ISSN 1870-4069

Optimización Distribuida de Redes Convolucionales para la Clasificación de Imágenes

Emmanuel F. Ramírez Hernández[1], Héctor Rodríguez Rangel[1], Víctor González Huitrón[1], Juan J. Flores[2], Vicenç Puig[3]

[1] Instituto Tecnológico de Culiacán,
División de Estudios de Posgrado e Investigación, Culiacán, Sinaloa, México
[2] Universidad Michoacana de San Nicolás de Hidalgo,
Facultad de Ingeniería Eléctrica, División de Estudios de Postgrado,
Morelia, Michoacán, México
[3] Institut de Robòtica i Informàtica Industrial, Barcelona, España
eramirezhdez@gmail.com, hrodriguez@itculiacan.edu.mx,
victor.gonzalez@conacyt.mx, juanf@umich.mx, vicenc.puig@upc.edu

Resumen. Este artículo propone una mejora en los tiempos de ejecución en la optimización de los hiperpámetros de redes convolucionales en tareas de clasificación de imágenes. Para esto se realizó una modificación, distribuyendo la función de evaluación del algoritmo genético compacto, de esta forma se mejoraron los tiempos de ejecución de la evaluación de los individuos de cada generación. Esta estrategia se utilizó para optimizar los tiempos de búsqueda esperando encontrar una mejor configuración de hiperparámetros de una Red Neuronal Convolucional (RNC). Además, se propuso una arquitectura para soportar la ejecución de procesos distribuidos utilizando Rabbit MQ; esta arquitectura es capaz de proporcionar propiedades deseables para sistemas distribuidos como alta disponibilidad, escalabilidad y tolerancia a fallos. Este trabajo fue probado utilizando la base de datos MNIST encontrando una mejora en los tiempos de ejecución en comparación con los algoritmos genéticos simples. Obteniendo una precisión comparable con los mostrados en la literatura.

Palabras clave: optimización de hiperparámetros, cómputo distribuido, red neuronal convolucional.

Distributed Optimization of Convolutional Networks for Image Classification

Abstract. This article proposes an improvement in the execution times in the optimization of the hyperparameters of Convolutional Neural Networks in image classification tasks. For this purpose, an improvement was made by distributing the evaluation function of the compact genetic algorithm, improving the time in the execution of the evaluation of the

individuals of each generation. This strategy was used to optimize scanning times and hope to find a better configuration of hyperparameters of a convolutional neural network (RNC). In addition, an architecture was proposed to support the execution of distributed processes using Rabbit MQ, this architecture is capable of providing desirable properties for distributed systems such as high availability, scalability and fault tolerance. This work was tested using the MNIST database and found an improvement in execution times compared to simple genetic algorithms. Obtaining an accuracy comparable to those shown in the literature.

Keywords: hyperparameters optimization, distiburted computing, convolutional neural network.

1. Introducción

En la actualidad las redes de aprendizaje profundo han estado revolucionando el ámbito de la inteligencia artificial, gracias a los resultados obtenidos mediante su uso. Las redes de aprendizaje profundo han empezado a utilizarse debido a que la capacidad de cómputo ha incrementado. Dicho incremento en la capacidad de cómputo ha hecho posible implementar estos modelos que son costosos en tiempo y procesamiento.

Normalmente las redes de aprendizaje profundo cuentan con millones de parámetros y requieren grandes cantidades de información para la optimización de estos y obtener el resultado esperado. Si incrementamos el tamaño de la red de aprendizaje profundo y el conjunto de datos podremos obtener modelos con mejores resultados pero a costa de un tiempo computacional considerablemente alto [1].

En la actualidad los tiempos en el entrenamiento de una red convolucional dependen del tamaño del conjunto de datos de entrenamiento y de las configuraciones asignadas. Si el conjunto de datos para el entrenamiento es demasiado extenso (para tener un modelo de calidad necesitamos que el conjunto de datos sea extenso, variado y balanceado) costará mas tiempo en finalizar su entrenamiento.

En este artículo se propone una metodología basada en distribuición de la evaluación de los individuos de un algoritmo genético compacto para la optimización de una red neuronal convolucional; de esta manera se podrá explorar un número mayor de posibles soluciones en un menor tiempo.

