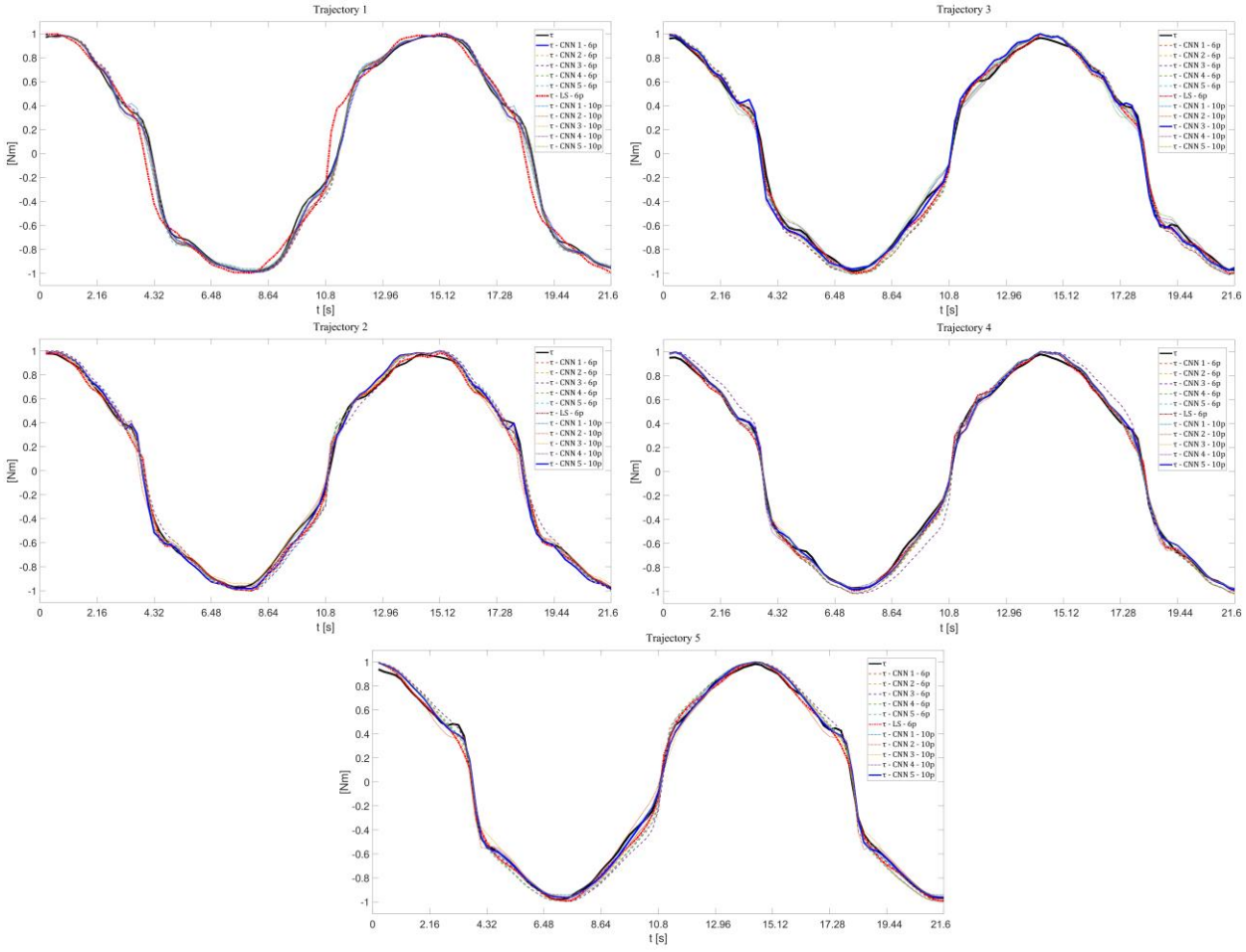



Figura 7-29 Reconstrucción de $\tau$ del eje X.

Las trayectorias experimentales realizadas para identificar los parámetros del robot cartesiano se muestran en la figura 7-29 para el eje X y la figura 7-30 para el eje Y. La posición experimental, la velocidad y la aceleración se utilizan para identificar los parámetros dinámicos de un verdadero robot cartesiano de tres grados de libertad. Los parámetros dinámicos también se identifican utilizando mínimos cuadrados (LS) para comparar con una metodología muy conocida.

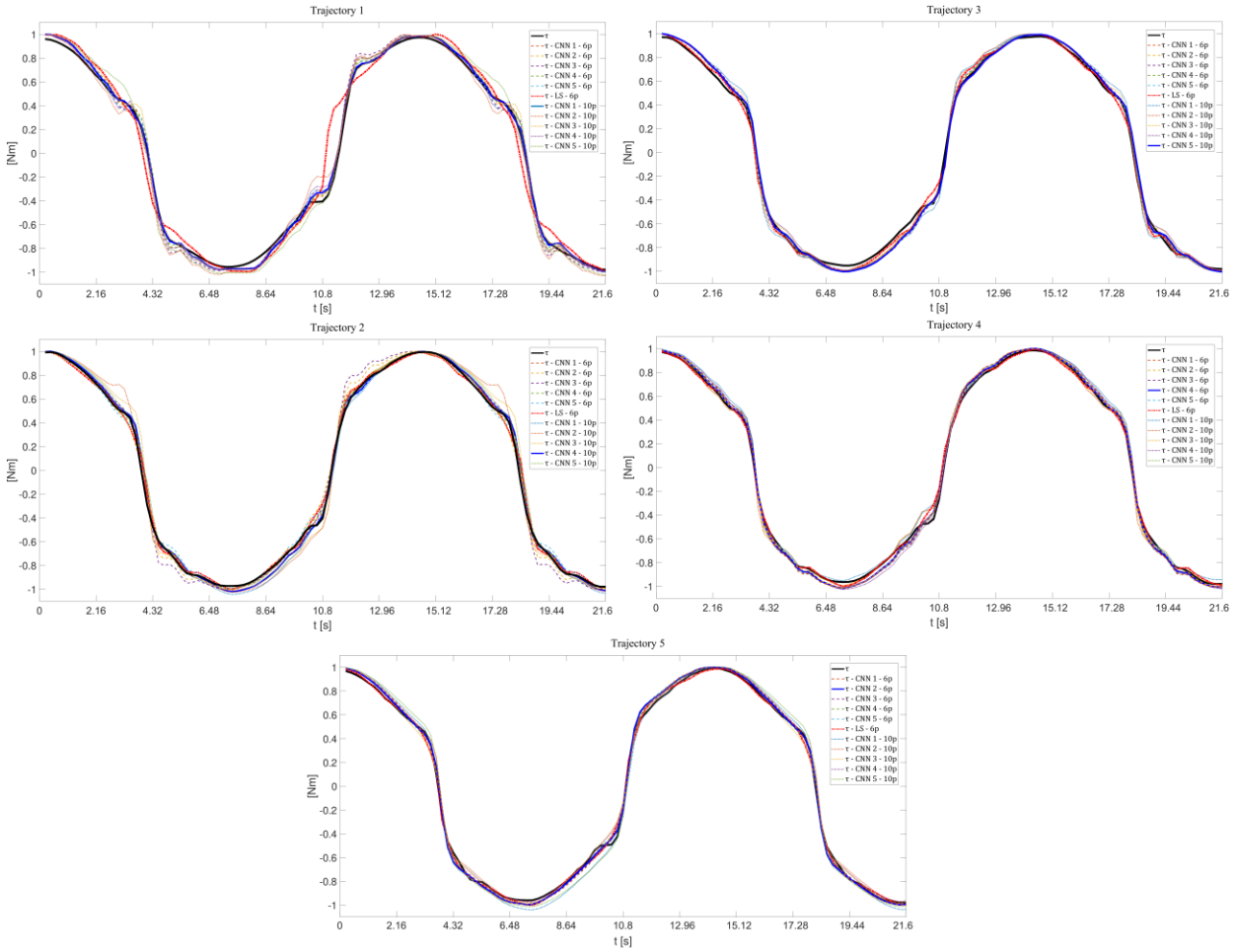



Figura 7-30 Reconstrucción de τ del eje Y.

