



Casa abierta al tiempo

Universidad Autónoma Metropolitana

Posgrado en Ciencias y Tecnologías de la Información

División de Ciencias Básicas e Ingeniería

Unidad Iztapalapa

“EVOLUCIÓN DE PARÁMETROS DE OPTIMIZACIÓN DE REDES
NEURONALES PROFUNDAS PARA EL RECONOCIMIENTO Y CLASIFICACIÓN
DE IMÁGENES”

Tesis que presenta

Lic. Yafte Aaron Flores Morales

Matrícula

2183802184

Para obtener el grado de

Maestro en Ciencias

(Ciencias y Tecnología de la información)

Directores

Dra. Graciela Román Alonso

Dr. Juan Villegas Cortez

Jurado

Presidente Dr. Francisco Fernandez de Vega

Secretario Dr. Eric Alfredo Rincon Garcia

Vocal Dr. Juan Villegas Cortez

Unidad Iztapalapa

Universidad Autónoma Metropolitana

División de Ciencias Básicas e Ingeniería

Ciudad de México, Iztapalapa, noviembre del 2022



Universidad Autónoma Metropolitana

Posgrado en Ciencias y Tecnologías de la Información

División de Ciencias Básicas e Ingeniería

Unidad Iztapalapa

Evolución de parámetros de optimización de redes neuronales profundas
para el reconocimiento y clasificación de imágenes

Tesis que presenta

Lic. Yafte Aaron Flores Morales

Para obtener el grado de

Maestro en Ciencias

(Ciencias y Tecnología de la información)

Asesores

Dra. Graciela Román Alonso

Dr. Juan Villegas Cortez

Unidad Iztapalapa,
Universidad Autónoma Metropolitana
División de Ciencias Básicas e Ingeniería
Ciudad de México, 21 de noviembre de 2022

Lic. Yafte Aaron Flores Morales
Universidad Autónoma Metropolitana

EVOLUCIÓN DE PARÁMETROS DE
OPTIMIZACIÓN DE REDES NEURONALES
PROFUNDAS PARA EL RECONOCIMIENTO Y
CLASIFICACIÓN DE IMÁGENES

RESUMEN

La importancia de la aplicación del reconocimiento de patrones para el reconocimiento de imágenes, se hace notar cuando el número y la complejidad de las imágenes aumenta. Las redes neuronales artificiales han demostrado su efectividad para atacar problemas de patrones complejos en los últimos quince años aproximadamente y han evolucionado hacia arquitecturas híbridas y profundas. La construcción de las redes neuronales artificiales se ha dividido en dos etapas; la etapa de entrenamiento y la etapa de prueba, en la etapa de entrenamiento es necesario de conocimiento experto para la selección y ajuste de los parámetros de las redes neuronales artificiales.

En este trabajo presentamos la implementación de un sistema construido con cómputo evolutivo y cómputo paralelo, que optimiza los parámetros de una red neuronal profunda gruesa-fina basada en el estado del arte. Esta red neuronal profunda gruesa-fina realiza una alimentación paralela de los patrones a diferentes niveles de granularidad: fina, mediana y gruesa; buscando conformar un análisis del patrón característico robusto aplicado al reconocimiento y clasificación de imágenes. La red neuronal profunda de la cual partimos, se ha modificado y orientado, de patrones de reconocimiento de actividades humanas (HAR), hacia atacar la complejidad de las imágenes, con resultados prometedores del 99 % de reconocimiento en las pruebas realizadas con la base de datos MNIST.

Se realizaron dos versiones del sistema; Algoritmo evolutivo secuencial y Algoritmo evolutivo paralelo, este último basado en el modelo de comunicación de paso de mensajes, utilizando el protocolo MPI. Además, se utilizaron tarjetas gráficas (GPU) y computador multiprocesador para el manejo masivo de operaciones. Finalizando con una sección de pruebas del sistema, con diferentes bases de datos de reconocimiento de actividad humana y de imágenes, analizando y comparando los resultados con el estado del arte.

Palabras Clave: RNC · Cómputo evolutivo · Clasificación · Optimización · Aprendizaje profundo · Reconocimiento de patrones · HAR · Reconocimiento de imágenes · Algoritmo genético · Cómputo paralelo · MPI

ABSTRACT

The importance of the application of pattern recognition for image recognition becomes apparent when the number and complexity of images increases. Artificial neural networks have proven their effectiveness in tackling complex pattern problems in the last fifteen years or so and have evolved into deep, hybrid architectures. The construction of neural networks is divided into two stages; the training stage and the test stage, in these stages expert knowledge is needed for the selection and adjustment of the parameters of the neural networks.

In this work we present the implementation of a system that optimizes the parameters of a deep neural network based on the state of the art by means of evolutionary computation and parallel computation. This deep Coarse-Fine neural network performs a parallel feeding of the patterns at different levels of granularity: fine, medium and coarse; seeking to form an analysis of the robust characteristic pattern applied to image recognition and classification. We have modified and oriented the deep network from which we started, from patterns of recognition of human activities (HAR), towards attacking the complexity of the images, with promising results of 99 % recognition in the tests carried out with the database MNIST.

Two versions of the system were made; Sequential Evolutionary Algorithm and Parallel Evolutionary Algorithm, the latter based on the message passing communication model, using the MPI protocol. In addition, the use of graphic cards (GPU) and multiprocessor computer for the massive handling of operations. We end with a set of system tests with different mechanisms applied to the system, analyzing and comparing the results with the state of the art.

Keywords: CNN · Evolutionary Computation · Classifier · Optimization · Deep Learning · Pattern Recognition · HAR · Computer Vision · Genetic Algorithm · Parallel Computing · MPI

CONTENIDO

Contenido	VII
Lista de figuras	XI
Lista de tablas	XIII
Lista de algoritmos	XV
Lista de acrónimos	XVII
Dependencias lógicas de capítulos	XIX
Parte I Presentación	1
1 Introducción	3
1.1 Definición del problema.	3
1.1.1 Pregunta de investigación fundamental.	4
1.1.2 Hipótesis	4
1.1.3 Justificación.	4
1.2 Objetivos.	4
1.2.1 Objetivo general	4
1.2.2 Objetivos particulares	4
1.3 Metodología	5
Parte II Antecedentes	7
2 Marco teórico	9

2.1	Historia	9
2.2	Redes neuronales biológicas	9
2.3	Aprendizaje automático.	10
2.4	Aprendizaje profundo	10
2.5	Redes neuronales artificiales	11
2.6	Redes neuronales convolucionales	13
2.6.1	Capa de entrada	13
2.6.2	Convolución	13
2.6.3	Función de activación	15
2.6.4	Reducción	16
2.6.5	Aplanado	17
2.6.6	Capa totalmente conectada	17
2.6.7	Dropout	18
2.6.8	Softmax	19
2.6.9	Capa de salida	19
2.7	Cómputo evolutivo	19
2.8	Algoritmos genéticos	19
2.8.1	Componentes de un algoritmo genético.	20
2.8.2	Operadores genéticos.	21
2.9	Cómputo paralelo	23
2.9.1	Modelos de comunicación	23
2.9.2	Tiempo de ejecución	24
3	Estado del arte	25
3.1	Trabajos relacionados	25
3.2	Red neuronal gruesa-fina	26
3.3	Reconocimiento de actividad humana	26
3.3.1	Métodos basados en ML	27
3.3.2	Métodos basados en DL	28
3.4	Reconocimiento de imágenes	30
3.4.1	Métodos basados en ML	30
3.4.2	Métodos basados en DL	30
	Parte III Propuestas	31
4	Propuesta principal	33
4.1	Ajuste de la red neuronal gruesa-fina	33
4.2	Algoritmo genético secuencial	36
4.2.1	Definición del individuo	37
4.2.2	Operadores genéticos.	40
4.2.3	Función de aptitud propuesta	40
4.2.4	Parámetros del algoritmo genético	41
4.3	Algoritmo genético paralelo	42
5	Bases de datos seleccionadas	45

5.1	UCI HAR	45
5.2	HAPT	47
5.3	WISDM v1.1	47
5.4	WISDM v2.0	49
5.5	MNIST	53
Parte IV Resultados y conclusiones		55
6	Evaluación de la red neuronal gruesa-fina	57
6.1	Plataforma de prueba.	57
6.2	Resultados del ajuste de la red neuronal gruesa-fina	57
6.2.1	Resultados del conjunto de datos UCI HAR	58
6.2.2	Resultados del conjunto de datos HAPT	58
6.2.3	Resultados del conjunto de datos WISDM v1.1	59
6.2.4	Resultados del conjunto de datos WISDM v2.0	60
6.2.5	Resultados del conjunto de datos MNIST	60
6.3	Conclusiones y análisis sobre la implementación de la red neuronal propuesta hacia imágenes y nuevos conjuntos de datos	60
7	Evaluación de la propuesta principal	63
7.1	Plataforma de prueba.	63
7.2	Parámetros del algoritmo genético.	63
7.3	Resultados del conjunto de datos UCI HAR	64
7.4	Resultados del conjunto de datos UCI HAPT	66
7.5	Resultados del conjunto de datos WISDM v1.1.	68
7.6	Resultados del conjunto de datos WISDM v2.0.	69
7.7	Resultados del conjunto de datos MNIST.	71
7.8	Escalabilidad	73
7.9	Comparación con el estado del arte	74
7.9.1	UCI HAR	74
7.9.2	UCI HAPT	75
7.9.3	WISDM v1.1	76
7.9.4	WISDM v2.0	77
7.9.5	MNIST	78
8	Conclusiones y Trabajo futuro	79
Apéndices		81
A	Código	83
A.1	Principal	84
A.2	Genético	84
A.3	Conjunto de datos	90
A.4	Red neuronal convolucional	92
B	Artículos publicados	99
C	Glosario	115

LISTA DE FIGURAS

Figura 1.1	Diagrama de bloques de la metodología propuesta.	6
Figura 2.1	Red neuronal biológica.	10
Figura 2.2	Diagrama de Venn que muestra algunas áreas de la inteligencia artificial.	11
Figura 2.3	Red neuronal artificial clásica.	12
Figura 2.4	Pesos asociados a las conexiones de cada neuronal de la red neuronal artificial.	12
Figura 2.5	Elementos de una CNN.	14
Figura 2.6	Estructura de una imagen en el espacio RGB.	14
Figura 2.7	Ejemplo de la operación de convolución con diferentes filtros.	15
Figura 2.8	Ejemplo de la operación de Pooling con diferentes mecanismos.	17
Figura 2.9	Ejemplo de la transformación de la imagen aplicando maxpooling.	18
Figura 2.10	Ejemplo de la tarea de Dropout.	18
Figura 2.11	Diagrama de flujo de un algoritmo evolutivo general.	20
Figura 2.12	Diagrama de flujo de un algoritmo genético clásico.	20
Figura 2.13	Mecanismo del operador elitismo, selecciona los mejores tres individuos.	21
Figura 2.14	Mecanismo del operador de cruza.	22
Figura 2.15	Mecanismo del operador de Mutación.	22
Figura 2.16	Combinación de operadores genéticos.	23
Figura 3.1	Modelo de red neuronal convolucional gruesa-fina. [6]	26
Figura 4.1	Modelo de red neuronal convolucional gruesa-fina.	34
Figura 4.2	Metodología del algoritmo genético con una red neuronal gruesa-fina.	36
Figura 4.3	Estructura final del individuo propuesto.	38
Figura 4.4	Ejemplo de la estructura propuesta del individuo a evolucionar y los parámetros que involucra la red neuronal.	40
Figura 4.5	Porcentaje de individuos generados por los operadores genéticos para una nueva población.	41
Figura 4.6	Algoritmo genético paralelo.	42
Figura 5.1	Adecuación y estructura del conjunto de datos UCI HAR.	46
Figura 5.2	Distribución del conjunto de datos UCI HAR.	46
Figura 5.3	Adecuación y estructura del conjunto de datos HAPT.	47
Figura 5.4	Distribución de clases del conjunto de prueba de HAPT.	48
Figura 5.5	Adecuación y metodología y estructura del conjunto de datos WISDM v1.1.	49
Figura 5.6	Distribución de clases del conjunto de datos WISDM v1.1.	50
Figura 5.7	Adecuación y estructura del conjunto de datos WISDM v2.0.	51
Figura 5.8	Distribución de procesamiento del conjunto de datos WISDM v2.0.	52
Figura 5.9	Ejemplo de imágenes con resolución de 28×28 píxeles del conjunto de datos MNIST.	53

Figura 5.10	Distribución de procesamiento del conjunto de datos MNIST.	54
Figura 6.1	Evaluación del conjunto entrenamiento de HAR.	58
Figura 6.2	Evaluación del conjunto prueba de HAPT.	59
Figura 6.3	Evaluación del conjunto entrenamiento de WISDM v1.1.	59
Figura 6.4	Evaluación del conjunto prueba de WISDM v2.0.	60
Figura 6.5	Evaluación del conjunto prueba de MNIST.	61
Figura 7.1	Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.	65
Figura 7.2	Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.	67
Figura 7.3	Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.	68
Figura 7.4	Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.	70
Figura 7.5	Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.	72
Figura 7.6	Comparación de escalabilidad de los modelos genéticos con la base de datos UCI HAR.	73
Figura 7.7	Comparación de resultados obtenidos con el estado del arte de la base de datos UCI HAR.	74
Figura 7.8	Comparación de resultados obtenidos con el estado del arte de la base de datos HAPT.	75
Figura 7.9	Comparación de resultados obtenidos con el estado del arte de la base de datos WISDM v1.1.	76
Figura 7.10	Comparación de resultados obtenidos con el estado del arte de la base de datos WISDM v2.0.	77
Figura 7.11	Comparación de resultados obtenidos con el estado del arte de la base de datos MNIST.	78

LISTA DE TABLAS

Tabla 3.1	Comparación de exactitud de UCI HAR [4].	28
Tabla 3.2	Comparación de exactitud de UCI HAPT [51].	29
Tabla 3.3	Comparación de exactitud de WISDM 1.1v [30].	29
Tabla 3.4	Comparación de exactitud de WISDM 2.0v [30] [65] [39].	29
Tabla 3.5	Comparación de exactitud de MNIST [32].	30
Tabla 4.1	Parámetros pre-establecidos para la red neuronal profunda gruesa-fina. . .	35
Tabla 4.2	Estructura propuesta del individuo.	38
Tabla 5.1	Número de instancias por clase del conjunto de datos UCI HAR.	45
Tabla 5.2	Número de instancias por clase de HAPT procesada y particionada.	49
Tabla 5.3	Número de instancias por clase del conjunto de datos WISDM v1.1.	50
Tabla 5.4	Número de instancias por clase de WISDM v2.0 procesada y particionada.	51
Tabla 5.5	Comparación de clases contenidas en los conjuntos de datos de reconoci- miento de activada humana.	52
Tabla 5.6	Número de instancias por clase de la base de datos MNIST.	53
Tabla 6.1	Librerías utilizadas en la ejecución.	58
Tabla 6.2	Comparación de exactitud de diferentes técnicas del estado del arte.	61
Tabla 7.1	Parámetros utilizados en los modelos evolutivos.	64
Tabla 7.2	Resultados de los modelos genéticos con el conjunto de datos UCI HAR. . .	64
Tabla 7.3	Parámetros optimizados encontrados por los modelos evolutivos.	65
Tabla 7.4	Resultados de los modelos genéticos con el conjunto de datos UCI HAPT. . .	66
Tabla 7.5	Parámetros optimizados encontrados por los modelos evolutivos.	67
Tabla 7.6	Resultados de los modelos genéticos con el conjunto de datos WISDM v1.1.	68
Tabla 7.7	Parámetros optimizados encontrados por los modelos evolutivos.	69
Tabla 7.8	Resultados de los modelos genéticos con el conjunto de datos WISDM v2.0.	69
Tabla 7.9	Parámetros optimizados encontrados por los modelos evolutivos.	70
Tabla 7.10	Resultados para MNIST.	71
Tabla 7.11	Parámetros optimizados encontrados por los modelos evolutivos.	72
Tabla 7.12	Comparación de exactitud de UCI HAR [4].	74
Tabla 7.13	Comparación de exactitud de UCI HAPT [51].	75
Tabla 7.14	Comparación de exactitud de WISDM v1.1 [30].	76
Tabla 7.15	Comparación de exactitud de WISDM v2.0 [30] [65] [39].	77
Tabla 7.16	Comparación de exactitud de MNIST [32].	78

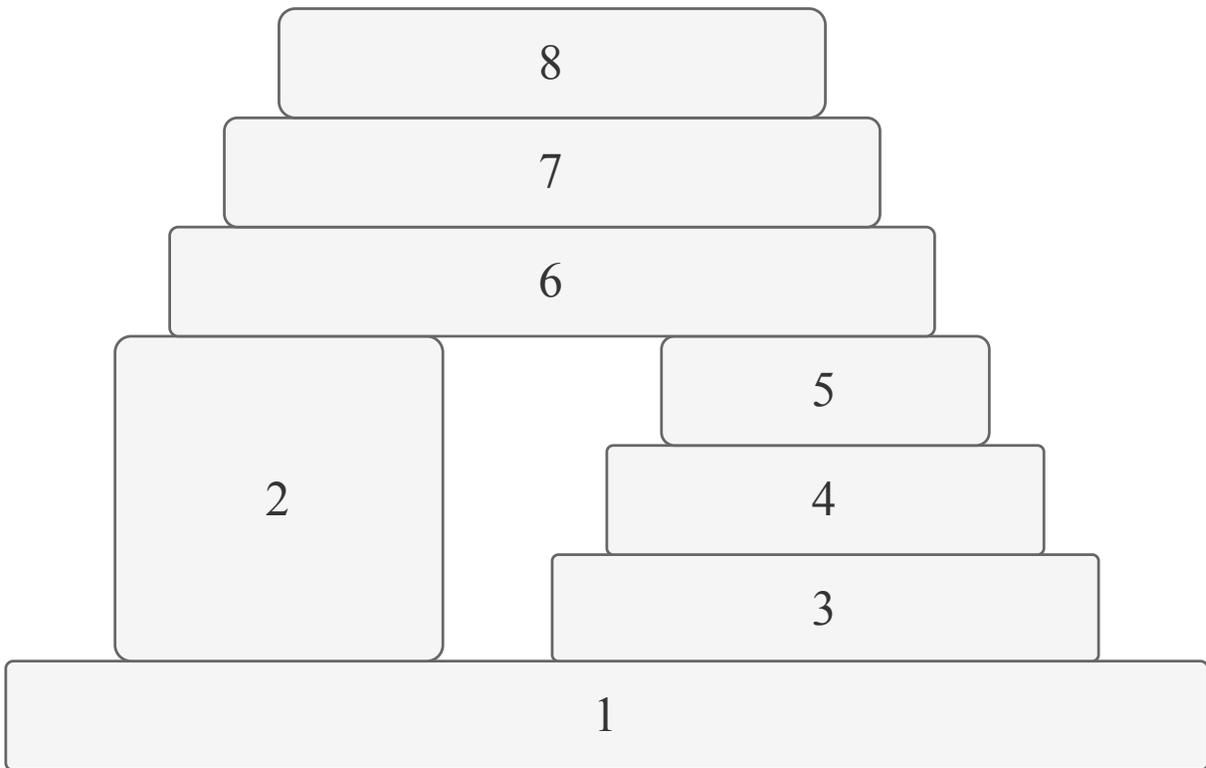
LISTA DE ALGORITMOS

Algoritmo 4.1	Algoritmo genético	37
Algoritmo 4.2	Algoritmo genético paralelo en el nodo <i>id</i>	43

LISTA DE ACRÓNIMOS

RNA	Red Neuronal Artificial	2.1
ANN	Artificial Neural Network	2.1
ILSVRC	ImageNet Large Scale Visual Recognition Challenge.....	2.1
IA	Inteligencia Artificial	2.3
ML	Machine learning	2.3
DL	Deep Learning	2.4
RNN	Recurrent Neural Network	2.5
RELU	Rectified Linear Units	2.6
CNN	Convolutional Neural Network	2.6
RNC	Red Neuronal Convolutacional	2.6
AG	Algoritmo Genético	2.7
EE	Estrategia Evolutiva	2.7
PG	Programación Genética	2.7
MPI	Message Passing Interface.....	2.9
GPU	Graphics Processor Unit	2.9
RGB	Red, Green, Blue	3.1
LSTM	Long Short Term Memory	3.1
SVM	Support Vector Machine	3.1

DEPENDENCIAS LÓGICAS DE CAPÍTULOS



CAPÍTULOS

Capítulo 1	Introducción
Capítulo 2	Marco Teórico
Capítulo 3	Estado del Arte
Capítulo 4	Red neuronal gruesa-fina
Capítulo 5	Implementación
Capítulo 6	Propuesta principal
Capítulo 7	Evaluación
Capítulo 8	Conclusiones y Trabajo futuro

Parte I

PRESENTACIÓN

No hay que ir para atrás ni para darse impulso.

LAO TSÉ

1

INTRODUCCIÓN

Este capítulo presenta un preámbulo del tema de investigación, se detalla el problema de estudio, pregunta de investigación fundamental, se plantea la hipótesis, la justificación, se muestra el alcance del trabajo de investigación mediante los objetivos generales y particulares, finalmente se propone la metodología en diagrama de bloques.

1.1 Definición del problema

La clasificación de imágenes de forma automatizada sigue siendo un problema abierto, al día de hoy se han trabajado diversas metodologías para resolver el problema. En los últimos años las redes neuronales profundas han sido populares en el procesamiento de datos a gran escala y muchos trabajos han demostrado que son herramientas prometedoras en muchos campos, en especial, en la clasificación [43].

Las imágenes en su estudio se diversifican en metodologías que orientan el reconocimiento de los mismos en tipo de patrones y por tipo de valores usados (binario, natural, entero, real), determinar si están o no acotados en sus valores, analizar el comportamiento de sus componentes a partir de las herramientas estadísticas, o bien de analizar sus componentes principales o cuáles determinan su comportamiento en el tiempo. Los clasificadores estadísticos, las redes neuronales artificiales, la lógica difusa, los algoritmos genéticos, la optimización por enjambres de partículas y la programación genética, son algunas de las técnicas más usadas en el área de aplicación del reconocimiento de patrones. Específicamente los algoritmos genéticos han mostrado ser una alternativa para atacar problemas de reconocimiento de patrones, tanto para elegir las posibles combinaciones de atributos necesarias para un entrenamiento de algoritmos de aprendizaje, como para la modificación de estructuras de los mismos algoritmos en cómo procesan la información de forma secuencial.

Cuando se trabaja con redes neuronales profundas es importante definir sus parámetros, muchas veces estos parámetros son determinados por medio de la idea básica de ensayo y error, ésta se basa en ejecutar la red neuronal y modificar sus parámetros con base a mejoras en la clasificación y tiempo en cada ejecución.

1.1.1 Pregunta de investigación fundamental

¿Por medio del cómputo evolutivo y cómputo paralelo se podrá optimizar los parámetros de una red neuronal profunda para el reconocimiento y clasificación de imágenes?

1.1.2 Hipótesis

La red neuronal profunda puede optimizarse por medio del cómputo evolutivo y el cómputo paralelo.

1.1.3 Justificación

Los métodos actuales de ajuste de parámetros se hacen de manera manual, por tanto, el tiempo de ajuste de los parámetros depende del conocimiento del experto, el tamaño y la complejidad del conjunto de datos, entonces, este proceso de ajuste no automático de parámetros puede llegar a ser largo, y si no se obtienen los ajustes de parámetros adecuados es difícil obtener buenos resultados en la clasificación y reconocimiento de imágenes. Por lo tanto, determinar los parámetros de manera automática de la red neuronal profunda permite una mejora en la clasificación y en tiempo de ejecución para el reconocimiento y clasificación de imágenes. También se pretende aprovechar las características del cómputo evolutivo y cómputo paralelo para reducir el tiempo de entrenamiento de la red neuronal profunda.

1.2 Objetivos

En este proyecto de investigación se propone una nueva estrategia de red neuronal profunda integrando cómputo evolutivo, con un enfoque paralelo. Además, optimizar la red neuronal profunda con ayuda de la determinación de los parámetros de forma automática con un algoritmo genético. A continuación se enlistan los objetivos generales y objetivos particulares.

1.2.1 Objetivo general

Optimizar una red neuronal profunda por medio de cómputo evolutivo para la identificación y clasificación de imágenes.

1.2.2 Objetivos particulares

1. Estudiar e identificar las topologías y estructuras generales de las redes profundas probadas en la literatura.

2. Estudiar el estado del arte del cómputo evolutivo desde la optimización combinatoria y los algoritmos genéticos.
3. Experimentar con la implementación de la red neuronal profunda en el ámbito de un problema de reconocimiento y clasificación de imágenes.
4. Determinar los parámetros a optimizar y modificar de forma permisible en una red profunda por medio del cómputo evolutivo.
5. Aplicar al menos una red neuronal profunda optimizada en un problema específico de reconocimiento y clasificación de imágenes, con base a la optimización desarrollada.

1.3 Metodología

La metodología de este trabajo esta compuesta en cinco etapas, a continuación se describen las etapas de la metodología. La Figura 1.1 muestra un diagrama de bloques de la metodología propuesta.

1. Determinación del tipo de red neuronal profunda a implementar con base al estudio del estado del arte para el reconocimiento de imágenes.
2. Implementación de la red neuronal profunda para el reconocimiento de imágenes.
3. Determinación de parámetros para la optimización de la red neuronal profunda implementada.
4. Optimización de la red neuronal profunda mediante cómputo evolutivo y cómputo paralelo.
5. Entrenamiento y prueba del sistema con una base de datos de un tipo de imágenes, realizar análisis y comparaciones con los resultados obtenidos.

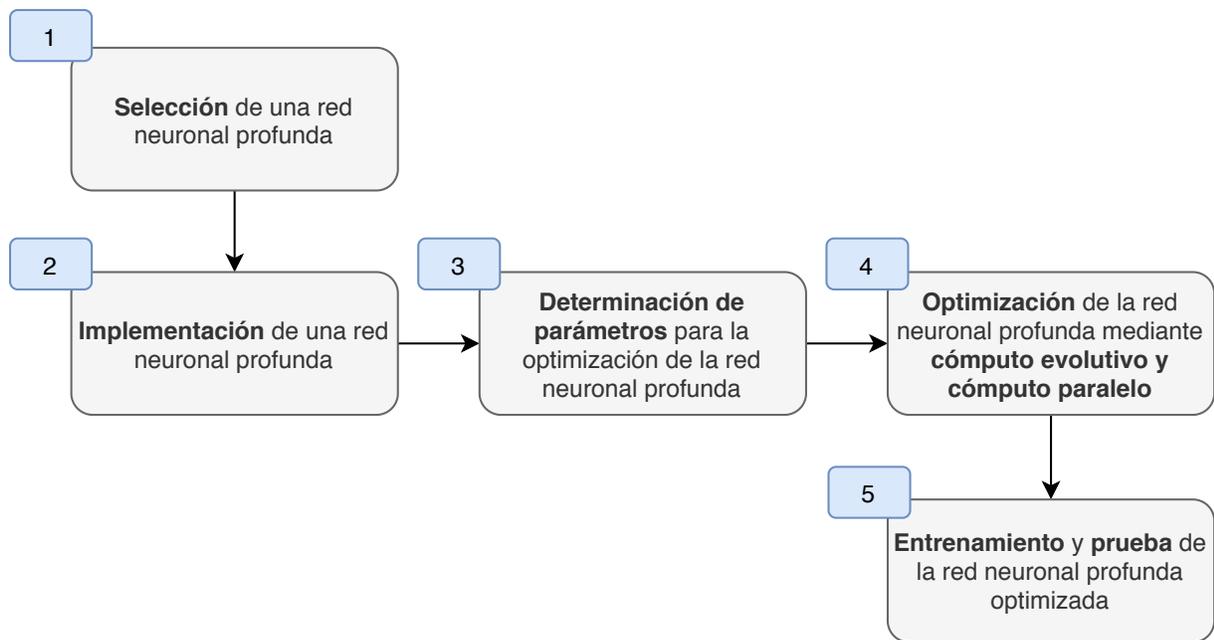


Figura 1.1: Diagrama de bloques de la metodología propuesta.

Parte II

ANTECEDENTES

Si el presente trata de juzgar el pasado, perderá el futuro.
WINSTON CHURCHILL

2

MARCO TEÓRICO

El propósito principal del capítulo consiste en abarcar los fundamentos teóricos que permitan abordar la propuesta principal de investigación y orientar el trabajo de un modo coherente, se explica el reconocimiento de imágenes, la clasificación, las redes neuronales artificiales, el cómputo evolutivo y el cómputo paralelo, así como sus principales características, ventajas, desventajas y aplicaciones.

2.1 Historia

En 1943 Warren McCulloch de la universidad de Illinois y Walter Pitts de la universidad Chicago, desarrollaron un modelo de la neurona biológica inspirada en la red neuronal del cerebro. Al conjunto de neuronas artificiales interconectadas se le llama *red neuronal artificial* (RNA o por sus siglas en inglés ANN). La tarea principal que se pretende emular son las habilidades para el reconocimiento de patrones del cerebro desde una mera perspectiva bio-inspirada. Actualmente existen diferentes modelos de estas redes como: las redes neuronales recurrentes, redes neuronales convolucionales, redes neuronales profundas, entre otras ¹.

En los últimos años las redes neuronales convolucionales han ganado fama, en especial en la competencia *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [26] que se hace cada año desde el 2010; desde AlexNet [28] hasta ResNet [24] agregando nuevos algoritmos de aprendizaje y capas durante este proceso. Aunque Yann LeCun y su grupo de trabajo ya habían desarrollado las bases desde finales de los 90 del siglo pasado [33], con el avance del hardware han cobrado mayor fuerza, debido a que las unidades de procesamiento de gráficos o *graphics processing unit* (GPUs) pueden ser utilizadas para entrenar en horas una red neuronal artificial, a comparación de hace 5 años que se hacía en días o semanas.

2.2 Redes neuronales biológicas

Las redes neuronales artificiales están basadas en el cerebro de los seres vivos. Las partes utilizadas para modelar la red neuronal biológica son: las dendritas, la sinapsis, el cuerpo celular y el axón. [19] La información llega a través de las dendritas, con las conexiones llamadas sinapsis que sirven como unión entre neuronas. Después, la información pasa por el cuerpo celular para ser procesada. Este cuerpo celular da resultados a la salida utilizando unas fibras largas llamadas

¹Britannica Academic, s.v. Artificial intelligence (AI), accessed October 12, 2020, <https://bidi.uam.mx:6402/levels/collegiate/article/artificial-intelligence/9711219105.toc>

axón que sirven para enviar información a otra neurona. En la Figura 2.1 se muestra una neurona biológica perteneciente al sistema nervioso central. Al conjunto de neuronas interconectadas se le llama red neuronal.

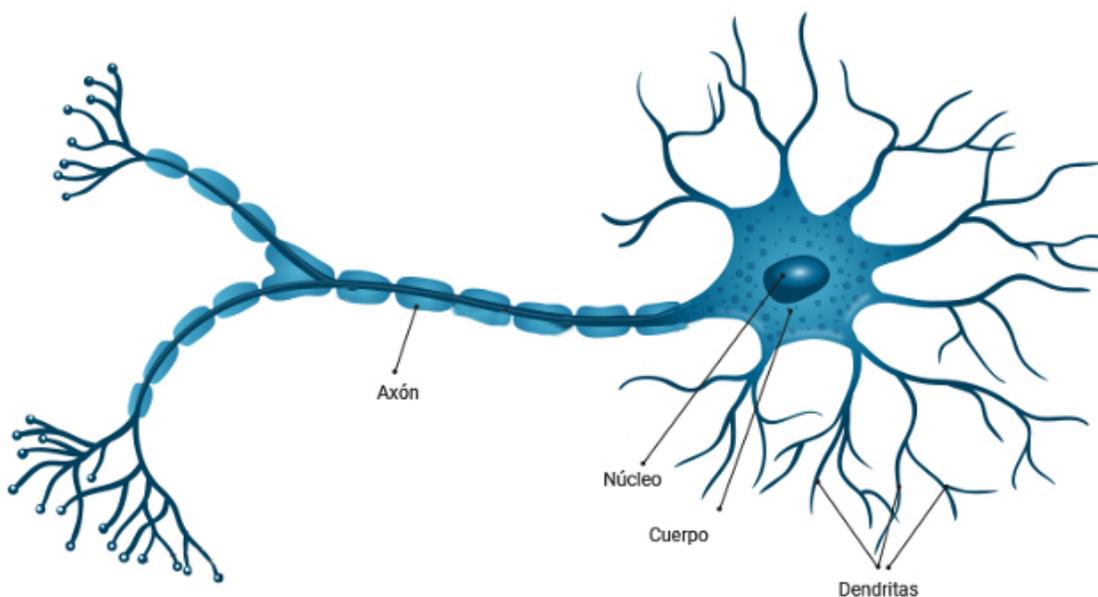


Figura 2.1: Red neuronal biológica.

2.3 Aprendizaje automático

El aprendizaje automático o Machine Learning (ML) forma parte de la inteligencia artificial (IA), este enfoque se basa en la idea de que los sistemas pueden aprender de los datos, identificar patrones y tomar decisiones con mínima intervención humana [21]. Este aprendizaje permite a los sistemas la capacidad de identificar patrones en datos masivos para hacer predicciones y realizar tareas específicas de forma autónoma. Algunas técnicas de aprendizaje automático son por ejemplo árboles de decisión, Bayes ingenuo, K-vecinos mas cercanos, etc. [41]

2.4 Aprendizaje profundo

El aprendizaje profundo o Deep Learning (DL) son sistemas basados en RNA multicapa, estas redes permiten aprender características de los patrones, que bien pueden ser imágenes completas como patrones de entrada dado el tamaño de las mismas, y estas redes van adaptando características del conjunto de datos de entrenamiento a medida que se van entrenando.

El DL es un subcampo del aprendizaje automático, permite que los modelos computacionales compuestos de múltiples capas de procesamiento aprendan representaciones de datos con múltiples niveles de abstracción. Actualmente es una de las técnicas de aprendizaje artificial más utilizada, ya que emplea un conjunto de algoritmos que trabajan en múltiples capas para aprender representaciones a partir de datos. En una imagen los datos pueden ser representados por medio de un vector de píxeles; al analizar los datos de las imágenes se tendrán que identificar

las características más representativas, algunas características hacen más fácil reconocer objetos sobre la imagen, e. g., en un tablero de ajedrez identificar los caballos.²

En la Figura 2.2 muestra un diagrama de Venn que muestra cómo el aprendizaje profundo es un tipo de aprendizaje, que a su vez es un tipo de aprendizaje automático, que se utiliza para muchos enfoques de la IA.

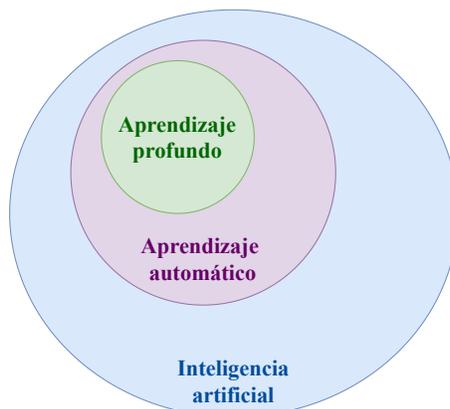


Figura 2.2: Diagrama de Venn que muestra algunas áreas de la inteligencia artificial.

2.5 Redes neuronales artificiales

En los últimos años el enfoque de la red neuronal artificial ha sido ampliamente adoptado; el hardware ha ayudado especialmente en disminuir los tiempos de entrenamiento. La red neuronal tiene ventajas como: es adaptable a diferentes conjuntos de datos, capacidad de generalización para diferentes tipos de imágenes, entre otros aspectos. El perceptrón multicapa es el modelo más popular de red neuronal en la clasificación de imágenes basada en la red neuronal biológica, se divide en *i*) capa de entrada, *ii*) capas ocultas y *iii*) capas de salida. Cuando una red neuronal artificial tiene más de una capa oculta se dice que es una red neuronal profunda, entre mayor sea el número de capas ocultas más profunda será. En la Figura 2.3 se puede ver la representación de la red neuronal artificial modelando las partes de una red neuronal biológica, la capa de entrada contiene cinco neuronas, también cuenta con cinco capas ocultas y finalmente la capa de salida cuenta con tres neuronas [42].