Este trabajo está estructurado de la siguiente manera: En la Sección 2 se encuentra una breve descripcioń de algunos algoritmos y metodologías de optimización para modelos de aprendizaje profundo. En la Sección 3 y 4 se muestran la definición, estructura y funcionamiento de una red convolucional y de los algoritmos genéticos compactos, respectivamente.. En la Sección 5 se detalla la arquitectura propuesta para la distribución de los procesos del algoritmo genético compacto. La descripción de la implementación de la arquitectura propuesta se

encuentra en la Sección 6. Para finalizar en las Secciones 7 y 8 tenemos los resultados de los experimentos realizados y conclusiones, respectivamente.

2. Trabajos relacionados

La búsqueda de soluciones mejores y mas rápidas ha sido un gran potenciador para un un gran número de trabajos para la búsqueda en la optimización de los procesos de entrenamiento de redes de aprendizaje profundo.

Las redes de aprendizaje profundo son notablemente eficientes descubriendo estructuras de correlación en datos sin supervisión. Por lo tanto son ampliamente utilizadas en el análisis de procesamiento de lenguaje natural y visión computacional. En el artículo presentado por Hegde et al. [1] se han realizado diferentes formas de paralelizar el aprendizaje en entornos multinúcleo y distribuidos. También se han analizado de forma empírica la acelaración del entrenamiento de una RNC utilizando CPU y GPU.

En el trabajo de Brownlee [2] se ha considerado el problema de entrenar una red de aprendizaje profundo con miles de millones de parámetros utilizando miles de núcleos de un CPU. En este artículo se presenta un marco de trabajo llamado *DistBelief* que permite utilizar clústeres informáticos con miles de máquinas para entrenar modelos grandes. Por último en este trabajo se han desarrollado dos algoritmos para el entrenamiento distribuido a gran escala tales como: SGD y Sandblaster.

El trabajo presentado por Travis Densell [4] expone un nuevo algoritmo que es llamado *Evolutionary exploration of augmenting convolutional topologies (EXACT)*, el cual es capaz de evolucionar la estructura de la red neuronal convolucional. *EXACT* está modelado con base al algoritmo *Neuroevolution of augmenting topologies (NEAT)* que permite escalar en ambientes computacionales de manera distribuida y evolucionar las redes neuronales convolucionales. A diferencia de Travis Densell se realizó la aplicación del *Algoritmo genético compacto*.

En el trabajo presentado por Castillo, Pedro A [3] se utiliza el protocolo transferencia de estado representacional (REST) basado en HTTP como protocolo de comunicación. Además la función de evaluación de los individuos es la ejecución de una red neuronal multi-capa. Por último Castillo, Pedro A. et al. no describen algún mecanismo en caso de que suceda una falla en la ejecución de la función de aptitud por lo cual es posible tener la pérdida de individuos.

3. Redes neuronales convolucionales

El objetivo principal de las redes neuronales convolucionales (RNC) es aprender características de orden superior que se encuentran en los datos a través de convoluciones. La eficacia de las RNC en el reconocimiento de imágenes es una de las principales razones por las cuales el mundo reconoce el poder del aprendizaje profundo [5].

Las RNC extraen la información mediante convoluciones y estas características sirven de entrada a la capa de clasificación. En términos generales, se puede decir que las RNC básicamente están conformadas por tres grupos de capas:

1. Capa de entrada.
2. Capa de extracción de características (Convolución).
3. Capa de clasificación (aprendizaje).

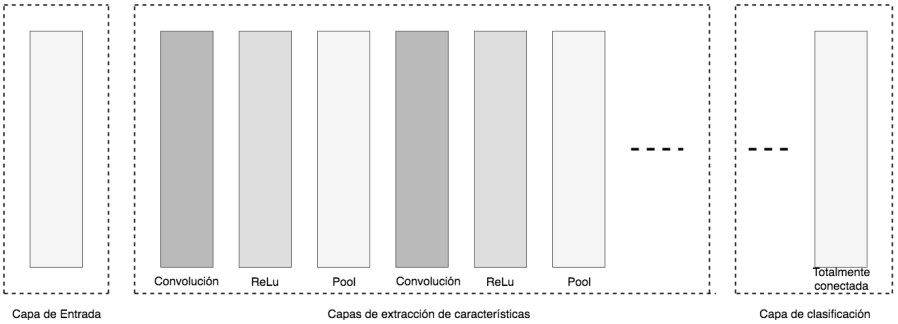

Fig. 1. Arquitectura general de una red convolucional.