La figura 7-29 y 7-30 muestran el par real τ, el par reconstruido τ-CNN con la metodología propuesta de seis y diez parámetros y τ-LS de los ejes X e Y. En cada figura el par original se muestra en negro, y en línea continua azul el par reconstruido con mayor porcentaje de similitud.

La tabla 7-14 muestra los valores de similitud del par experimental del eje X y su reconstrucción mediante redes neuronales para los dos modelos dinámicos y LS, y la tabla 7-15 para el eje Y. Para pruebas con trayectorias experimentales (XP1, XP2, XP3, XP4, XP5, YP1, YP2, YP3, YP4, YP5), los parámetros extraídos con el algoritmo propuesto superan los resultados de LS. En la similitud de los pares del eje X con los reconstruidos solo en una trayectoria, el basado en el modelo de seis parámetros obtuvo mejores resultados. En comparación, en el eje Y, dos trayectorias obtuvieron mejores resultados con este modelo. El modelo de diez parámetros tiene los valores más altos para el resto de las trayectorias.

Tabla 7-14 Evaluación experimental de la similitud entre par real y reconstruido del eje X, modelo de seis parámetros (6p) y diez parámetros (10p) con redes neuronales y mínimos cuadrados (LS).

| | XP1 | XP2 | XP3 | XP4 | XP5 |
|---|---|---|---|---|---|
| CNN-1 6p | **96.79%** | 95.64% | 95.63% | 95.43% | 94.40% |
| CNN-2 6p | 95.86% | 95.51% | 95.68% | 95.26% | 95.32% |
| CNN-3 6p | 96.17% | 94.61% | 95.61% | 94.77% | 95.38% |

| | | | | | |
|---|---|---|---|---|---|
| CNN-4 6p | 96.45% | 95.84% | 95.80% | 95.65% | 95.52% |
| CNN-5 6p | 96.51% | 95.33% | 95.84% | 95.56% | 95.37% |
| LS | 95.80% | 94.17% | 95.26% | 94.56% | 94.64% |
| CNN-1 10p | 95.77% | 95.80% | 95.37% | 95.82% | 96.58% |
| CNN-2 10p | 96.49% | 94.45% | 95.68% | 95.38% | 94.89% |
| CNN-3 10p | 95.89% | 95.96% | **96.51%** | 95.50% | 95.46% |
| CNN-4 10p | 96.64% | 95.59% | 95.35% | 95.21% | 96.98% |
| CNN-5 10p | 96.58% | **96.46%** | 94.69% | **96.24%** | **97.20%** |

Tabla 7-15 Evaluación experimental de la similitud entre par real y reconstruido del eje Y, modelo de seis parámetros (6p) y diez parámetros (10p) con redes neuronales y mínimos cuadrados (LS).

| | YP1 | YP2 | YP3 | YP4 | YP5 |
|---|---|---|---|---|---|
| CNN-1 6p | 96.62% | 96.24% | 97.42% | 97.69% | 97.40\% |
| CNN-2 6p | 96.58% | 95.34% | 97.04% | 97.86% | **97.68%** |
| CNN-3 6p | 96.19% | 95.00% | 97.26% | 97.77% | 97.42\% |
| CNN-4 6p | 96.66% | 96.25% | 97.38% | **97.87%** | 97.62\% |
| CNN-5 6p | 96.09% | 95.81% | 97.37% | 97.72% | 97.54\% |
| LS | 93.29% | 95.01% | 95.24% | 95.81% | 96.05\% |
| CNN-1 10p | **97.17%** | 97.09% | 96.42% | 95.48% | 96.85\% |
| CNN-2 10p | 94.46% | 96.21% | 96.32% | 96.33% | 96.59\% |
| CNN-3 10p | 95.52% | 95.10% | 96.48% | 95.97% | 96.64\% |
| CNN-4 10p | 96.48% | **97.59%** | 96.94% | 96.77% | 96.78\% |
| CNN-5 10p | 96.66% | 95.75% | **97.47%** | 97.46% | 97.22\% |

## 7.4 Congreso 1: 4° Coloquio Nacional Doctoral LKE 2021


**LINK del evento:**

**https://meet.google.com/pte-sfzu-jhc**

El Coloquio Doctoral LKE es un conjunto de sesiones simultáneas organizadas de acuerdo a la temática de los trabajos aprobados, durante las cuales los doctorandos presentan sus propuestas de disertación y los resultados parciales hasta el día de la presentación **sobre sus proyectos de investigación**.

Los miembros de los paneles les proporcionan pautas y sugerencias que los ayudarán a identificar las fortalezas y debilidades de dichas propuestas, para mejorar sus contenidos y prepararse para la sustentación final ante sus comités de tesis.

Los proyectos doctorales que se reciban para el Coloquio Doctoral estarán sujetos a un proceso de doble revisión por un jurado, el que seleccionará los trabajos. Los trabajos aceptados serán publicados.

Los trabajos serán programados para su presentación oral y serán publicados como capítulo de un libro, sin embargo, sólo los trabajos con mas alta evaluaciones serán invitados a participar para ser publicados como capítulo de libro con editorial internacional United Academic Journals (UA Journals, http://www.uajournals.com/es/).

Este año las presentaciones serán en línea debido a la pandemia de COVID19.

Figura 7-31 Convocatoria Coloquio Nacional.

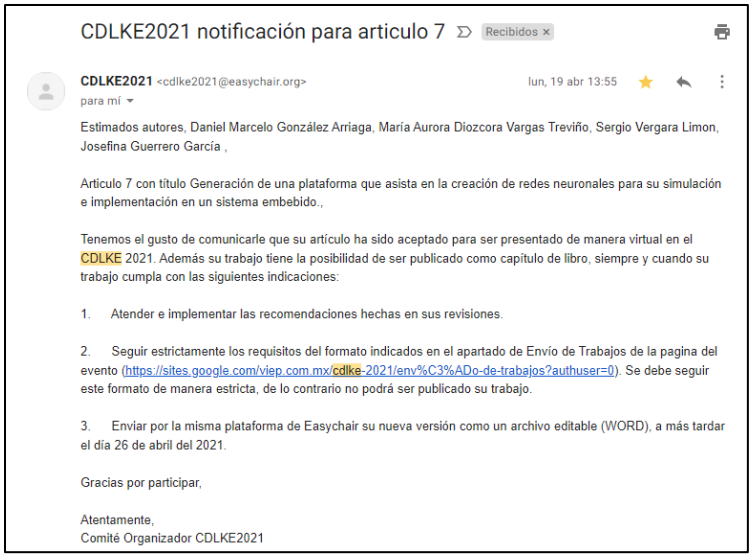
El cuarto Coloquio Nacional Doctoral LKE 2021 fue celebrado el día 3 y 4 de junio del 2021 donde fueron presentados los trabajos de investigación del doctorado LKE.