Las conexiones entre neuronas tienen asociado un peso, se realiza una combinación lineal con los valores de entrada de cada neurona y se aplica una función de activación para añadir no linealidad a la red neuronal, en la Subsección 2.6.3 se presentan más detalles de la función de activación. En la Figura 2.4 se muestran los pesos asociados de cada conexión entre neuronas y la combinación lineal entre los pesos y los valores de entrada. Cada neurona realiza esta operación recibiendo los pesos y los valores de entrada, finalmente la neurona devuelve el valor calculado a las neuronas de la siguiente capa conectadas con ella.

²Britannica Academic, s.v. Neural network, accessed October 12, 2020, <https://bidi.uam.mx:6402/levels/collegiate/article/neural-network/126495>.

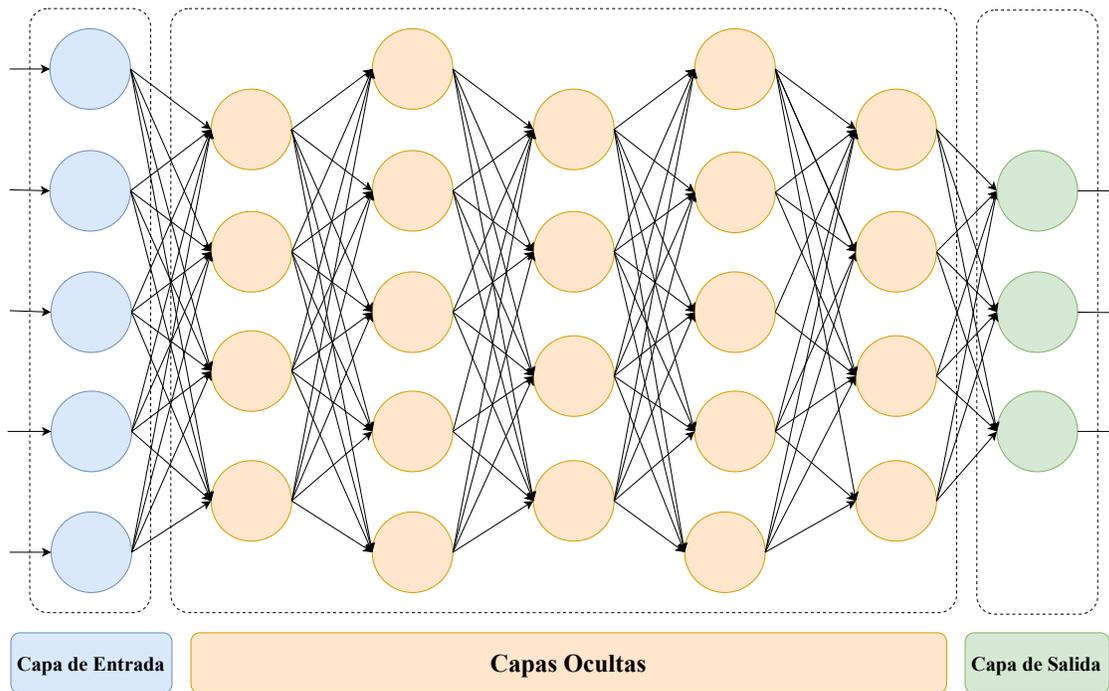


Figura 2.3: Red neuronal artificial clásica.

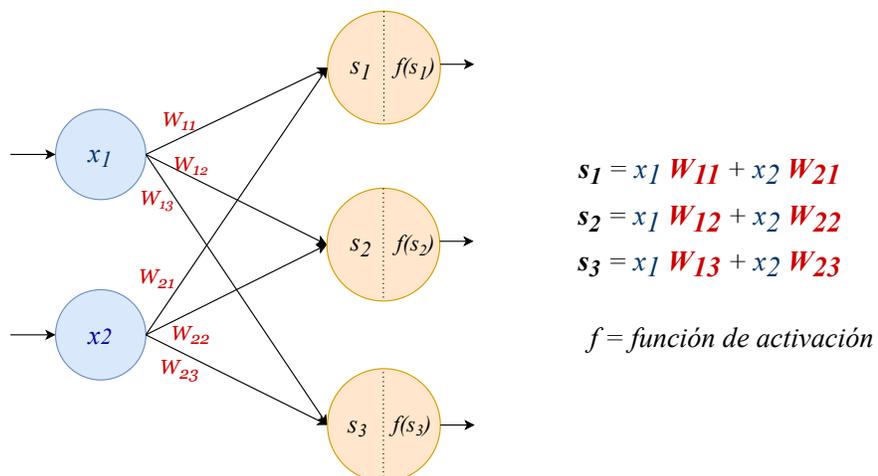


Figura 2.4: Pesos asociados a las conexiones de cada neuronal de la red neuronal artificial.

2.6 Redes neuronales convolucionales

Entre las distintas redes neuronales artificiales se encuentra una que nos interesa en esta propuesta, la red neuronal convolucional (RNC) o Convolutional Neural Network (CNN), este tipo de red neuronal se encuentra en el área de DL por estar diseñada en forma de redes neuronales profundas, compuesta por múltiples capas ocultas y capas de convolución.

Las CNN se inspiraron en la estructura del sistema visual. Al día de hoy se encuentran entre los sistemas con mejores resultados para el reconocimiento de patrones; son utilizados para el procesamiento de imágenes, vídeo, voz y audio. Estas redes han demostrado ser una herramienta posible de implementarse en distintas áreas; en medicina para la segmentación de imágenes de ultrasonido musculoesquelético o para la detección de sangrado gastrointestinal, en la industria automotriz para la identificación y colocación automática de partes. La CNN es una red neuronal profunda, tiene entre sus capas a la operación de convolución que realiza una operación entre las matrices de la imagen y una matriz llamada filtro o kernel, utilizada para obtener las características de imágenes como son borde, enfoque, sombras, entre otras.

En general la CNN es una estructura jerárquica de red neuronal artificial compuesta por los siguientes elementos: entrada, convolución, reducción (pooling), función de activación, una capa totalmente conectada y al final la capa de salida. Las capas de convolución y reducción pueden repetirse varias veces dependiendo del modelo y son utilizadas para obtener las características más representativas de la imagen. La capa totalmente conectada, puede ser una o varias, unifica los resultados para realizar la clasificación. Al final en la capa de salida se muestra que tan probable es que la imagen de entrada pertenezca a una categoría. En la Figura 2.5 se muestra el proceso que sigue la CNN, donde se encuentra la imagen de entrada (Preparación de datos), después por la capa de convolución (Convolución), luego se reduce con pooling (Maxpooling), y pasa por una función de activación como ReLU (Activación). Al final de la figura se encuentra un aplanado de las características extraídas (Aplanado) seguido una capa totalmente conectada con las funciones de dropout y softmax. En las siguientes subsecciones se detallan las partes de la CNN, las cuales son; *a*) Capa de entrada, *b*) Convolución, *c*) Función de activación, *d*) Reducción, *e*) Aplanado, *f*) Capa totalmente conectada, *g*) Dropout, *g*) Softmax, *h*) Capa de salida.

2.6.1 Capa de entrada

Los datos de entrada en una red neuronal convolucional suelen ser vectores bidimensionales de $(M \times N)$ como imágenes, conjunto de señales, matrices, etc. En imágenes cada valor de la matriz representa un píxel, si se trabaja en el espacio RGB serían 3 matrices de $(M \times N)$ teniendo una matriz por canal. En la Figura 2.6 se muestra un ejemplo de una imagen representada por sus matrices en el espacio RGB.

2.6.2 Convolución

Una convolución se puede ver como un producto escalar o también llamado producto punto entre un conjunto de píxeles de una matriz y un filtro también llamado kernel.

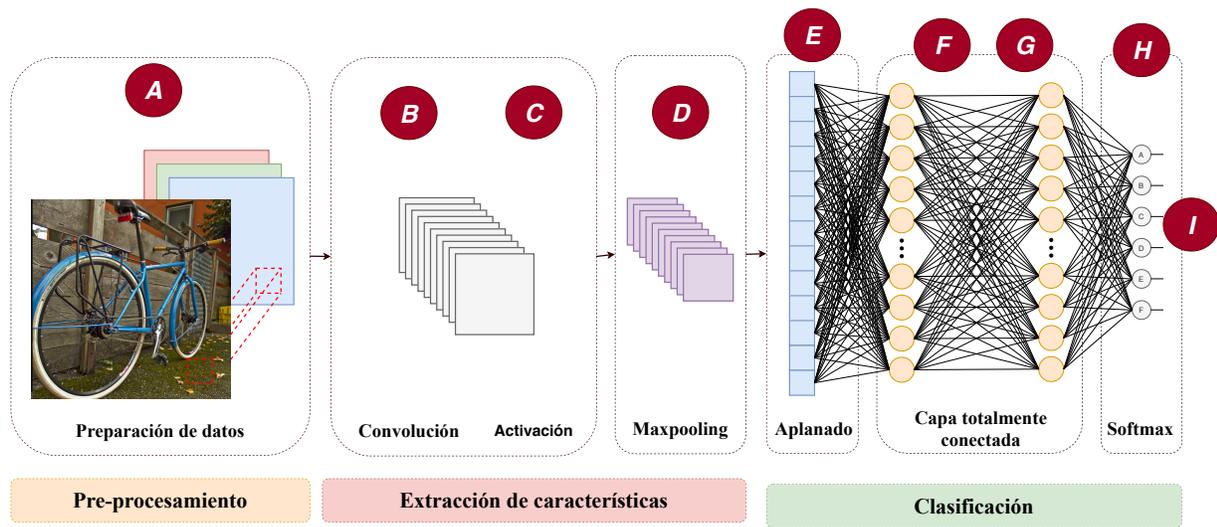


Figura 2.5: Elementos de una CNN.

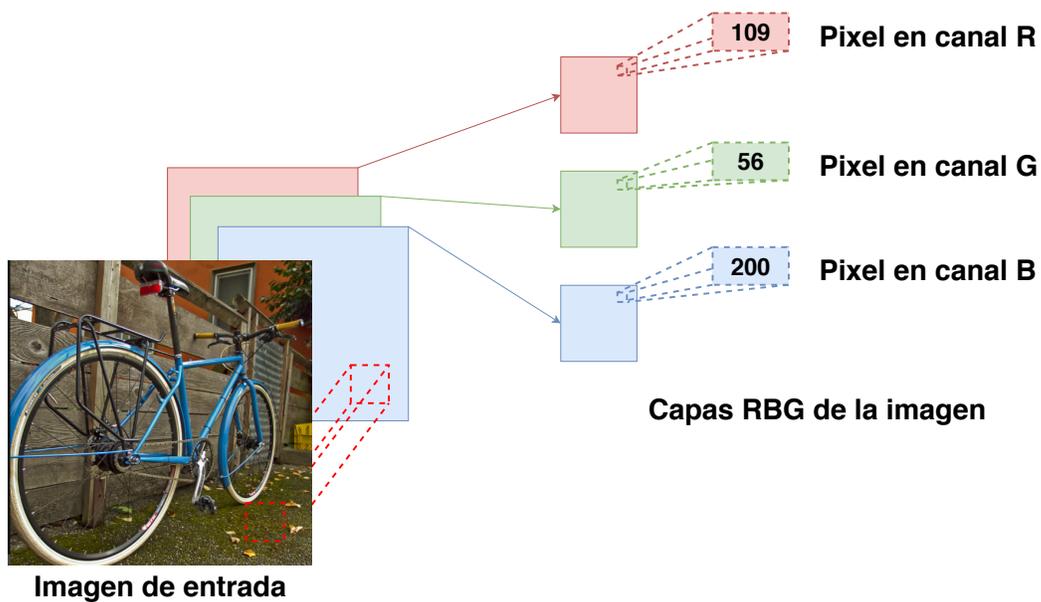


Figura 2.6: Estructura de una imagen en el espacio RGB.

$$C[x, y] = I[x, y] * K[x, y] = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I[m, n] * K[x - m, y - n] \quad (2.1)$$

donde:

- *: Indica la operación de convolución discreta bidimensional
- I : Dato de entrada
- K : Filtro

En la Figura 2.7 se muestra el resultado de aplicar la operación de convolución con tres diferentes filtros a una misma imagen. Al iniciar el entrenamiento de la CNN los filtros se inicializan aleatoriamente con valores numéricos, los valores de los filtros cambian en el transcurso del entrenamiento para adaptarse a las imágenes de entrada.

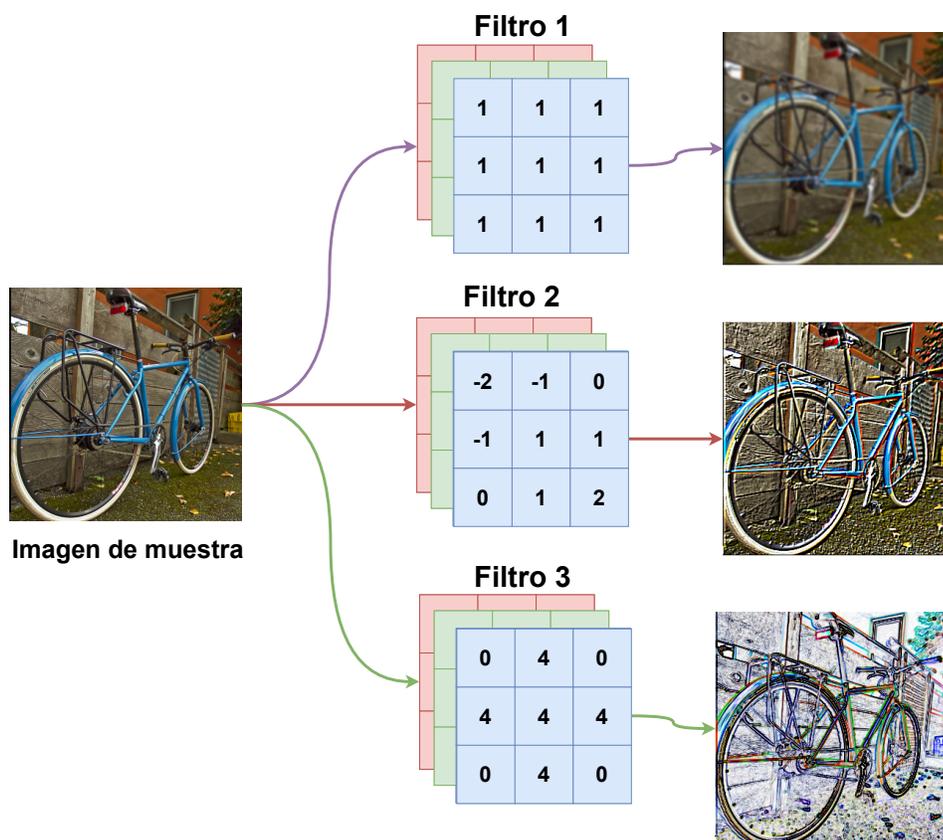


Figura 2.7: Ejemplo de la operación de convolución con diferentes filtros.

2.6.3 Función de activación

La función de activación es un componente clave en el entrenamiento de una CNN, la tarea de una función de activación es la de introducir no linealidad a la red, permitiendo así una aproximación a una función general para una mejor clasificación de los datos. Existen distintas

funciones, siendo las más importantes la sigmoide, la tangente hiperbólica, y la ReLU, las cuales se definen matemáticamente a continuación:

- **Sigmoide:**

$$Y(x) = \frac{1}{1 + e^{-x}} \quad (2.2)$$

- **Tangente hiperbólica:**

$$Y(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \quad (2.3)$$

- **ReLU;**

$$Y(x) = \max(0, x) \quad (2.4)$$

- **ReLU6:**

$$y = \min(\max(x, 0), 6) \quad (2.5)$$

- **SeLU:**

$$Y(x) = \lambda \begin{cases} x & \text{si } x > 0 \\ \alpha(e^x - 1) & \text{si } x \leq 0 \end{cases} \quad (2.6)$$

2.6.4 Reducción

La capa de reducción o pooling (traducción al inglés) tiene por objetivo principal reducir la salida de la capa de convolución, de modo que la cantidad de características también serán reducidas, manteniendo la información mas representativa de la imagen. Así se reduce la resolución de atributos y los hace robustos al ruido y distorsión. Existen diferentes tipos de pooling pero los más utilizados son:

- **Average-pooling:** Obtiene el valor promedio de los elementos j de una región R_i de una imagen $I(x,y)$ de tamaño $N \times N$. La podemos definir como:

$$Y_1(x) = \frac{1}{k \times k} \sum_{i \in [k \times k]} R_i \quad (2.7)$$

Donde:

$$R_i \in I(x, y) \forall \{x, y\} \in \{1 \dots N\}$$

- **Max-pooling:** Obtiene el valor máximo de una región R_i de tamaño $k \times k$ de una imagen $I(x,y)$ de tamaño $N \times N$. La podemos definir como:

$$Y_2(x) = \max(R_i)_{i \in [k \times k]} \quad (2.8)$$

Donde:

$$R_i \in I(x,y) \forall \{x,y\} \in \{1 \dots N\}$$

En la Figura 2.8 se muestra la operación de maxpooling y averagepooling con una región o también llamada mascara de (2×2) . En la Figura 2.9 muestra el efecto de la operación de maxpooling de una imagen.

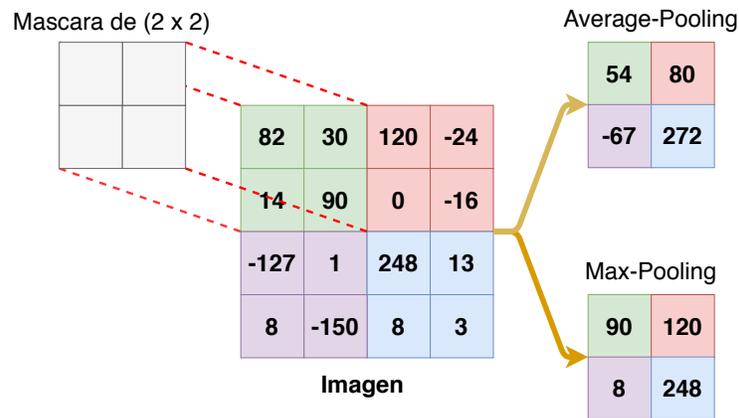


Figura 2.8: Ejemplo de la operación de Pooling con diferentes mecanismos.

2.6.5

Aplanado

En esta etapa se reciben las características extraídas de los datos de entrada llamadas *Mapas de características* donde serán convertidas a un vector unidimensional. El objetivo del aplanado de los mapas características es la de adaptar los datos a una red neuronal clásica totalmente conectada.

2.6.6

Capa totalmente conectada

El vector unidimensional es introducido en cada uno de los nodos de entrada de una red neuronal clásica totalmente conectada, el trabajo principal de la red neuronal clásica totalmente conectada es clasificar los datos de entrada.

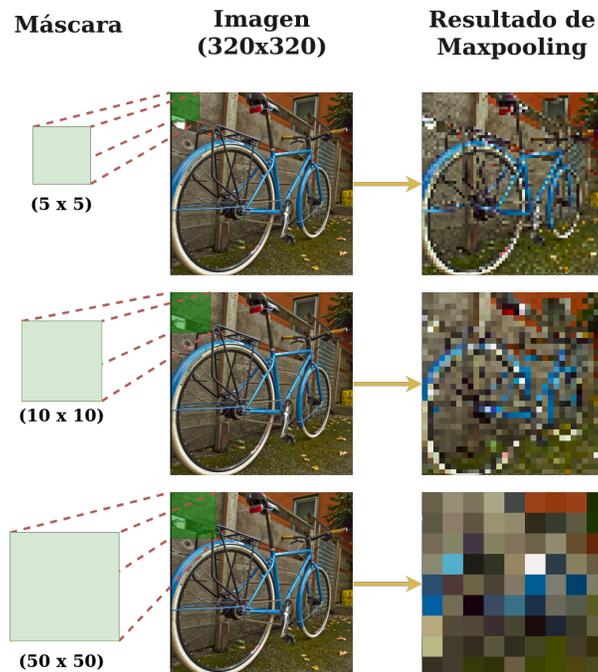


Figura 2.9: Ejemplo de la transformación de la imagen aplicando maxpooling.

2.6.7

Dropout

El mecanismo Dropout tiene por objetivo reducir el sobre-entrenamiento. Desconecta un cierto número de neuronas de forma aleatoria en cada iteración del entrenamiento, así los nodos habilitados ayudan a que la red sea más específica. En la Figura 2.10 se muestran tres casos con diferentes valores del Dropout [58].

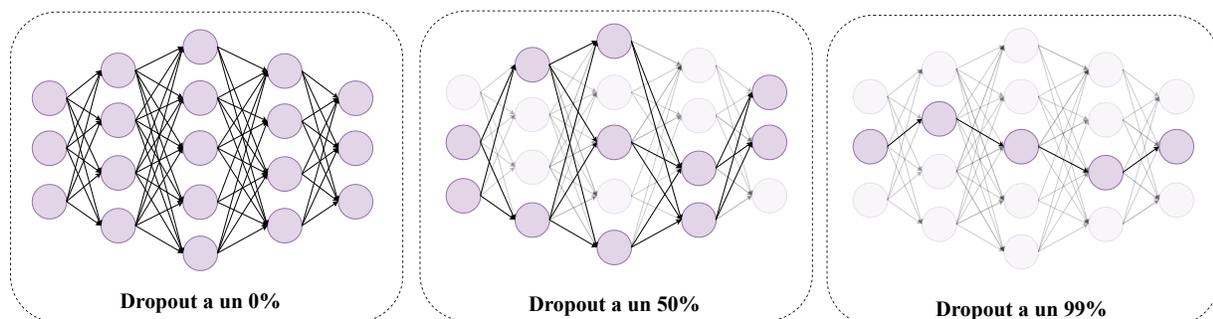


Figura 2.10: Ejemplo de la tarea de Dropout.

2.6.8 Softmax

El mecanismo Softmax es una función que se utiliza generalmente en la última capa totalmente conectada para predecir la categoría en la que pertenece la imagen de entrada con un valor entre 0 y 1. La definición matemática se muestra en la siguiente ecuación:

$$s_j(x) = \frac{e^{x_j}}{\sum_{k=1}^n e^{x_k}} \quad \forall_j = 1 \dots n \quad (2.9)$$

2.6.9 Capa de salida

El trabajo principal de esta capa es mostrar la predicción realizada por la red neuronal profunda, donde muestra las probabilidades de éxito de que la imagen pertenezca a cada clase. Es importante recalcar que pueden ser varios los resultados de salida con diferentes probabilidades de éxito, esto depende de la arquitectura de la CNN.

2.7 Cómputo evolutivo

En el pasado se ha estudiado el comportamiento de la naturaleza, estudios como la teoría evolutiva propuesta por Charles Darwin, el seleccionismo de August Weismann o la genética de Gregor Mendel, son trabajos que han servido para establecer los fundamentos para el desarrollo de técnicas y algoritmos en el área de la computación. El proceso de evolución de las especies fue visto como un mecanismo de aprendizaje mediante el cual la naturaleza dota a las especies de características para hacerlas más aptas para sobrevivir [54].

Así, surgió el cómputo evolutivo (CE) el cual comprende el estudio y la creación de algoritmos, técnicas, procesos y estrategias, basadas en dicho mecanismo de aprendizaje. Lawrence Fogel pionero de la computación evolutiva, la define como una área de las ciencias de la computación que estudia las simulaciones de la evolución natural inspiradas en la teoría del neodarwinismo, los principales enfoques de investigación propuestos son: Los Algoritmos Genéticos (**AG**), Estrategias Evolutivas (**EE**) y la Programación Evolutiva (**PG**) [18].

En los años 1950s Friedberg usó un proceso evolutivo para la resolución de problemas informáticos, esto representó algunos de los primeros trabajos en aprendizaje automático y describió el uso de un algoritmo evolutivo para la programación automática [16]. En la Figura 2.11 se muestra de forma general un diagrama de flujo de un algoritmo evolutivo.

2.8 Algoritmos genéticos

El Dr. David E. Goldberg define los AG como algoritmos de búsqueda basados en los mecanismos biológicos de la selección natural y la genética. [20] Los AG surgen por el trabajo del Dr. John Henry Holland en los años 70s, Holland es considerado como el padre de los AG [57]. El interés de estudiar los AG está dado por su robustez, equilibrio entre eficiencia y eficacia necesaria para la supervivencia, son algoritmos de búsqueda basados en los mecanismos de selección natural y la genética. Los componentes básicos son; selección, cruce y mutación. [20] En la

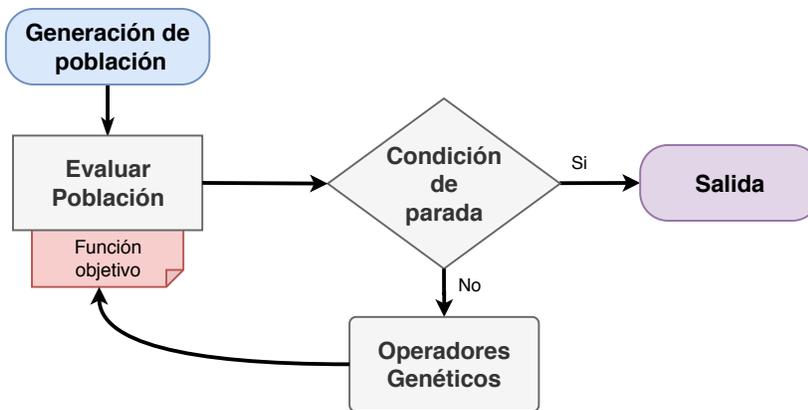


Figura 2.11: Diagrama de flujo de un algoritmo evolutivo general.

Figura 2.12 se muestra el esquema general de un algoritmo genético, a continuación se describe cada parte del esquema.

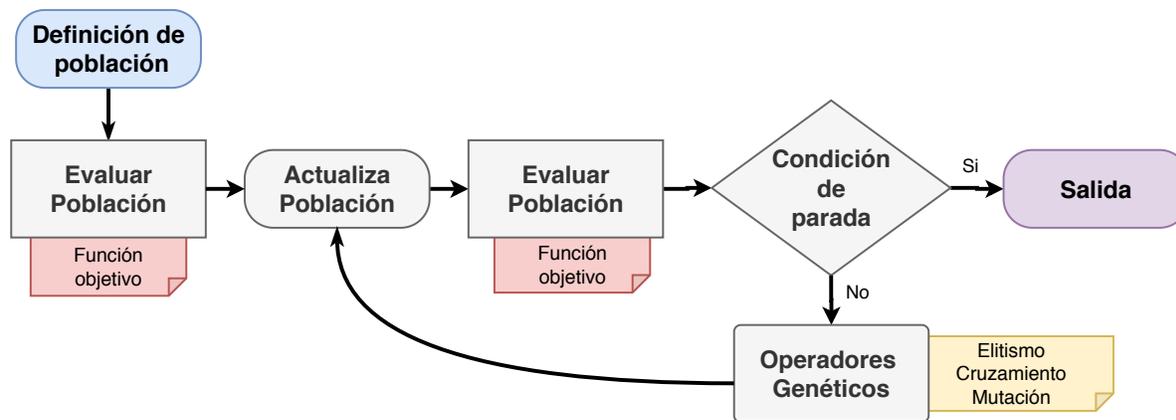


Figura 2.12: Diagrama de flujo de un algoritmo genético clásico.

2.8.1

Componentes de un algoritmo genético

- **Definición de población:** Se genera aleatoriamente una población. La población consiste en un conjunto de individuos que formaran la primera generación a evolucionar.
- **Evaluar población:** El objetivo de evaluar la población es otorgar a cada individuo una aptitud o calidad asociada para la comparación entre ellos, con el fin de saber qué solución es mejor que otra.
- **Operadores genéticos:** Los operadores genéticos son operaciones que se realizan entre individuos de la población. Estas operaciones son: Mutación, Cruza, Elitismo. Estos operadores se describen en la Sección 2.8.2.

- **Actualiza población:** Tiene como objetivo remplazar los individuos de la población con los individuos generados por los operadores genéticos.
- **Condición de parada:** Es un mecanismo que indica al algoritmo cuando parar puede ser un número de iteraciones fijado, alcanzar un error mínimo o alcanzar una eficiencia preestablecida.

2.8.2 Operadores genéticos

Un operador genético es una función empleada en los algoritmos genéticos para mantener la diversidad genética de una población. La variación genética es necesaria para el proceso de evolución. Los operadores genéticos utilizados en los algoritmos genéticos son análogos a aquellos que ocurren en el mundo natural: el *elitismo* es equivalente a la supervivencia del más apto en el mundo natural; la *cruza* equivale a la reproducción sexual y la *mutación* equivale a la mutación biológica. Los operadores genéticos considerados son;

1. **Elitismo:** Este operador consiste en seleccionar de una población inicial un número determinado de individuos considerados los mejores para la siguiente generación. En la Figura 2.13 muestra el mecanismo de este operador genético con la selección de los tres mejores individuos de la población.

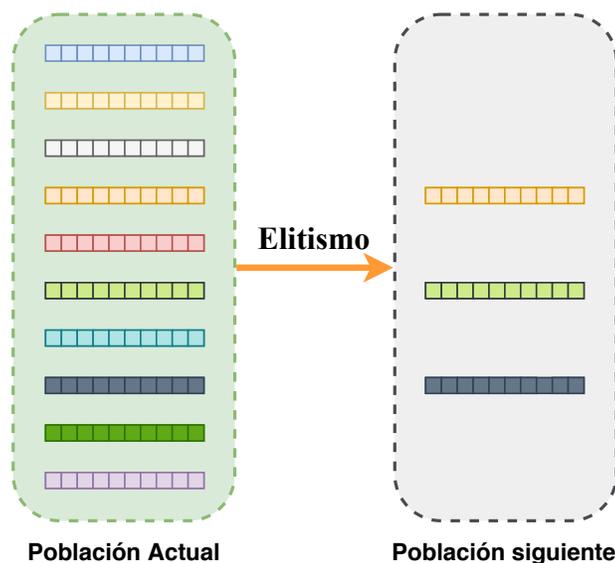


Figura 2.13: Mecanismo del operador elitismo, selecciona los mejores tres individuos.

2. **Cruza:** Este operador consiste en seleccionar dos individuos (padres) de la población de manera aleatoria, ambos tendrán un factor de corte aleatorio que consiste en la selección de un punto de corte, finalmente se combinan las partes de los padres generando dos nuevos individuos (hijos). Los mejores individuos entre padres e hijos se seleccionan para la siguiente generación. En la Figura 2.14 muestra el mecanismo de este operador genético con un factor de corte aleatorio, seleccionando los mejores dos individuos al final.

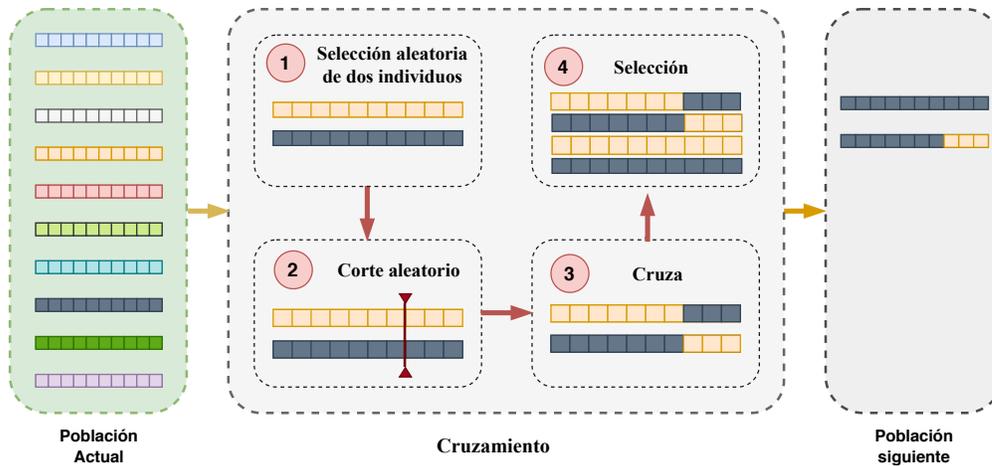


Figura 2.14: Mecanismo del operador de cruce.

3. **Mutación:** Este operador provoca que un individuo de la población pueda variar sus genes de forma aleatoria. El individuo a mutar se selecciona de manera aleatoria. En la Figura 2.15 muestra el mecanismo de este operador genético con un factor de mutación de 10%.

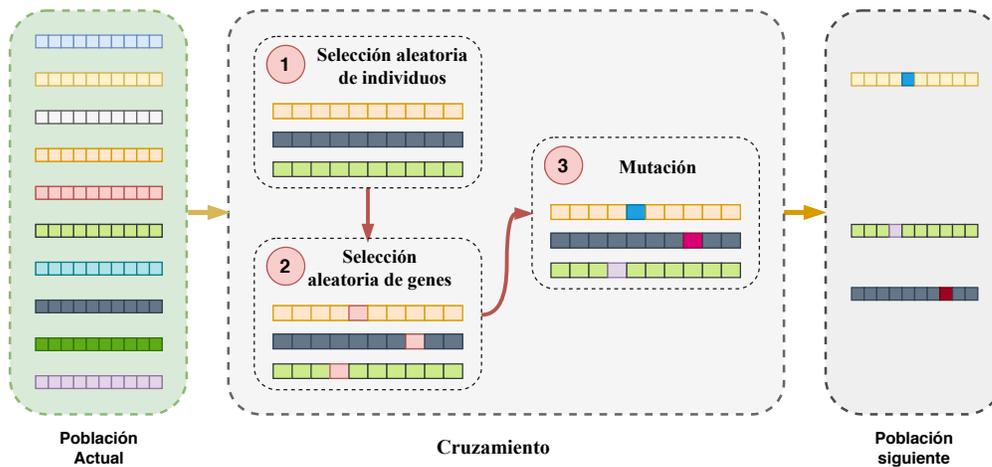


Figura 2.15: Mecanismo del operador de Mutación.

Por lo general, se aplican los operadores genéticos para crear una nueva población. En la Figura 2.16 se muestra una combinación de los operadores genéticos para formar una nueva población, la proporción de individuos generados por cada operador genético se define con constantes llamadas; *porcentaje de mutación*, *porcentaje de cruce* y *porcentaje de elitismo*.

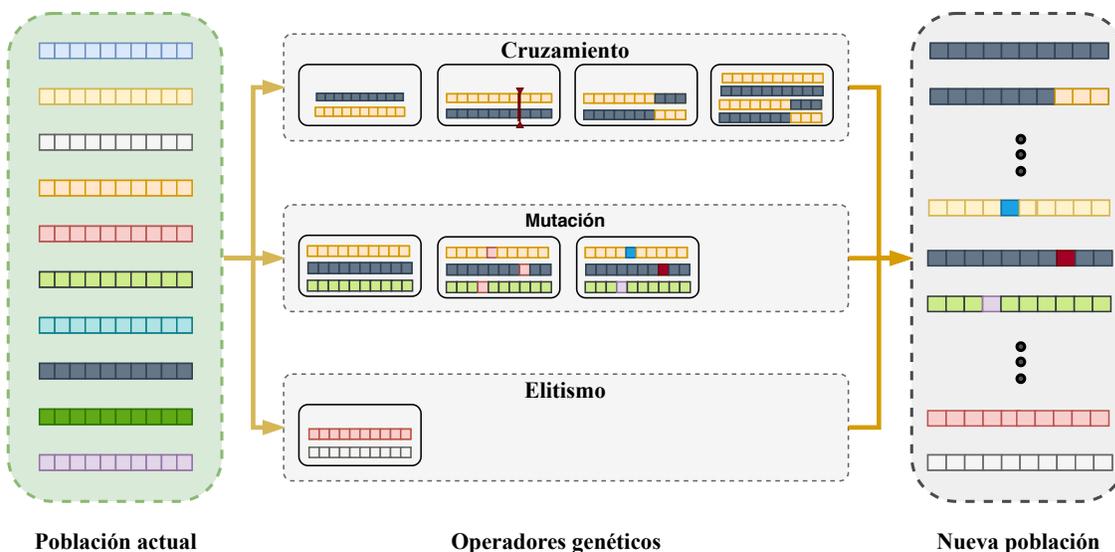


Figura 2.16: Combinación de operadores genéticos.