3.1. Capas de entrada

La capa de entrada es donde ingresan los datos crudos de una imagen para ser procesados en la red. Normalmente los datos de entrada especifican sus dimensiones, como el ancho y alto. Cuando la imagen se encuentra a color se maneja una dimensión adicional denominada canal. Típicamente el número de canales son tres, para el valor de RGB de cada pixel.

3.2. Capa de extracción de características

La capa de extracción de características tiene un patrón general repetitivo de la siguiente secuencia:

1. Capa de convolución.
2. Capa de pooling (Agrupamiento).

Las capas de convolución. Son consideradas el núcleo mas importante en la construcción de la arquitectura para una RNC. Estas capas transforman los datos de entrada usando un parche de neuronas conectado localmente desde una capa anterior. La capa calculará el producto punto entre la región de las neuronas de la capa de entrada y los pesos a los que se encuentran conectadas localmente en la capa de salida.

La salida resultante generalmente contará con las mismas dimensiones espaciales (o dimensiones espaciales más pequeñas) pero a veces aumenta el número de elementos en la tercera dimensión de la salida (dimensión de profundidad).

Las capa de pooling (Agrupamiento) Encuentran una serie de características en las imágenes y construyen progresivamente características de un orden superior.

Estas capas se insertan comúnmente entre capas convolucionales sucesivas. Se busca seguir capas agrupadas para reducir progresivamente el tamaño espacial (ancho y alto) de la representación de datos. La agrupación de capas reduce la representación de datos progresivamente en la red y ayuda a controlar el sobre ajuste.

3.3. Capa de clasificación

Estas capas son en las que se encuentran una o más capas completamente conectadas para tomar las características de orden superior y producir probabilidades o puntajes de clase. Estas capas están completamente conectadas a todas la neuronas de la capa anterior. La salida de estas capas produce normalmente una salida bidimensional de las dimensiones [b $\times$ N], donde b es el número de ejemplos en el mini lote y N es el número de clases que estamos interesados en calificar.

La Figura 2 muestra las diferentes capas trabajando en conjunto para realizar una clasificación desde la capa de entrada, pasando por un conjunto de capas de extracción de características (convolución y agrupamiento) y por último pasando por la capa de clasificación y otorgando un resultado.

4. Algoritmo genético compacto

El Algoritmo Genético Compacto (AGc) es una técnica de búsqueda u optimización probabilística, la cual está relacionada con el algoritmo genético y otros algoritmos evolutivos que están inspirados en la teoría de la evolución por medio de la selección natural. El objetivo AGc es simular el comportamiento del algoritmo genético con menores requerimientos (sin requerir que una población sea mantenida en memoria) [6]. Esto se logra utilizando un vector de probabilidades en lugar de la población entera. Las soluciones candidatas son probabilísticamente generadas desde la cadena de características. Las características que proveen una mejor solución se utilizan para realizar cambios en un vector de probabilidades [2].

El Algoritmo genético inicia definiendo el vector de probabilidad, a partir del dicho vector, dos individuos se generan y evalúan. El proceso de evaluación de los individuos prueba los hiperparámetros propuestos por el AGc. Los individuos generados compiten y de acuerdo al ganador se actualiza el vector de probabilidad. Este proceso se repite hasta que el criterio de convergencia se cumple.