El trabajo enviado se tituló Generación de una plataforma que asista en la creación de redes neuronales para su simulación e implementación en un sistema embebido.

En la figura 7-32 se muestra el correo de aceptación enviado por el comité organizador.

En la figura 7-33 se muestra la evidencia de participación en el programa del congreso.

En la figura 7-34 se muestra la constancia de participación en el congreso.



CDLKE2021 notificación para articulo 7 Recibidos ×

**CDLKE2021** <cdlke2021@easychair.org> lun, 19 abr 13:55

para mí

Estimados autores, Daniel Marcelo González Arriaga, María Aurora Diozcora Vargas Treviño, Sergio Vergara Limon, Josefina Guerrero García ,

Articulo 7 con título Generación de una plataforma que asista en la creación de redes neuronales para su simulación e implementación en un sistema embebido.,

Tenemos el gusto de comunicarle que su artículo ha sido aceptado para ser presentado de manera virtual en el CDLKE 2021. Además su trabajo tiene la posibilidad de ser publicado como capítulo de libro, siempre y cuando su trabajo cumpla con las siguientes indicaciones:

1. Atender e implementar las recomendaciones hechas en sus revisiones.

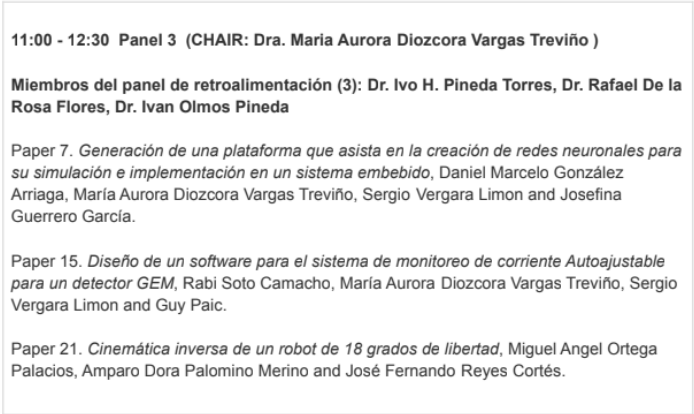
2. Seguir estrictamente los requisitos del formato indicados en el apartado de Envío de Trabajos de la pagina del evento (https://sites.google.com/viep.com.mx/cdlke-2021/env%C3%ADo-de-trabajos?authuser=0). Se debe seguir este formato de manera estricta, de lo contrario no podrá ser publicado su trabajo.

3. Enviar por la misma plataforma de Easychair su nueva versión como un archivo editable (WORD), a más tardar el día 26 de abril del 2021.

Gracias por participar,

Atentamente,
Comité Organizador CDLKE2021

Figura 7-32 Correo de aceptación al congreso.



**11:00 - 12:30 Panel 3 (CHAIR: Dra. Maria Aurora Diozcora Vargas Treviño )**

**Miembros del panel de retroalimentación (3): Dr. Ivo H. Pineda Torres, Dr. Rafael De la Rosa Flores, Dr. Ivan Olmos Pineda**

Paper 7. *Generación de una plataforma que asista en la creación de redes neuronales para su simulación e implementación en un sistema embebido*, Daniel Marcelo González Arriaga, María Aurora Diozcora Vargas Treviño, Sergio Vergara Limon and Josefina Guerrero García.

Paper 15. *Diseño de un software para el sistema de monitoreo de corriente Autoajustable para un detector GEM*, Rabi Soto Camacho, María Aurora Diozcora Vargas Treviño, Sergio Vergara Limon and Guy Paic.

Paper 21. *Cinemática inversa de un robot de 18 grados de libertad*, Miguel Angel Ortega Palacios, Amparo Dora Palomino Merino and José Fernando Reyes Cortés.

Figura 7-33 Participación en el programa del congreso.

La Benemérita Universidad Autónoma de Puebla

A través de la Facultad de Ciencias de la Computación

otorga la presente

# CONSTANCIA

A: Daniel Marcelo González Arriaga, María Aurora Diozcora Vargas Treviño, Sergio Vergara Limon, Josefina Guerrero García

Por presentar el artículo titulado "Generación de una plataforma que asista en la creación de redes neuronales para su simulación e implementación en un sistema embebido", en el 4° Coloquio Nacional Doctoral de Ingeniería del Lenguaje y del Conocimiento.

**"Pensar bien, para vivir mejor"**

Puebla de Z., a 04 de junio de 2021.

**M.I. María del Consuelo Molina García**

Directora.

Figura 7-34 Constancia de participación.