2.9 Cómputo paralelo

La computación paralela es una técnica de programación en la que muchas instrucciones se ejecutan simultáneamente, se basa en el principio *divide y vencerás* en el que los problemas grandes se pueden dividir en partes más pequeñas [66].

La importancia de la computación paralela radica en el rendimiento de los sistemas de computación, esto es, disminuir los tiempos de ejecución y la capacidad de procesar grandes volúmenes de información de manera efectiva. Además de aprovechar el poder de cómputo de las nuevas tecnologías paralelas [46].

Las diferentes arquitecturas para el cómputo paralelo mas usuales son; los computadores con multiprocesadores, los cluster y las tarjetas gráficas o también llamadas GPUs [48].

2.9.1 Modelos de comunicación

Existen dos principales modelos de comunicación dentro del cómputo paralelo; Memoria compartida y Paso de mensajes.

- **Memoria compartida:** En este modelo todas las tareas comparten una zona de memoria, Existen librerías basadas en este modelo como OpenMP para la programación concurrente.
- **Paso de mensajes:** En este modelo cada tarea corre en paralelo y se comunican mediante el envío y recepción de mensajes. Las tareas se distinguen entre si, mediante un identificador único del sistema. Existen estándares basadas en este modelo, por ejemplo *Message Passing Interface* (MPI), en este protocolo las tareas representadas con procesos que se identifican mediante un entero mayor a cero. En MPI todas los procesos ejecutan el mismo programa [48].

En una comunicación donde las tareas quedan suspendidas hasta que la transferencia de mensaje se ha efectuado completamente se le llama un sistema de **comunicación síncrona**. En un sistema de **comunicación asíncrona** las tareas no quedan suspendidas, no es necesario que las tareas estén presentes. En un sistema con comunicación asíncrono se requiere un sistema de buffer de mensajes entre el emisor y receptor.

2.9.2 Tiempo de ejecución

El tiempo de ejecución de un sistema paralelo puede ser medido de la misma forma que un sistema secuencial. Midiendo el tiempo inicial de la primera tarea ejecutada hasta la finalización de la última tarea del sistema. Así, definamos t_p como el tiempo de ejecución paralela de un algoritmo. Este tiempo esta dividido en dos partes, tales que:

$$t_p = t_{comp} + t_{comm} \quad (2.10)$$

donde:

t_p = tiempo de ejecución paralela.

t_{comp} = tiempo de cómputo.

t_{comm} = tiempo de comunicación.

En un sistema paralelo con granularidad gruesa donde cada tarea realiza más cálculos que comunicaciones, el tiempo de comunicaciones suele despreciarse. ya que, el costo de cálculo es mucho mayor que el costo de comunicación, se desprecia t_{comm} igualándolo a cero. por lo tanto, *el tiempo de ejecución de un programa paralelo con granularidad gruesa es el tiempo en que termina la última tarea del sistema paralelo [66].*

3

ESTADO DEL ARTE

En este capítulo se presenta una recopilación de resultados de investigaciones relacionados con la propuesta principal de este trabajo. El estado del arte muestra un vistazo a lo último realizado con redes neuronales convolucionales aplicadas al reconocimiento de actividades humanas y el reconocimiento de imágenes con diferentes bases de datos.

3.1 Trabajos relacionados

Las CNN se han utilizado en los últimos años para resolver muchos problemas diferentes de inteligencia artificial, proporcionando avances significativos en algunos dominios y conduciendo a resultados de última generación. Sin embargo, las topologías de CNN implican muchos parámetros diferentes y, en la mayoría de los casos, su diseño sigue siendo un proceso manual que implica esfuerzo y una cantidad significativa de prueba y error.

En el trabajo de Baldominos [1], se ha explorado la aplicación de la *neuroevolución* al diseño automático de topologías CNN, introduciendo un marco común para esta tarea y desarrollando dos soluciones novedosas basadas en algoritmos genéticos y evolución gramatical. La propuesta se evaluó utilizando el conjunto de datos MNIST para el reconocimiento de dígitos escritos a mano, logrando resultados competitivos comparados con el estado del arte sin ningún tipo de aumento de datos o preprocesamiento.

Otros enfoques presenta Desell [17] con mejoras en un algoritmo de neuroevolución llamado Exploración evolutiva de topologías convolucionales aumentadas (EXACT), que es capaz de evolucionar la estructura de las redes neuronales convolucionales (CNN). EXACT tiene implementaciones multiprocesador y paralelas. Las mejoras incluyen el desarrollo de un nuevo operador de mutación, que aumentó la tasa de evolución en un orden de magnitud y también demostró ser significativamente más confiable en la generación de nuevas CNN que el método tradicional.

Como vimos, las redes neuronales convolucionales ofrecen un buen rendimiento frente al procesamiento de imágenes. Sin embargo, sus arquitecturas están diseñadas manualmente para diferentes tipos de imágenes. En [44], se presentó un enfoque de evolución de red basado en el algoritmo genético para buscar los genes más adecuados para optimizar las estructuras de red. Los resultados experimentales en imágenes de perfusión de tomografía computarizada demuestran la capacidad del método para seleccionar los genes más aptos para construir redes de alto rendimiento, llamadas EvoNets, los resultados se comparan favorablemente con los métodos más modernos.

En [45] se examinan la evolución de los modelos y tendencias más eficientes en el desarrollo de la arquitectura de redes neuronales convolucionales, que actualmente se utilizan para la clasificación de imágenes. Más precisamente, las características clave de la arquitectura y sus variaciones anuales se revelan en el contexto de la creciente eficiencia de la aplicación práctica de estas redes.

Se han desarrollado técnicas heurísticas como en [60] llamada estrategia de matriz mínima media para determinar los nodos CNN de alto nivel más adecuados para la evaluación de la aptitud. Esta estrategia previa a la evolución determina los nodos comunes de CNN de alto nivel que muestran altos valores de activación para una familia de imágenes que comparten una característica de imagen de interés.

3.2 Red neuronal gruesa-fina

En el estado del arte existen diferentes tipos de arquitecturas de redes neuronales, una nueva estrategia es la de combinar tres diferentes capas de convolución; capa de convolución fina, capa de convolución media y la capa de convolución gruesa propuesta por Avilés [6]. La idea de combinar estas tres capas radica en la extracción de características a tres diferentes nivel de abstracción, para la evaluación de la red neuronal gruesa-fina se utilizaron solamente conjunto de datos para el reconocimiento de actividad humana. En la Figura 3.1 podemos ver la arquitectura de la CNN, en la Sección 4.1 se detalla a profundidad esta red neuronal gruesa-fina.

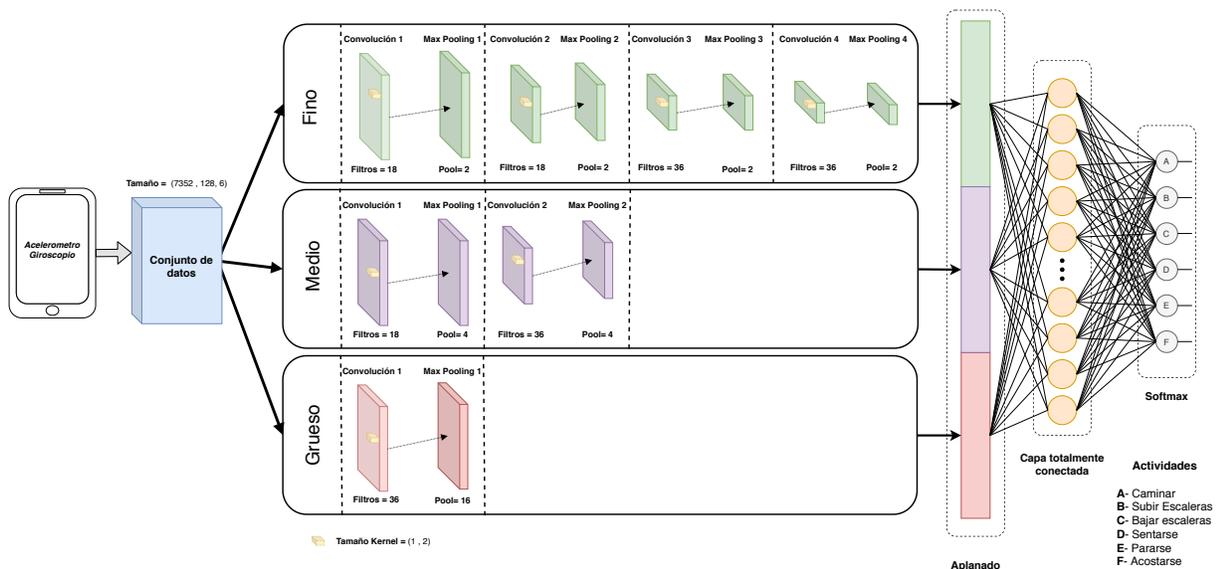


Figura 3.1: Modelo de red neuronal convolucional gruesa-fina. [6]

3.3 Reconocimiento de actividad humana

En la literatura existen dos enfoques principales para tratar la tarea del reconocimiento de actividad humana, del inglés *Human Activity Recognition* (HAR), el primer enfoque se basa en

reconocer diferentes actividades por medio de cámaras de video, Jalal [27] obtiene buenos resultados con este enfoque, utilizando dispositivos de vídeo para reconocer actividades realizadas por un individuo. Sin embargo, el costo de procesar video reduce la eficiencia de los algoritmos por el alto consumo de memoria y cómputo, debido a esto muchos autores utilizan diferentes técnicas capaces de procesar el gran volumen de datos producidos por sensores de video.

Martinez [40] utiliza visión por computadora y otros enfoques para el reconocimiento HAR obteniendo resultados arriba del 90%. El segundo enfoque se basa en la utilización de sensores como giroscopio y acelerómetro colocados en diferentes partes del cuerpo del individuo, trabajando con señales obtenidas de estos sensores. En este trabajo nos centraremos en el segundo enfoque usando diferentes bases de datos obtenidas del estado del arte, las bases de datos seleccionadas contienen señales de sensores de dispositivos móviles.

El área de investigación de HAR ha recibido atención debido a la tendencia creciente de sus aplicaciones en diferentes áreas, la reducción en el precio de los sensores integrados en los dispositivos portátiles y la masificación de su producción [31]. Diferentes artículos realizan estudios sobre características importantes de los sensores; tipo, costo, capacidad, cantidad. sin embargo, en la mayoría de estos se utilizan teléfonos inteligentes debido a la relación calidad-precio y la accesibilidad [59].

El HAR es un problema tratado por diferentes técnicas de la IA, el ML y el DL. Existen métodos basados en ML como máquinas de soporte vectorial o árboles de decisión, estos métodos utilizan características estadísticas como media, mínimo, máximo, desviación estándar, asimetría, curtosis, ángulos, entropía, etc [15, 62].

Un enfoque diferente para la tarea de extracción de características se basa en DL donde se centra en la utilización de Redes Neuronales Convolucionales. La importancia de las CNN radica en el tratamiento de dos puntos principales: (i) La extracción de características más robustas a partir de la implementación de la operación de convolución, y (ii) la posibilidad de trabajar con una mayor cantidad de patrones complejos en longitud o cantidad de atributos, así como el manejo de valores en punto flotante. Existen diversos conjuntos de datos para el HAR que ayudan a medir el rendimiento de los modelos. En este trabajo se ocuparon cuatro conjuntos de datos: UCI HAR [4], WISDM v1.1 [30], WISDM v2.0 [30, 65] y HAPT [51]. Estas bases de datos se detallan a profundidad en el Capítulo 5.

3.3.1 Métodos basados en ML

Existen diferentes trabajos con UCI HAR basados en aprendizaje automático, Anguita [5] mediante máquinas de soporte vectorial logra atacar el problema HAR obteniendo buenos resultados, Xiangbin Zhu [71] utiliza cadenas de markov.

Con UCI HAPT, Taufeeq [61] obtiene excelentes resultados utilizando arboles de decisión. Zheng [70] utiliza máquinas de soporte vectorial combinado un modelo de agrupación dispersa de dos capas.

Con WIDSM 1.1v, Kishor walse [63] utiliza árboles de decisión. Cagatay Catal [9] realiza una

combinación de diferentes técnicas, árboles de decisión, regresión logística y perceptrón multicapa.

Con WISDM 2.0v, Majid Ali Khan Quaid [47] por medio de un algoritmo genético logra alcanzar buenos resultados.

3.3.2 Métodos basados en DL

Existen diferentes trabajos con UCI HAR basados en aprendizaje profundo, Avilés Cruz *et al.* [6] obtienen el 100% de exactitud en la clasificación utilizando una CNN con tres diferentes niveles de alimentación paralela. Cho *et al.* [12] utilizaron el paradigma de divide y vencerás y una CNN para identificar acciones realizadas por humanos. Yong Zhang *et al.* [69] construyeron una CNN capaz de adaptarse a diferentes longitudes de datos.

Con HAPT, Yong Zhang *et al.* [69], es el único trabajo encontrado con enfoque DL, construyeron una CNN capaz de adaptarse a diferentes longitudes de datos.

Con WISDM 1.1v Avilés Cruz *et al.* [6] obtienen el 100% de exactitud en la clasificación utilizando una CNN con tres diferentes niveles de alimentación paralela. Xinxin Han [23] utiliza una CNN con tres niveles de alimentación. Daniele Ravi [49] crea una CNN tradicional probando diferentes capas de convolución.

Con WISDM 2.0v Girmaw Abebe [2] obtiene un buen resultado combinando una CNN con memorias de largo plazo (LSTM). Ignatov [25] utiliza una CNN tradicional con métodos de preprocesamiento.

En las Tablas 3.1 a la 3.4 se presentan trabajos relacionados con las bases de datos UCI HAR, UCI HAPT, WISDM 1.0v y WISDM 2.0v basadas en diferentes técnicas de ML y DL.

Nombre	Año	Exactitud	Enfoque	Método
Avilés Cruz [6]	2019	100.0	DL	CNN
Yong Zhang [69]	2018	98.00	DL	U-CNN
San-Segundo [53]	2016	98.00	ML	Cadena de Márkov
Andrey Ignatov [25]	2018	97.63	DL	CNN
Heeryon Cho [11]	2018	97.62	DL	CNN+Sharpen
Xiangbin Zhu [71]	2016	96.61	ML	Cadena de Márkov
Davide Anguita [4]	2013	96.00	ML	SVM
Charissa Ann Ronao [52]	2016	95.75	DL	CNN+Fourier
Charissa Ann Ronao [52]	2016	94.79	DL	CNN

Tabla 3.1: Comparación de exactitud de UCI HAR [4].

Nombre	Año	Exactitud	Enfoque	Método
Taufeeq [61]	2016	100.0	ML	Random Forrest
Zheng [70]	2018	96.26	ML	TASG+SVM
Bustoni [7]	2019	96.00	ML	Random Forrest
Zheng [70]	2018	95.83	ML	TASG+RNN
Yong Zhang [69]	2018	93.10	DL	U-CNN

Tabla 3.2: Comparación de exactitud de UCI HAPT [51].

Nombre	Año	Exactitud	Enfoque	Método
Avilés cruz [6]	2019	100.0	DL	CNN
Xinxin Han [23]	2020	98.83	DL	CNN+FusionTrial
Yan Xu [67]	2017	98.80	ML	ECDF-PCA-MLCF
Daniele Ravi [49]	2017	98.60	DL	CNN
Mohammad Abu [3]	2016	98.23	DL	CNN
Daniel Ravi [50]	2016	98.20	DL	CNN
Kishor walse [63]	2016	98.09	ML	Random forrest
Yong Zhang [69]	2018	97.00	DL	U-CNN
Haoxi Zhang [68]	2020	96.40	DL	CNN-multiHead
Jeniffer Kwapisz [30]	2011	91.70	ML	Multi Perceptron
Cagatay Catal [9]	2015	91.60	ML	Ensemble learning
Jeniffer Kwapisz [30]	2011	85.10	ML	J48
Jeniffer Kwapisz [30]	2011	78.10	ML	Regresión logística

Tabla 3.3: Comparación de exactitud de WISDM 1.1v [30].

Nombre	Año	Exactitud	Enfoque	Método
Girmaw Abebe [2]	2017	97.90	DL	CNN+LSTM+Handcrafted
Majid Ali Khan Quaid [47]	2019	94.02	ML	Algoritmo Genético
Andrey Ignatov [25]	2018	93.32	DL	CNN
Daniele Ravi [49]	2017	92.70	DL	CNN
Sarbagya Ratna Shakya [56]	2018	92.22	DL	CNN

Tabla 3.4: Comparación de exactitud de WISDM 2.0v [30] [65] [39].

3.4 Reconocimiento de imágenes

Una de las áreas donde los avances han sido más notables es el reconocimiento de imágenes, en parte gracias al desarrollo de nuevas técnicas de DL. En este trabajo se ocupó el conjunto de datos: MNIST que se detalla a profundidad en el Capítulo 5, en la investigación del estado del arte se encontraron un número considerable de trabajos que utilizan Mnist para evaluar modelos enfocados en el reconocimiento de imágenes, sin embargo, solo se presentan los trabajos con buenos resultados.

3.4.1 Métodos basados en ML

En el estado de arte no se han visto métodos basados en ML eficientes para el reconocimientos de imágenes con MNIST. Huseyin [29] muestra varias técnicas de aprendizaje automático con la utilización de la base de datos MNIST, donde K-vecinos más cercanos logró el mejor porcentaje de exactitud con 97.31 %, la maquina de soporte vectorial (SVM) logró un porcentaje de 93.78 %, mientras que los arboles de decisión obtuvieron una exactitud de 94.82 %.

3.4.2 Métodos basados en DL

Con la base de datos MNIST [32], Li Wan [64] por medio de una CNN y un enfoque Dropconnect obtiene un buen resultado. Dan Claudiu Cireşan [13] construye una red neuronal multicolumna. Ming Liang [36] por medio de una red neuronal recurrente logra mejores resultados que una CNN clásica. En la Tabla 3.5 muestra una comparación de distintos métodos basados en DL y ML.

Nombre	Año	Exactitud	Enfoque	Método
Li Wan [64]	2013	99.79	DL	CNN Dropconnect
Dan Claudiu Cireşan [13]	2012	99.77	DL	CNN Multi-Column
Ikuro Sato [55]	2015	99.77	DL	CNN APAC
Jian Ren Chang [10]	2015	99.76	DL	CNN Batch
Chen Yu Lee [35]	2015	99.71	DL	CNN Pooling fuction
Ming Liang [36]	2015	99.69	DL	Recurrent CNN
Zhibin Liao [38]	2015	99.69	DL	Normalisation CNN
Benjamin Graham [22]	2014	99.68	DL	CNN Fraccional
Zhibin Liao [37]	2015	99.67	DL	CNN Multi-scale
Dan Claudiu Cireşan [14]	2010	99.65	DL	Big Deep CNN
Huseyin [29]	2019	97.31	ML	K-NN
Huseyin [29]	2019	94.82	ML	Random forest
Huseyin [29]	2019	93.78	ML	SVM

Tabla 3.5: Comparación de exactitud de MNIST [32].

Parte III

PROPUESTAS

El trabajo duro es un talento. La capacidad de seguir intentándolo cuando otros abandonan es un talento.

GARRY KASPAROV

4

PROPUESTA PRINCIPAL

En este capítulo se describe a detalle el funcionamiento de la red neuronal profunda gruesa-fina, que sirve de punto de partida para nuestra propuesta; se describen los componentes de esta red neuronal profunda, los parámetros utilizados y detalles técnicos. También, se presentan los detalles de la una propuesta evolutiva en dos partes: el algoritmo genético en ejecución secuencial y el algoritmo genético en ejecución paralela. Se muestran los esquemas del funcionamiento general y los detalles técnicos de la implementación del algoritmo genético, para la optimización de parámetros de la red neuronal gruesa-fina.

4.1 Ajuste de la red neuronal gruesa-fina

La idea es implementar una CNN propuesta por Avilés [6] con diferentes niveles de extracción paralela de características para la clasificación y reconocimiento de imágenes, estos niveles de extracción están divididos en grueso, medio y fino. En la Figura 4.1 se muestra la arquitectura de la red neuronal, la idea principal está basada en extraer diferentes niveles de características, donde el nivel fino extrae características más sutiles de los datos, el nivel medio extrae características más robustas y el nivel grueso extrae características más burdas como líneas, círculos, colores, etc. La arquitectura presentada en la Figura 4.1 consta de tres niveles:

1. **Fino:** Está compuesto por cuatro capas de convolución, cada capa de convolución es seguida por una capa de agrupación máxima, los mapas de características son reducidos en cada capa. Las primeras dos capas de convolución cuentan con 18 filtros y las últimas dos capas cuentan con 36 filtros, el tamaño del filtro es de (1×2) para todas las capas de convolución. La ventana de agrupación máxima es de (1×2) y el paso es de 2.
2. **Medio:** Está compuesto por dos capas de convolución, cada capa de convolución es seguida de una capa de agrupación máxima, los mapas de características son reducidos en cada capa. La primera capa de convolución cuenta con 18 filtros y la última cuenta con 36 filtros, el tamaño del filtro es de (1×2) para todas las capas de convolución. La ventana de agrupación máxima es de (1×4) y el paso es de 4.
3. **Grueso:** Está compuesto por una capa de convolución, la capa de convolución es seguida por una capa de agrupación máxima. la capa de convolución cuenta con 36 filtros de tamaño (1×2) . La ventana de agrupación máxima es de (1×16) y el paso es de 16.

La salida de los tres niveles se agrupan y se aplanan, formando un vector que pasa a una etapa de clasificación por medio de una red neuronal totalmente conectada. Finalmente la salida

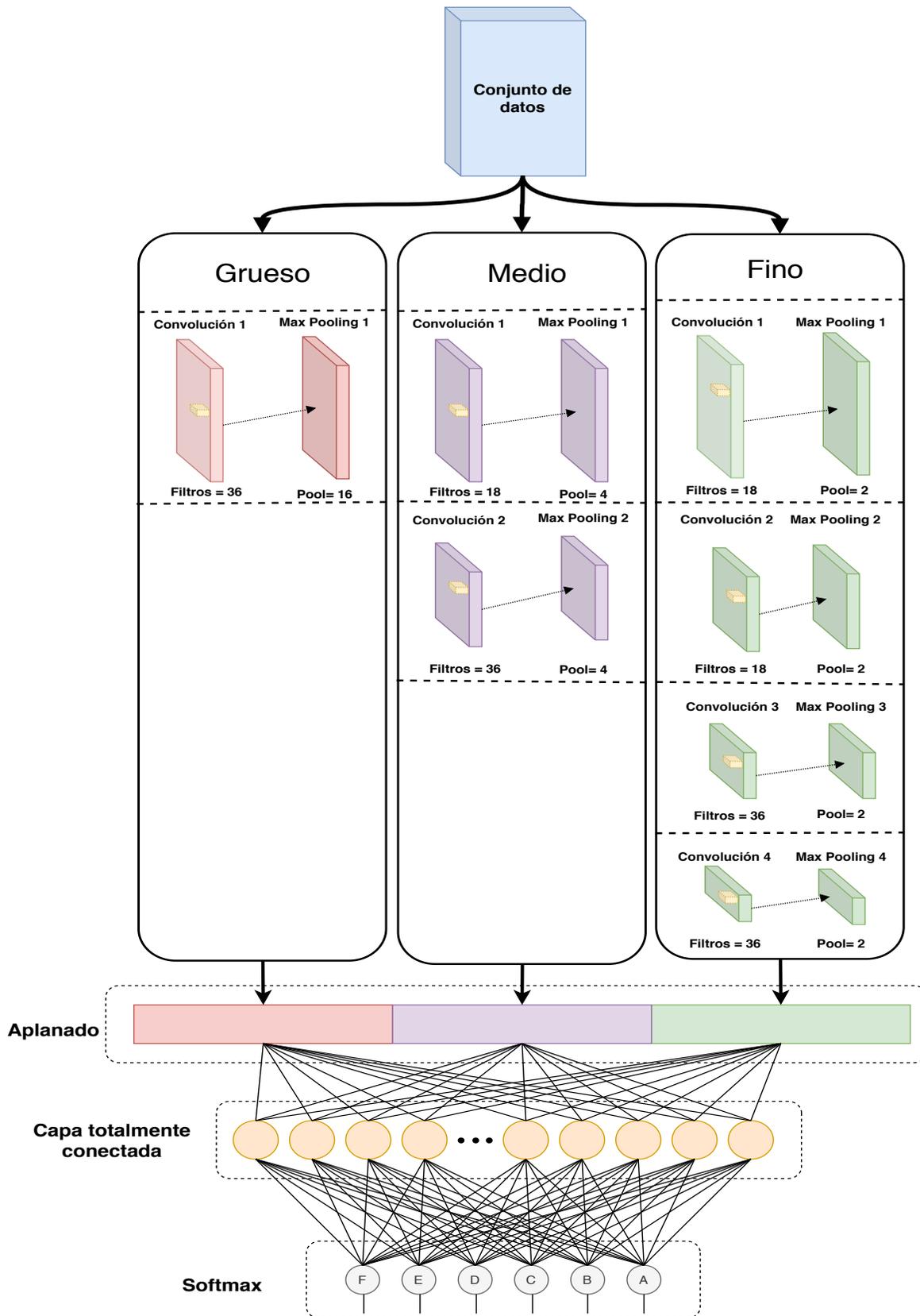


Figura 4.1: Modelo de red neuronal convolucional gruesa-fina.

de la capa totalmente conectada se pasa a una capa softmax que calcula la distribución de probabilidad de las clases [21]. En la Tabla 4.1 se muestran los parámetros utilizados por la red neuronal gruesa-fina, la selección de estos parámetros están basados en el trabajo realizado por Avilés [6].

Parámetros	Valor	Tipo
Número de épocas	500	Entero
Tamaño de lote	300	Entero
Dropout	0.8	Flotante
Función activación	ReLU	Función
Función perdida	Entropía cruzada	Función
Número de filtros	18,36	Entero
Dimensión filtro (fila)	1	Entero
Dimensión filtro (columna)	2	Entero
Dimensión maxpooling (fila)	1	Entero
Dimensión maxpooling (columna)	2,4,16	Entero
Paso de maxpooling	2,4,16	Entero
Optimizador	ADADELTA	Función

Tabla 4.1: Parámetros pre-establecidos para la red neuronal profunda gruesa-fina.

Los ajustes realizados a la red neuronal gruesa-fina hacia la clasificación y reconocimiento de patrones se describen a continuación:

1. **Canales:** Reducción del número de canales en el procesamiento de las imágenes, la propuesta original cuenta con seis canales para conjunto de datos para el reconocimiento de actividad humana basadas en estampas de tiempo, sin embargo, para las imágenes fue necesario una reducción a tres canales para conjuntos de datos para el reconocimiento de imágenes en el espacio RGB o un solo canal para imágenes en escala de grises.
2. **Ajuste de los niveles de extracción de características:** El número de niveles de extracción de características depende del tamaño de los datos, esto implica que si el tamaño de los datos es muy grande la red neuronal gruesa-fina aumenta los niveles y las capas de convolución en cada nivel, si el tamaño de los datos es pequeño se reduce el tamaño de los niveles.

Como veremos en el siguiente capítulo, proponemos una implementación de optimización de los parámetros sobre la estructura de la red neuronal propuesta gruesa-fina, a tal modo de mejorarla para diferentes propósitos con base a un algoritmo evolutivo, que para nuestro caso funcionó muy bien un algoritmo genético, y posteriormente se presenta la implementación paralela de toda la propuesta.

4.2 Algoritmo genético secuencial

La Figura 4.2 muestra a grandes rasgos cómo está conformada la propuesta, se trata de un algoritmo genético con una red neuronal profunda gruesa-fina. La idea principal de tal combinación es la optimización de los parámetros de la red neuronal gruesa-fina. Una de las dificultades de entrenar una red neuronal profunda es la de calibrar los parámetros iniciales de la red por el método de prueba y error. Algunos de los parámetros principales de la red neuronal convolucional son la tasa de aprendizaje, dropout, número filtros, dimensión de filtros, etc. A continuación se describe el proceso de optimización y cómo se plantea.¹

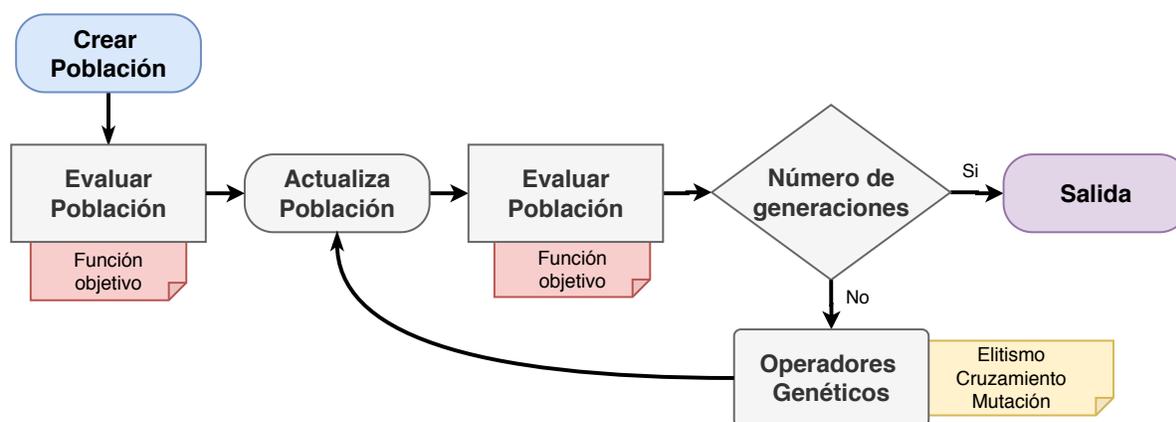


Figura 4.2: Metodología del algoritmo genético con una red neuronal gruesa-fina.

Como sabemos las ventajas de un algoritmo evolutivo es el equilibrio de la eficiencia y eficacia, y su robustez. El algoritmo genético es un algoritmo de búsqueda inspirado en la selección natural y en la genética.

En el listado 4.1, se presenta la generalidad del algoritmo genético, y sus partes. Los parámetros de entrada del algoritmo genético están declarados por el símbolo σ_i , donde σ_1 es el número de individuos permitidos en una población generada, σ_2 es el factor de mutación, σ_3 es el porcentaje de individuos de la población creados por el operador de mutación, σ_4 es el porcentaje de individuos de la población creados por el operador de cruce y finalmente σ_5 es el número de generaciones.

¹Britannica Academic, s.v. Genetic algorithm, accessed October 12, 2020, <https://bidi.uam.mx:6402/levels/collegiate/article/genetic-algorithm/476767>.

Algoritmo 4.1 Algoritmo genético**Entrada:** Parámetros evolutivos $K = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$.

```

1: generacion = 0
2: poblacion = construyePoblacion(  $\sigma_1$  )
3: fitness = evalua(poblacion)
4: mejor = dameMejor(poblacion, fitness)
5: mientras generacion  $\leq \sigma_5$  &&  $fitness[mejor] < 100$  hacer
6:   E = elitismo( poblacion )
7:   C = cruza( poblacion,  $\sigma_4$  )
8:   M = mutacion( poblacion,  $\sigma_2, \sigma_3$  )
9:   poblacion = E  $\cup$  C  $\cup$  M
10:  fitness = evalua(poblacion)
11:  mejor = dameMejor(poblacion, fitness)
12:  generacion = generacion + 1
13: fin mientras
14: devolver elitismo( poblacion )

```

En la línea 1 se define un contador para registrar el número de generación actual. En la línea 2 el método *construyePoblacion()* construye una lista de individuos donde serán guardados en la variable población. En la línea 3 el método *evalua(poblacion)* devuelve una lista de las aptitudes asociadas a cada individuo de la población. En la línea 4 el método *dameMejor(poblacion, fitness)* regresa el índice del mejor individuo de la población actual. En la línea 5 se declara una estructura de control donde se repetirá la acción mientras no se hayan completado todas la generaciones definidas en σ_5 y la aptitud del mejor individuo sea menor que el 100 %. En las líneas 6, 7 y 8 representan los operadores genéticos elitismo, cruza y mutación, donde éstos regresan un conjunto de individuos para formar la nueva población. En la línea 9 se realiza la unión de los individuos generados por cada operador genético. En la línea 12 se devuelve al mejor individuo encontrado a lo largo de todas las generaciones.

4.2.1 Definición del individuo

El individuo del algoritmo genético es una colección de números enteros normalizados, los cuales representan los valores de los parámetros de la red neuronal gruesa-fina, en la ecuación 4.1 se define la estructura del individuo, una tupla de n elementos ordenados.

$$(x_1, x_2, x_3, \dots, x_n) \quad x_{1,2,\dots,n} \in [1, \sigma] \quad (4.1)$$

Donde $(x_1, x_2, x_3, \dots, x_n)$ representa la estructura del individuo, x_i representa el valor normalizado de un parámetro de la red neuronal gruesa-fina, σ es una constante entera que representa la escala de la normalización.

El tamaño de los individuos definidos para este trabajo será considerando, $n = 9$, $\sigma = 100$. El individuo esta formado por 9 genes, en la Figura 4.3 muestra la estructura del individuo y en la Tabla 4.2 se muestran los parámetros con su respectivo dominio en una red neuronal artificial.

Ejemplo de un individuo con los siguientes parámetros $n = 9$ y $\sigma = 100$



Figura 4.3: Estructura final del individuo propuesto.

Gen	Parámetros	Limite
x_1	Número de épocas	(1,50)
x_2	Tamaño de lote	(100,500)
x_3	Dropout	(0,0.99)
x_4	Función activación	0: Ninguno 1: Tanh 2: ReLU 3: Selu 4: Sigmoid 5: ReLU6
x_5	Convoluciones	(1,30)
x_6	Dimensión filtro	(1,10)
x_7	Dimensión Maxpooling	(1,10)
x_8	Optimizador	0: Adagrad 1: RMSprop 2: Adadelta 3: Adamax 4: Nadam 5: Adam 6: SGD
x_9	Tasa de aprendizaje	(0.5, 0.0005)

Tabla 4.2: Estructura propuesta del individuo.

$$(x_1, x_2, x_3 \dots, x_9) = (35, 20, 15, 8, 99, 50, 1, 40, 32) \quad (4.2)$$

donde:

- $x_1 = 35 =$ valor normalizado para **Número de épocas**
- $x_2 = 20 =$ valor normalizado para **Tamaño de lote**
- $x_3 = 15 =$ valor normalizado para **Dropout**
- $x_4 = 08 =$ valor normalizado para **Función de activación**
- $x_5 = 99 =$ valor normalizado para **Convoluciones**
- $x_6 = 50 =$ valor normalizado para **Dimensión filtro**
- $x_7 = 01 =$ valor normalizado para **Dimensión Maxpooling**
- $x_8 = 40 =$ valor normalizado para **Optimizador**
- $x_9 = 32 =$ valor normalizado para **Tasa de aprendizaje**

Note que el individuo con los valores normalizados x_i es utilizado por el algoritmo genético, sin embargo, para ser utilizado por la red neuronal artificial debe de ser transformado a sus valores de dominio mostrados en la Tabla 4.2. A continuación se explica la transformación de cambio de escala por cada parámetro.

Dado un individuo I

$$I = (x_1, x_2, x_3, \dots, x_n) \quad x_{1,2,\dots,n} \in [1, \sigma] \quad (4.3)$$

Definimos una función para el cambio de escala de dos conjuntos

$$f : [1, \sigma] \mapsto [\gamma, \beta]$$

donde $[\gamma, \beta]$ son los dominios definidos en la Tabla 4.2. Entonces para cada x_i del individuo definimos la función $f(x_i)$.

$$p_i = f(x_i) = \frac{(x_i - 1)(\beta - \gamma)}{(\sigma - 1)} + \gamma \quad (4.4)$$

Por ejemplo, si tomamos el parámetro dropout donde los valores están dentro de intervalo $[0,1)$, entonces para x_3 realizamos las siguientes operaciones:

$$f : [1, 100] \mapsto [0, 0.99]$$

$$p_3 = f(x_3) = \frac{(x_3 - 1)(0.99 - 0)}{(100 - 1)} + 0$$

$$p_3 = f(x_3) = \frac{(x_3 - 1)(0.99)}{99}$$

por lo tanto si el algoritmo genético calcula, $x_3 = 15$ entonces $f(15) = 0.14$ para el dropout en la red neuronal artificial.