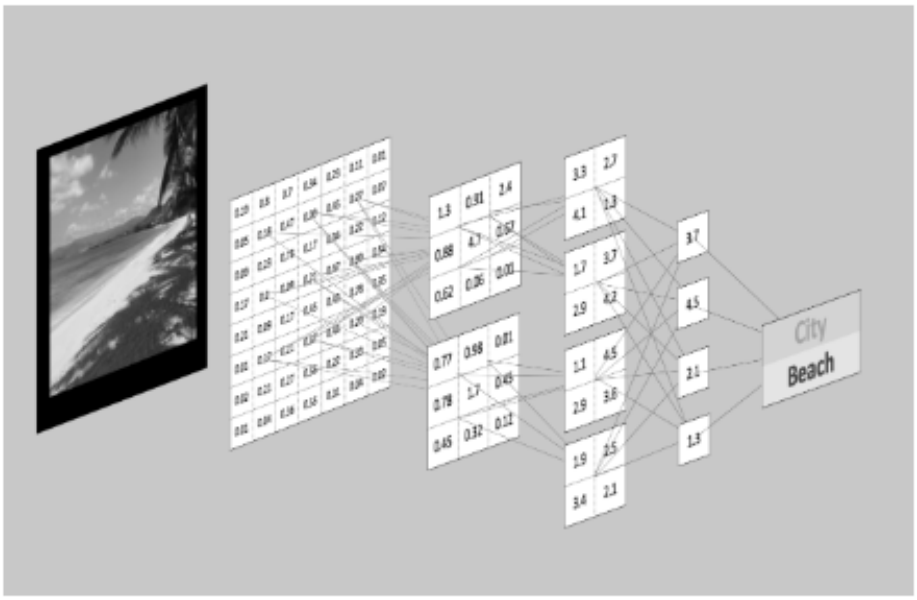

Fig. 2. Ejemplo de las diferentes capas para la obtención de una clasificación [5].

5. Arquitectura propuesta

Esta sección describe la arquitectura propuesta para llevar a cabo este trabajo. En la Figura 3 se muestra de forma general la arquitectura utilizada.

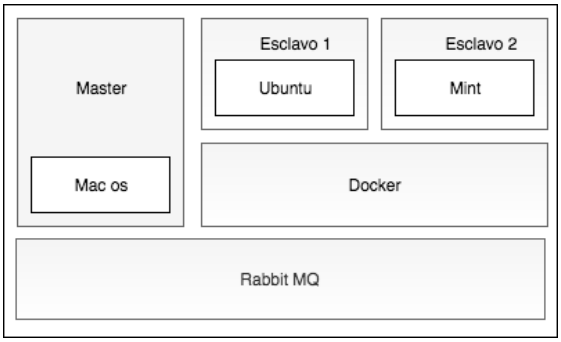

Fig. 3. Arquietctura general

El componente *esclavo* es el encargado de realizar la ejecución de la función de aptitud del individuo, esto quiere decir que en este caso el componente esclavo realizará el entrenamiento de la RNC según la configuración especificada por el

individuo. Al finalizar la ejecución del entrenamiento, este componente retorna el valor de aptitud especificado.

El componente *Maestro* es el encargado de coordinar los procesos que se realizan en el algoritmo genético compacto. Este componente realiza la generación de los individuos, la recepción de individuos entrenados y la competencia de los individuos. Además es el encargado de realizar la asignación del individuo que será procesado por el esclavo [7].

El componente llamado Docker es el encargado de la virtualización de los ambientes requeridos para realizar la ejecución de los procesos necesarios para el funcionamiento correcto del sistema. Docker nos permite utilizar entornos de aplicación aislados llamados contenedores. Esto es diseñado para el apoyo de los desarrolladores para poder replicar el ambiente utilizado en los esclavos sin mayor complejidad.

El último componente *RabbitMQ*, un "intermediario de mensajes"de código abierto que implementa el Protocolo Avanzado de Message Queue Server (AMQP), tiene como función el garantizar que los datos (los mensajes) vayan de un productor a los consumidores, para que estos mensajes sean procesados. El principal destinatario de los mensajes es la "cola", un acumulador de datos potencialmente ilimitado, que reside dentro de RabbitMQ. Si el productor y los consumidores están conectados a la misma cola, pueden comunicarse sin que realmente se conozcan entre sí. Esto convierte a RabbitMQ en una poderosa herramienta para la distribución escalable y confiable para realizar la segmentación de los procesos.