## 7.5 Congreso 2: 8th International Symposium on Language & Knowledge Engineering

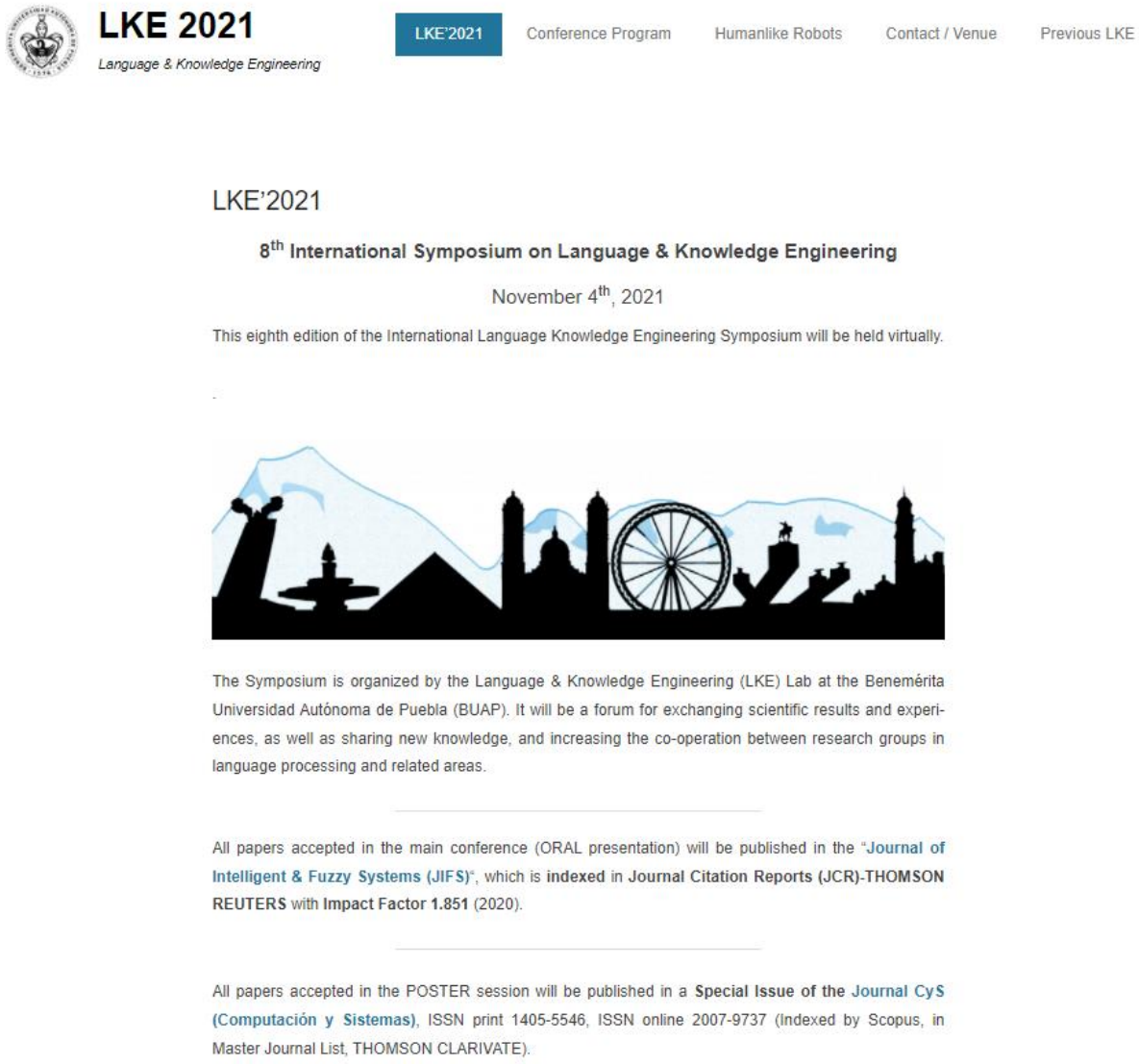


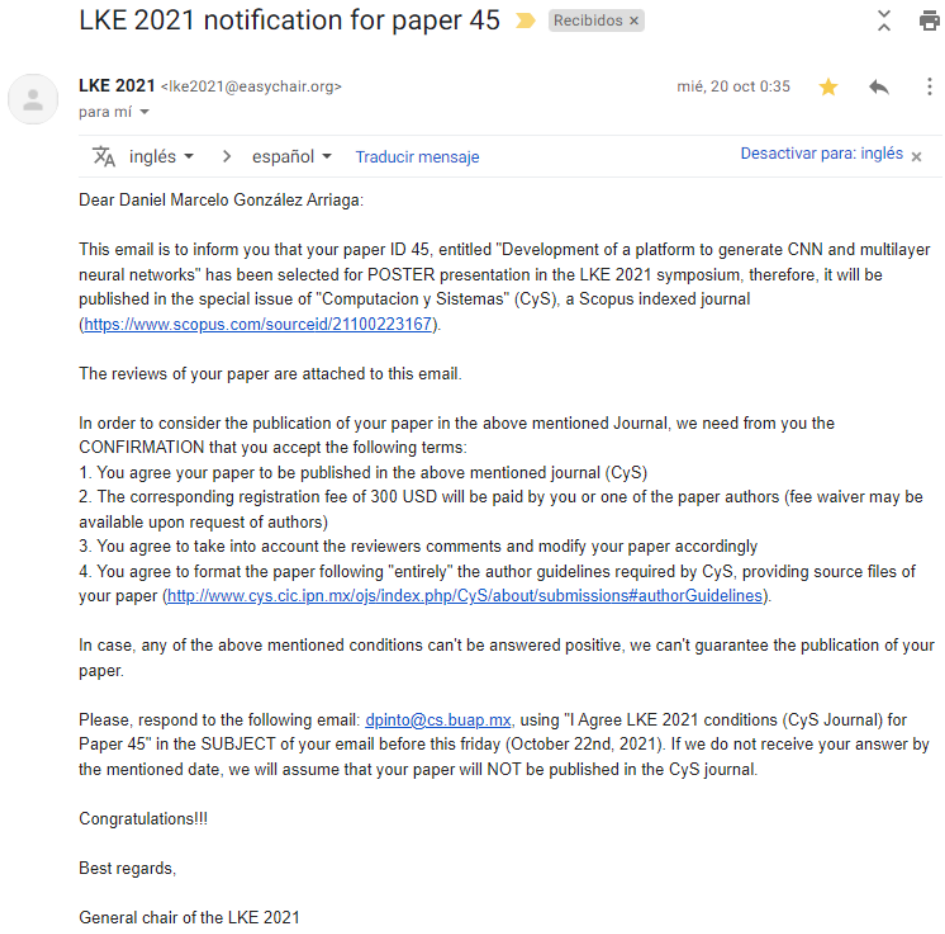

Figura 7-35 Convocatoria del congreso 8th International Symposium on Language & Knowledge Engineering.

El 8th International Symposium on Language & Knowledge Engineering fue celebrado el día 4 de noviembre del 2021. Se presentaron los resultados obtenidos hasta el momento de esta investigación. El trabajo enviado se tituló Development of a platform to generate CNN and multilayer neural networks.

En la figura 7-36 se muestra el correo de aceptación enviado por el comité organizador.

En la figura 7-37 se muestra la evidencia de participación en el programa del congreso.

En la figura 7-38 se muestra la constancia de participación en el congreso.



LKE 2021 notification for paper 45 Recibidos ×

**LKE 2021** <lke2021@easychair.org> mié, 20 oct 0:35

para mí

inglés › español Traducir mensaje Desactivar para: inglés ×

Dear Daniel Marcelo González Arriaga:

This email is to inform you that your paper ID 45, entitled "Development of a platform to generate CNN and multilayer neural networks" has been selected for POSTER presentation in the LKE 2021 symposium, therefore, it will be published in the special issue of "Computacion y Sistemas" (CyS), a Scopus indexed journal (https://www.scopus.com/sourceid/21100223167).

The reviews of your paper are attached to this email.

In order to consider the publication of your paper in the above mentioned Journal, we need from you the CONFIRMATION that you accept the following terms:
1. You agree your paper to be published in the above mentioned journal (CyS)
2. The corresponding registration fee of 300 USD will be paid by you or one of the paper authors (fee waiver may be available upon request of authors)
3. You agree to take into account the reviewers comments and modify your paper accordingly
4. You agree to format the paper following "entirely" the author guidelines required by CyS, providing source files of your paper (http://www.cys.cic.ipn.mx/ojs/index.php/CyS/about/submissions#authorGuidelines).

In case, any of the above mentioned conditions can't be answered positive, we can't guarantee the publication of your paper.

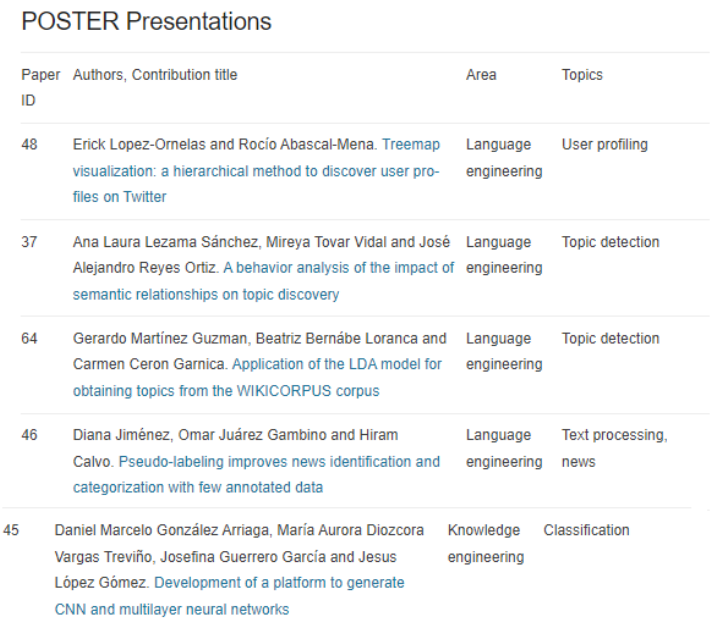
Please, respond to the following email: dpinto@cs.buap.mx, using "I Agree LKE 2021 conditions (CyS Journal) for Paper 45" in the SUBJECT of your email before this friday (October 22nd, 2021). If we do not receive your answer by the mentioned date, we will assume that your paper will NOT be published in the CyS journal.