Un individuo representa un conjunto de parámetros de la red neuronal gruesa-fina, esto significa que los individuos generados por el algoritmo genético cambiarán la estructura y el comportamiento de la red neuronal gruesa-fina. En la Figura 4.4 se observan las partes de la red neuronal gruesa-fina que son modificadas por el algoritmo genético.

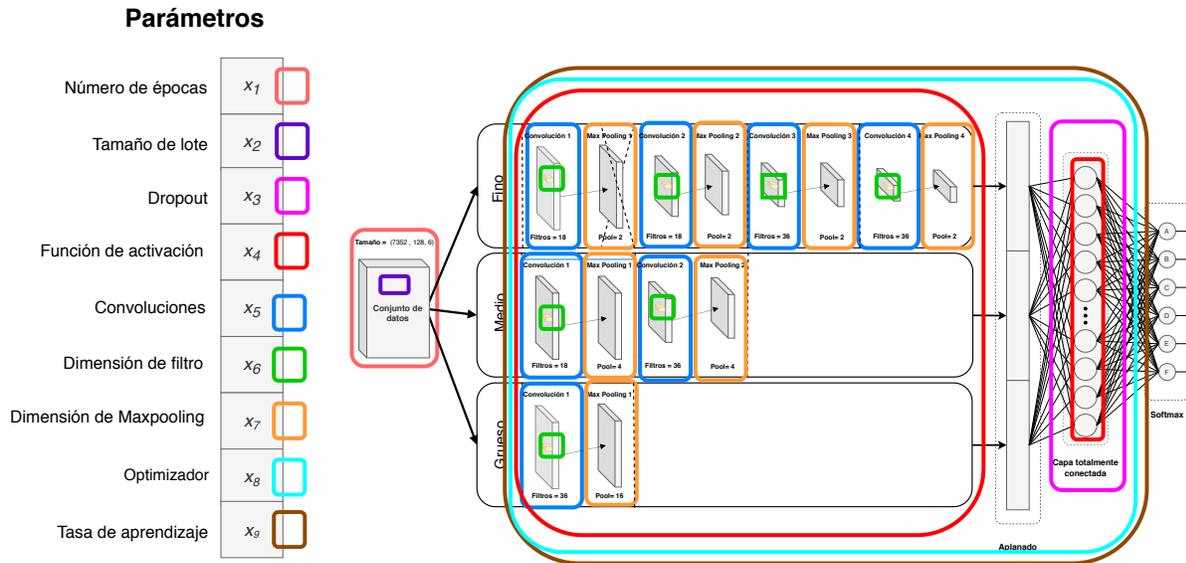


Figura 4.4: Ejemplo de la estructura propuesta del individuo a evolucionar y los parámetros que involucra la red neuronal.

4.2.2 Operadores genéticos

Los operadores genéticos definidos en la Sección 2.8.2 son considerados en este trabajo, con las siguientes características;

1. **Elitismo:** Este operador sólo considera al mejor individuo de la población, se considera una selección con repetición.
2. **Cruza:** Los individuos a cruzar se seleccionan de manera aleatoria de la población, de la misma forma, el punto de cruce se selecciona de manera aleatoria en los individuos, se considera una selección con repetición.
3. **Mutación:** El individuo a mutar se selecciona de manera aleatoria de la población y sufre una variación aleatoria de algunos de sus genes con base al factor de mutación, de esta forma la mutación proporciona diversificación en la generación de individuos. En este trabajo se considera una selección con repetición.

En la Figura 4.5 se muestran los porcentajes de individuos generados por cada operador genético. Donde elitismo tiene el 10 %, cruza el 30 % y mutación el 60 %.

4.2.3 Función de aptitud propuesta

La función de aptitud o exactitud considerada es el porcentaje de recuperación o reconocimiento, como se presenta en la Ecuación 4.5. La red neuronal gruesa-fina funge como función objetivo del algoritmo genético.

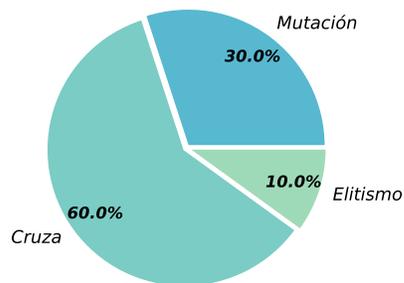


Figura 4.5: Porcentaje de individuos generados por los operadores genéticos para una nueva población.

$$Aptitud = \frac{VP + VN}{VP + FP + FN + VN} \quad (4.5)$$

donde:

- VP = Verdaderos positivos.
- VN = Verdaderos negativos.
- FP = Falsos positivos.
- FN = Falsos negativos.

4.2.4

Parámetros del algoritmo genético

Los parámetros del algoritmo genético considerados son los siguientes; *i*) Número de generaciones, *ii*) Tamaño de población, *iii*) Porcentaje de cruza, *iv*) Porcentaje de mutación, *v*) Factor de mutación y *vi*) Función de aptitud para medir la aproximación de mejor individuo.

4.3 Algoritmo genético paralelo

Uno de los problemas principales de implementar un algoritmo genético integrando una RNC es el tiempo de ejecución, es por esto que se propone una versión paralela, en la Figura 4.6 mostramos la arquitectura del modelo paralelo. Los componentes genéticos para la versión del modelo paralelo son iguales a los utilizados en el modelo secuencial de la sección 4.2. Estos componentes son; definición del individuo, operadores genéticos, función de aptitud, parámetros del algoritmo genético. La diferencia radica en el comportamiento y ejecución del algoritmo genético cuando hay mas de un nodo.

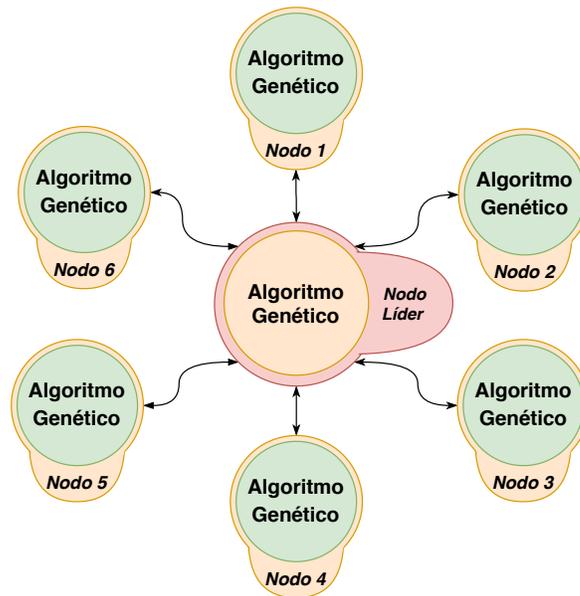


Figura 4.6: Algoritmo genético paralelo.

El sistema paralelo consta de un conjunto de nodos siguiendo la Interfaz de Paso de Mensajes, contiene un nodo líder y un conjunto de nodos secundarios. El nodo líder es el encargado de organizar y de emitir el mejor individuo de cada generación. Los nodos secundarios tienen el objetivo de ejecutar su propio algoritmo genético y cada generación tendrán la tarea de compartir con el líder el mejor individuo. observar que cada uno de los nodos funciona como la versión secuencial de la sección 4.2. La propuesta es un modelo de islas síncrono con topología de estrella. [8]

El Algoritmo 4.2 propuesto muestra el funcionamiento general de los nodos. Todos los nodos ejecutarán el algoritmo genético paralelo, cuentan con un identificador único mayor a cero. Observar que el sistema es considerado un sistema síncrono, ya que los nodos secundarios esperan el mejor individuo global en cada generación.

Los parámetros de entrada del algoritmo 4.2 están declarados por el símbolo σ_i , donde σ_1 es el número de individuos permitidos en una población generada, σ_2 es el factor de mutación, σ_3 es el porcentaje de individuos de la población creados por el operador de mutación, σ_4 es el porcentaje de individuos de la población creados por el operador de cruce y finalmente σ_5 es el número de generaciones.

Algoritmo 4.2 Algoritmo genético paralelo en el nodo *id***Entrada:** Parámetros evolutivos $K = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$.

```

1: generacion = 0
2: poblacion = construyePoblacion(  $\sigma_1$  )
3: fitness = evalua(poblacion)
4: mejorLocal = dameMejor(poblacion, fitness)
5: mientras generacion  $\leq \sigma_5$  && fitness[mejorLocal] < 100 hacer
6:   E = elitismo( poblacion )
7:   C = cruza( poblacion,  $\sigma_4$ )
8:   M = mutacion( poblacion,  $\sigma_2, \sigma_3$  )
9:   poblacion = E  $\cup$  C  $\cup$  M
10:  mejorLocal = elitismo( poblacion )
11:  mejores = gather( mejorLocal, lider )
12:  si id == lider entonces
13:    mejor = elitismo( mejores )
14:  fin si
15:  mejor = Broadcast(mejor, todos)
16:  poblacion = poblacion  $\cup$  mejor
17:  fitness = evalua(poblacion)
18:  mejorLocal = dameMejor(poblacion, fitness)
19:  generacion = generacion + 1
20: fin mientras
21: devolver elitismo( poblacion )

```

En la línea 1 se define un contador para registrar el número de generación actual. En la línea 2 el método *construyePoblacion*(σ_1) construye una lista de individuos donde serán guardados en la variable población. En la línea 3 el método *evalua*(*poblacion*) devuelve una lista de las aptitudes asociadas a cada individuo de la población.

En la línea 4 el método *dameMejor*(*poblacion*, *fitness*) regresa el índice del mejor individuo de la población actual. En la línea 5 se declara una estructura de control donde se repetirá la acción mientras no se hayan completado todas la generaciones definidas en σ_5 y la aptitud del mejor individuo sea menor que el 100%. En las líneas 6, 7 y 8 representan los operadores genéticos elitismo, cruza y mutación, donde éstos regresan un conjunto de individuos para formar la nueva población.

En la línea 9 se realiza la unión de los individuos generados por cada operador genético. En las líneas 10-15 se realiza la comunicación de todos los nodos, notemos los mecanismos de comunicación por paso de mensajes en las líneas 11 y 15, donde se utiliza el mecanismo *recolectar*() y *transmitir*(), en el protocolo MPI es *gather*() y *broadcast*()).

En la línea 11 el nodo líder recolecta al mejor individuo local de cada nodo del modelo, después en la línea 13 el nodo líder selecciona al mejor individuo global. En la línea 15 el nodo líder transmite el mejor individuo global a todos los otros nodos. En las líneas 16-19 se actualizan los datos para la siguiente generación. En la línea 21 se devuelve al mejor individuo encontrado a lo largo de todas las generaciones.

5

BASES DE DATOS SELECCIONADAS

En este capítulo se describen las bases de datos utilizadas para la evaluación y comparación de los modelos propuestos. Estas bases de datos son divididas según su propósito, para el reconocimiento de actividad humana y para el reconocimiento de imágenes. Para el reconocimiento de actividad humana se consideran; UCI HAR [4], WISDM v1.1 [30], WISDM v2.0 [30] y HAPT [51]. Para el reconocimiento de imágenes se considera MNIST [32].

5.1 UCI HAR

UCI HAR es un conjunto de señales preprocesadas de seis distintas actividades humanas: caminar, subir escaleras, bajar escaleras, sentarse, pararse, acostarse. Estas señales fueron tomadas con un teléfono inteligente colocado en la cintura de treinta voluntarios, se tomaron muestras del giroscopio y acelerómetro integrados en el teléfono con una velocidad de 50 Hz. Las señales del giroscopio y del acelerómetro están compuestas por tres ejes, (X, Y, Z). El conjunto de datos HAR cuenta con un conjunto de entrenamiento de 7,352 (71.3%) instancias y con un conjunto de prueba de 2,947 (28.7%) instancias, cada instancia tiene 128 atributos y 6 canales, debido a la frecuencia de 50Hz las 128 muestras representan 2.56 segundos ($50 * 2.56 = 128$). En la Figura 5.1 se muestra la estructura de los datos. En la Tabla 5.1 se muestra la cantidad de instancias por clase. En las Figuras 5.2a y 5.2b muestran la distribución por clase de los conjuntos de datos UCI HAR.

Actividad	Entrenamiento	Porcentaje	Prueba	Porcentaje
Bajar escaleras	986	13.4 %	420	14.25 %
Subir escaleras	1,073	14.6 %	471	15.98 %
Caminar	1,226	16.7 %	496	16.83 %
Sentarse	1,286	17.5 %	491	16.66 %
Pararse	1,374	18.7 %	532	18.05 %
Acostarse	1,407	19.1 %	537	18.22 %
Total	7,352	100 %	2,947	100 %

Tabla 5.1: Número de instancias por clase del conjunto de datos UCI HAR.

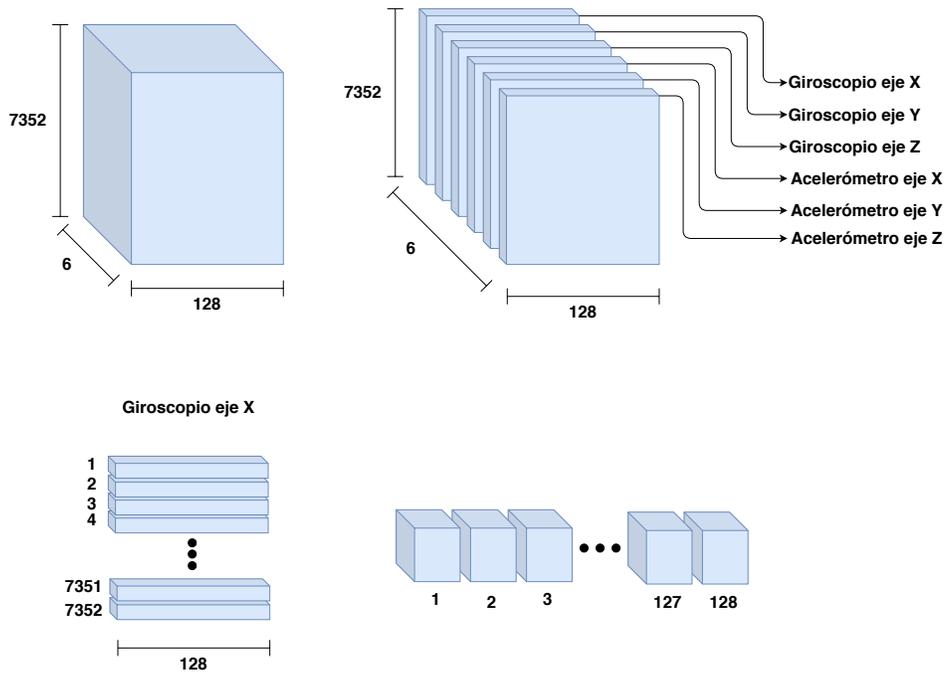


Figura 5.1: Adecuación y estructura del conjunto de datos UCI HAR.

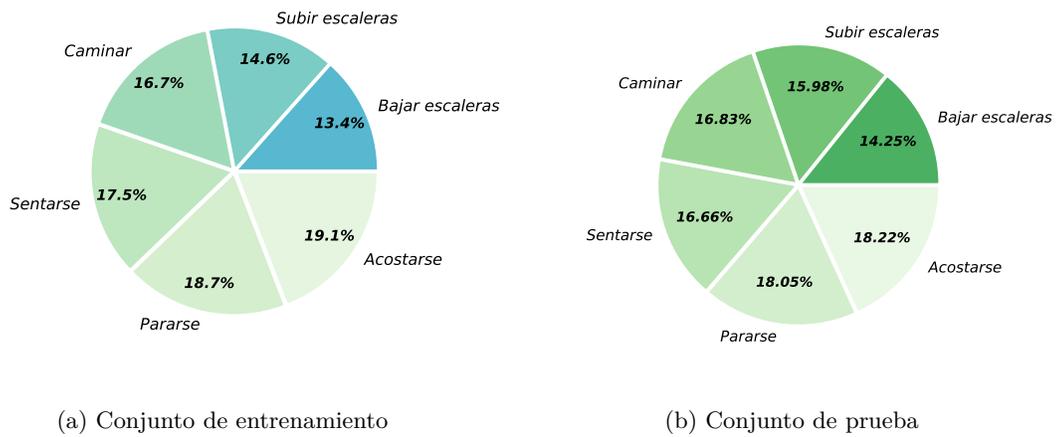


Figura 5.2: Distribución del conjunto de datos UCI HAR.

5.2 HAPT

Tenemos que *Human Activities and Postural Transitions* (HAPT) es una actualización del conjunto de datos UCI HAR, tiene doce clases: caminar, subir escaleras, bajar escaleras, sentarse, pararse, acostarse, pararse a sentarse, sentarse a pararse, sentarse a acostarse, acostarse a sentarse, pararse a acostarse, acostarse a pararse. HAPT proporciona un conjunto de datos no procesados.

Se realizó un preprocesamiento aplicando el método de ventana deslizante con empalme de 50 % con tamaño de ventana de 128 muestras, equivalente a 6.4 segundos, se aplicó un filtro Butterworth de tercer orden con frecuencia de corte de 10 Hz . El total de instancias generadas fueron de 12,742 instancias con 128 atributos y 6 canales. Se crearon dos particiones aleatoriamente; 8,919 instancias para el conjunto de entrenamiento (70 %) y 3,823 para el conjunto de prueba (30 %). En la Figura 5.3 se muestra la metodología y estructura de los datos. En la Tabla 5.2 se muestra la cantidad de instancias por clase del conjunto de entrenamiento y del conjunto de prueba. En las Figuras 5.4a y 5.4b muestran la distribución por clase de los conjuntos de datos UCI HAPT.

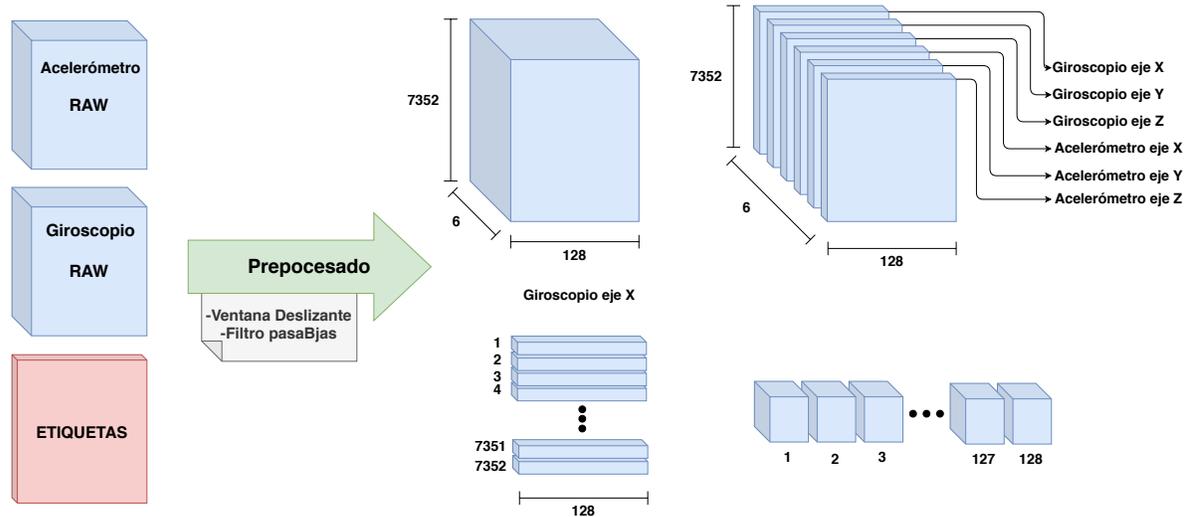
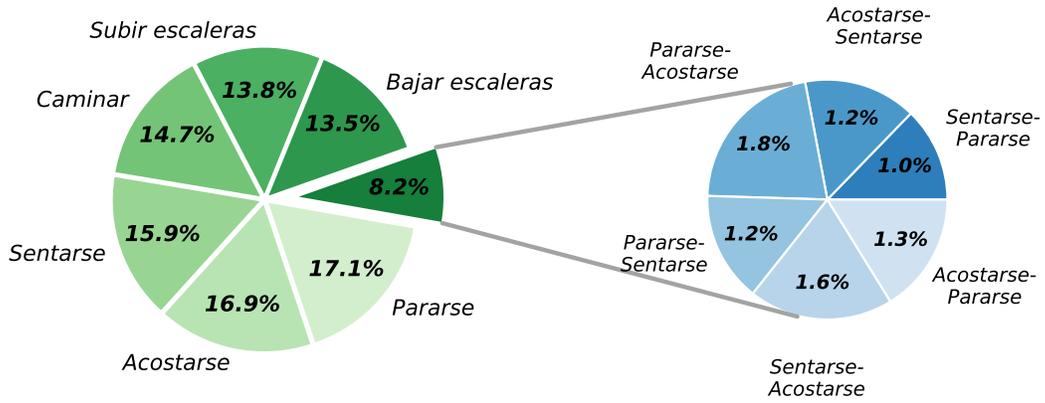


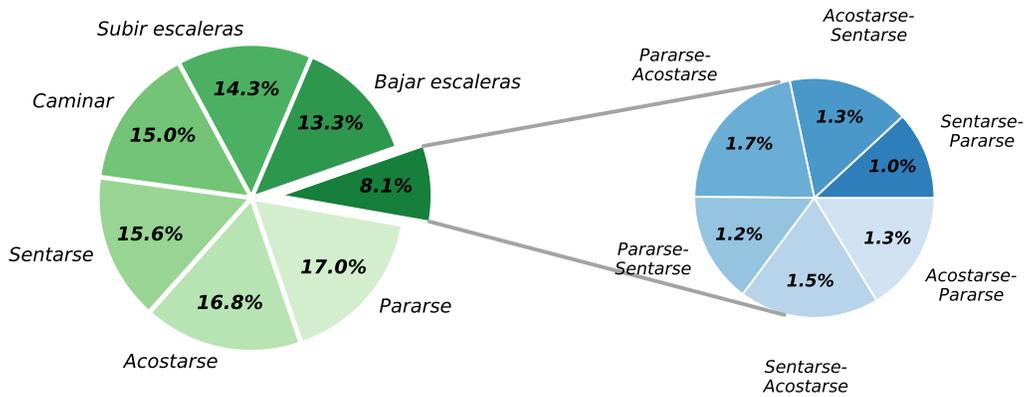
Figura 5.3: Adecuación y estructura del conjunto de datos HAPT.

5.3 WISDM v1.1

El conjunto de datos de Wireless Sensor Data Mining (WISDM) v1.1, contiene 1,098,204 señales no preprocesadas de seis diferentes actividades humanas: caminar, subir escaleras, bajar escaleras, sentarse, pararse, trotar. Estas señales fueron tomadas con un teléfono inteligente que midió la aceleración en tres ejes (X,Y,Z) con una frecuencia de 20 Hz . Para el preprocesamiento de las señales se utilizó el método de ventana deslizante con empalme de 50 % y un tamaño de ventana de 128 muestras equivalente a 6.4s, se aplicó un filtro de Butterworth de tercer orden con frecuencia de corte de 10 Hz . El total de instancias generadas fueron de 17,158 instancias con 128 atributos y 3 canales. Se crearon dos particiones aleatoriamente; 12,010 instancias para



(a) Conjunto de entrenamiento de HAPT



(b) Conjunto de prueba de HAPT

Figura 5.4: Distribución de clases del conjunto de prueba de HAPT.

Actividad	Procesada	Entrenamiento	Prueba
Sentarse-Pararse	123	93	30
Pararse-Sentarse	155	108	47
Acostarse-Sentarse	169	111	59
Acostarse-Pararse	170	118	51
Sentarse-Acostarse	195	142	53
Pararse-Acostarse	223	157	66
Bajar escaleras	1,691	1,202	489
Subir escaleras	1,817	1,227	590
Caminar	1,905	1,314	591
Sentarse	1,983	1,421	562
Acostarse	2,144	1,503	641
Pararse	2,167	1,523	644
Total	12,742	8,919	3,823

Tabla 5.2: Número de instancias por clase de HAPT procesada y particionada.

el conjunto entrenamiento (70%) y 5,148 para el conjunto de prueba (30%). En la Figura 5.5 se muestra la metodología y estructura de los datos. La Tabla 5.3 muestra la cantidad de instancias. En las Figuras 5.6a y 5.6b muestra la distribución de datos por clase.

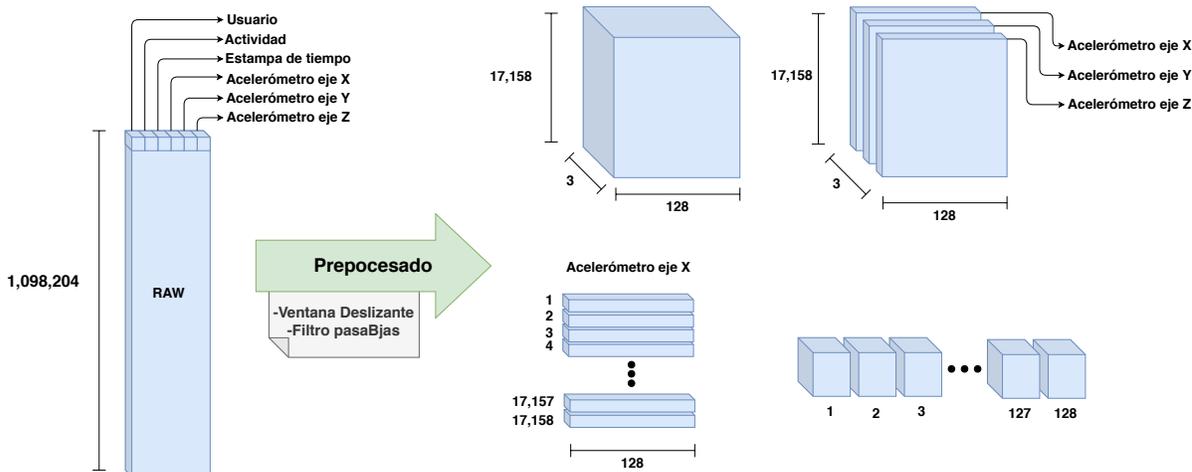


Figura 5.5: Adecuación y metodología y estructura del conjunto de datos WISDM v1.1.

5.4

WISDM v2.0

WISDM v2.0 es una actualización del conjunto de datos WISDM v1.1, es un conjunto de 2,980,765 señales no preprocesadas de seis diferentes actividades humanas: caminar, subir escaleras, sentarse, pararse, trotar, acostarse. Estas señales fueron tomadas con un teléfono inteligente el cual midió la aceleración en tres ejes (X,Y,Z) con una frecuencia de 20 Hz. Se utilizó el mismo

Actividad	Sin procesamiento	Procesada	Entrenamiento	Prueba
Pararse	48,395	757	544	213
Sentarse	59,939	936	650	286
Bajar escaleras	100,427	1,565	1,089	476
Subir escaleras	122,869	1,927	1,327	600
Trotar	342,176	5,346	3,737	1,609
Caminar	424,398	6,627	4,663	1,964
Total	1,098,204	17,158	12,010	5,148

Tabla 5.3: Número de instancias por clase del conjunto de datos WISDM v1.1.

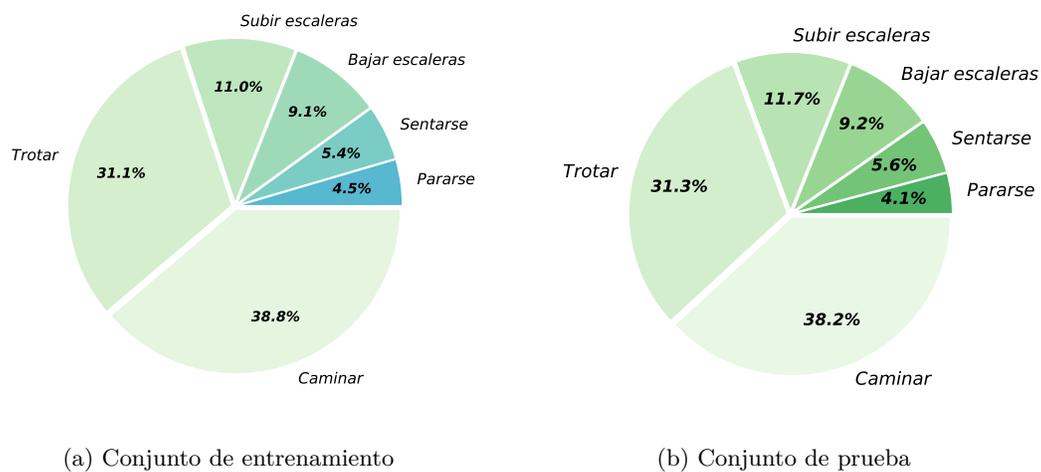


Figura 5.6: Distribución de clases del conjunto de datos WISDM v1.1.

preprocesamiento aplicado al conjunto de datos WISDM v1.1, se aplicó el método de ventana deslizante con empalme de 50 %, con un tamaño de ventana de 128 muestras equivalente a 6.4s, se aplicó un filtro de Butterworth de tercer orden con frecuencia de corte de $10Hz$.

El total de instancias generadas fueron de 46,573 instancias con 128 atributos y 3 canales. Se crearon dos particiones aleatoriamente; 32,601 instancias para el conjunto entrenamiento (70 %) y 13,972 para el conjunto de prueba (30 %). En la figura 5.7 se muestra la metodología y estructura de los datos. La Tabla 5.4 muestra la distribución de estas instancias por clase. En la figura 5.4 muestra la distribución de datos procesados por clase.

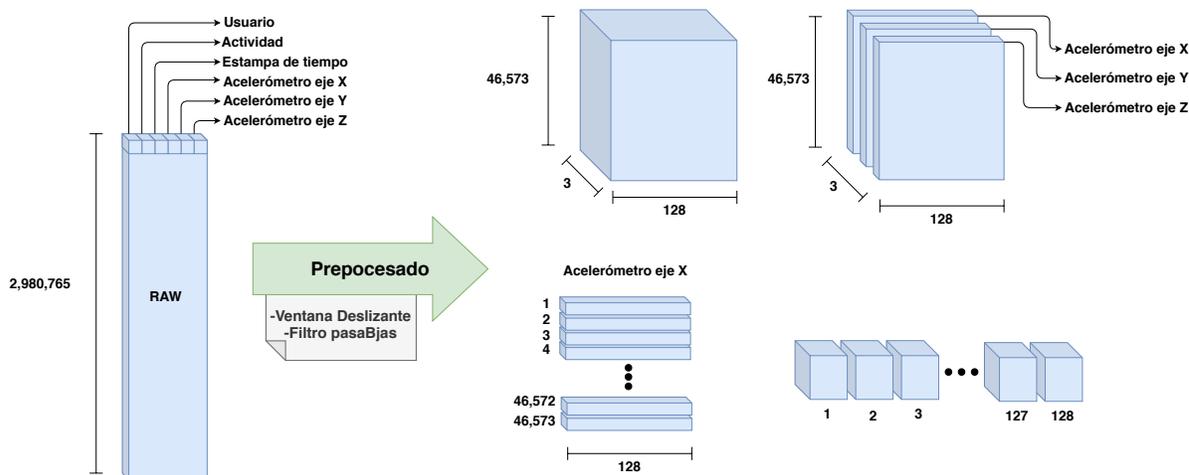


Figura 5.7: Adecuacion y estructura del conjunto de datos WISDM v2.0.

Actividad	Sin procesamiento	Procesada	Entrenamiento	Prueba
Escaleras	57,425	896	644	252
Acostarse	275,967	4,315	2,972	1,343
Pararse	288,873	4,512	3,141	1,371
Trotar	438,871	6,859	4,833	2,026
Sentarse	663,706	10,373	7,336	3,037
Caminar	1,255,92	19,618	13,675	5,943
Total	2,980,765	46,573	32,601	13,972

Tabla 5.4: Número de instancias por clase de WISDM v2.0 procesada y particionada.

La Tabla 5.5 compara las clases contenidas en cada conjunto de datos para el reconocimiento de actividad humana descritas anteriormente; UCI HAR, UCI HAPT, WISDM v1.1 y WISDM v2.0. Notemos que el único conjunto de datos preprocesado es la de UCI HAR, mientras que las demás son datos crudos entregados por el acelerómetro y giroscopio, según corresponda.

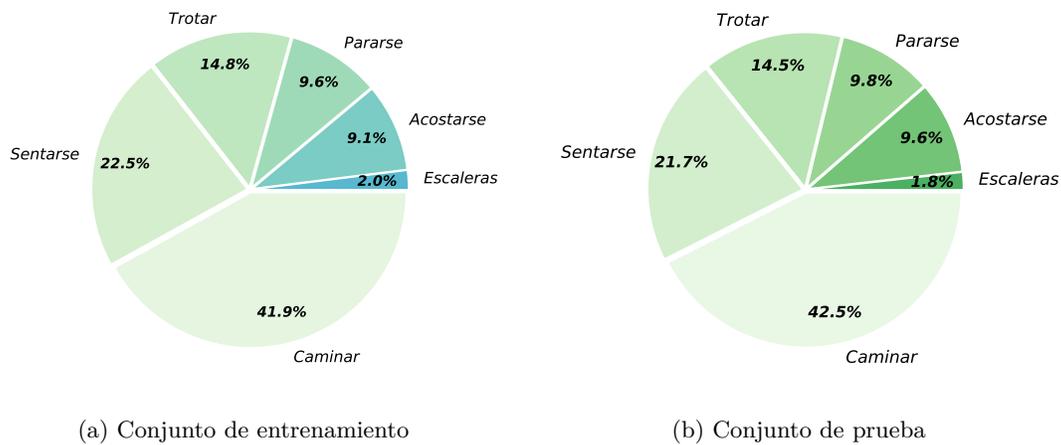


Figura 5.8: Distribución de procesamiento del conjunto de datos WISDM v2.0.

Clases	UCI HAPT	UCI HAR	WISDM v1.1	WISDM v2.0
Pararse-Sentarse	*			
Sentarse-Pararse	*			
Sentarse-Acostarse	*			
Acostarse-Sentarse	*			
Pararse-Acostarse	*			
Acostarse-Pararse	*			
Subir escaleras	*	*	*	
Bajar escaleras	*	*	*	
Acostarse	*	*		*
Caminar	*	*	*	*
Sentarse	*	*	*	*
Pararse	*	*	*	*
Trotar			*	*
Escaleras				*
Total clase	12	6	6	6

Tabla 5.5: Comparación de clases contenidas en los conjuntos de datos de reconocimiento de activada humana.

5.5 MNIST

Modified National Institute of Standards and Technology (MNIST) [34], es una base de datos de números escritos a mano, consta de 10 clases que representan los números del 0 al 9. Los dígitos están representadas por medio de imágenes en escala de grises con un tamaño de 28×28 píxeles. MNIST esta conformado por un total de 70,000, donde 60,000 son de entrenamiento y 10,000 de prueba. Debido a que las imágenes están en escala de grises solo cuentan con un canal, así la dimensión del conjunto de datos es $(28,28,1)$. En la figura 5.9 se muestran algunos ejemplos. En la Tabla 5.6 se muestra la cantidad de instancias por clase. En las figuras 5.10a y 5.10b muestra la distribución de las clases de la base de datos MNIST del conjunto de entrenamiento y conjunto de prueba.



Figura 5.9: Ejemplo de imágenes con resolución de 28×28 píxeles del conjunto de datos MNIST.

Dígito	Entrenamiento	Prueba
Cero	5,923	980
Uno	6,742	1,135
Dos	5,958	1,032
Tres	6,131	1,010
Cuatro	5,842	982
Cinco	5,421	892
Seis	5,918	958
Siete	6,265	1,028
Ocho	5,851	974
Nueve	5,949	1,009
Total	60,000	10,000

Tabla 5.6: Número de instancias por clase de la base de datos MNIST.