6. Implementación del algoritmo genético compacto distribuido

En esta sección se presenta la implementación de la arquitectura propuesta, la distribución de los procesos y la coordinación entre los componentes descritos anteriormente. Además revisaremos los cambios que se necesitaron realizar al algoritmo genético compacto para realizar la distribución [8].

El nodo maestro incializará los parámetros configurados tales como el tamaño de cromosoma y el número de generaciones que creara el AGCD. Para la definición del cromosoma se creara con base al tamaño del cromosoma especificado y se tomarán en cuenta los siguientes hiperpárametros.

1. Generales
   - *a*) **Épocas:** Es un número entero positivo que limita el número de pasos que el conjunto de validación es evaluado.
   - *b*) **Aprendizaje:** Es un número de punto flotante que representa la magnitud de la actualización por cada época de entrenamiento.
   - *c*) **Entrenamiento:** Es un número que representa la proporción de datos utilizados en el conjunto de entrenamiento.
   - *d*) **Optimizador:** Son funciones que calculan gradientes para una medida de pérdida y aplican gradiente a variables. Los tres posibles valores

de este parámetro son Stochastic gradient descent(SGD), AdamOptimizer(ADAM) y RMSPropOptimizer (RMSProp).

e) **Activación:** Funciones que proveen la no linealidad. Este parámetro puede tomar dos valores: una función rectificadora (relu) y una función para suavizar la no linealidad (elu).

2. Convolucionales
   a) **Tamaño del Filtro:** Es un número que representa el tamaño del filtro que recorrerá la matriz que representa la imagen.
   b) **Strides:** Una lista de números enteros que representa el número de características que serán recorridas de izquierda a derecha en la matriz, moviendo el filtro por cada dimensión en el vector de entrada.
   c) **Padding:** Indica si el filtro puede ir más allá de los límites de la matriz. Los valores que puede tomar este parámetro son dos *same* y *valid*.
   d) **Pooling:** Reduce la dimensionalidad de la entrada permitiendo hacer suposiciones de características contenidas en una región de la entrada. Los valores que puede tomar este parámetro son *maxpooling* y *avgpooling*.
   e) **Dropout:** Es una técnica de regularización para reducir el sobre entrenamiento en redes neuronales. Los valores que puede tomar son: 0.3, 0.4, 0.5, y 0.6.

Para la obtención de cada uno los modelos se realizó la búsqueda de los diferentes hiperparámetros con los cuales se obtuviera el modelo que mejor se apegue a los datos de entrenamiento utilizando un AGc para la búsqueda de estos. Las Tablas 1 y 2 muestran los hiperparárametros tomados en cuenta, clasificados en Generales y Convolucionales, respectivamente.

Tabla 1. Hiperparámetros generales y sus posibles valores.

Parámetro	Rango
Épocas	**[20, 40, 60, 80, 100, 120, 160]**
Aprendizaje	**[0.0001, 0.0006, 0.0011, 0.0016, 0.0021, 0.0026, 0.0031]**
Entrenamiento	**[0.70, 0.80, 0.90, 1.00]**
Optimizador	**[SGD, ADAM, RMSprop]**
Activación	**[relu, elu]**

Tabla 2. Hiperparámetros convolucionales y sus posibles valores.

Parámetro	Rango
Tamaño del Filtro	**[3, 4, 5, 6]**
Strides	**[2, 3, 4, 5]**
Padding	**[valid, same]**
Polling	**[MaxPooling2D, AveragePooling2D]**
Dropout	**[0.3, 0.4, 0.5, 0.6]**

La Figura 4 muestra un ejemplo de que posiciones del cromosoma proporcionado por el AGc corresponden a los diferentes hiperparámetros. El cromosoma es un vector de valores binarios, el cual es decodificado. e.g. para obtener el hiperparámetro Épocas se toman como referencia las 3 primeras posiciones, se codifica a decimal y así se determina que valor corresponde a dicha variable.

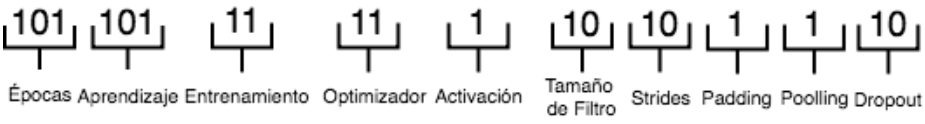

Fig. 4. Codificación de las diferentes hiperparámetros en el cromosoma del AGc.