Congratulations!!!

Best regards,

General chair of the LKE 2021

Figura 7-36 Correo de aceptación al congreso.



POSTER Presentations

| Paper ID | Authors, Contribution title | Area | Topics |
|---|---|---|---|
| 48 | Erick Lopez-Ornelas and Rocío Abascal-Mena. Treemap visualization: a hierarchical method to discover user profiles on Twitter | Language engineering | User profiling |
| 37 | Ana Laura Lezama Sánchez, Mireya Tovar Vidal and José Alejandro Reyes Ortiz. A behavior analysis of the impact of semantic relationships on topic discovery | Language engineering | Topic detection |
| 64 | Gerardo Martínez Guzman, Beatriz Bernábe Loranca and Carmen Ceron Garnica. Application of the LDA model for obtaining topics from the WIKICORPUS corpus | Language engineering | Topic detection |
| 46 | Diana Jiménez, Omar Juárez Gambino and Hiram Calvo. Pseudo-labeling improves news identification and categorization with few annotated data | Language engineering | Text processing, news |
| 45 | Daniel Marcelo González Arriaga, María Aurora Diozcora Vargas Treviño, Josefina Guerrero García and Jesus López Gómez. Development of a platform to generate CNN and multilayer neural networks | Knowledge engineering | Classification |

Figura 7-37 Participación en el programa del congreso.


Figura 7-38 Constancia de participación.

## 7.6 Publicación 1: Computación y Sistemas

Como parte de la participación en el congreso 8th International Symposium on Language & Knowledge Engineering en la modalidad de poster, el articulo enviado será publicado en la revista Computación y sistema indexado por Scopus (Scopus Preview - Scopus - Computación y Sistemas, n.d.).

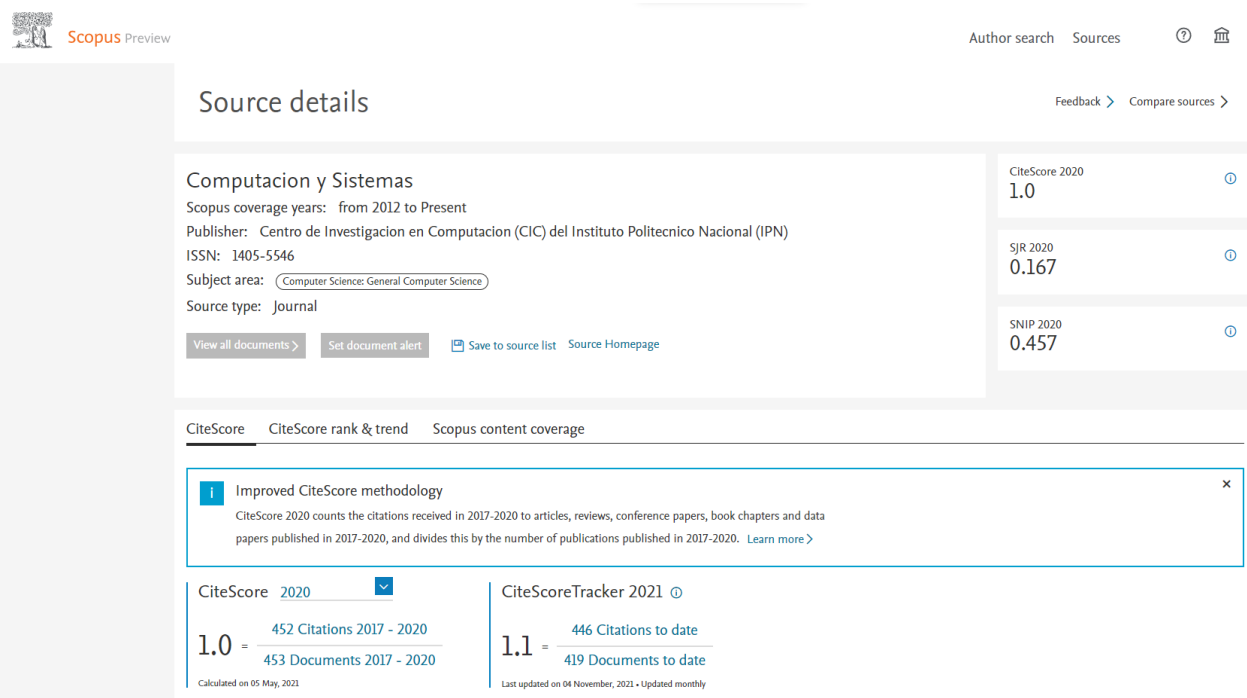



Figura 7-39 Pagina Scopus de revista Computación y Sistemas

ISSN 2007-9737

# Development of a Platform for Generation of CNN and Multilayer Neural Networks

Daniel Marcelo González-Arriaga[1], María Aurora Diozcora Vargas-Treviño[2], Josefina Guerrero García [1], Jesús López Gómez[3]

[1] Benemérita Universidad Autónoma de Puebla, Facultad de Ciencias de la Computación, México

[2] Benemérita Universidad Autónoma de Puebla, Facultad de Ciencias de la Electrónica, México

[3] Universidad Juárez Autónoma de Tabasco, División Académica de Ingeniería y Arquitectura, México

daniel.gonzaleza@alumno.buap.mx, {aurora.vargas,josefina.guerrero}@correo.buap.mx, jesus.lopezg@ujat.mx

**Abstract.** This research presents the design of a platform that assists in the generation of convolutional (CNN) and multilayer neural networks to provide a user-friendly interface for the design, formation, and development of neural networks. This platform is developed in LabVIEW as this software allows to perform inter-faces and generate an executable for use. It aims to reduce the development time of neural networks by providing an assistant-like graphical interface that guides the user through various common scenarios (data import, neural network construction and adjustment), allows the user to focus on solving their problems without having to write code, edit text files, or manually analyze recorded data. The user interface with the options offered is described. The way the neural network is generated is described. The results generated with the platform are presented producing an image with the proposed methodology applying a complete convolution layer. The usefulness of this platform is explained by presenting a case where there is a significant improvement in the development of a neural network, in time and reduction of errors.

**Keywords**: CNN, multilayer, software, platform, LabVIEW platform, MATLAB platform.

## 1 Introduction

The artificial neural network is a model extracted from the biological neural network. In the biological neural network, "when a neuron receives an exciting input that is large enough compared to its inhibitory input, it sends a peak of electrical activity through its axon. Learning occurs by changing the effectiveness of synapses so that the influence of one neuron on another changes [1].

Artificial intelligence is becoming a widely used tool for its robust applicability to problems, particularly those that cannot be solved well by humans, for example, in medicine where algorithms are used to identify subjects with a family history of an inherited disease or an increased risk of a chronic disease or in the evaluation of changes in human performance in such situations-rehabilitation [2].