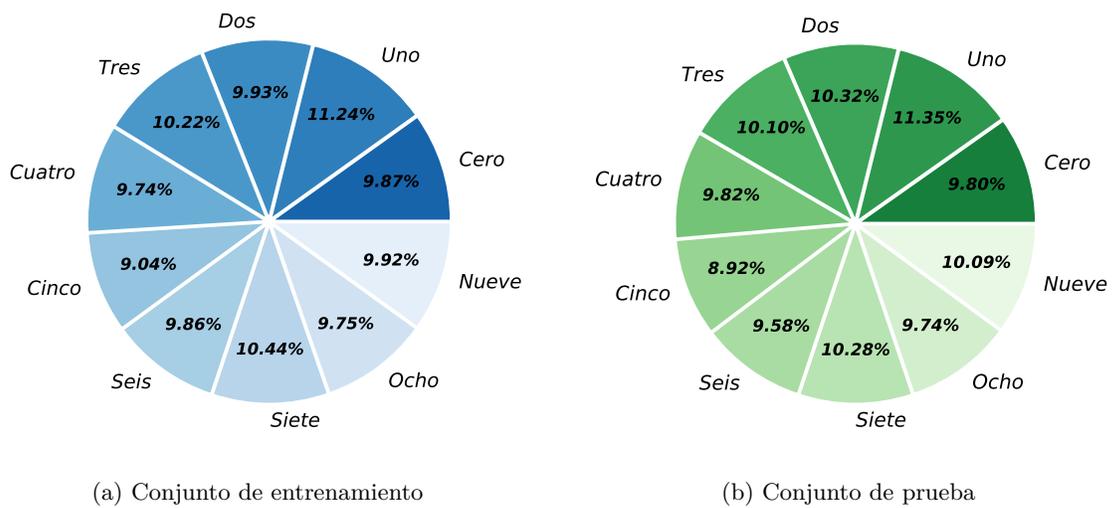


Figura 5.10: Distribución de procesamiento del conjunto de datos MNIST.

Parte IV

RESULTADOS Y CONCLUSIONES

Con el fin de mejorar tu juego, debes de estudiar los finales antes que todo, ya que mientras que los finales pueden ser estudiados y dominados por sí mismos, el medio juego y la apertura deben de ser estudiados en relación con los finales.

J. R. CAPABLANCA EX CAMPEÓN DEL MUNDO

6

EVALUACIÓN DE LA RED NEURONAL GRUESA-FINA

En este capítulo se presenta la evaluación de la red neuronal profunda gruesa-fina para el reconocimiento de actividad humana y el reconocimiento de imágenes, usando las diferentes bases de datos seleccionadas en los capítulos anteriores; MNIST, UCI HAR, UCI HAPT, WISDM v1.1 y WISDM v2.0. También, se presenta el análisis y conclusiones de los resultados.

6.1 Plataforma de prueba

La implementación se realizó en una computadora tipo Workstation con las siguientes características:

- **Procesador:** Intel Core i7-8750H
- **Sistema operativo:** Linux Ubuntu 18.04 LTS
- **Procesadores:** 6 CPUs con una velocidad promedio de 4.1 GHz con Turbo Boost
- **GPU:** Nvidia GeForce GTX 1050, 768 núcleos, memoria de 4 GB.
- **Memoria RAM:** 8 GB.
- **Librerías:** Las librerías instaladas para la ejecución son mostradas en la Tabla 6.1.

6.2 Resultados del ajuste de la red neuronal gruesa-fina

Los resultados se analizaron en dos etapas: entrenamiento y prueba. La primera etapa consiste en entrenar la red neuronal propuesta con el conjunto de datos de entrenamiento. Por otro lado, la segunda etapa consiste en evaluar la red neuronal obtenida por medio del conjunto de datos de prueba, el cual es ajeno al conjunto de entrenamiento. Para medir el rendimiento de la red neuronal gruesa-fina se utilizó el porcentaje de reconocimiento, definido como se muestra en la Ecuación 4.5.

Librería	Versión
Python	3.6.10
TensorFlow	2.1
Keras	2.3.1
Conda	4.8.3
Numpy	1.18.1
Pandas	1.0.1
Mpi4py	3.0.3
Cuda	10.1.2

Tabla 6.1: Librerías utilizadas en la ejecución.

6.2.1 Resultados del conjunto de datos UCI HAR

En las Figuras 6.1a y 6.1b se muestra la evaluación de los conjuntos de entrenamiento y prueba. Se alcanzó el 100 % de recuperación y el parámetro de pérdida alcanzó el valor 0 % al evaluar el conjunto de prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento fue de 4.83 minutos.

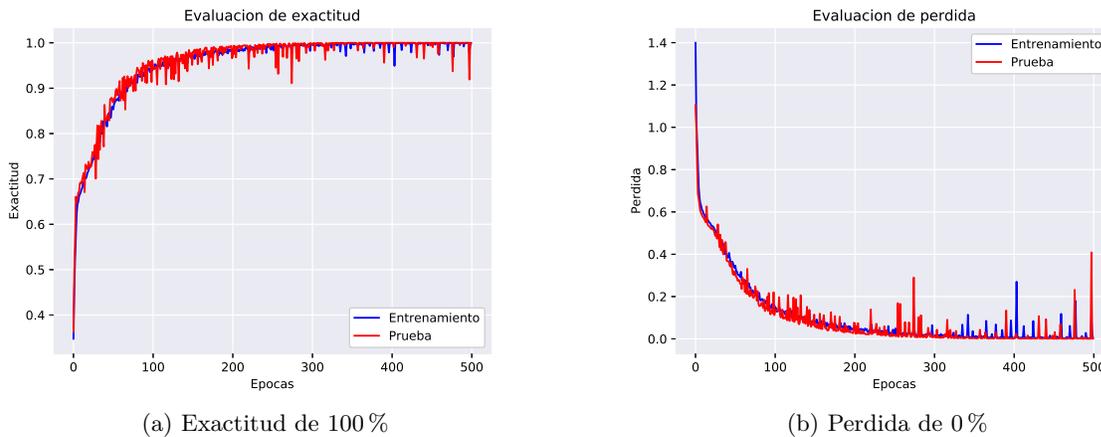


Figura 6.1: Evaluación del conjunto entrenamiento de HAR.

6.2.2 Resultados del conjunto de datos HAPT

HAPT resulta un conjunto de datos difícil, ya que no cuenta con un equilibrio en el número de instancias por clase (como se ve en la Tabla 5.2), sin embargo, se alcanzó una exactitud del 95.27 % con una pérdida de 4.73 % al evaluar el conjunto de prueba. En las Figuras 6.2a y 6.2b se muestra la evaluación de los conjuntos de entrenamiento y prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento fue de 5.30 minutos.

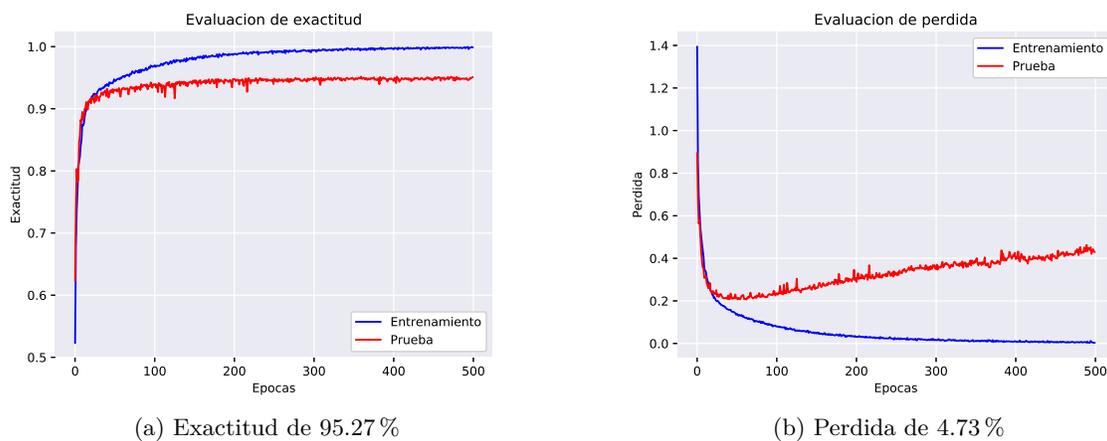


Figura 6.2: Evaluación del conjunto prueba de HAPT.

6.2.3

Resultados del conjunto de datos WISDM v1.1

Se alcanzó una recuperación de 98.12 % y valor de pérdida de 1.88 % al evaluar el conjunto de prueba. En la Figura 6.2.3 se muestra la evaluación de los conjuntos de entrenamiento y prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento fue de 7.06 minutos.

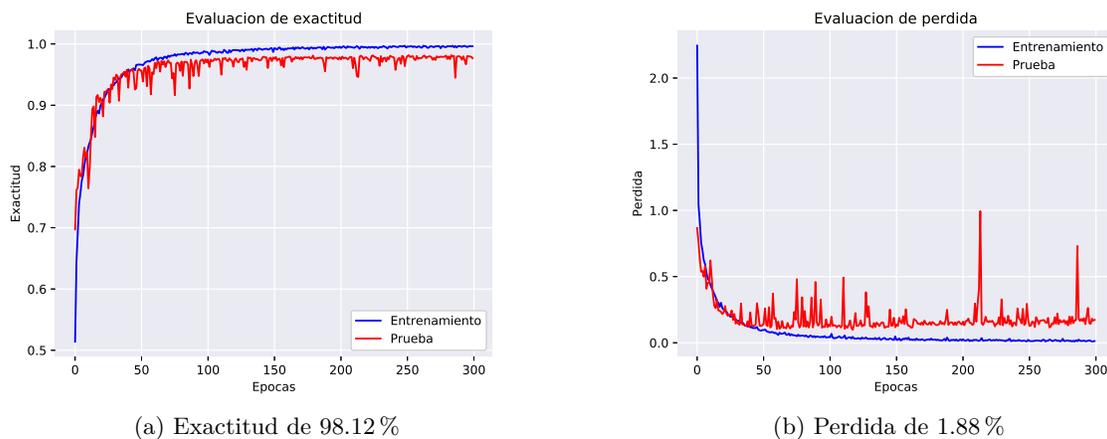


Figura 6.3: Evaluación del conjunto entrenamiento de WISDM v1.1.

6.2.4 Resultados del conjunto de datos WISDM v2.0

WISDM v2.0 es una versión más grande que la v1.1, esto aumenta la dificultad. Los resultados obtenidos con el conjunto de datos WISDM v2.0 alcanzaron un valor de recuperación de 94.75 % y un valor de pérdida de 5.21 % al evaluar el conjunto de prueba. En la Figura 6.2.4 se muestra la evaluación de los conjuntos de entrenamiento y prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento fue de 14.85 minutos.

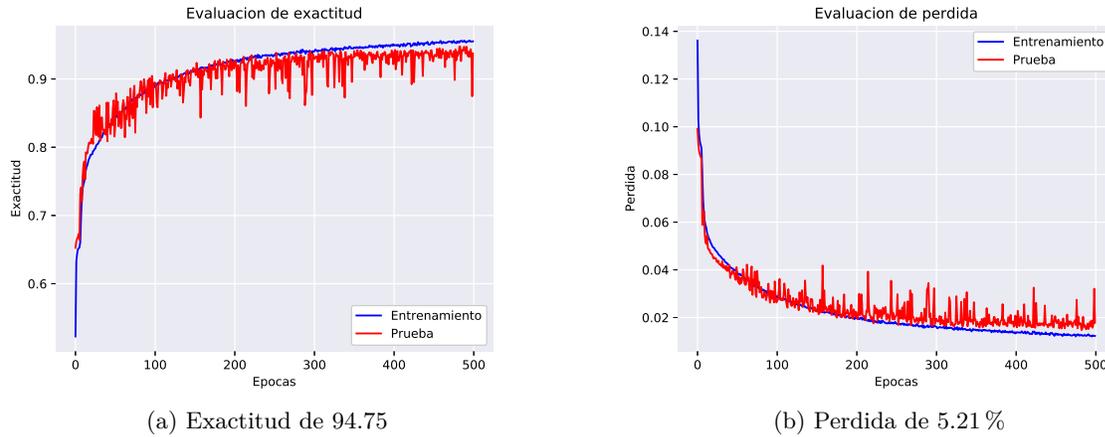


Figura 6.4: Evaluación del conjunto prueba de WISDM v2.0.

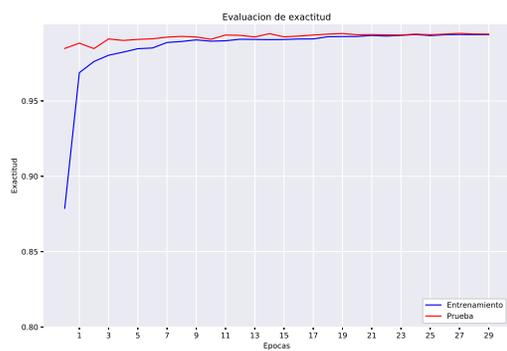
6.2.5 Resultados del conjunto de datos MNIST

MNIST es una base de datos muy popular para la evaluación de modelos. Los resultados obtenidos alcanzaron un valor de exactitud de 99.47 % y un porcentaje de error de 0.53 % al evaluar el conjunto de prueba. La Figura 6.2.5 muestra la evaluación del conjunto de entrenamiento y del conjunto de prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento y con 30 etapas fue de 30.75 minutos.

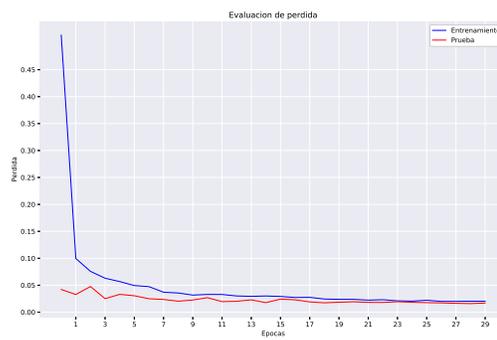
La Tabla 6.2 muestra una comparación de los resultados obtenidos con otros resultados en el estado del arte.

6.3 Conclusiones y análisis sobre la implementación de la red neuronal propuesta hacia imágenes y nuevos conjuntos de datos

Se presentó una nueva implementación de la propuesta de una red neuronal profunda previamente reportada en la literatura para aplicarse sólo para reconocimiento y clasificación de patrones HAR, tal que la estrategia se basa en la extracción paralela en tres etapas: fina, media y gruesa. Sin embargo, es necesario una mejora en la optimización de los parámetros, ya que como hemos



(a) Exactitud de 99.47%



(b) Pérdida de 0.53%

Figura 6.5: Evaluación del conjunto prueba de MNIST.

Base Datos	Propuesta	Avilés [6]	Zhang [69]	Ravi [49]	Abebe [2]	Taufeeq [61]	Wan [64]
UCI HAR	100	100	98.4	-	-	-	-
UCI HAPT	95.27	-	93.1	-	-	100	-
WISD v1.1	98.12	100	97.0	98.6	-	-	-
WISD v2.0	94.75	-	-	92.7	97.9	-	-
MNIST	99.47	-	-	-	-	-	99.79

Tabla 6.2: Comparación de exactitud de diferentes técnicas del estado del arte.

presentado en la implementación para imágenes, como meta más ambiciosa y de mayor alcance para aplicaciones en visión por computadora, es necesario poder hacerle modificaciones que mejoren en rapidez y capacidad de extracción de características propias para imágenes como entidades numéricas más complejas.

La implementación aquí presentada está ubicada en el nivel 25 del *Ranking classification datasets results* ¹, esta página se tomó como referencia para la comparación de resultados de diferentes técnicas y enfoques del conjunto de datos MNIST, ya que para aplicaciones prácticas es factible su uso, pero consideramos que es posible su mejora si realizamos modificaciones a la capa de extracciones paralelas, buscando robustecer a los patrones orientados a problemas de imágenes más complejas.

Si bien la base de datos MNIST es una buena aproximación para validar posibles aplicaciones reales, existen problemas de imágenes más complejas como lo son el rostro humano y los escenarios naturales, donde en un futuro trabajo podremos mostrar los avances en tales campos.

¹Ranking classification datasets results: https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

7

EVALUACIÓN DE LA PROPUESTA PRINCIPAL

En este capítulo se presenta la evaluación los dos modelos propuestos para la optimización de los parámetros de la red neuronal profunda gruesa-fina, también se describe y se muestran las pruebas realizadas usando las diferentes bases de datos seleccionadas; MNIST, UCI HAR, UCI HAPT, WISDM v1.1 y WISDM v2.0. Además, se compara los resultados obtenidos de los modelos propuestos con el estado del arte.

7.1 Plataforma de prueba

La implementación se realizó en una computadora tipo Workstation con las siguientes características:

- **Procesador:** Intel Core i7-8750H
- **Sistema operativo:** Linux Ubuntu 18.04 LTS
- **Procesadores:** 6 CPUs con una velocidad promedio de 4.1 GHz con Turbo Boost
- **GPU:** Nvidia GeForce GTX 1050, 768 núcleos, memoria de 4 GB.
- **Memoria RAM:** 8 GB.
- **Librerías:** Las librerías instaladas para la ejecución de los modelos son mostradas en la Tabla 6.1 de la sección anterior.

En las siguientes secciones se muestran los resultados obtenidos con el modelo secuencial y el modelo paralelo, estos resultados están divididos con base a las cinco bases de datos seleccionadas para el reconocimiento de actividad humana y el reconocimiento de imágenes.

7.2 Parámetros del algoritmo genético

El criterio de selección de los parámetros de los modelos evolutivos propuestos fue por medio de ensayo y error, sin embargo, fueron acotados debido al tiempo de ejecución en cada prueba. Los parámetros son utilizados en las cinco bases de datos seleccionadas, en la Tabla 7.1 se muestran los parámetros usados en los modelos evolutivos.

Parámetros genético	Genético secuencial		Genético paralelo	
	GPU	CPU	GPU	CPU
Generaciones	15	15	15	15
Tamaño de población	10	10	10	10
Porcentaje de elitismo	10 %	10 %	10 %	10 %
Porcentaje de cruce	60 %	60 %	60 %	60 %
Porcentaje de mutación	30 %	30 %	30 %	30 %
Factor de mutación	0.1	0.1	0.1	0.1

Tabla 7.1: Parámetros utilizados en los modelos evolutivos.

7.3 Resultados del conjunto de datos UCI HAR

A continuación se mostrará el proceso y características de la ejecución de los dos modelos evolutivos propuestos. En la Sección 7.2 se muestran los parámetros seleccionados para los modelos evolutivos propuestos. En la Tabla 7.2 muestra el resultado de la ejecución del algoritmo genético secuencial y el algoritmo genético paralelo visto en el Capítulo 4, los resultados están divididos por el modo de ejecución, esto es, ejecución con CPU y ejecución con GPU.

Observe la diferencia de la eficiencia en el tiempo de ejecución del uso de las GPUs contra la CPU. La ejecución del modelo secuencial con CPU logró obtener una exactitud de 99.4 % con un tiempo elevado de 12.07 hrs. La ejecución del modelo secuencial con GPU logró una exactitud del 99.4 % con un error del 1.6 % y un tiempo de 4.50 hrs. La exactitud obtenida en los dos modos de ejecución (CPU y GPU) del modelo secuencial fue la misma, sin embargo, se redujo el tiempo de ejecución en un 63 % aproximadamente.

Por otro lado, la ejecución del modelo paralelo con CPU logró obtener una exactitud de 100 % con un tiempo de 4.10 hrs. La ejecución del modelo paralelo con GPU también logró una exactitud del 100 %, sin embargo, el tiempo de ejecución fue de 3.45 hrs. reduciendo el tiempo de ejecución en un 15 % aproximadamente.

Base de datos	CPU		GPU	
	Exactitud	Tiempo	Exactitud	Tiempo
UCI HAR				
Secuencial	99.4 %	12.07 hr	99.4 %	4.50 hr
Paralelo	100 %	4.10 hr	100 %	3.45 hr

Tabla 7.2: Resultados de los modelos genéticos con el conjunto de datos UCI HAR.

En la Figura 7.1a nos muestra la ganancia en exactitud en cada generación utilizando la GPU y en la Figura 7.1b nos muestra la pérdida de error en cada generación utilizando la GPU.

En la Tabla 7.3 podemos ver los parámetros optimizados de la red neuronal gruesa-fina encontrados por los modelos evolutivos propuestos.

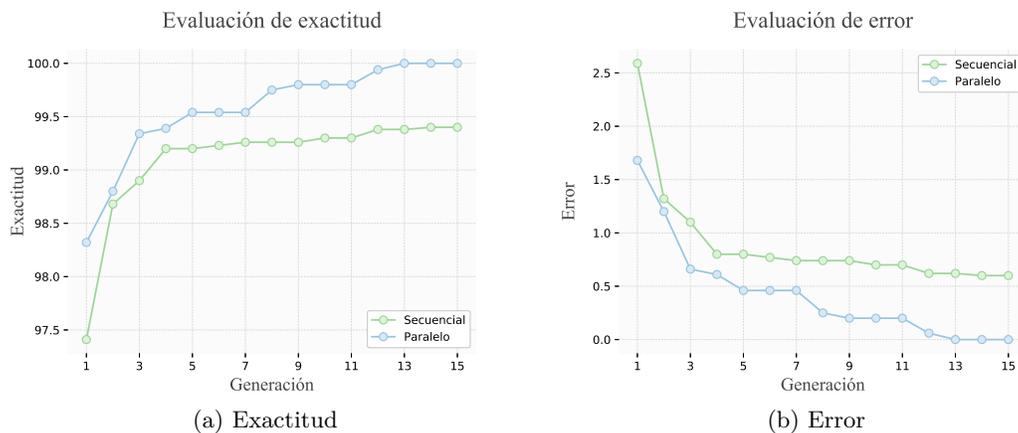


Figura 7.1: Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.

UCI HAR	Genético secuencial		Genético paralelo	
	GPU	CPU	GPU	CPU
Número de épocas	530	540	570	502
Tamaño de lote	300	350	305	320
Dropout	0.95	0.92	0.97	0.94
Función activación	ReLU	ReLU6	ReLU6	ReLU6
Convoluciones	26, 52	26, 56	26, 56	26, 56
Dimensión Filtro	(1 × 2)	(1 × 2)	(1 × 2)	(8 × 8)
Dimensión Maxpooling	(1 × 2)	(1 × 2)	(2 × 2)	(2 × 2)
Optimizador	Adadelta	Adadelta	Adadelta	Adadelta
Tasa de aprendizaje	0.005	0.005	0.005	0.005

Tabla 7.3: Parámetros optimizados encontrados por los modelos evolutivos.

7.4 Resultados del conjunto de datos UCI HAPT

A continuación se mostrará el proceso y características de la ejecución de los dos modelos evolutivos propuestos. En la Sección 7.2 se muestran los parámetros seleccionados para los modelos evolutivos propuestos. En la tabla 7.4 muestra el resultado de la ejecución del algoritmo genético secuencial y el algoritmo genético paralelo visto en el Capítulo 4, los resultados están divididos por el modo de ejecución, esto es, ejecución con CPU y ejecución con GPU.

La ejecución del modelo secuencial con CPU logró obtener una exactitud de 95.27% de clasificación con un tiempo elevado de 13.76 hrs., mientras que la ejecución del modelo secuencial con GPU logró una exactitud del 95.27% con un error del 4.73% y un tiempo de 6.3 hrs., de modo que notamos la diferencia de la eficiencia en tiempo del uso de las GPUs contra la CPU, reduciendo un 54% de tiempo en la ejecución.

Por otro lado, la ejecución del modelo paralelo con CPU logró obtener una exactitud de 95.80% de clasificación con un tiempo de 5.2 hrs., en cuanto a la ejecución del modelo paralelo con GPU logró una exactitud del 95.80% con un error del 4.20% y un tiempo de 4.4 hrs., reduciendo el tiempo de ejecución en un 15% con la utilización de la GPU.

Entonces comparando el tiempo de ejecución entre el modelo secuencial y el modelo paralelo con la utilización de GPU notamos una ganancia de tiempo de 2 hrs.

Base de datos	CPU		GPU	
UCI HAPT	Exactitud	Tiempo	Exactitud	Tiempo
Secuencial	95.27 %	13.76 hr	95.27 %	6.3 hr
Paralelo	95.80 %	5.2 hr	95.80 %	4.4 hr

Tabla 7.4: Resultados de los modelos genéticos con el conjunto de datos UCI HAPT.

Notemos que se encontró una mejora en la exactitud de manera automática sin intervención experta. En la Figura 7.2a nos muestra la ganancia en exactitud en cada generación y en la Figura 7.2b nos muestra la pérdida de error en cada generación de los modelos evolutivos ejecutados con GPU.

En la Tabla 7.5 podemos ver los parámetros optimizados de la red neuronal gruesa-fina encontrados por los modelos evolutivos propuestos.

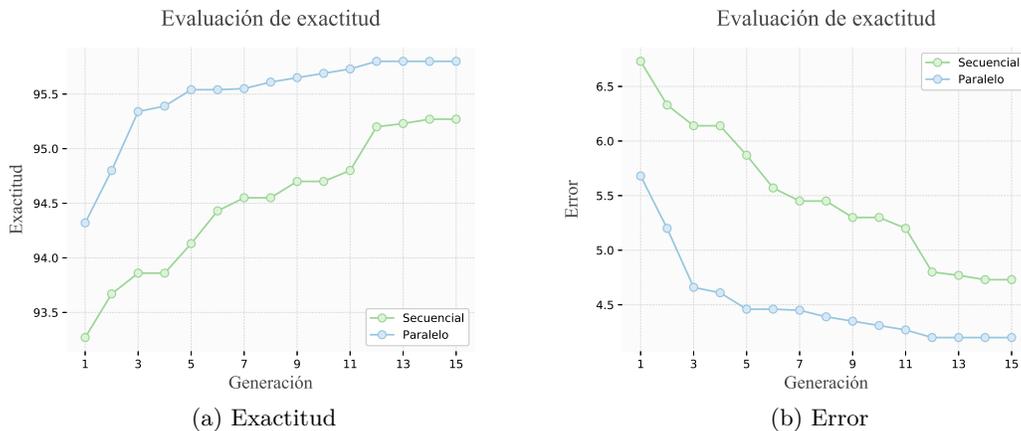


Figura 7.2: Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.

UCI HAPT	Genético secuencial		Genético paralelo	
	GPU	CPU	GPU	CPU
Número de épocas	33	33	33	33
Tamaño de lote	209	209	220	210
Dropout	0.76	0.76	0.80	0.83
Función activación	ReLU	ReLU6	ReLU6	ReLU6
Convoluciones	26, 52	26, 56	26, 56	26, 56
Dimensión Filtro	(8×8)	(8×8)	(8×8)	(8×8)
Dimensión Maxpooling	(2×2)	(2×2)	(2×2)	(2×2)
Optimizador	Adadelta	Adadelta	Adadelta	Adadelta
Tasa de aprendizaje	0.005	0.005	0.005	0.005

Tabla 7.5: Parámetros optimizados encontrados por los modelos evolutivos.

7.5

Resultados del conjunto de datos WISDM v1.1

A continuación se mostrará el proceso y características de la ejecución de los dos modelos evolutivos propuestos. En la Sección 7.2 se muestran los parámetros seleccionados para los modelos evolutivos propuestos. En la tabla 7.6 muestra el resultado de la ejecución del algoritmo genético secuencial y el algoritmo genético paralelo visto en el Capítulo 4, los resultados están divididos por el modo de ejecución, esto es, ejecución con CPU y ejecución con GPU.

La ejecución del modelo secuencial con CPU logró obtener una exactitud de 98.30 % con un tiempo elevado de 17.65 hrs. La ejecución del modelo secuencial con GPU logró una exactitud del 98.30 % con un error del 2.70 % y un tiempo de 8.51 hrs. A pesar de que el porcentaje de clasificación es la misma notamos una ganancia del 46 % en el tiempo de ejecución.

Por otro lado, la ejecución del modelo paralelo con CPU logró obtener una exactitud de 98.92 % con un tiempo de 8.2 hrs. La ejecución del modelo paralelo con GPU logró una exactitud del 98.92 % con un error del 2.08 % y un tiempo de 7.48 hrs., a pesar que el porcentaje de exactitud es la misma, notamos una ganancia en el tiempo de ejecución de 8 % aproximadamente.

Base de datos	CPU		GPU	
WISDM v1.1	Exactitud	Tiempo	Exactitud	Tiempo
Secuencial	98.30 %	17.65 hr	98.30 %	9.51 hr
Paralelo	98.92 %	9.2 hr	98.92 %	8.48 hr

Tabla 7.6: Resultados de los modelos genéticos con el conjunto de datos WISDM v1.1.

En la Figura 7.3a nos muestra la ganancia en exactitud en cada generación y en la Figura 7.3b nos muestra la pérdida de error en cada generación.

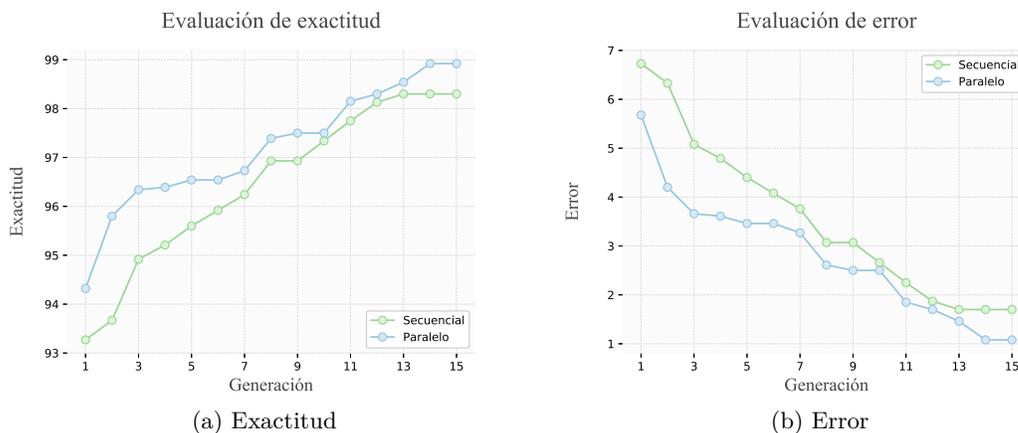


Figura 7.3: Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.

En la Tabla 7.7 podemos ver los parámetros optimizados de la red neuronal gruesa-fina encontrados por los modelos evolutivos propuestos.

WISDM v1.1	Genético secuencial		Genético paralelo	
	GPU	CPU	GPU	CPU
Parámetros				
Número de épocas	33	33	33	33
Tamaño de lote	209	209	220	210
Dropout	0.76	0.76	0.80	0.83
Función activación	ReLU	ReLU6	ReLU6	ReLU6
Convoluciones	26, 52	26, 56	26, 56	26, 56
Dimensión Filtro	(8 × 8)	(8 × 8)	(8 × 8)	(8 × 8)
Dimensión Maxpooling	(2 × 2)	(2 × 2)	(2 × 2)	(2 × 2)
Optimizador	Adadelta	Adadelta	Adadelta	Adadelta
Tasa de aprendizaje	0.005	0.005	0.005	0.005

Tabla 7.7: Parámetros optimizados encontrados por los modelos evolutivos.

7.6

Resultados del conjunto de datos WISDM v2.0

A continuación se mostrará el proceso y características de la ejecución de los dos modelos evolutivos propuestos. En la Sección 7.2 se muestran los parámetros seleccionados para los modelos evolutivos propuestos. En la tabla 7.8 muestra el resultado de la ejecución del algoritmo genético secuencial y el algoritmo genético paralelo visto en el Capítulo 4, los resultados están divididos por el modo de ejecución, esto es, ejecución con CPU y ejecución con GPU.

La ejecución del modelo secuencial con CPU logró obtener una exactitud de 95.1 % con un tiempo elevado de 32.125 hrs., mientras que la ejecución del modelo secuencial con GPU logró la misma exactitud de 95.1 %, pero con tiempo de ejecución de 24.4 hrs. Note la ganancia de tiempo aproximadamente del 25 %.

Por otro lado, la ejecución del modelo paralelo con CPU logró obtener una exactitud de 95.4 % con un tiempo de 23.2 hrs., mientras que la ejecución del modelo paralelo con GPU logró una exactitud del 95.4 % y un tiempo de 20.5 hrs. Notamos una ganancia de tiempo de 11 % aproximadamente.

Base de datos	CPU		GPU	
	Exactitud	Tiempo	Exactitud	Tiempo
WISDM v2.0				
Secuencial	95.1 %	32.125 hr	95.1 %	24.4 hr
Paralelo	95.4 %	23.2 hr	95.4 %	20.5 hr

Tabla 7.8: Resultados de los modelos genéticos con el conjunto de datos WISDM v2.0.

Notemos que se encontró una mejora en la exactitud de manera automática sin intervención experta. En la Figura 7.4a nos muestra la ganancia en exactitud en cada generación y en la Figura 7.4b nos muestra la pérdida de error en cada generación.

En la Tabla 7.9 podemos ver los parámetros optimizados de la red neuronal gruesa-fina encontrados por los modelos evolutivos propuestos.

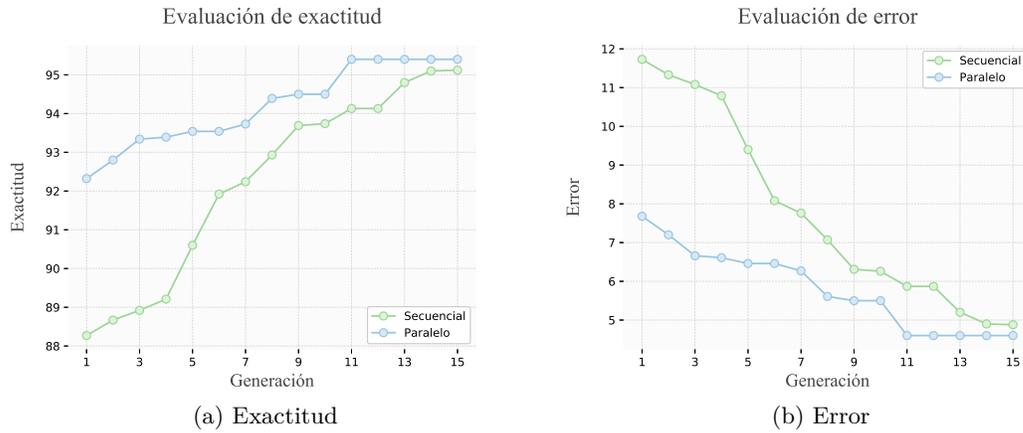


Figura 7.4: Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.

WISDM v2.0	Genético secuencial		Genético paralelo	
	GPU	CPU	GPU	CPU
Número de épocas	33	33	33	33
Tamaño de lote	209	209	220	210
Dropout	0.76	0.76	0.80	0.83
Función activación	ReLU	ReLU6	ReLU6	ReLU6
Convoluciones	26, 52	26, 56	26, 56	26, 56
Dimensión Filtro	(8×8)	(8×8)	(8×8)	(8×8)
Dimensión Maxpooling	(2×2)	(2×2)	(2×2)	(2×2)
Optimizador	Adadelta	Adadelta	Adadelta	Adadelta
Tasa de aprendizaje	0.005	0.005	0.005	0.005

Tabla 7.9: Parámetros optimizados encontrados por los modelos evolutivos.

7.7 Resultados del conjunto de datos MNIST

A continuación se mostrará el proceso y características de la ejecución de los dos modelos evolutivos propuestos. En la Sección 7.2 se muestran los parámetros seleccionados para los modelos evolutivos propuestos. En la tabla 7.10 muestra el resultado de la ejecución del algoritmo genético secuencial y el algoritmo genético paralelo visto en el Capítulo 4, los resultados están divididos por el modo de ejecución, esto es, ejecución con CPU y ejecución con GPU.

La ejecución del modelo secuencial con CPU logró obtener una exactitud de 99.50 % con un tiempo elevado de 35.3 hrs., mientras que la ejecución del modelo secuencial con GPU logró una exactitud del 99.50 % y un tiempo de 21.80 hrs.

Por otro lado, la ejecución del modelo paralelo con CPU logró obtener una exactitud de 99.50 % con un tiempo de 10.23 hrs. La ejecución del modelo paralelo con GPU logró una exactitud del 99.50 % y un tiempo de 21.80 hrs.

A pesar que no hubo diferencia en los porcentajes de exactitud de la clasificación, hubo una ganancia en el tiempo de ejecución, siendo la versión paralela con GPU la de mayor ganancia en tiempo de ejecución.

Base de datos	CPU		GPU	
	Exactitud	Tiempo	Exactitud	Tiempo
MNIST				
Secuencial	99.50 %	35.3 hr	99.50 %	21.80 hr
Paralelo	99.50 %	11.23 hr	99.50 %	10.26 hr

Tabla 7.10: Resultados para MNIST.