Una vez preparado el nodo maestro, los nodos esclavos serán inicializados con la arquitectura seleccionada de la RNC (se muestra en la Figura 5) y realizando el cargado de datos que se le ha otorgado. El nodo esclavo una vez cargado e inicializado se suscriben a la cola de Rabbit MQ llamada *individuos* y a partir de ese momento los nodos esclavos estarán listos para procesar los individuos que el nodo maestro les proporcione.

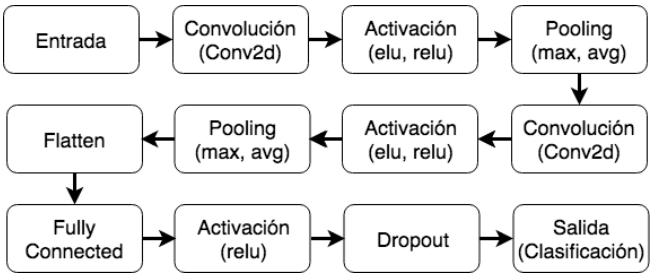

Fig. 5. Arquitectura de la CNN.

Una vez inicializado el nodo maestro y los nodos esclavos, el nodo maestro procederá a iniciar el proceso del AGc como se indica en la linea 11 y 12 del algoritmo 1 en las cuales indica que se empezará con la creación de la primera generación. En la linea 13 y 14 del algoritmo 1 se indica que los individuos serán enviados a la cola de Rabbit MQ llamada *individuos* a la cual los nodos esclavos ya han sido registrados previamente y tendrá que esperar mientras los nodos esclavos finalizan su proceso como se indica en la linea 15 del mismo algoritmo.

Algoritmo 1 Algoritmo Genético Compacto Distribuido.

```
1: procedure AGcD(n, l, g)          ▷ n como tamaño de población, l longitud del
   cromosoma
2:     Global individualTrainedOne
3:     Global individualTrainedTwo
4:     Global individualsTraining
5:     Se subscribe a la cola de Rabbit MQ
6:     subscribeQueueIndividualTrained(recieveIndividualTrained);
7:     Inicializar el vector de probabilidad
8:     for i:= 1 to l do p[i] := 0.5; do
9:         individualsTraining = True;
10:        Generar 2 individuos desde el vector
11:        a:= generate(p);
12:        b:= generate(p);
13:        sendIndividualToQueueIndividualGenerated(a);
14:        sendIndividualToQueueIndividualGenerated(b);
15:        while individualsTrained do
16:            wait();
17:        end while
18:        //Finalizado el entrenamiento se compiten los individuos
19:        winner,loser := compete(a,b);
20:        //Actualizar el vector utilizando el mejor
21:        for i := 1 to l do
22:            if winner[i] ≠ loser[i] then
23:                if winner[i] = 1 then
24:                    p[i] := p[i] + 1/n
25:                else
26:                    p[i] := p[i] − 1/n;
27:                end if
28:            end if
29:        end for
30:        // Revisar si el vector ha convergido
31:        for i := 1 to l do
32:            if p[i] > 0 & p[i] < 1 then
33:                return to step 4;
34:            end if
35:        end for
36:    end for
37:    return p
38: end procedure
```

Los nodos esclavos toman un individuo de la cola y empiezan la ejecución de la función de aptitud, en el caso de la RNC es la ejecución del entrenamiento tomando como métrica la *precisión*. Una vez finalizado el proceso de entrenamiento en los nodos esclavos, estos mandan un mensaje a la cola de *individuosEntrenados* de la cual el nodo maestro está escuchando; una vez que los dos individuos hayan finalizado su trabajo, libera los recursos para iniciar la competencia de los

individuos entrenados como lo indica la linea 12 del algoritmo 2. Seleccionado el individuo con mayor aptitud (individuo que tenga mayor precisión) como lo indica la linea 19 de algoritmo 2, se procede a la actualización del vector de probabilidades con base al individuo ganador y una vez actualizado el vector de probabilidades se verifica si el problema ha convergido. En caso de que no haya convergencia el nodo maestro procede a la siguiente generación; este proceso se repite hasta que el problema haya convergido o haya finalizado el numero de generaciones especificadas.