There is a particular type of artificial neural network that makes a difference in practice, which is precisely the one that corresponds to networks used to process signals: convolutional networks. Their success in solving computer vision problems

Computación y Sistemas, Vol. 26, No. 1, 2022, pp. 171–182
doi: 10.13053/CyS-26-1-4162

Figura 7-40 Primera página de articulo enviado.

## 7.7 Congreso ICECCME 2023 y publicación en IEEE Xplore

Como parte de la participación en el congreso 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), el articulo enviado será publicado en la revista IEEE Xplore indexado por Scopus.

*Proc. of the International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME 2023)*
*19-20 July 2023, Tenerife, Canary Islands, Spain*

# A deep learning approach to identify dynamic parameters in a cartesian robot

Daniel Marcelo González-Arriaga*, María Aurora D. Vargas Treviño†, Sergio Vergara Limon†
Carlos Leopoldo Carreón Díaz de León* and Josefina Guerrero-Garcíal*
*Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla, Puebla, México
Email: daniel.gonzalez1@alumno.buap.mx
†Facultad de Ciencias de la Electrónica
Benemérita Universidad Autónoma de Puebla, Puebla, México

***Abstract*—This article proposes a deep-learning approach to identify dynamic parameters in a Cartesian robot. The parametric identification of a robot allows us to know its dynamic model. The dynamic model will enable us to perform control the robot in a better way. Two structures of convolutional neural networks generated with a platform that assists in developing previously developed neural networks are proposed. The reconstruction of the pair is compared with the parameters identified by both networks to find the best one for this purpose. The construction of the pair using simulated data reaches a similarity of 99.39% and using experimental data 95.32%.**

***Index Terms*—Cartesian robot, convolutional neural network, mechatronics, parameter identification, robotics**

### I. Introduction

Parametric identification is a line of research that makes possible the dimensionality of a mathematical model of a particular robot. The identification of a robot consists in finding the number of parameters associated with the ordinary differential equation. These parameters are generally linearly independent. The dynamic parameters are determined using the position, velocity, acceleration, and torque.

In [1], it's propose using neural networks to make the black box identification of the robot's forward dynamics. It predicts a step forward of the positions, speeds, and accelerations of the robot's joints based on the knowledge of the positions of the joints, the speeds in the previous time step, and the torques of action. In [2], it proposes a method of identifying dynamic load parameters based on the artificial neural network. The parameter difference algorithm is used to calculate the influence parameters of the operating environment of the industrial robot and eliminate interference. In [3], deep learning is applied to help identify the identification of dynamic parameters of a robot of 6 degrees of freedom (DoF), for the compensation of uncertain factors, the authors propose a new approach based on the deep neural network called the Uncertainty Compensation Model (UCM).

This article has one main contributions: The parameter identification of a cartesian robot with a smooth Coulomb friction model. Therefore, the dynamic model has nonlinear parameters. The parameter identification algorithm is tested on a real two DoF cartesian robot.

This article is divided into 4 sections. Section 2 presents the methodology of the dynamic model used and the neural networks developed, Section 3 shows the results of the formation and execution of the neural networks presented, and the values identified by the proposed methodology, Section 4 presents the conclusions.

### II. Methodology

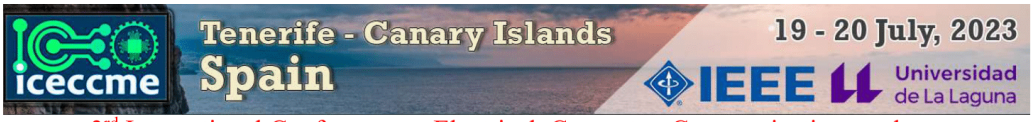
It is proposed to use neural networks to identify five parameters of the dynamic model of a Cartesian robot. This chapter presents the dynamic model used, presenting the proposed convolutional neural networks (CNN). These CNN were generated and trained on a platform that assists in generating the code of execution.

#### *A. Model of a two Dof Cartesian robot*

The dynamic model of a manipulator robot contains in its mathematical structure parameters such as centers of gravity, masses, moments of inertia and friction coefficients [4]. These parameters are unknown; this is the case for most commercial robots where the manufacturer does not provide their nominal values. While there are control theory tools such as adaptive schemes and robust controllers that allow for errors in dynamic parameters, knowledge of these is crucial for most schemes based on the dynamic model of a robot. The parametric identification problem has led to the derivation of several identification schemes that have become an attractive tool for determining the dynamic parameters of manipulating robots, especially when it is difficult to measure them directly. However, the non-linear nature of the dynamic model of manipulative robots makes the parametric identification task non-trivial. System identification can be defined as characterizing a dynamic system by observing its measurable behavior. When a priori information about the model of a system does not exist or is too complex, for example, in electrocardiographic signals, identification techniques are used to construct a model only

979-8-3503-2297-2/23/$31.00 © 2023 IEEE

Figura 7-41 Primera página de articulo enviado.




3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering

**ICECCME'23**

Tenerife-Canary Islands

SPAIN

***Acceptance Letter***

01.04.2023

**Dear Daniel Marcelo Gonzalez-Arriaga, Marıa Aurora D. Vargas Trevino, Sergio Vergara Limon, Carlos Leopoldo Carreon Dıaz de Leon, Josefina Guerrero-Garcıal;**

We are glad to inform you that your paper entitled **"ID 595 A deep learning approach to identify dynamic parameters in a cartesian robot"** is **accepted** as an oral or video presentation in the 2nd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME 2023) and for publication at the IEEE Conference Proceedings.

ICECCME'23 will take place in the TENERIFE-CANARY ISLANDS, SPAIN, on 19-20 July 2023. More details will be provided as the deadline approaches.

We strictly follow "no podium, no paper" policy and only the papers that are presented at the conference will be submitted to IEEE Explore for publication. At least one author of an accepted paper must register (as a full participant) and participate in ICECCME 2023 online or physically for the paper to be included in the proceedings. **The payment details will be announced soon. Kindly note that your registration becomes valid only after your payment.**

According to the conference regulations, only those papers which have been duly registered and presented on the conference day are considered for submission to IEEE Explore. The conference program will be communicated in due course.

We look forward to seeing you for a fruitful research and innovation event and for a great time in the wonderful environment of the Tenerife-Canary Islands, Spain.

Yours sincerely,

**Prof. Dr. Deivis Avila Prats**
ICECCME Conference Chair

Universidad de La Laguna,
Tenerife-Spain
E-mail: davilapr@ull.edu.es

Figura 7-42 Carta de aceptación ICECCME 23.

## Journal Bibliometrics

The Journal Citation Reports™ (JCR) and CiteScore metrics provide quantitative tools for ranking, evaluating, categorizing, and comparing journals. These metrics—including Journal Impact Factor™, Eigenfactor® Score, Article Influence® Score, and CiteScore metrics are available where applicable. These metrics examine the influence and impact of scholarly research journals.

The values displayed for the journal bibliometrics fields in IEEE *Xplore* are based on the Journal Citation Report from Clarivate from the 2021 report released in June 2022. The values displayed for CiteScore metrics are from Scopus 2021 report released in June 2022.

### Journal Impact Factor

Journal Impact Factor is the average number of times articles from a journal published in the past two years have been cited in the JCR year. The Journal Impact Factor is a so-called *popularity* measure which relies on the crude number of citations, each of them counting the same independently of the quality of the source.