Notemos que se encontró una mejora en la exactitud de manera automática sin intervención experta. En la Figura 7.5a nos muestra la ganancia en exactitud en cada generación y en la Figura 7.5b nos muestra la pérdida de error en cada generación.

En la Tabla 7.11 podemos ver los parámetros optimizados de la red neuronal gruesa-fina encontrados por los modelos evolutivos propuestos.

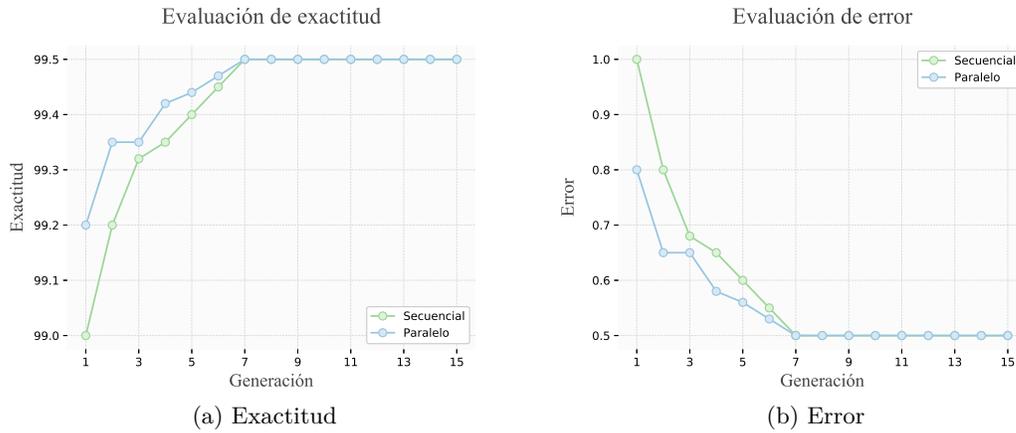


Figura 7.5: Gráficas que muestran la ganancia de los modelos evolutivos ejecutados con GPU.

MNIST	Genético secuencial		Genético paralelo	
	GPU	CPU	GPU	CPU
Número de épocas	33	33	33	33
Tamaño de lote	209	209	220	210
Dropout	0.76	0.76	0.80	0.83
Función activación	ReLU	ReLU6	ReLU6	ReLU6
Convoluciones	26, 52	26, 56	26, 56	26, 56
Dimensión Filtro	(8×8)	(8×8)	(8×8)	(8×8)
Dimensión Maxpooling	(2×2)	(2×2)	(2×2)	(2×2)
Optimizador	Adadelata	Adadelata	Adadelata	Adadelata
Tasa de aprendizaje	0.005	0.005	0.005	0.005

Tabla 7.11: Parámetros optimizados encontrados por los modelos evolutivos.

7.8 Escalabilidad

En la figura 7.6

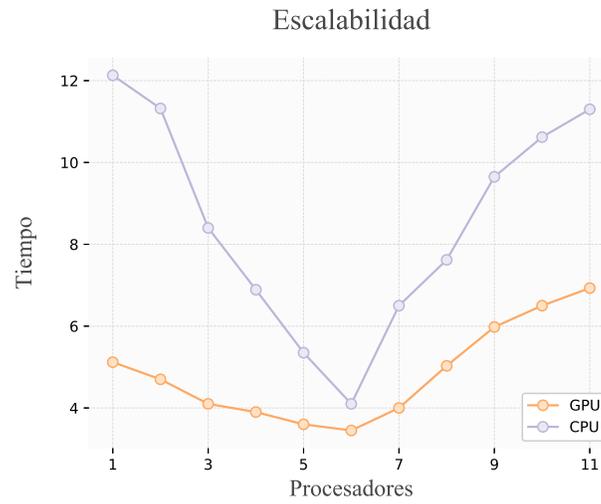


Figura 7.6: Comparación de escalabilidad de los modelos genéticos con la base de datos UCI HAR.

7.9 Comparación con el estado del arte

En esta sección se muestra las tablas del estado del arte con el objetivo de comparar los modelos implementados en este trabajo con otras técnicas enfocadas al ML y DL.

7.9.1 UCI HAR

En la Tabla 7.12 se muestra la comparación de resultados de la base de datos UCI HAR, donde se nota una mejoría sobre técnicas de aprendizaje automático como las cadenas de Márkov y máquinas de soporte vectorial. Observar que el método (CNN gruesa-fina) alcanzó el 100%, sin embargo, la calibración de los parámetros no fue automática. Tiene una En la Figura 7.7 muestra una gráfica de barras comparando las diferentes técnicas de aprendizaje automático y aprendizaje profundo con la propuesta de este trabajo.

Nombre	Año	Exactitud	Enfoque	Método
Avilés Cruz [6]	2019	100.0	DL	CNN
Yafte (CNN-GF)	2020	100.0	DL	CNN gruesa-fina
Yafte (CNN-P)	2020	100.0	DL	CNN+Genético+Paralelo
Yafte (CNN-S)	2020	99.40	DL	CNN+Genético+Secuencial
Yong Zhang [69]	2018	98.40	DL	U-CNN
San-Segundo [53]	2016	98.00	ML	Cadena de Márkov
Andrey Ignatov [25]	2018	97.63	DL	CNN
Heeryon Cho [11]	2018	97.62	DL	CNN+Sharpen
Xiangbin Zhu [71]	2016	96.61	ML	Cadena de Márkov
Davide Anguita [4]	2013	96.00	ML	SVM
Charissa Ann Ronao [52]	2016	95.75	DL	CNN+Furrier
Charissa Ann Ronao [52]	2016	94.79	DL	CNN

Tabla 7.12: Comparación de exactitud de UCI HAR [4].

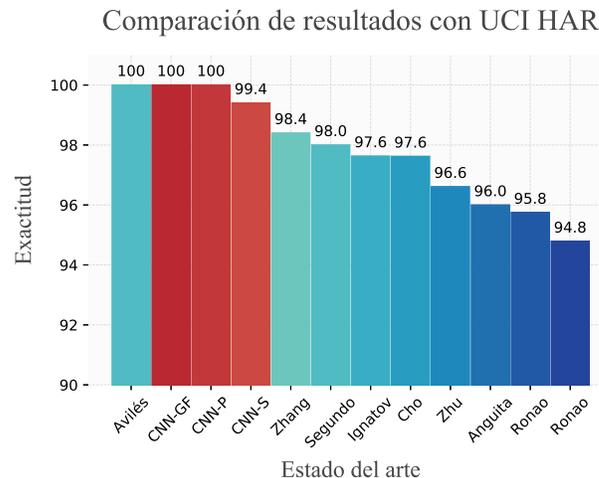


Figura 7.7: Comparación de resultados obtenidos con el estado del arte de la base de datos UCI HAR.

7.9.2 UCI HAPT

En la Tabla 7.9.2 se muestra la comparación de resultados de la base de datos UCI HAPT. Donde se alcanzó el mejor resultado comparando con enfoques de aprendizaje profundo, sin embargo la técnica de Random Forest sobresalió en esta base de datos. En la Figura 7.8 muestra una gráfica de barras comparando las diferentes técnicas de aprendizaje automático y aprendizaje profundo con la propuesta de este trabajo.

Nombre	Año	Exactitud	Enfoque	Método
Taufeeq [61]	2016	100.0	ML	Random Forrest
Zheng [70]	2018	96.26	ML	TASG+SVM
Bustoni [7]	2019	96.00	ML	Random Forrest
Zheng [70]	2018	95.83	ML	TASG+RNN
Yafte (CNN-P)	2020	95.80	DL	CNN+genético+paralelo
Yafte (CNN-GF)	2020	95.27	DL	CNN gruesa-fina
Yafte (CNN-S)	2020	95.27	DL	CNN+genético+secuencial
Yong Zhang [69]	2018	93.10	DL	U-CNN

Tabla 7.13: Comparación de exactitud de UCI HAPT [51].

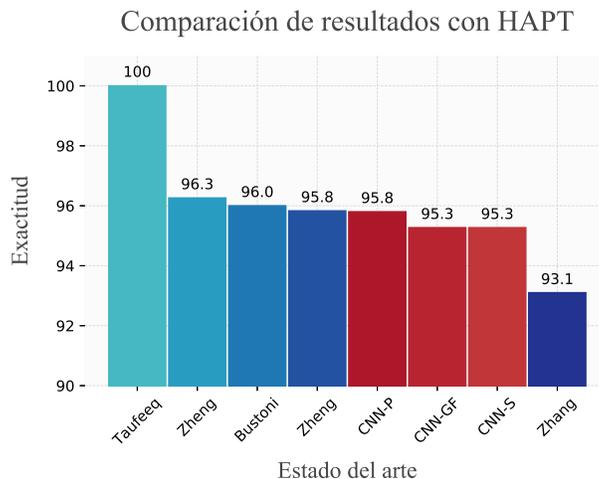


Figura 7.8: Comparación de resultados obtenidos con el estado del arte de la base de datos HAPT.

7.9.3 WISDM v1.1

En la Tabla 7.14 se muestra la comparación de resultados de la base de datos WISDM v1.1. En esta base de datos los resultados sobrepasaron al modelo secuencial, observar que los enfoques basados en DL han tenido mejores resultado. En la Figura 7.9 muestra una gráfica de barras comparando las diferentes técnicas de aprendizaje automático y aprendizaje profundo con la propuesta de este trabajo.

Nombre	Año	Exactitud	Enfoque	Método
Avilés cruz [6]	2019	100.0	DL	CNN
Yafte (CNN-P)	2020	98.92	DL	CNN+genético+paralelo
Xinxin Han [23]	2020	98.83	DL	CNN+FusionTrial
Yan Xu [67]	2017	98.80	ML	ECDF-PCA-MLCF
Daniele Ravi [49]	2017	98.60	DL	CNN
Yafte (CNN-S)	2020	98.30	DL	CNN+genético+secuencial
Mohammad Abu [3]	2016	98.23	DL	CNN
Daniel Ravi [50]	2016	98.20	DL	CNN
Yafte (CNN-GF)	2020	98.12	DL	CNN gruesa-fina
Kishor walse [63]	2016	98.09	ML	Random forrest
Yong Zhang [69]	2018	97.00	DL	U-CNN
Haoxi Zhang [68]	2020	96.40	DL	CNN-multiHead
Jeniffer Kwapisz [30]	2011	91.70	ML	Multi Perceptron
Cagatay Catal [9]	2015	91.60	ML	Ensemble learning
Jeniffer Kwapisz [30]	2011	85.10	ML	J48
Jeniffer Kwapisz [30]	2011	78.10	ML	Regresión logística

Tabla 7.14: Comparación de exactitud de WISDM v1.1 [30].

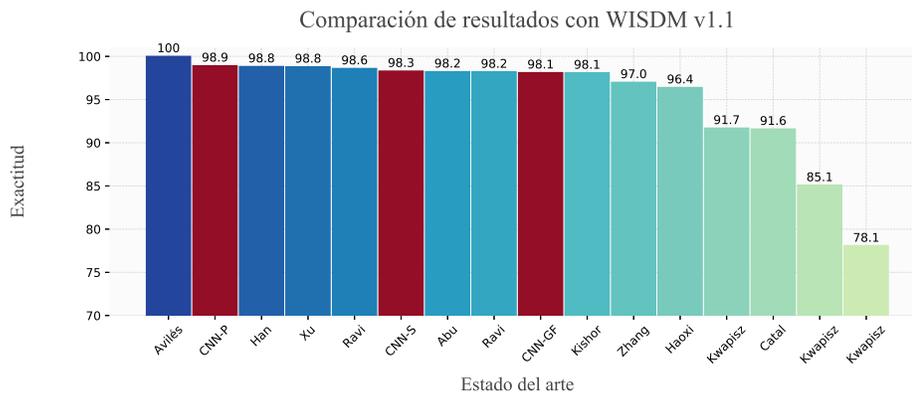


Figura 7.9: Comparación de resultados obtenidos con el estado del arte de la base de datos WISDM v1.1.

7.9.4

WISDM v2.0

En la Tabla 7.15 se muestra la comparación de resultados de la base de datos WISDM v2.0. En la Figura 7.10 muestra una gráfica de barras comparando las diferentes técnicas de aprendizaje automático y aprendizaje profundo con la propuesta de este trabajo.

Nombre	Año	Exactitud	Enfoque	Método
Girmaw Abebe [2]	2017	97.90	DL	CNN+LSTM+Handcrafted
Yafte (CNN-P)	2020	95.40	DL	CNN+genético+paralelo
Yafte (CNN-S)	2020	95.10	DL	CNN+genético+secuencial
Yafte (CNN-GF)	2020	94.75	DL	CNN gruesa-fina
Majid Ali Khan Quaid [47]	2019	94.02	ML	Algoritmo Genético
Andrey Ignatov [25]	2018	93.32	DL	CNN
Daniele Ravi [49]	2017	92.70	DL	CNN
Sarbagya Ratna Shakya [56]	2018	92.22	DL	CNN

Tabla 7.15: Comparación de exactitud de WISDM v2.0 [30] [65] [39].

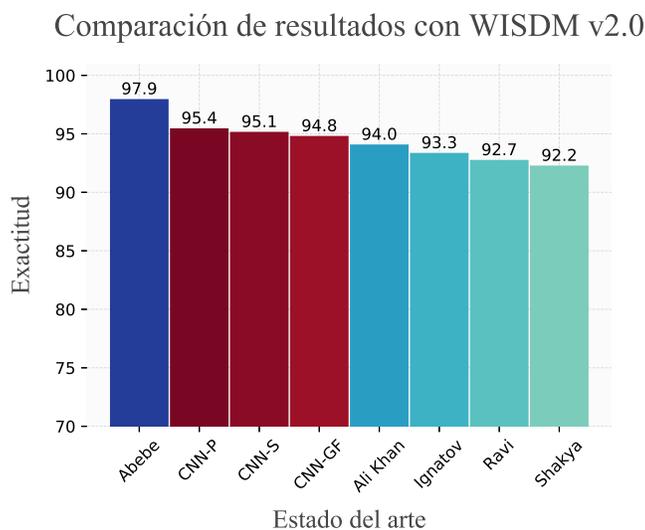


Figura 7.10: Comparación de resultados obtenidos con el estado del arte de la base de datos WISDM v2.0.

7.9.5 MNIST

En la Tabla 7.16 se muestra la comparación de resultados de la base de datos MNIST. Los modelos propuestos se colocaron arriba sobrepasando un número variado de técnicas de DL. En la Figura 7.11 muestra una gráfica de barras comparando las diferentes técnicas de aprendizaje profundo.

Nombre	Año	Exactitud	Enfoque	Método
Li Wan [64]	2013	99.79	DL	CNN Dropconnect
Dan Claudiu Cireşan [13]	2012	99.77	DL	CNN Multi-Column
Ikuro Sato [55]	2015	99.77	DL	CNN APAC
Jian Ren Chang [10]	2015	99.76	DL	CNN Batch
Chen Yu Lee [35]	2015	99.71	DL	CNN Pooling fuction
Ming Liang [36]	2015	99.69	DL	Recurrent CNN
Zhibin Liao [38]	2015	99.69	DL	Normalisation CNN
Benjamin Graham [22]	2014	99.68	DL	CNN Fraccional
Zhibin Liao [37]	2015	99.67	DL	CNN Multi-scale
Dan Claudiu Cireşan [14]	2010	99.65	DL	Big Deep CNN
Yafte (CNN-P)	2020	99.50	DL	CNN+genético+paralelo
Yafte (CNN-S)	2020	99.50	DL	CNN+genético+secuencial
Yafte (CNN-GF)	2020	99.47	DL	CNN gruesa-fina

Tabla 7.16: Comparación de exactitud de MNIST [32].

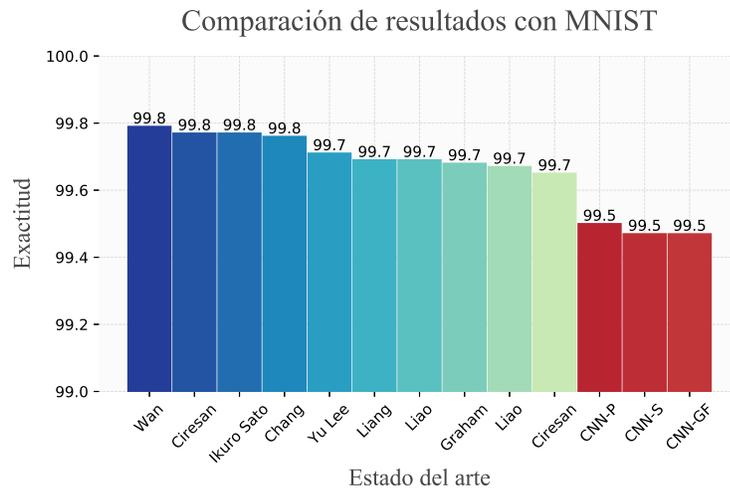


Figura 7.11: Comparación de resultados obtenidos con el estado del arte de la base de datos MNIST.

8

CONCLUSIONES Y TRABAJO FUTURO

En este capítulo final se pretende interpretar, enmarcar las principales aportaciones y resultados del trabajo de investigación presentado, también se describe el rumbo de la investigación, el trabajo futuro a realizar y puntos esenciales que se consideran de vital importancia.

La combinación del cómputo evolutivo y el aprendizaje profundo en redes neuronales ha sido beneficiosa. Las ventajas de optimizar los parámetros de una red neuronal comprende el mejoramiento en calidad de clasificación y selección automática de los parámetros, esto es, reduciendo la intervención experta y el tiempo de ajuste de los parámetros neuronales, la calibración de parámetros puede ser muy fina y requieren conocimiento experto. Notar el poder del cómputo paralelo en conjunto con la utilización de tarjetas gráficas, los resultados demostraron una mejora en tiempo y exactitud.

A pesar de los buenos resultados obtenidos en el uso del cómputo evolutivo, se podría pensar en otro tipo de heurística como; recocido simulado, búsquedas basadas en enjambres, búsqueda dispersa para enfocarnos en la diversificación de la búsqueda o incluso estudiar el comportamiento con simulación Monte Carlo. También sería interesante aplicar la optimización de parámetros en arquitecturas mas simples para realizar un estudio de los parámetros.

APÉNDICES

CONTENIDO:

-
- Apéndice A Código
 - Apéndice B Artículos publicados
 - Apéndice C Glosario
-

A

CÓDIGO

En esta sección se presenta parte del código implementado de la propuesta de investigación. El objetivo es dar un pequeño bosquejo de las técnicas de programación, paquetes y librerías utilizadas. Observar que se utilizó programación orientada a objetos en Python y Keras.

Archivos:

A.1 Principal: Principal.py

A.2 Genético: Genetico.py

A.3 Conjunto de datos: Datos.py

A.4 Red Neuronal Convolutacional: RedNeuronalConvolutacional.py

A.1 Principal

```

1 from __future__ import print_function
2 from mpi4py import MPI
3
4 from time import time
5
6 from Genetico import Genetico
7 from Solucion import Solucion
8
9
10 #Variables de estado paralelo
11 comm = MPI.COMM_WORLD
12 numProcesos = comm.size
13 identificador = comm.rank
14
15 #Variables del problema
16 lider = 0
17
18 #Medicion del tiempo unicial
19 tiempo_inicial = time()
20
21 print( "Inicio del nodo %d" % (identificador) )
22 #print( "Nodo %d: Inicio del algoritmo genetico" % (identificador) )
23
24 #-----Algoritmo Genetico paralelo-----
25 genetico = Genetico()
26 solucion = genetico.ejecutaParalelo( comm, identificador, lider )
27
28 #Comunicacion de la mejor solucion
29 if identificador == lider:
30     print( "Nodo %d: Mejor Solucion Encontra:" % ( identificador ) )
31     solucion.imprime()
32
33 #Medicion del tiempo final
34 tiempo_final = time()
35 tiempo_ejecucion = tiempo_final - tiempo_inicial
36 print( "Nodo %d: El tiempo de ejecucion fue: %f s" % ( identificador,
37     tiempo_ejecucion) )
38
39 #Comunicacion del tiempo final al lider
40 tiempo_ejecucion = comm.gather(tiempo_ejecucion, root = lider)
41
42 if identificador == lider:
43     #print( "Nodo Lider %d: tiempo final de todos es de: %s " % (identificador,
44     tiempo_ejecucion) )
45     tiempo_final_programa = max(tiempo_ejecucion)
46     print( "Nodo Lider %d: Tiempo final del programa es: %s " % (identificador,
47     tiempo_final_programa) )
48
49 #Muerte de los procesos
50 print( "Nodo %d: Termine" % (identificador) )
51 comm.Barrier() # Espera a que todos lleguen aqui

```

A.2 Genético

```

1 from __future__ import print_function
2 from mpi4py import MPI

```

```

3 from Solucion import Solucion
4 from random import randint, uniform
5 from RedNeuronalConvolutacional import RedNeuronalConvolutacional
6 from time import time
7
8 import matplotlib.pyplot as plt
9 import numpy as np
10
11 class Genetico:
12
13     def __init__(self):
14         #Variables Problema
15         self.tamCuerpo = 9
16         self.limiteInferior = 1
17         self.limiteSuperior = 100
18         #Variables del modelo
19         self.memoria = [ ]
20         self.generaciones = 15
21         self.numeroPoblacion = 10
22         self.factorMutacion = 0.1
23         self.porcentajeCruza = 0.6
24         self.porcentajeMutacion = 0.4
25         #Variables de eficiencia
26         self.llamadasFO = 0
27
28
29         #Red Neuronal
30         self.RNNC = RedNeuronalConvolutacional()
31
32     def ejecuta(self):
33         #Ejecucion del algoritmo genetico
34         solucion = self.genetico()
35         return solucion
36
37     def ejecutaParalelo(self, comunicador, identificador, lider):
38         #Ejecucion del algoritmo genetico
39         solucion = self.geneticoParalelo( comunicador, identificador, lider )
40         return solucion
41
42     def geneticoParalelo(self, comunicador, identificador, lider):
43         print("Nodo= " ,identificador, " -----Algoritmo Genetico
44               Paralelo-----")
45         #Inicializa poblacion
46         poblacion = self.inicializaPoblacion( )
47         mejor = self.elitismo( poblacion )
48         #self.imprimePoblacion( poblacion )
49
50         #Para graficar el error
51         #x = range(0, self.generaciones) #generaciones
52         #exactitud = [] #EXactitud
53         #error = [] #Error
54
55         #Seleccion de nueva poblacion
56         for i in range(0, self.generaciones):
57             print("\n-----\n-----Nodo= " ,
58                   identificador, "Generacion ",i , " -----", "\n
59                   -----\n")
60             nuevaPoblacion = [ ]

```

```

59     #print("----- Elitismo -----")
60     #Elitismo
61     individuos = self.elitismo( poblacion + [mejor] ) #Se anade el mejor de
62     todos para ser considerado
63     nuevaPoblacion.append( individuos )
64     self.imprimePoblacion( nuevaPoblacion )
65
66     #print("\n----- Cruzamiento -----")
67     #Cruzamiento
68     individuos = self.cruza( poblacion, self.porcentajeCruza)
69     nuevaPoblacion = nuevaPoblacion + individuos
70     #self.imprimePoblacion( nuevaPoblacion )
71
72     #print("\n----- Mutacion -----")
73     #Mutacion
74     individuos = self.mutacion( poblacion , self.porcentajeMutacion )
75     nuevaPoblacion = nuevaPoblacion + individuos
76
77     #print("\n----- Actualizar -----")
78     #Actualizamos
79     poblacion = nuevaPoblacion
80
81     #Guardamos el mejor
82     #exactitud.append( self.elitismo( poblacion ).valor )
83     #error.append( 1.0-self.elitismo( poblacion ).valor )
84
85     #print("-----Compartir el mejor con todos-----")
86     mejor = self.elitismo( poblacion ) #Todos solicitan el mejor de su
87     poblacion actual
88     mejores = comunicador.gather(mejor, root = lider) # Lo envian al lider.
89     if identificador == lider:
90         mejor = self.elitismo( mejores )
91
92     mejor = comunicador.bcast(mejor, root = lider) #El lider comunica a todos
93     el mejor
94
95     #Graficamos el error
96     #self.graficarError(x,error)
97     #self.graficarExactitud(x,exactitud)
98     #mejor = self.elitismo( poblacion )
99     return mejor
100
101 def genetico(self):
102     print("-----Algoritmo Genetico
103     -----")
104     #Inicializa poblacion
105     poblacion = self.inicializaPoblacion( )
106     #self.imprimePoblacion( poblacion )
107
108     #Para graficar el error
109     x = range(0, self.generaciones) #generaciones
110     exactitud = [] #EXactitud
111     error = [] #Error
112
113     #Seleccion de nueva poblacion
114     for i in range(0, self.generaciones):

```

```

114     print("\n-----\n----- Generacion "
,i , " -----", "\n-----\n")
115     nuevaPoblacion = []
116
117     #print("----- Elitismo -----")
118     #Elitismo
119     individuos = self.elitismo( poblacion )
120     nuevaPoblacion.append( individuos )
121     self.imprimePoblacion( nuevaPoblacion )
122
123     #print("\n----- Cruzamiento -----")
124     #Cruzamiento
125     individuos = self.cruza( poblacion , self.porcentajeCruza)
126     nuevaPoblacion = nuevaPoblacion + individuos
127     #self.imprimePoblacion( nuevaPoblacion )
128
129     #print("\n----- Mutacion -----")
130     #Mutacion
131     individuos = self.mutacion( poblacion , self.porcentajeMutacion )
132     nuevaPoblacion = nuevaPoblacion + individuos
133
134     #print("\n----- Memoria -----")
135     #Guardamos en Memoria
136     self.actualizaMemoria( nuevaPoblacion )
137     #self.imprimePoblacion( self.memoria )
138
139     #print("\n----- Actualizar -----")
140     #Actualizamos
141     poblacion = nuevaPoblacion
142
143     #Guardamos el mejor
144     exactitud.append( self.elitismo( poblacion ).valor )
145     error.append( 1.0-self.elitismo( poblacion ).valor )
146
147     #Graficamos el error
148     self.graficarError(x,error)
149     self.graficarExactitud(x,exactitud)
150     mejor = self.elitismo( poblacion )
151     return mejor
152
153 def graficarError(self, x, y):
154     plt.plot(x,y)
155     # Anado una malla al grafico
156     plt.grid()
157     plt.title('Error')
158     plt.xlabel('Generaciones')
159     plt.ylabel('Error')
160     plt.show()
161
162 def graficarExactitud(self, x, y):
163     plt.plot(x,y)
164     # Anado una malla al grafico
165     plt.grid()
166     plt.title('Exactitud')
167     plt.xlabel('Generaciones')
168     plt.ylabel('Exactitud')
169     plt.show()
170
171 def inicializaPoblacion(self):

```

```
172 poblacion = []
173 #print("-----1. Inicializa poblacion-----")
174 for i in range( self.numeroPoblacion ):
175     cuerpo = [randint(self.limiteInferior,self.limiteSuperior) for _ in range(
176         self.tamCuerpo )] #[setoma:setoma]
177     valor = self.funcionObjetivo( cuerpo )
178     solucion = Solucion( cuerpo, valor )
179     poblacion.append( solucion )
180
181
182     return poblacion
181
182 def funcionObjetivo(self, cuerpo ):
183     valor = self.RNNC.dameSolucion(cuerpo)
184     Exactitud = valor[1]
185     self.llamadasFO = self.llamadasFO + 1
186     return Exactitud
187
188 def elitismo( self, poblacion ):
189     mejor = poblacion[0]
190     for solucion in poblacion:
191         if solucion.valor > mejor.valor :
192             mejor = solucion
193     return mejor
194
195 def cruza( self, poblacion, porcentajeCruza ):
196     tamCruza = int(self.numeroPoblacion*porcentajeCruza)
197     #print("Numero resultante de soluciones con cruza= ", tamCruza )
198     cruza = []
199     for i in range(0, tamCruza, 2 ):
200         tomaUno = randint(1, self.numeroPoblacion) -1
201         tomaDos = randint(1, self.numeroPoblacion) -1
202         #print("1. solucion tomada= ", tomaUno, "\n2. solucion tomada= ", tomaDos
203     )
204     factorCorte = randint( 2, self.tamCuerpo-1 )
205     #print("factor corte= ", factorCorte )
206     solucionUno = poblacion[tomaUno].cuerpo
207     solucionDos = poblacion[tomaDos].cuerpo
208     #self.imprimePoblacion([poblacion[tomaUno],poblacion[tomaDos]])
209     #Definimos los nuevo individuos [ si toca:no toca)
210     solucionUno_a = solucionUno[:factorCorte]
211     solucionUno_b = solucionUno[factorCorte:]
212     solucionDos_a = solucionDos[:factorCorte]
213     solucionDos_b = solucionDos[factorCorte:]
214     #print("1. Solucion A= ", solucionUno_a)
215     #print("1. Solucion B= ", solucionUno_b)
216     #print("2. Solucion A= ", solucionDos_a)
217     #print("2. Solucion B= ", solucionDos_b)
218
219     #Intercambio 1
220     solucionNueva1 = solucionUno_a + solucionDos_b
221     valor1 = self.funcionObjetivo( solucionNueva1 )
222     solucion1 = Solucion(solucionNueva1, valor1)
223
224     #Intercambio 2
225     solucionNueva2 = solucionDos_a + solucionUno_b
226     valor2 = self.funcionObjetivo( solucionNueva2 )
227     solucion2 = Solucion(solucionNueva2, valor2)
228
229     #print("Cruza la familia de 4")
```

```

229     #self.imprimePoblacion([ poblacion[tomaUno], poblacion[tomaDos], solucion1
, solucion2] )
230     #input()
231     #Comparar soluciones
232     mejorSol1, mejorSol2 = self.dameDosMejores( poblacion[tomaUno], poblacion[
tomaDos], solucion1, solucion2 )
233
234     #print("Cruza los mejores 2")
235     #self.imprimePoblacion([ mejorSol1, mejorSol2 ] )
236     #input()
237     cruza.append( mejorSol1 )
238     cruza.append( mejorSol2 )
239
240     #print("Iteracion= ",i," \n Soluciones Cruza")
241     #self.imprimePoblacion(cruza)
242     return cruza
243
244 def dameDosMejores(self, uno,dos,tres,cuatro):
245     mejor1 = []
246     mejor2 = []
247
248     bolsa = [uno.valor, dos.valor, tres.valor, cuatro.valor]
249     #print("Original ")
250     #print(bolsa)
251     orden = np.sort( bolsa )
252     #print("Ordenado")
253     #print(orden)
254     for solucion in [ uno,dos,tres,cuatro ]:
255         if solucion.valor == orden[3]:
256             mejor1 = solucion
257             break
258     for solucion in [ uno,dos,tres,cuatro ]:
259         if solucion.valor == orden[2]:
260             mejor2 = solucion
261             break
262
263     return mejor1, mejor2
264
265
266 def mutacion( self, poblacion, porcentajeMutacion):
267     tamMutacion = int( self.numeroPoblacion*porcentajeMutacion)
268     #print("Numero resultante de soluciones con mutacion= ", tamMutacion )
269     mutacion = []
270     for i in range( tamMutacion ):
271         toma = randint(1, self.numeroPoblacion)-1 #[toma,toma]
272         #print("solucion tomada= ", toma )
273         solucionTomada = poblacion[toma].cuerpo
274         #print("Solucion Tomada= ", solucionTomada )
275         solucionNueva = solucionTomada[:]
276         for j in range( self.tamCuerpo ):
277             aleatorio = uniform(0, 1)
278             if aleatorio <= self.factorMutacion:
279                 #solucionNueva[j] = (solucionNueva[j] + 1) % 2 #Binaria
280                 solucionNueva[j] = randint( self.limiteInferior, self.limiteSuperior )
281         valor = self.funcionObjetivo( solucionNueva )
282         solucion = Solucion(solucionNueva, valor)
283         #print("Solucion mutada= ", solucionNueva)
284         mutacion.append( solucion )
285     return mutacion

```

```

286
287 def actualizaMemoria( self, nuevaPoblacion ):
288     esta = False
289     for solucionNueva in nuevaPoblacion:
290         for solucion in self.memoria :
291             if self.sonIguales(solucionNueva.cuerpo, solucion.cuerpo) :
292                 esta = True
293                 break
294             #ultimo
295             if esta == False :
296                 self.memoria.append( solucionNueva )
297
298 def repetidoMemoria( self, cuerpo ):
299     for i in range( len(self.memoria) ):
300
301         if self.sonIguales(self.memoria[i].cuerpo, cuerpo) :
302             return i
303     return -1
304
305 def imprimePoblacion(self, poblacion):
306     for solucion in poblacion:
307         solucion.imprime()
308
309 def sonIguales( self, lista1, lista2 ):
310     for i in range(len(lista1)):
311         if(lista1[i] != lista2[i]):
312             return False
313     return True

```

A.3 Conjunto de datos

```

1 import sys
2 import os
3
4 #Importar librerias
5 import tensorflow as tf
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from tensorflow.keras.datasets import mnist
9 import tensorflow.keras.backend as K
10
11 class Datos:
12     def __init__( self ):
13         #-----
14         #-MNIST
15         #-----
16         # input image dimensions
17         img_rows, img_cols = 28, 28
18
19         # the data, split between train and test sets
20         (selfinstancias, selfclases), (selfinstanciasPrueba, selfclasesPrueba) =
21         mnist.load_data()
22
23         if K.image_data_format() == 'channels_first':
24             selfinstancias = selfinstancias.reshape(selfinstancias.shape[0], 1,
25             img_rows, img_cols)
26             selfinstanciasPrueba = selfinstanciasPrueba.reshape(self.