Algoritmo 2 Algoritmo de control de la siguiente generación.

```
1: procedure RECIEVEINDIVIDUALS(individualRecieved) ▷ individualRecieved es el
   individuo ya entrenado
2:     Global individualTrainedOne
3:     Global individualTrainedTwo
4:     Global individualsTraining
5:     Global individualsTrainingCounter
6:     Se recibe el individuo desde la cola
7:     individual = json.load(individualRecieved);
8:     contadorIndivudosEntrenados = contadorIndivudosEntrenados + 1
9:     if individualsTrainingCounter == 2 then
10:        individualTrainedTwo = individualRecieved;
11:        individualsTrainingCounter = 0
12:        individualsTrained = False
13:    else
14:        individualTrainedOne = individualRecieved;
15:    end if
16: end procedure
```

7. Resultados

Se realizaron una serie de experimentos para demostrar que el algoritmo genético compacto distribuido nos otorga una mejora en tiempos de procesamiento. La configuración de los experimentos llevados a cabo fue la siguiente:

1. Un nodo maestro, un nodo esclavo.
2. Un nodo maestro, dos nodos esclavos.

La Figura 6 especifica las dos configuraciones con las cuales se realizaron los experimentos para comprobar que la metodología logró reducir los tiempos de procesamiento obteniendo la misma precisión. El nodo maestro es un equipo con un procesador Intel Core i5 de 2.9 GHz, con 8 GB de memoria RAM DDR3 con una frecuencia de 1600 MHz y una tajeta GPU NVIDIA GeForce GTX 660M de 512 MB. El esclavo número uno es un equipo con un procesador Intel core i5-7500 de 3.40 GHz, con 8 GB de memoria RAM DDR3 con una frecuencia 1600 MHz y una tarjeta de video GPU NVIDIA GeForce GTX 1080 con 8 Gb

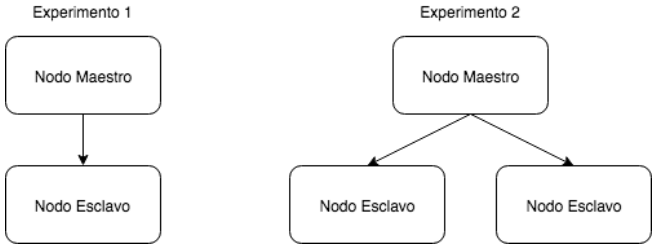

Fig. 6. Representación de la configuración de los experimentos realizados.

en RAM. El esclavo número dos es un equipo con un procesador Intel Xenon v4 de 1.70 GHz, y 32 GB de memoria RAM DDR3 a una frecuencia 1600 MHz y una tarjeta GPU NVIDIA GeForce GTX 1080 con 8 Gb en RAM.

La base de datos utilizada en los experimentos es el conocido "MNIST Database of Handwritten Digits"que contiene 10 clases posibles [9]. Esta base de datos contiene 60,000 imágenes de entrenamiento y 10,000 para realizar pruebas. Todas las imágenes están en escala de grises, sus tamaños son de 28 x 28 pixeles.

A continuación se presentan los resultados obtenidos con la configuración de un esclavo y un nodo maestro. En la tabla 3 se observa la representación del cromosoma con los tiempos de ejecución y la precisión obtenida. Podemos concluir que el tiempo promedio de ejecución es de 9 horas con 42 minutos y 59 segundos con una precisión promedio de 99.43 %.

Tabla 3. Resultados obtenidos en experimentos con un nodo maestro y un nodo esclavo.