### Eigenfactor Score

Eigenfactor Score takes into account the number of times articles from a journal published in the last five years have been cited in the JCR year while also considering which journals have contributed these citations. Since citations are in this case weighted dependently on the source, The Eigenfactor Score belongs to the class of so-called *prestige* measures. The Eigenfactor Score represents the probability of reading a specific journal in the entire collection and therefore high-scoring journals have a greater influence in the scientific community.

### Article Influence Score

Article Influence Score is also a *prestige* measure and has all the features of the Eigenfactor Score, with an additional normalization to the number of published papers. Hence it can be considered the average influence of a journal's articles over the first five years after publication.

### CiteScore

CiteScore is a measure reflecting the yearly average number of citations to recent articles published in a journal. The calculation is based on the citations recorded in the Scopus database using citations for articles published in the preceding four years.

Find out more about IEEE Journal Rankings.

Figura 7-43 IEEE Xplore detalles.

# 7.8 Certificado de ingles TOEFL

TOEFL ITP Score Report

Name of Institution: UNIV DEL VALLE DE PUEBLA

Name: GONZALEZ DANIEL
Student Number:

DOB: 04/22/1993 Sex: M Degree: Times Taken TOEFL: None

Native Country: Mexico
Native Language: Spanish

Scaled Scores:

| | | |
|---|---|---|
| Listening Comprehension: | 47 | Test Date: 04/26/2019 |
| Structure & Written Expression: | 41 | Form: TOEFL ITP |
| Reading Comprehension: | 52 | |
| Total Score: | 467 | |

ETS TOEFL ITP

Student's File Copy
Do Not Copy

The face of this document has a security background. The back contains a watermark. Hold at an angle to view.

The *TOEFL ITP* Assessment Series is designed to be used for placement, progress monitoring, and exit purposes. *TOEFL ITP* scores can also be used for admissions to programs and institutions where English is not the dominant language of instruction for content courses. Learn more at www.ets.org/toefl_itp/use.

127888-16573 • FB718R100 • Printed in U.S.A. I.N. 770462



Figura 7-44 Certificado de ingles TOEFL.

# Capitulo 8. Conclusiones

El modelado del sistema utilizando UML permite mostrar de manera clara la secuencia de ejecución y la manera en que interactúan los diferentes componentes de la plataforma.

La plataforma es capaz de generar una red neuronal y realizar su entrenamiento con los datos de configuración del usuario, así como los conjuntos de datos etiquetados.

Los resultados obtenidos con las redes neuronales presentadas en el capítulo 7 validan el correcto funcionamiento de la plataforma para realizar el entrenamiento de las redes neuronales y su posterior ejecución con un conjunto de datos desconocido.

En la sección 7.2 se describió un método para identificar los parámetros de un robot cartesiano de 2 grados de libertad con una metodología basada en CNN. Se propusieron cuatro estructuras diferentes de CNN para probar la plataforma desarrollada. Fueron implementados y entrenados usando la plataforma, y muestra que CNN 2 es el entrenamiento más rápido, y llegó al error mínimo de toda CNN. Se realizó la comparación de rendimiento en la reconstrucción de par utilizando un algoritmo de comparación de similitud utilizando las señales construidas con el modelo dinámico. La CNN 2 tiene el mejor rendimiento para torque 1 con 99.39% de similitud y la CNN 3 con 98.44% para torque 2. Por otro lado, utilizando las señales de torque experimentales, un menor porcentaje de similitud está presente. El mejor rendimiento es presentado por CNN 1 para torque 1 con 95.32% y en CNN 2 con torque 2 con 92.60% de similitud.

En la sección 7.3 se realizó la identificación paramétrica de dos ejes de un robot cartesiano de tres grados de libertad, con datos simulados y experimentales para probar la metodología con un robot real. Se validaron los resultados del par numérico y los resultados experimentales del entrenamiento de CNN. En ninguna de las trayectorias LS tuvo el mayor porcentaje de similitud. Analizando el valor mínimo y máximo del porcentaje de similitud, LS obtuvo 93.19% y 96.05%, CNN basado en el modelo de seis parámetros obtuvo 94.61% y 97.87% y CNN basada en el modelo de diez parámetros es del 94,45% y el 97,59%.

Al crear diferentes estructuras de redes neuronales para un mismo propósito, la plataforma facilita la comparación de diferentes redes neuronales para un mismo propósito, esto permite elegir la que mejor desempeño tenga para el propósito de cada usuario, ya sea un menor uso computacional, un menor tiempo de entrenamiento, o una mejor precisión.

Es de gran utilidad contar con diferentes funciones de activación para cada aplicación. Se considera agregar más funciones de activación en un futuro cercano y la posibilidad de que el usuario ingrese una función personalizada.

## Referencias


Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Research, G. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. https://arxiv.org/abs/1603.04467v2

Amatriain, X., & Arumi, P. (2011). Frameworks generate domain-specific languages: A case study in the multimedia domain. *IEEE Transactions on Software Engineering*, *37*(4), 544–558. https://doi.org/10.1109/TSE.2010.48

Atencia, M., Joya, G., & Sandoval, F. (2005). Hopfield Neural Networks for Parametric Identification of Dynamical Systems. *Neural Processing Letters*, *21*(2), 143–152. https://doi.org/10.1007/s11063-004-3424-3

Baehr, S. Skambraks, S. Neuhaus, S. Kiesling, C. Becker, J. (2017). A neural network on FPGAs for the z-vertex track trigger in Belle II. *Journal of Instrumentation*, *12*(3). https://doi.org/10.1088/1748-0221/12/03/C03065

Bai, J., Lu, F., Zhang, K., & others. (2019). ONNX: Open Neural Network Exchange. In *GitHub repository*. GitHub.

Berzal, F. (2018). Redes Neuronales & Deep Learning. In *Departamento de Ciencias de la Computacion e IA*.

Booch, Grady., Rumbaugh, James., & Jacobson, Ivar. (2006). *El Lenguaje Unificado de Modelado : guía del usuario*. Addison-Wesley.

Borelli, F. F., Biondi, G. O., & Kamienski, C. A. (2020). BIoTA: A Buildout IoT Application Language. *IEEE Access*, *8*(Cmcc), 126443–126459. https://doi.org/10.1109/ACCESS.2020.3003694

Chao, G. (2009). Human-computer interaction: Process and principles of human-computer interface design. *Proceedings - 2009 International Conference on Computer and Automation Engineering, ICCAE 2009*, 230–233. https://doi.org/10.1109/ICCAE.2009.23

Contreras, F. R., Alvarez, B., Sanchez, P., & Pastor, J. A. (2016). U-DSL: A domain specific language for ubiquitous systems. *IEEE Latin America Transactions*, *14*(10), 4416–4420. https://doi.org/10.1109/TLA.2016.7786324

Delaval, G., & Rutten, É. (2007). A domain-specific language for multitask systems, applying discrete controller synthesis. *Eurasip Journal on Embedded Systems*, *2007*. https://doi.org/10.1155/2007/84192

Downing, K. L. (2015). Intelligence Emerging: Adaptivity and Search in Evolving Neural Systems. In *The MIT Press*.