```

```

instanciasPrueba.shape[0], 1, img_rows, img_cols)
26     input_shape = (1, img_rows, img_cols)
27     else:
28         self.instancias = self.instancias.reshape(self.instancias.shape[0],
img_rows, img_cols, 1)
29         self.instanciasPrueba = self.instanciasPrueba.reshape(self.
instanciasPrueba.shape[0], img_rows, img_cols, 1)
30         input_shape = (img_rows, img_cols, 1)
31
32     self.instancias = self.instancias.astype('float32')
33     self.instanciasPrueba = self.instanciasPrueba.astype('float32')
34     self.instancias /= 255
35     self.instanciasPrueba /= 255
36     print('x_train shape:', self.instancias.shape)
37     print(self.instancias.shape[0], 'train samples')
38     print(self.instanciasPrueba.shape[0], 'test samples')
39
40
41     #Variables del Problema 5
42     self.numInstancias = len( self.instancias )
43     self.numRegistros = len( self.instancias[0] )
44     self.numAtributos = len( self.instancias[0][0] )
45     self.numCanales = len( self.instancias[0][0][0] )
46     self.numClases = len( np.unique( self.clases ) )
47
48
49     #Variables del Problema 5
50     self.numInstanciasPrueba = len( self.instanciasPrueba )
51     self.numRegistrosPrueba = len( self.instanciasPrueba[0] )
52     self.numAtributosPrueba = len( self.instanciasPrueba[0][0] )
53     self.numCanalesPrueba = len( self.instanciasPrueba[0][0][0] )
54     self.numClasesPrueba = len( np.unique( self.clasesPrueba ) )
55
56 #-----
57 # Carga la base de datos
58 #-----
59 def cargaDatos(self, ruta):
60     #Lee Datos
61     archivo = open( ruta , "r" )
62     datos = []
63     for linea in archivo.readlines():
64         datos.append(linea.split())
65     archivo.close()
66     for i in range( len( datos ) ):
67         for j in range( len( datos[0] ) ):
68             valor = float( datos[i][j] )
69             datos[i][j] = valor
70     return datos
71 #-----
72 # Carga las clases de pruebas
73 #-----
74 def cargaDatosClases(self, ruta):
75     #Lee Datos
76     archivo = open( ruta , "r" )
77     datos = []
78     for linea in archivo.readlines():
79         datos.append(linea.split())
80     archivo.close()
81     for i in range( len( datos ) ):

```

```

82     for j in range( len( datos[0] ) ):
83         valor = int( datos[i][j] )
84         datos[i][j] = valor
85     return datos

```

A.4**Red neuronal convolucional**

```

1  import os
2  import sys
3  import numpy as np
4  import tensorflow as tf
5  from Datos import Datos
6  import matplotlib.pyplot as plt
7  import csv
8  #Quitar mensajes de
9  #0 = all messages are logged (default behavior)
10 #1 = INFO messages are not printed
11 #2 = INFO and WARNING messages are not printed
12 #3 = INFO, WARNING, and ERROR messages are not printed
13 import os
14 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
15
16 class RedNeuronalConvolucional:
17     #-----
18     # Constructor
19     #-----
20     def __init__(self):
21         #CargaDatos
22         datos = Datos()
23
24         #-----Entrenamiento
25         #Instancias
26         self.instancias = datos.instancias
27         #Clases
28         self.clases = datos.clases
29         #Varaibles
30         self.numInstancias = datos.numInstancias
31         self.numRegistros = datos.numRegistros
32         self.numAtributos = datos.numAtributos
33         self.numCanales = datos.numCanales
34         self.numClases = datos.numClases
35         #-----Prueba
36         #Instancias
37         self.instanciasPrueba = datos.instanciasPrueba
38         #Clases
39         self.clasesPrueba = datos.clasesPrueba
40         #Varaibles
41         self.numInstanciasPrueba = datos.numInstanciasPrueba
42         self.numRegistrosPrueba = datos.numRegistrosPrueba
43         self.numAtributosPrueba = datos.numAtributosPrueba
44         self.numCanalesPrueba = datos.numCanalesPrueba
45         self.numClasesPrueba = datos.numClasesPrueba
46
47         #Archivo a escribir
48 #     with open('memoria.csv', 'w') as file:
49 #         fieldnames = ["numEpocas","tamLote", "tamLoteEvaluar","optimizer", "
numDropout","momento","nesterovv", "amsgradd", "weight_decay", "learning_Rate
","tamFiltroFilas","tamFiltroColum","tamMaxFilas","tamMaxColum", "
numFiltroMultiplo","Activacion","f(x)"]

```

```

50 #     self.writer = csv.DictWriter(file, fieldnames=fieldnames)
51 #     self.writer.writeheader()
52
53
54 def transformar( self, Ovalor, Nminimo, Nmaximo):
55     Omaximo = 100
56     Ominimo = 1
57     Otramo = Omaximo - Ominimo
58     Ntramo = Nmaximo - Nminimo
59     Nvalor = ( Ovalor - Ominimo ) * Ntramo / Otramo + Nminimo
60     return Nvalor
61
62 def imprimeParametros( self, parametros):
63     nombres = ["numEpocas", "tamLote", "tamLoteEvaluar", "optimizer", "numDropout"
64     , "momento", "nesterovv", "amsgradd", "weight_decay", "learning_Rate",
65     "tamFiltroFilas", "tamFiltroColum", "tamMaxFilas", "tamMaxColum", "
66     numFiltroMultiplo", "Activacion"]
67     optimizador = ["None", "RMSprop", "Adadelta", "Adamax", "Nadam", "Adam"
68     , "SGD"]
69     activacion = ["Adagrad", "Thang", "relu", "selu", "sigmoid", "relu6"]
70     for i in range( len( parametros ) ):
71         if i == 10 :#Dime Filtro
72             print("i = ",i, "\t", "Filtro", " = ( ", parametros[i], " , ", parametros[i
73             +1], ")")
74         else:
75             if i == 12 : #Dime Maxpool
76                 print("i = ",i, "\t", "MaxPool", " = ( ", parametros[i], " , ", parametros[
77                 i+1], ")")
78             else:
79                 if i == 3 : #optimizador
80                     print("i = ",i, "\t", "Optimizador = ", parametros[i] , " : ",
81                     optimizador [ parametros[i] ])
82                 else:
83                     if i == 15:#activacion
84                         print("i = ",i, "\t", "Activacion = ", parametros[i] , " : ",
85                         activacion [ parametros[i] ])
86                     if i!=10 and i!=11 and i!=12 and i!=13 and i!=3 and i!=15:
87                         print("i = ",i, "\t", nombres[i], " = ", parametros[i])
88                     #print("i = ",i, "\t", "Filtro", " = ( ", parametros[11], " , ", parametros
89                     [12], ")")
90                     #print("i = ",i, "\t", "MaxPool", " = ( ", parametros[13], " , ", parametros
91                     [14], ")")
92
93 #-----
94 # Empieza la Red Neuronal Convolutional
95 #-----
96 def dameSolucion(self, parametros):
97     #Limpiamos memoria para que borre todo los procesos anteriores
98     tf.keras.backend.clear_session()
99
100     #Parametros de la red evolucionados
101     print("parametros sin transformar= ", parametros)
102     #----COMENTARIO paramentos = self.transformar( parametro , inicio, final )
103     -----
104     numEpocas = round( self.transformar( parametros[0], 1, 50 ) )
105     tamLote = round( self.transformar( parametros[1], 100, 500) )
106     #tamLoteEvaluar = round( self.transformar( parametros[3], 100, 500) )
107     optimizer = round( self.transformar( parametros[2], 0, 6) )
108     numDropout = self.transformar( parametros[3], 0, 0.95)

```

```

98     #momento = self.transformar( parametros[5], 0.1, 1.0)
99     #nesterovv = False if round( self.transformar( parametros[6], 0, 1) ) == 0
    else True
100    #amsgradd = False if round( self.transformar( parametros[7], 0, 1) ) == 0
    else True
101    #weight_decay = 5 * pow(10, (-1)*round( self.transformar( parametros[8], 4,
    6) ) )
102    learning_Rate = 5 * pow(10,(-1)*round( self.transformar( parametros[4], 1,
    5) ) )
103    activate = round( self.transformar( parametros[ 5 ], 0, 5) )
104    tamFiltroFilas = round( self.transformar( parametros[ 6 ], 1, 10 ) )
105    #tamFiltroColum = round( self.transformar( parametros[ 2 ], 2, 10 ) )
106    tamMaxFilas = round( self.transformar( parametros[ 7 ], 2, 3 ) )
107    #tamMaxColum = round( self.transformar( parametros[ 2 ], 2, 10 ) )
108    numFiltroMultiplo = round( self.transformar( parametros[ 8 ], 2, 30) )
109
110    #Parametros de la red
111    #numEpocas = 2      #(1,100)
112    #numDropout = 0.9 #(0.1,1.0 )
113    #tamLote = 500     #(100,500)
114    #tamLoteEvaluar = tamLote #(100,500)
115    #optimizer = 2 #(0,6) {0:None ,1:RMSprop ,2:Adadelta ,3:Adamax ,4:Nadam ,5:
    Adam ,6:SGD}
116
117    #Parametros de optimizadores
118    momento = 0.7 #SGD #(0.1, 1.0)
119    nesterovv = True #SGD #(0,1)
120    amsgradd = False #Adam #(0,1)
121    weight_decay = 0.00005 #SGD #(1,5)
122    #learning_Rate = 0.05 #(1,5)
123    activate = 2 #(0,5) {0:Adagrad ,1:Thang ,2:relu ,3:selu ,4:sigmoid ,5:relu6
    }
124    #tamFiltroFilas = 3 #filtro(3,3)
125    tamFiltroColum = tamFiltroFilas #filtro(3,3)
126    #tamMaxFilas = 2 #filtro(1,2)
127    tamMaxColum = tamMaxFilas #filtro(1,2)
128    #numFiltroMultiplo = 18
129
130    #Parametros Extraccion de caracteristicas
131    #Capa Fina
132    numeroFiltrosC1F = numFiltroMultiplo
133    numeroFiltrosC2F = numFiltroMultiplo*2
134    #Capa Gruesa
135    numeroFiltrosC1G = numFiltroMultiplo*2
136
137    # convert class vectors to binary class matrices
138    clases = tf.keras.utils.to_categorical(self.clases, self.numClases)
139    clasesPrueba = tf.keras.utils.to_categorical(self.clasesPrueba, self.
    numClases)
140
141    #Preparar los datos de entrada
142    entrada = tf.keras.Input(shape=( self.numRegistros , self.numAtributos, self
    .numCanales) , name='dataHAR')
143
144    #Definir el tamaño de los filtros
145    dimensionFiltrosF = (tamFiltroFilas,tamFiltroColum)
146    dimensionFiltrosG = (tamFiltroFilas,tamFiltroColum)
147
148    #Definir el tamaño de Maxpooling

```

```

149     dimensionMaxPoolingF = ( tamMaxFilas , tamMaxColum )
150     dimensionMaxPoolingG = ( tamMaxFilas*tamMaxFilas , tamMaxColum*tamMaxFilas )
151
152     #Definimos el paso de MaxPooling
153     pasoMaxPoolingC1F = tamMaxFilas
154     pasoMaxPoolingC2F = tamMaxFilas
155     pasoMaxPoolingC1G = tamMaxFilas*tamMaxFilas
156
157     #Definir funcion de activacion
158     activador = []
159     activador.append( None )
160     activador.append( tf.tanh )
161     activador.append( tf.nn.relu )
162     activador.append( tf.nn.selu )
163     activador.append( tf.sigmoid )
164     activador.append( tf.nn.relu6 )
165
166
167     self.imprimeParametros( [numEpocas,tamLote, tamLoteEvaluar,optimizer,
numDropout,momento,nesterovv, amsgradd, weight_decay, learning_Rate,
tamFiltroFilas ,tamFiltroColum,tamMaxFilas ,tamMaxColum,numFiltroMultiplo,
activate] )
168
169
170     #input()
171
172     #-----
173     #FINA
174     #-----
175     #Capa convolucion 1
176     fina_conv_1 = tf.keras.layers.Conv2D(numeroFiltrosC1F,dimensionFiltrosF,
padding = 'same',activation = activador[activate],input_shape = (self.
numRegistros, self.numAtributos, self.numCanales)) (entrada)
177     fina_max_1 = tf.keras.layers.MaxPooling2D( dimensionMaxPoolingF, strides =
pasoMaxPoolingC1F)( fina_conv_1 )
178     #Capa convolucion 2
179     fina_conv_2 = tf.keras.layers.Conv2D(numeroFiltrosC2F,dimensionFiltrosF,
padding = 'same', activation=activador[activate]) (fina_max_1)
180     fina_max_2 = tf.keras.layers.MaxPooling2D(dimensionMaxPoolingF, strides=
pasoMaxPoolingC2F) (fina_conv_2)
181
182     fina = tf.keras.Model(entrada, fina_max_2, name='Fina')
183
184
185
186     #-----
187     #GRUESA
188     #-----
189     #Capa convolucion 1
190     gruesa_conv_1 = tf.keras.layers.Conv2D(numeroFiltrosC1G,dimensionFiltrosG,
padding = 'same', activation=activador[activate], input_shape = (self.
numRegistros, self.numAtributos, self.numCanales)) (entrada)
191     gruesa_max_1 = tf.keras.layers.MaxPooling2D(dimensionMaxPoolingG, strides=
pasoMaxPoolingC1G) (gruesa_conv_1)
192     gruesa = tf.keras.Model(entrada, gruesa_max_1, name='Gruesa')
193
194     #Combinar capas
195     merged = tf.keras.layers.concatenate([ fina.output, gruesa.output ])
196     #-----

```

```

197 #Capa de clasificacion
198 #-----
199 aplanado = tf.keras.layers.Flatten() ( merged )
200 numeroNeuronas = aplanado.get_shape()[1]
201 densa = tf.keras.layers.Dense( numeroNeuronas, activation= activador[
activate] ) ( aplanado )
202 drop = tf.keras.layers.Dropout( numDropout ) (densa)
203 soft = tf.keras.layers.Dense(self.numClases, activation=tf.nn.softmax) (
drop)
204
205 #-----
206 #Construye modelo
207 #-----
208 modelo = tf.keras.Model(inputs=[entrada], outputs=soft)
209
210 #-----
211 #compilar el modelo
212 #-----
213 optimizador = []
214 optimizador.append( tf.keras.optimizers.Adagrad( learning_rate=
learning_Rate))
215 optimizador.append( tf.keras.optimizers.RMSprop( learning_rate= 0.01, rho
=0.90))
216 optimizador.append( tf.keras.optimizers.Adadelta(learning_rate= 1.00, rho
=0.95))
217 optimizador.append( tf.keras.optimizers.Adamax( learning_rate=
learning_Rate, beta_1=0.9, beta_2=0.999))
218 optimizador.append( tf.keras.optimizers.Nadam( learning_rate=
learning_Rate, beta_1=0.9, beta_2=0.999))
219 optimizador.append( tf.keras.optimizers.Adam( learning_rate=
learning_Rate, beta_1=0.9, beta_2=0.999, amsgrad=amsgradd ))
220 optimizador.append( tf.keras.optimizers.SGD( learning_rate=
learning_Rate, momentum=momento, decay=weight_decay, nesterov=nesterovv))
221
222 modelo.compile( optimizer = optimizador[ optimizer ], loss='
categorical_crossentropy', metrics = ['acc'])
223
224 #-----
225 #Entrenar Modelo
226 #-----
227 history = modelo.fit(self.instancias, clases, validation_split= 0.1,
shuffle = True ,batch_size = tamLote, epochs = numEpocas, verbose=1)
228 #history = modelo.fit(self.instancias, self.clases, shuffle = True,
batch_size = tamLote, epochs = numEpocas)
229 #-----
230 #Evaluar de la red
231 #-----
232 resultado = modelo.evaluate(self.instanciasPrueba, clasesPrueba, batch_size
= tamLoteEvaluar, verbose = 0)
233
234 #-----
235 #Guardamos la red entrenada
236 #-----
237 # dir='./modelo/'
238 #
239 # if not os.path.exists(dir):
240 #     os.mkdir(dir)
241 #     modelo.save('./modelo/modelo.h5')
242 #     modelo.save_weights('./modelo/pesos.h5')

```

```
243
244 #-----
245 #INFORMACION
246 #-----
247 #modelo.summary()
248
249 #Guardamos imagen
250 # tf.keras.utils.plot_model(modelo, to_file='my_model.png')
251
252 # #Graficar exactitud de entrenamieto y validacion
253 # plt.plot(history.history['acc'])
254 # plt.plot(history.history['val_acc'])
255 # plt.title('Modelo de exactitud')
256 # plt.ylabel('Exactitud')
257 # plt.xlabel('Epocas')
258 # plt.legend(['Entrenamiento', 'Prueba'], loc='upper left')
259 # plt.show()
260
261 # #Graficar perdida de entrenamieto y validacion
262 # plt.plot(history.history['val_loss'])
263 # plt.title('Modelo Perdida')
264 # plt.ylabel('Perdida')
265 # plt.xlabel('Epocas')
266 # plt.legend(['Entrenamiento', 'Prueba'], loc='upper left')
267 # plt.show()
268
269 #Guardar en memoria
270 param = np.array( [[numEpocas,tamLote, tamLoteEvaluar,optimizer, numDropout,
momento,nesterovv, amsgradd, weight_decay, learning_Rate,tamFiltroFilas,
tamFiltroColum,tamMaxFilas,tamMaxColum,numFiltroMultiplo,activate,resultado
[1]] ] )
271 nombres = ["numEpocas","tamLote", "tamLoteEvaluar","optimizer", "numDropout"
,"momento","nesterovv", "amsgradd", "weight_decay", "learning_Rate", "
tamFiltroFilas","tamFiltroColum","tamMaxFilas","tamMaxColum","
numFiltroMultiplo","Activacion","f(x)"]
272
273 with open('memoria.csv', 'a+') as csvfile:
274     writer = csv.writer(csvfile)
275     writer.writerow(param)
276     print(" Escritura completa ")
277
278
279
280 return resultado
```


B

ARTÍCULOS PUBLICADOS

En esta sección parte del apéndice se mostrarán los artículos publicados durante el desarrollo del trabajo de investigación, además se incluirán los detalles de congresos y el pre-impresión de los artículos publicados.

Artículo:

COMIA 2020: Implementación de una red neuronal profunda en tres etapas paralelas

Artículo de investigación sometido al Congreso Mexicano de Inteligencia Artificial COMIA 2020, celebrado el 5 al 7 de agosto del 2020, bajo la organización de la Sociedad Mexicana de Inteligencia Artificial (SMIA), y la Universidad de Cd Juárez, Chihuahua. El artículo fue sometido a revisión ciega entre pares y aceptado para publicarse en el número especial de la revista indexada *Research in Computing Science*, del Centro de Investigación en Computación, del Instituto Politécnico Nacional; con la referencia tentativa a completar con la fecha de publicación como:

Yafte A. Flores-Morales et al. *Implementacion de una red neuronal profunda en tres etapas paralelas para el reconocimiento de actividades humanas e imagenes*. Por publicarse en: *Research in Computing Science* (indexada en: Latindex, DBPL) Aceptado: 1 de julio, 2020. ISSN 18704069.

Adicionalmente, el trabajo de investigación fue seleccionado para presentarse en ponencia en el marco de Visión por Computadora y Reconocimiento de Patrones. A continuación se mostrará el artículo aceptado.

Implementación de una red neuronal profunda en tres etapas paralelas para el reconocimiento de actividades humanas e imágenes

Yafte A. Flores-Morales¹, Juan Villegas-Cortez², Graciela Roman-Alonso¹, Arturo Zúñiga-López³, César Benavides-Álvarez¹, Salomón Cordero-Sánchez⁴

¹ Universidad Autónoma Metropolitana, Unidad Iztapalapa. Departamento de Ingeniería Eléctrica. Av. San Rafael Atlixco 186, Col. Vicentina, Cd de México 09340, México

² Universidad Autónoma Metropolitana, Unidad Azcapotzalco. Departamento de Sistemas. Av. San Pablo 180, Reynosa Tamaulipas, Cd de México 02200, México

³ Universidad Autónoma Metropolitana, Unidad Azcapotzalco. Departamento de Electrónica, Av. San Pablo 180, Reynosa Tamaulipas, Cd de México 02200, México

⁴ Universidad Autónoma Metropolitana, Unidad Iztapalapa. Departamento de Química. Av. San Rafael Atlixco 186, Col. Vicentina, CP 09340, Cd. de México, México.

yafteaaron@xanum.uam.mx, juanvc@azc.uam.mx, groman@xanum.uam.mx, azl@azc.uam.mx, cesarbenavides32@gmail.com, scordero@xanum.uam.mx

Resumen La importancia de la aplicación del reconocimiento de patrones para el reconocimiento de imágenes, aumenta cuando el número y la complejidad de las imágenes aumenta. Las redes neuronales artificiales han demostrado su efectividad para atacar problemas de patrones complejos en los últimos quince años aproximadamente y han evolucionado hacia arquitecturas híbridas profundas. En este artículo presentamos la implementación de una red neuronal profunda en tres capas paralelas, con una alimentación paralela de los patrones a tres niveles de granularidad: fina, mediana y gruesa; buscando conformar un análisis del patrón característico y justo aplicado a imágenes. La red profunda de la cual partimos hemos modificado y orientado, de patrones de reconocimiento de actividades humanas (HAR), hacia atacar la complejidad de las imágenes, con resultados prometedores del 99% de reconocimiento en las pruebas realizadas.

Keywords: RN, aprendizaje profundo, Clasificación, Reconocimiento de patrones, HAR, Reconocimiento de imágenes

Implementation of Three Parallel Layer Deep Neural Network for Human Activity and Image Recognition

Abstract. The recognition of patterns for the recognition of faces becomes an issue of extreme importance when the number and complexity of images increases. For the last fifteen years, artificial neural networks have shown its effectiveness to address complex pattern problems at the same time that they have evolved

to hybrid and deep architectures. This paper presents the implementation of a deep neural network developed in three parallel layers, and with parallel feeding of patterns with three levels of granularity: fine, medium and thick. This results in conforming an analysis of the characteristic robust pattern applied in the images. The initial deep network was modified and oriented to address the complexity of the images by integrating human activity recognition (HAR). The results are promising achieving 99 % of recognition of the performed tests.

Keywords. Deep Neural Networks, Deep Learning, Pattern recognition, HAR, Image Recognition.

1. Introducción

El área de reconocimiento de patrones en la inteligencia artificial ha ganado mucha importancia en los últimos años debido al tratamiento automático de grandes cantidades de datos complejos. El reconocimiento de patrones (RP) extrae de un conjunto de datos la información que permite establecer propiedades, vínculos y relaciones entre los datos, estas relaciones con los patrones permiten una descripción estructural o cuantitativa del objeto o de alguna otra entidad de interés en los datos [10, 14].

La aplicación del RP a diferentes estructuras de datos como señales o imágenes, ha dado lugar al desarrollo de una gran variedad de modelos, técnicas y algoritmos en las áreas de aprendizaje automático del inglés Machine Learning o ML) y aprendizaje profundo del inglés Deep Learning o DL), que permiten extraer patrones de las estructuras de datos con cierta eficiencia y eficacia. Algunas aplicaciones del reconocimiento de patrones son, por ejemplo, algoritmos sofisticados para reconocimiento de voz en los dispositivos móviles, o los videojuegos basados en realidad aumentada donde pueden detectar los movimientos del jugador por medio de una cámara. Sin embargo, reconocer movimientos de una persona no es tan fácil. La tarea de reconocer y clasificar acciones de un grupo de imágenes en el área de la IA, se le llama Reconocimiento de Actividad Humana (del inglés Human Action Recognition o HAR). En este artículo se propone implementar una red neuronal profunda con diferentes niveles de extracción de características basado en la red neuronal grueso-a-fino⁵ en [4] para el área HAR y el reconocimiento de patrones en imágenes.

El objetivo de este trabajo es dos puntos: (i), se busca evaluar la red neuronal basada en [4] con dos diferentes bases de datos WISDM v2.0 y HAPT, mismas que no se habían considerado previamente; y (ii), se prueba el rendimiento de la arquitectura de la red neuronal en el reconocimiento de imágenes.

El documento está organizado de la siguiente manera: en la Sección 2 presentamos el estado del arte del problema de la caracterización de HAR y el reconocimiento de imágenes en grandes cantidades desde la perspectiva del reconocimiento de patrones y de visión por computadora; en la Sección 3 presentamos la implementación y adecuación de la metodología de las bases de datos escogidas para ejemplificar los alcances. La Sección 4 comparte los resultados de los

⁵ Coarse-fine N. del T.

experimentos y una discusión de los mismos y por ultimo, en la Sección 5, las conclusiones y perspectivas de trabajo futuro son compartidos.

2. Estado del Arte

En la literatura existen dos enfoques principales para tratar la tarea HAR, el primer enfoque se basa en reconocer diferentes actividades por medio de cámaras de video, Jalal [16] obtiene buenos resultados con este enfoque, utiliza dispositivos de vídeo para reconocer actividades realizadas por un individuo. Sin embargo, el costo de procesar video reduce la eficiencia de los algoritmos por el alto consumo de memoria y cómputo, debido a esto muchos autores utilizan diferentes técnicas capaces de procesar el gran volumen de datos producidos por sensores de video, Martinez [22] utiliza visión por computador y otros enfoques para el reconocimiento de actividades obteniendo resultados de 90%. El segundo enfoque se basa en la utilización de sensores como geoscopio y acelerómetro colocados en diferentes partes del individuo trabajando con señales obtenidas de estos sensores. En esta propuesta nos centramos en este segundo enfoque.

El área de investigación de HAR ha recibido atención debido a la tendencia creciente de sus aplicaciones en diferentes áreas, la reducción en el precio de los sensores integrados en los dispositivos portátiles y la simplificación de su producción [18]. Diferentes artículos realizan estudios sobre características importantes de los sensores; tipo, costo, capacidad y cantidad. Sin embargo, en la mayoría de estos se utilizan teléfonos inteligentes debido a la relación calidad-precio y la accesibilidad [27].

El HAR es un problema tratado por diferentes técnicas de la IA, el ML y el DL. Existen métodos basados en ML como máquinas de soporte vectorial o árboles de decisión. Estos métodos utilizan características estadísticas como media, mínimo, máximo, desviación estándar, asimetría, curtosis, ángulos, entropía, etc [9, 29].

Un enfoque diferente para la tarea de extracción de características se basa en DL donde se centra en la realización de Redes Neuronales Convolucionales (del inglés Convolutional Neural Network o CNN), un tipo particular de las redes neuronales artificiales caracterizada por el uso de la convolución entre señales de entrada y valores discretos. Existen diferentes tipos de redes neuronales artificiales, la CNN utiliza como operación principal la convolución para la extracción de características de los datos, estas características representan rasgos particulares de los datos.

La importancia de las CNN radica en el tratamiento de dos puntos principales: *(i)* La extracción de características más robustas a partir de la implementación de la operación de convolución, y *(ii)* la posibilidad de trabajar con una gran cantidad de patrones complejos en longitud o cantidad de atributos, así como el manejo de valores en punto flotante.

Existen diversos conjuntos de datos para el HAR que ayudan a medir el rendimiento de los modelos. En este artículo se ocuparon cuatro conjuntos de datos: UCI HAR [2], WISDM v1.1 [17], WISDM v2 [17, 32] y HAPT [25].

2.1. Métodos basados en ML

UCI HAR, Anguita [3] mediante máquinas de soporte vectorial logra atacar el problema HAR obteniendo buenos resultados, Xiangbin Zhu [35] utiliza cadenas de markov. UCI HAPT, Taufeeq [28] obtiene excelentes resultados utilizando arboles de decisión. Zheng [34] utiliza máquinas de soporte vectorial combinado un modelo de agrupación dispersa de dos capas. WIDSM v1.1 Kishor walse [30] utiliza árboles de decisión. Cagatay Catal [5] realiza una combinación de diferentes técnicas, árboles de decisión, regresión logística y perceptrón multicapa. WIDSM v2, Majid Ali Khan Quaid [23] por medio de un algoritmo genético logra alcanzar buenos resultados.

2.2. Métodos basados en DL

HAR, Avilés Cruz *et al.* [4] obtienen el 100% utilizando una CNN con diferentes niveles de alimentación paralela. Cho *et al.* [6] utilizan el paradigma de divide y vencerás y una CNN para identificar acciones realizadas por el usuario. Yong Zhang *et al.* [33] construyeron una CNN capaz de adaptarse a diferentes longitudes de datos. En HAPT, Yong Zhang *et al.* [33] es el único trabajo encontrado con enfoque DL, construyeron una CNN capaz de adaptarse a diferentes longitudes de datos.

WIDSM v1.1 Avilés Cruz *et al.* [4] obtienen el 100% utilizando una CNN con 3 diferentes niveles de alimentación paralela. Jinxin Han [13] utiliza una CNN con tres niveles de alimentación. Dan Claudiu Cireşan [7] utiliza una CNN tradicional probando diferentes capas de resolución. WIDSM v2 Girmaw Abebe [1] obtiene un buen resultado combinando una CNN con memorias de largo plazo (LSTM). Ignatov [15] utiliza una CNN tradicional con métodos de preprocesamiento. MNIST, Li Wan [31] por medio de una CNN y un enfoque Dropconnect obtiene un buen resultado. Dan Claudiu Cireşan [7] construye una red neuronal multicolumna. Ming Liang [21] por medio de una red neuronal recurrente logra mejores resultados que una CNN clásica.

En las Tablas 1 a la 5 se presentan los trabajos mencionados de las diferentes técnicas de ML y DL.

Tabla 1. Comparación de exactitud de UCI HAR [2]

Nombre	Año	Exactitud %	Método
Avilés Cruz [4]	2019	100.0	CNN
Yong Zhang [33]	2018	98.4	U-CNN
San-Segundo [26]	2016	98.0	Marcov+ Estadística
Sergey Ignatov [15]	2018	97.63	CNN

3. Propuesta

La propuesta es implementar una CNN con diferentes niveles de extracción paralela de características, estos niveles de extracción están divididos en grueso,

Tabla 2. Comparación de exactitud de HAPT [25]

Nombre	Año	Exactitud %	Método
Taufeeq [28]	2016	100.0	Random Forrest
Zheng [34]	2018	96.26	TASG+SVM
Zheng [34]	2018	95.83	TASG+RNN
Yong Zhang [33]	2018	93.1	U-CNN

Tabla 3. Comparación de exactitud de WISDM 1.1v [17]

Nombre	Año	Exactitud %	Método
Avilés Cruz [4]	2019	100	CNN
Xinxin Han [13]	2020	98.83	CNN+Fusion
Daniel Ravi [24]	2017	98.2	CNN
kishor walse [30]	2016	98.09	Random forrest
Yong Zhang [33]	2018	97.0	U-CNN

Tabla 4. Comparación de exactitud de WISDM 1.0v [17] [31]

Nombre	Año	Exactitud %	Método
Girmaw Abebe [1]	2017	97.9	CNN+LSTM
Majid Ali Khan Quaid [23]	2019	94.6	Genético
andrey Ignatov [15]	2018	92.7	CNN tamaño 100
Daniele Ravi [24]	2017	92.7	CNN

Tabla 5. Comparación de exactitud de MNIST [19]

Nombre	Año	Exactitud %	Método
Li Wan [31]	2015	99.79	CNN Dropconnect
Dan Claudiu Cireşan [7]	2012	99.57	CNN Multi-Column
Benjamin Graham [12]	2014	99.68	CNN Fraccional
Dan Claudiu Cireşan [7]	2010	99.65	Big Deep CNN

medio y fino. En la Figura 1 se muestra la arquitectura de la red neuronal [4], la idea principal está basada en extraer diferentes niveles de características, donde el nivel fino extrae características más sutiles de los datos, el nivel medio extrae características más robustas y el nivel grueso extrae características más burdas como líneas, círculos, colores, etc.

La arquitectura presentada en la Figura 1 consta de tres niveles:

1. **Fino:** Está compuesto por cuatro capas de convolución, cada capa de convolución es seguida por una capa de agrupación máxima, los mapas de características son reducidos en cada capa. Las primeras dos capas de convolución cuentan con 18 filtros y las últimas dos capas cuentan con 36 filtros, el tamaño del filtro es de (1×2) para todas las capas de convolución. La ventana de agrupación máxima es de (1×2) y el paso es de 2.

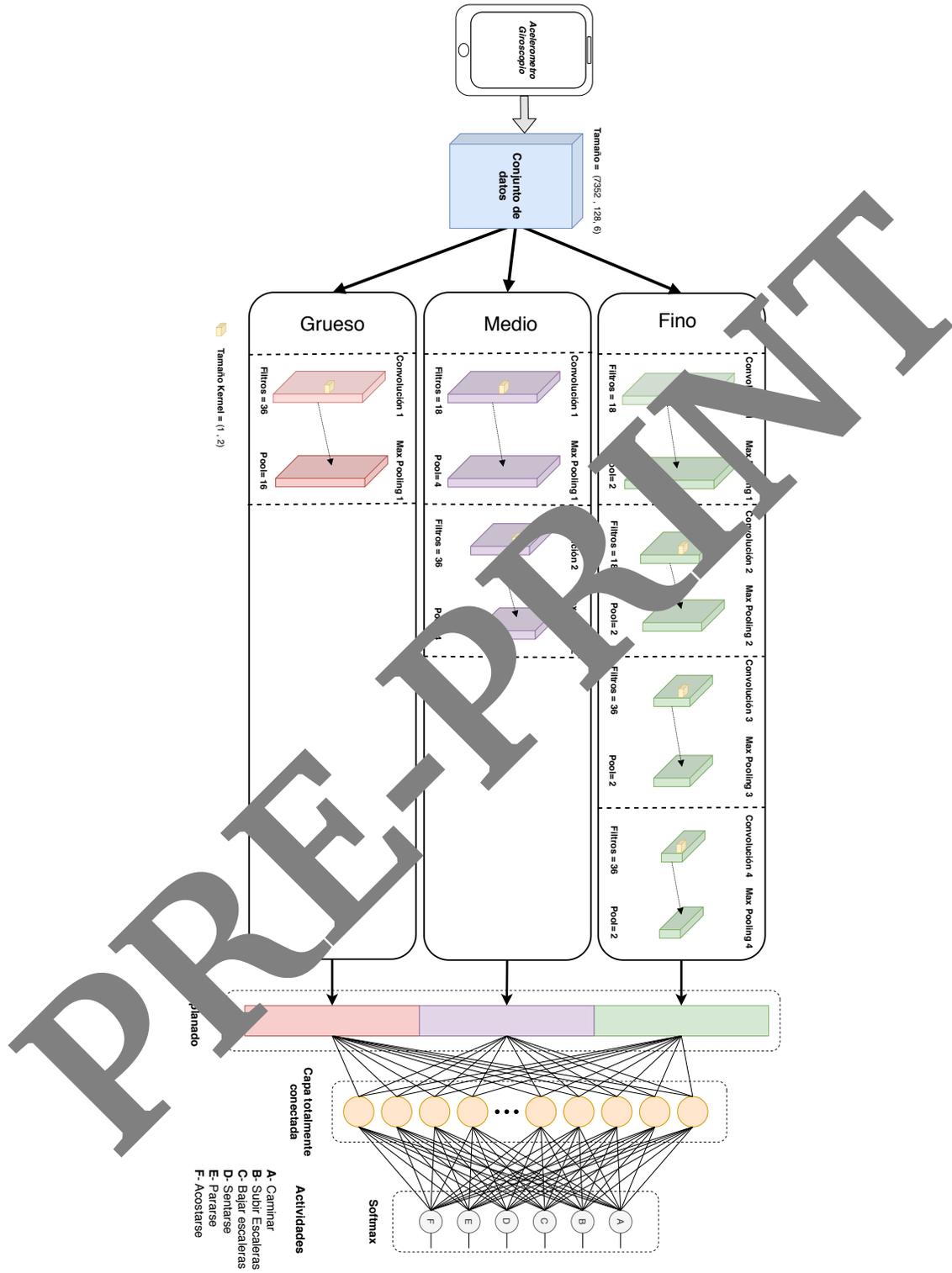


Fig. 1. Arquitectura de la red neuronal propuesta aplicada al reconocimiento de actividad humana [4].

2. **Medio:** Está compuesto por dos capas de convolución, cada capa de convolución es seguida de una capa de agrupación máxima, los mapas de características son reducidos en cada capa. La primera capa de convolución cuenta con 18 filtros y la última cuenta con 36 filtros, el tamaño del filtro es de (1×2) para todas las capas de convolución. La ventana de agrupación máxima es de (1×4) y el paso es de 4.
3. **Grueso:** Está compuesto por una capa de convolución, la capa de convolución es seguida por una capa de agrupación máxima. la capa de convolución cuenta con 36 filtros de tamaño (1×2) . La ventana de agrupación máxima es de (1×16) y el paso es de 16.

La salida de los tres niveles se agrupan y se aplanan, formando un vector que pasa a una etapa de clasificación por medio de una red neuronal totalmente conectada. Finalmente la salida de la capa totalmente conectada se pasa a una capa softmax que calcula la distribución de probabilidad de las clases [2]. En la Tabla 6 se muestran los parámetros utilizados por la red neuronal propuesta.

Tabla 6. Parámetros establecidos para la red neuronal profunda gruesa-fina

Parámetros	Valores	Tipo
Número de épocas	50	Entero
Tamaño de lote	300	Entero
Dropout	0.5	Flotante
Función activación	ReLU	Función
Función pérdida	Entropía cruzada	Función
Numero de filtros	18,36	Entero
Dimensión filtro (filas)	1	Entero
Dimensión filtro (columnas)	2	Entero
Dimensión maxpooling (filas)	1	Entero
Dimensión maxpooling (columnas)	2,4,16	Entero
Paso de maxpooling	2,4,16	Entero
Optimizador	Adadelta	Función

4 Implementación

La implementación se realizó en una computadora tipo Workstation con las siguientes características: sistema operativo Linux, 8 GB RAM, procesador Intel Core i7, GPU NVIDIA GTX-1050. Esta sección describe y muestra las pruebas realizadas de la red neuronal profunda gruesa-fina usando diferentes conjuntos de datos, como se detallan a continuación en su generalidad.

4.1 Conjuntos de datos

En los experimentos se utilizaron dos tipos de conjunto de datos: 1. Reconocimiento de actividad humana; UCI HAR [2], WISDM v1.1 [17], WISDM v2 [17] [32] y HAPT [25]. 2. Reconocimiento de imágenes; MNIST [19].

UCI HAR es un conjunto de señales preprocesadas de seis distintas actividades humanas: caminar, subir escaleras, bajar escaleras, sentarse, pararse, acostarse. Estas señales fueron tomadas con un teléfono inteligente colocado en la cintura de treinta voluntarios, se tomaron muestras del giroscopio y acelerómetro integrados en el teléfono con una velocidad de 50 Hz . Las señales del giroscopio y del acelerómetro están compuestas por tres ejes, (X, Y, Z) . En la Tabla 7 se muestra la cantidad de instancias por clase.

Tabla 7. Número de instancias por clase del conjunto de datos HAR

Actividad	Entrenamiento	Porcentaje	Prueba	Porcentaje
Bajar escaleras	986	13.4 %	420	14.25 %
Subir escaleras	1,073	14.6 %	471	15.98 %
Caminar	1,226	16.7 %	496	16.83 %
Sentarse	1,286	17.5 %	491	16.66 %
Pararse	1,374	18.7 %	532	18.37 %
Acostarse	1,407	19.1 %	537	18.52 %
Total	7,352	100 %	2,977	100 %

Human Activities and Postural Transitions (HAPT) es una actualización de HAR, tiene doce clases: caminar, subir escaleras, bajar escaleras, sentarse, pararse, acostarse, pararse a sentarse, sentarse a pararse, sentarse a acostarse, acostarse a sentarse, pararse a acostarse, acostarse a pararse. HAPT proporciona un conjunto de datos no procesados, se realizó el preprocesamiento aplicando el método de ventana deslizante con un paso de 5 y tamaño de ventana de 128 muestras equivalente a 6.4 segundos, se aplicó un filtro Butterworth de tercer orden con frecuencia de corte de 10 Hz . En la Tabla 8 se muestra la cantidad de instancias por clase del conjunto de entrenamiento y del conjunto prueba.

Tabla 8. Número de instancias por clase del conjunto de datos HAPT procesada y particionada

Actividad	Procesada	Entrenamiento	Prueba
Sentarse-Pararse	123	93	30
Pararse-Sentarse	155	108	47
Acostarse-Sentarse	169	111	59
Acostarse-Pararse	170	118	51
Sentarse-Acostarse	195	142	53
Pararse-Acostarse	223	157	66
Bajar escaleras	1,691	1,202	489
Subir escaleras	1,817	1,227	590
Caminar	1,905	1,314	591
Sentarse	1,983	1,421	562
Acostarse	2,144	1,503	641
Pararse	2,167	1,523	644
Total	12,742	8,919	3,823

Wireless Sensor Data Mining (WISDM) v1.1, contiene 1,098,204 señales no

preprocesadas de seis diferentes actividades humanas: caminar, subir escaleras, bajar escaleras, sentarse, pararse, trotar. Estas señales fueron tomadas con un teléfono inteligente que midió la aceleración en tres ejes (X,Y,Z) con una frecuencia de 20 Hz. Para el preprocesamiento de las señales se utilizó el método de ventana deslizante con empalme de 50% y un tamaño de ventana de 128 muestras equivalente a 6.4s, se aplicó un filtro de Butterworth de tercer orden con frecuencia de corte de 10 Hz. La Tabla 9 muestra la cantidad de instancias.

Tabla 9. Número de instancias por clase del conjunto de datos WISDM v1.1

Actividad	Sin procesamiento	Procesada	Entrenamiento	Prueba
Pararse	48,395	757	544	213
Sentarse	59,939	936	650	256
Bajar escaleras	100,427	1,565	1,089	476
Subir escaleras	122,869	1,927	1,327	600
Trotar	342,176	5,346	3,737	1,609
Caminar	424,398	6,627	4,663	1,964
Total	1,098,204	17,158	12,010	5,148

WISDM v2.0 es una actualización del conjunto de datos WISDM v1.1, es un conjunto de 2,980,765 señales no preprocesadas de seis diferentes actividades humanas: caminar, escaleras, sentarse, pararse, trotar, pararse. Estas señales fueron tomadas con un teléfono inteligente que midió la aceleración en tres ejes (X,Y,Z) con una frecuencia de 20 Hz. Se utilizó el mismo preprocesamiento de las señales aplicado en WISDM v1.1. La Tabla 10 muestra la distribución de estas instancias por clase. La Tabla 11 compara las clases contenidas en cada conjunto de datos.

Tabla 10. Número de instancias por clase del conjunto de datos WISDM v2.0 procesada y particionada

Actividad	Sin procesar	Procesada	Entrenamiento	Prueba
Escaleras	57,225	896	644	252
Acostarse	15,967	4,315	2,972	1,343
Pararse	288,873	4,512	3,141	1,371
Trotar	438,871	6,859	4,833	2,026
Sentarse	663,706	10,373	7,336	3,037
Caminar	1,255,92	19,618	13,675	5,943
Total	2,980,765	46,573	32,601	13,972

Mnist (Modified National Institute of Standards and Technology (MNIST) [20], es una base de datos de números escritos a mano, consta de 10 clases que representan los números del 0 al 9. Los dígitos están representadas por medio de imágenes en escala de grises con un tamaño de 28×28 píxeles. En la Tabla 12 se muestra la cantidad de instancias por clase.

Tabla 11. Clases contenidas en los conjuntos de datos

Dataset	HAPT	UCI HAR	WISDM v1.1	WISDM v2.0
Pararse-Sentarse	*			
Sentarse-Pararse	*			
Sentarse-Acostarse	*			
Acostarse-Sentarse	*			
Pararse-Acostarse	*			
Acostarse-Pararse	*			
Subir escaleras	*	*	*	
Bajar escaleras	*	*	*	
Acostarse	*	*		*
Caminar	*	*	*	*
Sentarse	*	*	*	*
Pararse	*	*	*	*
Trotar			*	*
Escaleras				*
Total clase	12	6	6	

Tabla 12. Número de instancias por clase de la base de datos MNIST

Dígito	Entrenamiento	Prueba
Cero	5,923	
Uno	6,742	1,101
Dos	5,958	1,009
Tres	6,131	1,210
Cuatro	5,797	1,122
Cinco	5,801	1,100
Seis	5,911	958
Siete	6,267	1,028
Ocho	5,851	974
Nueve	6,732	1,009
Total	60,603	10,000

4.2. Análisis de resultados

El análisis de resultados se llevó a cabo por medio de dos etapas; entrenamiento y prueba. La primera consiste en entrenar la red neuronal propuesta con el conjunto de entrenamiento. Por otro lado, la segunda consiste en evaluar la red neuronal por medio del conjunto de prueba, el cual es ajeno al conjunto de entrenamiento. Para medir el rendimiento de la red neuronal gruesa-fina se realizaron métricas de exactitud.

Resultados del conjunto de datos UCI HAR: En la Figura 2a se muestra la evaluación de los conjuntos de entrenamiento y prueba. Se alcanzó el 100 % de recuperación y el parámetro de pérdida alcanzó el valor 0 % al evaluar el conjunto de prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento fue de 4.83 minutos.

Resultados del conjunto de datos HAPT: HAPT resulta un conjunto de datos difícil, ya que no cuenta con un equilibrio en el número de instancias por clase (como se ve en la Tabla 8), sin embargo, se alcanzó una exactitud

del 95.27% con una pérdida de 4.73% al evaluar el conjunto de prueba. En la Figura 2b se muestra la evaluación de los conjuntos de entrenamiento y prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento fue de 5.30 minutos.

Resultados del conjunto de datos WISDM v1.1: Se alcanzó una recuperación de 98.12% y valor de pérdida de 1.88% al evaluar el conjunto de prueba. En la Figura 2c se muestra la evaluación de los conjuntos de entrenamiento y prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento fue de 7.06 minutos.

Resultados del conjunto de datos WISDM v2.0: WISDM v2.0 es una versión más grande que la v1.1, esto aumenta la dificultad. Los resultados obtenidos con el conjunto de datos WISDM v2.0 alcanzaron un valor de recuperación de 94.75% y un valor de pérdida de 5.21% al evaluar el conjunto de prueba. En la Figura 2d se muestra la evaluación de los conjuntos de entrenamiento y prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento fue de 14.85 minutos.

Resultados del conjunto de datos MNIST: MNIST es una base de datos muy popular para la evaluación de modelos. Los resultados obtenidos alcanzaron un valor de exactitud de 99.47% y un porcentaje de error de 0.53% al evaluar el conjunto de prueba. La Figura 2e muestra la evaluación del conjunto de entrenamiento y del conjunto de prueba. El tiempo de entrenamiento de la red neuronal con el conjunto de entrenamiento y con 30 etapas fue de 30.75 minutos.

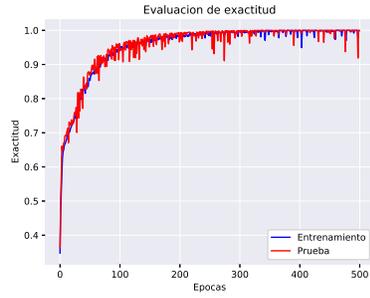
La Tabla 13 muestra una comparación de los resultados obtenidos con otros resultados en el estado del arte.

Tabla 13. Comparación de exactitud de diferentes técnicas del estado del arte

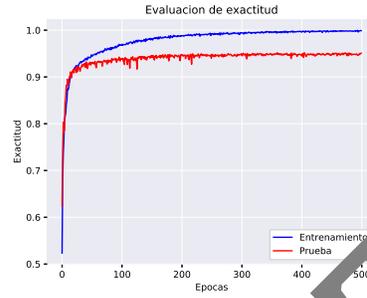
Base Datos	Propuesta	Avilés [4]	Zhang [24]	Yan [24]	Abebe [1]	Taufeeq [28]	Li Wan [31]
UCI HAR	100	100	94	-	-	-	-
HAPT	95.27	-	93.1	-	-	100	-
WISD v1.1	98.12	100	97.0	98.6	-	-	-
WISD v2.0	94.75	-	-	92.7	97.9	-	-
MNIST	99.47	-	-	-	-	-	99.79

5. Conclusiones

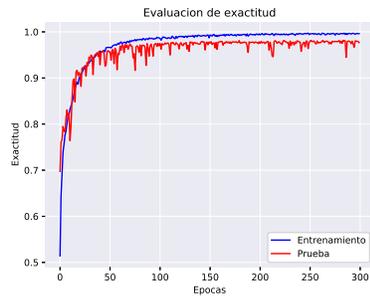
En este artículo se presentó una nueva implementación de la propuesta de una red neuronal profunda previamente reportada en la literatura para aplicarse sólo para reconocimiento y clasificación de patrones HAR, tal que la estrategia se basa en la extracción paralela en tres etapas: fina, media y gruesa. Sin embargo, es necesario una mejora en la optimización de los parámetros, ya que como hemos presentado en la implementación para imágenes, como meta más ambiciosa y de mayor alcance para aplicaciones en visión por computadora, es necesario poder hacerle modificaciones que mejoren en rapidez y capacidad de extracción de características propias para imágenes como entidades numéricas más complejas.



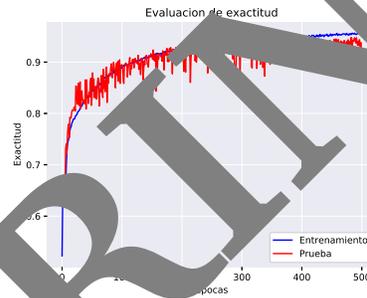
(a) UCI HAR: Exactitud de 100 %



(b) HAPT: Exactitud de 95.2 %



(c) WISDM v1.1: Exactitud de 98.2 %



(d) WISDM v2.0: Exactitud de 94.75 %



(e) MNIST: Exactitud de 99.47 %

Fig. 2. Evaluación de la red neuronal propuesta con las bases de datos seleccionadas

La implementación aquí presentada está ubicada en el nivel 25 del *Ranking classification datasets results*⁶, esta página se tomó como referencia para la comparación de resultados de diferentes técnicas y enfoques del conjunto de datos MNIST, ya que para aplicaciones prácticas es factible su uso, pero consideramos

⁶ Ranking classification datasets results: https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

que es posible su mejora si realizamos, como trabajo futuro, modificaciones a la capa de extracciones paralelas, buscando robustecer a los patrones orientados a problemas de imágenes más complejas.

Si bien la base de datos MNIST es una buena aproximación para validar posibles aplicaciones reales, existen problemas de imágenes más complejas como lo son el rostro humano y los escenarios naturales, donde en un futuro trabajo podremos mostrar los avances en tales campos.

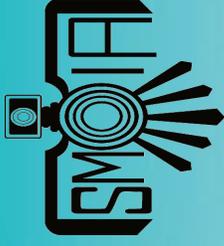
Referencias

1. Abebe, G., Cavallaro, A.: Inertial-vision: Cross-domain knowledge transfer for wearable sensors. pp. 1392–1400 (10 2017). <https://doi.org/10.1109/ICCVW.2017.165>
2. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.: A public domain dataset for human activity recognition using smartphones. pp. 24–26. *Computational Intelligence and Machine Learning* (01 2013)
3. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.: A public domain dataset for human activity recognition using smartphones (01 2013)
4. Avilés-Cruz, C., Ramírez, A., Zúñiga López, A., Villegas Cortés, J.: Coarse-to-fine convolutional deep-learning strategy for human activity recognition. *Sensors* **2019** (03 2019). <https://doi.org/10.3390/s19071556>
5. Catal, C., Tufekci, S., Pirmir, E., Kocabag, G.: On the use of ensemble of classifiers for accelerometer-based activity recognition. *Applied Soft Computing* **46** (01 2015). <https://doi.org/10.1016/j.asoc.2015.01.025>
6. Cho, H.; Yoon, S.: Divide and conquer-based 1d cnn for human activity recognition using test data sharpening. *Sensors* **2017** (06 2017). <https://doi.org/10.3390/s180410559>
7. Cireřan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. *Proceedings of CVPR, 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (02 2012). <https://doi.org/10.1109/CVPR.2012.48110>
8. Cireřan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation* **22**(12), 3207–3220 (2010). <https://doi.org/10.1162/NECO.a.00052>, PMID: 20858131
9. Davide Anguita, A.G.: A Public Domain Dataset for Human Activity Recognition in Smartphones. 21st European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning Bruges (Aachen 2013)
10. Butt, A., Chauhan, V., Khan, I.: Pattern recognition: an overview. *American Journal of Intelligent Systems* **2**, 23–27 (01 2012). <https://doi.org/10.1923/j.ajis.20120201.04>
11. Goodfellow, I., Bengio, Y., Courville, A.: *Deep Learning*. The MIT Press (2016)
12. Graham, B.: Fractional max-pooling (12 2014)
13. HAN, Y., YE, J., LUO, J., ZHOU, H.: The effect of axis-wise triaxial acceleration data fusion in cnn-based human activity recognition. *IEEE Transactions on Information and Systems* **E103.D**(4), 813–824 (2020). <https://doi.org/10.1587/transinf.2018EDP7409>
14. Hancock, E., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.: Advances in pattern recognition methodology and applications. *Pattern Recognition Letters* **34**, 359–360 (03 2013). <https://doi.org/10.1016/j.patrec.2012.11.001>
15. Ignatov, A.: Real-time human activity recognition from accelerometer data using convolutional neural networks. *Applied soft computing* **62**, 915–922 (9 2017). <https://doi.org/10.1016/j.asoc.2017.09.027>

16. Jalal, A., Kim, Y.H., Kim, Y.J., Kamal, S., Kim, D.: Robust human activity recognition from depth video using spatiotemporal multi-fused features. *Pattern Recognition* **61** (08 2016). <https://doi.org/10.1016/j.patcog.2016.08.003>
17. Kwapisz, J., Weiss, G., Moore, S.: Activity recognition using cell phone accelerometers. *SIGKDD Explorations* **12**, 74–82 (11 2010). <https://doi.org/10.1145/1964897.1964918>
18. Lane, N., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., Campbell, A.: A survey of mobile phone sensing. *IEEE Communications Magazine*, **IEEE** **48**, 140 – 150 (10 2010). <https://doi.org/10.1109/MCOM.2010.5560598>
19. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**, 2278 – 2324 (12 1998). <https://doi.org/10.1109/5.726791>
20. LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database. *AT&T Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> **2** (2010)
21. Liang, M., Hu, X.: Recurrent convolutional neural network for object recognition. pp. 3367–3375 (06 2015). <https://doi.org/10.1109/CVPR.2015.7298958>
22. Martínez, F., González-Fraga, J., Cuevas-Tello, J.C., Fernández, M.: Activity inference for ambient intelligence through haptic sensors in a healthcare environment. *Sensors (Basel, Switzerland)* **12**, 1068–1099 (12 2012). <https://doi.org/10.3390/s120101072>
23. Quid, M.A., Jalal, A.: Wearable sensors based human behavioral pattern recognition using statistical features and reweighted genetic algorithm. *Multimedia Tools and Applications* pp. 1–23 (12 2019). <https://doi.org/10.1007/s00422-019-08463-7>
24. Ravi, D., Wong, C., Lo, B., Yang, G.Z.: A deep learning approach to on-node sensor data analytics for mobile or wearable devices. *IEEE journal of biomedical and health informatics* **PP** (12 2016). <https://doi.org/10.1109/JBHI.2016.2633287>
25. Reyes-Ortiz, J., Oneto, L., Ghio, A., Lecun, Y., Parra, X.: Human activity recognition on smartphones with awareness of basic activities and postural transitions (01 2014)
26. San-Segundo, R., Lorenzo-Ortega, J., Martínez-González, B., Pardo, J.: Segmenting human activities based on haptic sensing smartphone inertial sensors. *Pervasive and Mobile Computing* **30** (01 2016). <https://doi.org/10.1016/j.pmcj.2016.01.004>
27. Stisen, A., Blunck, H., Bhattacharyya, S., Brentow, T., Kjærgaard, M., Dey, A., Sonne, T., Jensen, M.: Smart devices are different: Assessing and mitigating mobile sensing heterogeneities for activity recognition. pp. 127–140 (11 2015)
28. Uddin, M., Billah, M.M., Hossain, M.F.: Random forests based recognition of human activities and postural transitions on smartphone. In: 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV). pp. 250–255 (2016)
29. Vasilakos, A., Dobson, S., Benaru, L., Ciobanu, R.I.: Human physical activity recognition using smartphone sensors. *Sensors* **19**, 458 (01 2019)
30. Walse, K., Dharamar, R., Thakare, V.M.: Performance evaluation of classification algorithm dataset for human activity recognition (03 2016). <https://doi.org/10.1145/2905055.2905232>
31. Wang, S., Zeiler, M., Zhang, S., Lecun, Y., Fergus, R.: Regularization of neural networks using dropconnect (01 2013)
32. Weiss, G., Lockhart, J.: The impact of personalization on smartphone-based activity recognition. *AAAI Workshop - Technical Report* (01 2012)
33. Zhang, Y., Zhang, Y., Zhang, Z., Bao, J., Song, Y.: Human activity recognition based on time series analysis using u-net (09 2018)
34. Zheng, Z., Du, J., Sun, L., Huo, M., Chen, Y.: Tasg: An augmented classification method for impersonal har. *Mobile Information Systems* **2018**, 1–10 (12 2018)
35. Zhu, X., Qiu, H.: High accuracy human activity recognition based on sparse locality preserving projections. *PLOS ONE* **11**(11), 1–18 (11 2016)



COMIA
2020



UACJ

UNIVERSIDAD AUTÓNOMA
DE CIUDAD JUÁREZ



**LA SOCIEDAD MEXICANA DE INTELIGENCIA ARTIFICIAL
Y LA UNIVERSIDAD AUTÓNOMA DE CIUDAD JUÁREZ**

OTORGAN ESTE CERTIFICADO A:

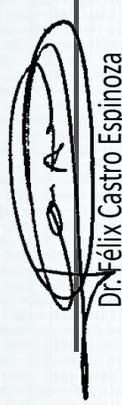
**Yafte Aaron Flores-Morales, Juan Villegas-Cortez, Graciela Roman-Alonso, Arturo Zuñiga-López, Cesar Benavides-Alvarez y
Salomón Cordero-Sánchez**

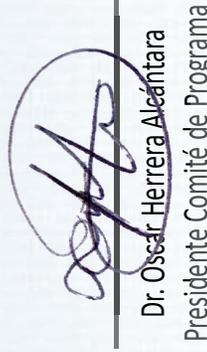
POR LA PRESENTACIÓN DEL ARTÍCULO TITULADO:

**Implementación de una red neuronal profunda en tres etapas paralelas para el reconocimiento de actividades humanas
e imágenes**

en el XII Congreso Mexicano de Inteligencia Artificial - COMIA 2020

Cd. Juárez, Chihuahua, México, del 5 al 7 de agosto de 2020


Dr. Félix Castro Espinoza
Presidente SMIA


Dr. Oscar Herrera Alcántara
Presidente Comité de Programa


Dr. Noe A. Castro Sánchez
Presidente Comité de Programa


Dr. Manuel de J. Nandayapa Alfaro
Comité Local COMIA

C

GLOSARIO

Tema	Definición	Sección
Cadenas de Márkov	En la teoría de la probabilidad, se conoce como cadena de Márkov o modelo de Márkov a un tipo especial de proceso estocástico discreto en el que la probabilidad de que ocurra un evento depende solamente del evento inmediatamente anterior. Esta característica de falta de memoria recibe el nombre de propiedad de Markov.	2.7
Red neuronal artificial	Un tipo de red neuronal artificial caracterizada por la introducción de variaciones aleatorias en la red, bien sea mediante la asignación de funciones de transferencia estocásticas a las neuronas artificiales, o bien asignando un peso estocástico a cada neurona.	2.5
Red neuronal convolucional	Una red neuronal convolucional es un tipo de red neuronal artificial utilizada en el reconocimiento de imágenes, esta se caracteriza por la utilización de la operación de convolución para la extracción de características.	2.6
MPI	La interfaz de paso de mensajes o MPI es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.	2.9

GPU	Una unidad de procesamiento gráfico o GPU (graphics processing unit) es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos u operación masivas.	2.9
CPU	La CPU, o unidad central de procesamiento, es la parte encargada de procesar todas las instrucciones y datos del software y del hardware, motivo por el cual constituye el elemento más importante del computador.	2.9
Inteligencia artificial	La Inteligencia artificial es el campo científico de la informática que se centra en la creación de programas y mecanismos que pueden mostrar comportamientos considerados inteligentes.	2.1
Visión por computadora	Es una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador.	2.6
Cluster	Un sistema distribuido de granjas de computadoras unidos entre sí, por una red de alta velocidad y que se comportan como si fuesen un único servidor.	2.9
Cómputo evolutivo	El cómputo evolutivo es una rama de la inteligencia artificial que involucra problemas de optimización combinatoria. Se inspira en los mecanismos de la Evolución biológica.	2.7
Optimización	La optimización es el proceso de establecer los valores de las variables de decisión de tal manera que se optimice el objetivo en cuestión. La solución óptima es un conjunto de variables de decisión que maximiza o minimiza la función objetivo mientras satisface las restricciones.	2.7

BIBLIOGRAFÍA

- [1] Evolutionary convolutional neural networks: An application to handwriting recognition. *Neurocomputing* 529, 7587 (2018), 484–9.
- [2] ABEBE, G., AND CAVALLARO, A. Inertial-vision: Cross-domain knowledge transfer for wearable sensors. pp. 1392–1400.
- [3] ABU ALSHEIKH, M., SELIM, A., NIYATO, D., DOYLE, L., LIN, S., AND TAN, H. Deep activity recognition models with triaxial accelerometers. In *AAAI Conference on Artificial Intelligence* (United States, 2016), vol. WS-16-01 - WS-16-15 of *AAAI Workshop - Technical Report*, AI Access Foundation, pp. 8–13.
- [4] ANGUITA, D., GHIO, A., ONETO, L., PARRA, X., AND REYES-ORTIZ, J. A public domain dataset for human activity recognition using smartphones. *Computational Intelligence and Machine Learning*, pp. 24–26.
- [5] ANGUITA, D., GHIO, A., ONETO, L., PARRA, X., AND REYES-ORTIZ, J. A public domain dataset for human activity recognition using smartphones.
- [6] AVILÉS-CRUZ, C., RAMÍREZ, A., ZÚÑIGA LÓPEZ, A., AND VILLEGAS CORTEZ, J. Coarse-fine convolutional deep-learning strategy for human activity recognition. *Sensors* 2019 (03 2019).
- [7] BUSTONI, I. A., HIDAYATULLOH, I., NINGTYAS, A., PURWANINGSIH, A., AND AZHARI, S. Classification methods performance on human activity recognition. *Journal of Physics: Conference Series* 1456 (01 2020), 012027.
- [8] CANTU-PAZ, E., AND GOLDBERG, D. On the scalability of parallel genetic algorithms. *Evolutionary computation* 7 (02 1999), 429–49.
- [9] CATAL, C., TUFEKCI, S., PIRMIT, E., AND KOCABAG, G. On the use of ensemble of classifiers for accelerometer-based activity recognition. *Applied Soft Computing* 46 (01 2015).
- [10] CHANG, J.-R., AND CHEN, Y.-S. Batch-normalized maxout network in network.
- [11] CHO, H., AND YOON, S. Divide and conquer-based 1d cnn human activity recognition using test data sharpening. *Sensors (Basel, Switzerland)* 18 (04 2018).
- [12] CHO, H.; YOON, S. Divide and conquer-based 1d cnn human activity recognition using test data sharpening. *Sensors* 2018 1055 (06 2017).
- [13] CIREŞAN, D., MEIER, U., AND SCHMIDHUBER, J. Multi-column deep neural networks for image classification. *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (02 2012).

- [14] CIREŞAN, D. C., MEIER, U., GAMBARDELLA, L. M., AND SCHMIDHUBER, J. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation* 22, 12 (2010), 3207–3220. PMID: 20858131.
- [15] DAVIDE ANGUITA, A. G. A Public Domain Dataset for Human Activity Recognition Using Smartphones. *21st European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning Bruges* (April 2013).
- [16] DE JONG, K., FOGEL, D., AND SCHWEFEL, H.-P. *A history of evolutionary computation*. 01 1997, pp. A2.3:1–12.
- [17] DESELL, T. Developing a volunteer computing project to evolve convolutional neural networks and their hyperparameters. *13th IEEE International Conference on eScience*, 8109119 (2017), 19–28.
- [18] FOGEL, D. B. *An Introduction to Evolutionary Computation*. IEEE, 1998, pp. 1–28.
- [19] FREEMAN, J. A. *Redes neuronales. Algoritmos, aplicaciones y técnicas de programación*. No. 1 edition. Addison-Wesley, 1993.
- [20] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, New York, 1989.
- [21] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. The MIT Press, 2016.
- [22] GRAHAM, B. Fractional max-pooling.
- [23] HAN, X., YE, J., LUO, J., AND ZHOU, H. The effect of axis-wise triaxial acceleration data fusion in cnn-based human activity recognition. *IEICE Transactions on Information and Systems E103.D*, 4 (2020), 813–824.
- [24] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.
- [25] IGNATOV, A. Real-time human activity recognition from accelerometer data using convolutional neural networks. *Applied soft computing* 62 (9 2017), 915–922.
- [26] ILSVRC. Concurso de reconocimiento de imágenes. <http://www.image-net.org/challenges/LSVRC/>, 2020.
- [27] JALAL, A., KIM, Y.-H., KIM, Y.-J., KAMAL, S., AND KIM, D. Robust human activity recognition from depth video using spatiotemporal multi-fused features. *Pattern Recognition* 61 (08 2016).
- [28] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.
- [29] KUSETOGULLARI, H., YAVARIABDI, A., CHEDDAD, A., GRAHN, H., AND HALL, J. Ardis: a swedish historical handwritten digit dataset. *Neural Computing and Applications* (11 2020).

- [30] KWAPISZ, J., WEISS, G., AND MOORE, S. Activity recognition using cell phone accelerometers. *SIGKDD Explorations* 12 (11 2010), 74–82.
- [31] LANE, N., MILUZZO, E., LU, H., PEEBLES, D., CHOUDHURY, T., AND CAMPBELL, A. A survey of mobile phone sensing. *IEEE Commun. Mag.* *Communications Magazine, IEEE* 48 (10 2010), 140 – 150.
- [32] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86 (12 1998), 2278 – 2324.
- [33] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.
- [34] LECUN, Y., CORTES, C., AND BURGESS, C. Mnist handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> 2 (2010).
- [35] LEE, C.-Y., GALLAGHER, P. W., AND TU, Z. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. *ArXiv abs/1509.08985* (2016).
- [36] LIANG, M., AND HU, X. Recurrent convolutional neural network for object recognition. pp. 3367–3375.
- [37] LIAO, Z., AND CARNEIRO, G. Competitive multi-scale convolution.
- [38] LIAO, Z., AND CARNEIRO, G. On the importance of normalisation layers in deep learning with piecewise linear activation units. pp. 1–8.
- [39] LOCKHART, J., WEISS, G., XUE, J., GALLAGHER, S., GROSNER, A., AND PULICKAL, T. Design considerations for the wisdm smart phone-based sensor mining architecture.
- [40] MARTINEZ, F., GONZÁLEZ-FRAGA, J., CUEVAS-TELLO, J. C., AND RODRIGUEZ, M. Activity inference for ambient intelligence through handling artifacts in a healthcare environment. *Sensors (Basel, Switzerland)* 12 (12 2012), 1072–99.
- [41] MITCHELL, T. M. *Machine Learning*, 1 ed. McGraw-Hill, Inc., USA, 1997.
- [42] NILSSON, N. J. *Artificial Intelligence: A New Synthesis*, 5 ed. Morgan Kaufmann Publishers, Inc. San Francisco, California, 1998.
- [43] OLAGUE, G. *Evolutionary Computer Vision*. 01 2016.
- [44] P, L. Neural network evolution using expedited genetic algorithm for medical image denoising. *13th IEEE International Conference on eScience*, 11070 (2017), 12–20.
- [45] P, L. Neural network evolution using expedited genetic algorithm for medical image denoising. *13th IEEE International Conference on eScience*, 11070 (2017), 12–20.
- [46] PACHECO, P. *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011.
- [47] QUAID, M. A., AND JALAL, A. Wearable sensors based human behavioral pattern recognition using statistical features and reweighted genetic algorithm. *Multimedia Tools and Applications* (12 2019), 1–23.
- [48] RAUBER, T., AND RÜNGER, G. *Parallel Programming - for Multicore and Cluster Systems*. 01 2010.

- [49] RAVÌ, D., WONG, C., LO, B., AND YANG, G.-Z. A deep learning approach to on-node sensor data analytics for mobile or wearable devices. *IEEE journal of biomedical and health informatics PP* (12 2016).
- [50] RAVÌ, D., WONG, C., LO, B., AND YANG, G.-Z. Deep learning for human activity recognition: A resource efficient implementation on low-power devices. pp. 71–76.
- [51] REYES-ORTIZ, J., ONETO, L., GHIO, A., ANGUIA, D., AND PARRA, X. Human activity recognition on smartphones with awareness of basic activities and postural transitions.
- [52] RONAO, C., AND CHO, S.-B. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Systems with Applications 59* (04 2016).
- [53] SAN-SEGUNDO, R., LORENZO-TRUEBA, J., MARTÍNEZ-GONZÁLEZ, B., AND PARDO, J. Segmenting human activities based on hmms using smartphone inertial sensors. *Pervasive and Mobile Computing 30* (01 2016).
- [54] SANTANA-QUINTERO, L., AND COELLO, C. Una introducción a la computación evolutiva y alguna de sus aplicaciones en economía y finanzas || an introduction to evolutionary computation and some of its applications in economics and finance. *Revista de Métodos Cuantitativos para la Economía y la Empresa 2* (12 2006).
- [55] SATO, I., NISHIMURA, H., AND YOKOI, K. Apac: Augmented pattern classification with neural networks.
- [56] SHAKYA, S., ZHANG, C., AND ZHOU, Z. Comparative study of machine learning and deep learning architecture for human activity recognition using accelerometer data.
- [57] SHARKEY, N. Evolutionary computation: the fossil record. *IEE Review 45*, 1 (Jan 1999), 40–40.
- [58] SOSSA, H., GARRO, B. A., VILLEGAS, J., AVILÉS, C., AND OLAGUE, G. Automatic design of artificial neural networks and associative memories for pattern classification and pattern restoration. In *Pattern Recognition* (Berlin, Heidelberg, 2012), J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, J. A. Olvera López, and K. L. Boyer, Eds., Springer Berlin Heidelberg, pp. 23–34.
- [59] STISEN, A., BLUNCK, H., BHATTACHARYA, S., PRENTOW, T., KJÆRGAARD, M., DEY, A., SONNE, T., AND JENSEN, M. Smart devices are different: Assessing and mitigating mobile sensing heterogeneities for activity recognition. pp. 127–140.
- [60] TANJIL, F. Deep learning concepts for evolutionary art. *8th International Conference on Computational Intelligence in Music, Sound, Art and Design*, 11453 (2019), 1–17.
- [61] UDDIN, M. T., BILLAH, M. M., AND HOSSAIN, M. F. Random forests based recognition of human activities and postural transitions on smartphone. In *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)* (2016), pp. 250–255.
- [62] VOICU, R.-A., DOBRE, C., BAJENARU, L., AND CIOBANU, R.-I. Human physical activity recognition using smartphone sensors. *Sensors 19* (01 2019), 458.
- [63] WALSE, K., DHARASKAR, R., AND THAKARE, V. M. Performance evaluation of classifiers on wisdm dataset for human activity recognition.

-
- [64] WAN, L., ZEILER, M., ZHANG, S., LECUN, Y., AND FERGUS, R. Regularization of neural networks using dropconnect.
- [65] WEISS, G., AND LOCKHART, J. The impact of personalization on smartphone-based activity recognition. *AAAI Workshop - Technical Report* (01 2012).
- [66] WILKINSON, B. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. No. 2 edition. Pearson, 2005.
- [67] XU, Y., SHEN, Z., ZHANG, X., GAO, Y., DENG, S., WANG, Y., FAN, Y., AND CHANG, E. Learning multi-level features for sensor-based human action recognition. *Pervasive and Mobile Computing* 40 (11 2016).
- [68] ZHANG, H., XIAO, Z., WANG, J., LI, F., AND SZCZERBICKI, E. A novel iot-perceptive human activity recognition (har) approach using multi-head convolutional attention. *IEEE Internet of Things Journal PP* (10 2019), 1–1.
- [69] ZHANG, Y., ZHANG, Y., ZHANG, Z., BAO, J., AND SONG, Y. Human activity recognition based on time series analysis using u-net, 09 2018.
- [70] ZHENG, Z., DU, J., SUN, L., HUO, M., AND CHEN, Y. Tasg: An augmented classification method for impersonal har. *Mobile Information Systems 2018* (12 2018), 1–10.
- [71] ZHU, X., AND QIU, H. High accuracy human activity recognition based on sparse locality preserving projections. *PLOS ONE* 11, 11 (11 2016), 1–18.



Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE EXAMEN DE GRADO

No. 00089

Matricula: 2183802184

Evolución de parámetros de optimización de redes neuronales profundas para el reconocimiento y clasificación de imágenes.

Con base en la Legislación de la Universidad Autónoma Metropolitana, en la Ciudad de México se presentaron a las 11:00 horas del día 28 del mes de enero del año 2021 POR VÍA REMOTA ELECTRÓNICA, los suscritos miembros del jurado designado por la Comisión del Posgrado:

DR. FRANCISCO FERNANDEZ DE VEGA
DR. JUAN VILLEGAS CORTEZ
DR. ERIC ALFREDO RINCON GARCIA



Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron para proceder al Examen de Grado cuya denominación aparece al margen, para la obtención del grado de:

MAESTRO EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: YAFTE AARON FLORES MORALES

YAFTE AARON FLORES MORALES
ALUMNO

y de acuerdo con el artículo 78 fracción III del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

APROBAR

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

REVISÓ

MTRA. ROSALIA SERRANO DE LA PAZ
DIRECTORA DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI

DR. JESUS ALBERTO OCHOA TAPIA

PRESIDENTE

DR. FRANCISCO FERNANDEZ DE VEGA

VOCAL

DR. JUAN VILLEGAS CORTEZ

SECRETARIO

DR. ERIC ALFREDO RINCON GARCIA

El presente documento cuenta con la firma –autógrafa, escaneada o digital, según corresponda- del funcionario universitario competente, que certifica que las firmas que aparecen en esta acta – Temporal, digital o dictamen- son auténticas y las mismas que usan los c.c. profesores mencionados en ella