Duan, C., & Singh, R. (2006). Dynamics of a 3dof torsional system with a dry friction controlled path. *Journal of Sound and Vibration*, *289*(4–5), 657–688. https://doi.org/10.1016/J.JSV.2005.02.029

Frank, E., Hall, M. A., Witten, I. H., & Kaufmann, M. (2016). *WEKA Workbench Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques."*

Geoffroy, C., Michaud, J. B., Tetrault, M. A., Clerk-Lamalice, J., Brunet, C. A., Lecomte, R., & Fontaine, R. (2015). Real time artificial neural network fpga implementation for triple coincidences recovery in pet. *IEEE Transactions on Nuclear Science*, *62*(3), 824–831. https://doi.org/10.1109/TNS.2015.2432754

Gharbi, S. O. S., Moalemi, P., Jamshidi, M., Sergey, P., & Shariati, M. A. (2016). Application of Machine Vision in Cheese Industries. *INTERNATIONAL JOURNAL OF PHARMACEUTICAL RESEARCH AND ALLIED SCIENCES*, *5*(3), 350+.

Guellal, A., Larbes, C., Bendib, D., Hassaine, L., & Malek, A. (2015). FPGA based on-line Artificial Neural Network Selective Harmonic Elimination PWM technique. *International Journal of Electrical Power and Energy Systems*, *68*, 33–43. https://doi.org/10.1016/j.ijepes.2014.11.030

Hajduk, Z. (2017). High accuracy FPGA activation function implementation for neural networks. *Neurocomputing*, *247*, 59–61. https://doi.org/10.1016/j.neucom.2017.03.044

Hajduk, Z. (2018). Reconfigurable FPGA implementation of neural networks. *Neurocomputing*, *308*, 227–234. https://doi.org/10.1016/j.neucom.2018.04.077

Hamet, P., & Tremblay, J. (2017). Artificial intelligence in medicine. *Metabolism: Clinical and Experimental*, *69*, S36–S40. https://doi.org/10.1016/j.metabol.2017.01.011

Hu, J., Liu, W., Cheng, S., Tian, H., Yuan, H., & Zhao, H. (2017). Real-time Pedestrian Detection using Convolutional Neural Network on Embedded Platform. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, *10*(1), 35–40. https://doi.org/10.4271/2016-01-1877

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, 675–678. https://doi.org/10.1145/2647868.2654889

Johanson, A. N., & Hasselbring, W. (2017). Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment. *Empirical Software Engineering*, *22*(4), 2206–2236. https://doi.org/10.1007/s10664-016-9483-z

Kahraman, G., & Bilgen, S. (2015). A framework for qualitative assessment of domain-specific languages. *Software and Systems Modeling*, *14*(4), 1505–1526. https://doi.org/10.1007/s10270-013-0387-8

Kakani, V., Nguyen, V. H., Kumar, B. P., Kim, H., & Pasupuleti, V. R. (2020). A critical review on computer vision and artificial intelligence in food industry. *Journal of Agriculture and Food Research*, *2*. https://doi.org/10.1016/J.JAFR.2020.100033

Lu, S., Qian, G., He, Q., Liu, F., Liu, Y., & Wang, Q. (2019). Insitu Motor Fault Diagnosis Using Enhanced Convolutional Neural Network in an Embedded System. *IEEE Sensors Journal*, *1748*(c), 1–1. https://doi.org/10.1109/jsen.2019.2911299

Meng, J., Chen, X., Gall, J., Kim, C.-S., Liu, Z., Piva, A., & Yuan, J. (2022). The Future of Computer Vision. *APSIPA Transactions on Signal and Information Processing*, *11*(1). https://doi.org/10.1561/116.00000009

Menotti, R., Cardoso, J. M. P., Fernandes, M. M., & Marques, E. (2012). LALP: A language to program custom FPGA-based acceleration engines. *International Journal of Parallel Programming*, *40*(3), 262–289. https://doi.org/10.1007/s10766-011-0187-0

*MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. (n.d.). Retrieved May 9, 2022, from http://yann.lecun.com/exdb/mnist/

Nazir, A., Alam, M., Malik, S. U. R., Akhunzada, A., Cheema, M. N., Khan, M. K., Ziang, Y., Khan, T., & Khan, A. (2017). A high-level domain-specific language for SIEM (design, development and formal verification). *Cluster Computing*, *20*(3), 2423–2437. https://doi.org/10.1007/s10586-017-0819-2

Purnamawati, S., Rachmawati, D., Lumanauw, G., Rahmat, R. F., & Taqyuddin, R. (2018). Korean letter handwritten recognition using deep convolutional neural network on android platform. *Journal of Physics: Conference Series*, *978*(1). https://doi.org/10.1088/1742-6596/978/1/012112

Qiao, Y., Shen, J., Xiao, T., Yang, Q., Wen, M., & Zhang, C. (2017). FPGA-accelerated deep convolutional neural networks for high throughput and energy efficiency. *Concurrency Computation* , *29*(20), 1–20. https://doi.org/10.1002/cpe.3850

Reyes Cortés, F. (2011). *Robótica. Control de Robots Manipuladores*. ALFAOMEGA.

*Scopus preview - Scopus - Computacion y Sistemas*. (n.d.). Retrieved November 23, 2021, from https://www.scopus.com/sourceid/21100223167

Slivnik, B. (2016). Measuring the complexity of domain-specific languages developed using MDD. *Software Quality Journal*, *24*(3), 737–753. https://doi.org/10.1007/s11219-015-9279-1

Szadkowski, Zbigniew Pytel, K. (2015). Artificial Neural Network as a FPGA Trigger for a Detection of Very Inclined Air Showers. *IEEE Transactions on Nuclear Science*, *62*(3), 1002–1009. https://doi.org/10.1109/TNS.2015.2421412

Tôn, L. P., & Truong, T. M. (2016). Linking Rules and Conceptual Model in a Domain Specific Language. *Proceedings - 2015 International Conference on Advanced Computing and Applications, ACOMP 2015*, 35–42. https://doi.org/10.1109/ACOMP.2015.25

Tran Tan, A., Falcou, J., Etiemble, D., & Kaiser, H. (2016). Automatic Task-Based Code Generation for High Performance Domain Specific Embedded Language. *International Journal of Parallel Programming*, *44*(3), 449–465. https://doi.org/10.1007/s10766-015-0354-9

Urrea, C., & Pascal, J. (2021). Design and validation of a dynamic parameter identification model for industrial manipulator robots. *Archive of Applied Mechanics*, *91*(5), 1981–2007. https://doi.org/10.1007/S00419-020-01865-2/METRICS

Voelter, M., Kolb, B., Birken, K., Tomassetti, F., Alff, P., Wiart, L., Wortmann, A., & Nordmann, A. (2019). Using language workbenches and domain-specific languages for safety-critical software development. *Software and Systems Modeling*, *18*(4), 2507–2530. https://doi.org/10.1007/s10270-018-0679-0

Wang, Y., & Li, Z. (2016). GridFOR: A Domain Specific Language for Parallel Grid-Based Applications. *International Journal of Parallel Programming*, *44*(3), 427–448. https://doi.org/10.1007/s10766-014-0348-z

Yin, H., Yi, W., & Hu, D. (2022). Computer vision and machine learning applied in the mushroom industry: A critical review. *COMPUTERS AND ELECTRONICS IN AGRICULTURE*, *198*. https://doi.org/10.1016/j.compag.2022.107015