Representación del cromosomas	**Tiempo**	**Precisión**
011 001 00 10 0 10 11 0 0 11	09:11:32	99.50 %
110 010 10 11 0 11 00 1 0 10	08:57:34	99.32 %
110 110 11 01 0 11 10 0 0 10	09:26:05	99.48 %
010 001 11 11 0 10 10 0 0 11	09:06:12	99.42 %
001 011 00 10 1 10 11 0 1 11	09:03:15	99.12 %
111 110 10 01 0 10 11 0 0 11	10:45:35	99.48 %
111 000 10 10 0 11 10 0 0 00	10:24:25	99.52 %
100 111 10 10 0 11 10 0 0 00	10:37:09	99.50 %
111 100 00 10 0 11 11 0 0 11	09:29:49	99.48 %
011 010 10 10 0 10 00 1 0 11	10:08:16	99.51 %
Promedio	**09:42:59**	**99.433 %**

Los resultados obtenidos con la configuración de dos esclavos y un nodo maestro. En la tabla 4 podemos observar una mejoría en tanto a tiempo de ejecución como de precisión. El promedio de tiempo de ejecución es de 6 horas 24 minutos con 11 segundos consiguiendo una mejora significativa en comparación a los métodos simples.

Tabla 4. Resultados obtenidos en experimentos con un nodo maestro y dos nodo esclavo

Representación del cromosomas	**Tiempo**	**Precisión**
100 010 11 11 0 10 01 0 0 00	06:48:52	99.51 %
100 001 01 01 0 11 01 0 0 00	06:28:07	99.49 %
110 101 00 11 0 01 10 0 0 00	06:23:53	99.48 %
011 010 10 11 0 11 00 0 0 11	06:33:21	99.47 %
111 010 11 11 0 01 11 0 0 11	05:48:26	99.47 %
100 011 01 01 0 11 01 0 0 00	06:48:38	99.45 %
101 011 00 01 0 10 01 1 0 01	06:20:52	99.46 %
110 100 10 11 0 10 10 0 0 01	06:02:53	99.51 %
110 000 01 10 0 10 01 1 0 10	06:22:39	99.49 %
100 010 01 10 0 10 01 1 0 10	06:32:29	99.41 %
Promedio	**06:24:11**	**99.474 %**

8. Conclusiones

Las principales contribuciones de este trabajo son el planteamiento del algoritmo genético compacto distribuido así como la arquitectura necesaria para su implementación, con lo cual se redujeron significativamente los tiempos de búsqueda de una configuración de hiperpárametros óptimos. También como observación para obtener un resultado en tiempos óptimo los nodos esclavos necesitan tener un hardware muy similar (capacidad de procesamiento), debido a que el AGCD esperará la finalización de los dos individuos, esto quiere decir que el tiempo de ejecución de una generación dependerá del equipo con menor capacidad de cómputo.

Tabla 5. Comparación de promedios de los experimentos realizados.

Configuración	**Tiempo**	**Precisión**
AGC simple	09:25:37	99.458 %
AGCD con un esclavo	09:42:59	99.433 %
AGCD con dos esclavos	**06:24:11**	**99.474 %**

Los resultados promedio obtenidos (ver la Tabla 5) se han comparado con otros trabajos tales como el de Tabik y et al. [10] en los cuales podemos observar que la precisión obtenida en este trabajo utilizando el MNIST original es muy similar e incluso superada, entonces podemos afirmar que en la metodología propuesta es comparable con el estado del arte en términos de precisión.

Es importante considerar la asignación de la configuración (individuo), ya que esta puede ser mas costosa en tiempo de ejecución, y al ser entregada al equipo con menor capacidad de procesamiento producirá que el equipo con mayor capacidad culminará la evaluación del individuo y esperará la finalización del

equipo con menor capacidad y mas carga de trabajo. También es necesario que los nodos esclavos y el nodo maestro se encuentren interconectados a través de una red y se recomienda que sea una conexión alámbrica.

Para finalizar podemos concluir que la implementación del algoritmo genético compacto distribuido podemos obtener una mejora significativa en los tiempos de ejecución comparado con el algoritmo genético compacto simple.

Referencias

1. Hegde, V., Usmani, S.: Parallel and distributed deep learning. Stanford University (2016)
2. Brownlee, J.: Clever algorithms: nature-inspired programming recipes. First Edition, Jason Brownlee, Australia (2011)
3. Desell, T.: Large scale evolution of convolutional neural networks using volunteer computing. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM (2017)
4. Castillo, P.A., Arenas, M.G., Mora, A.M., Laredo, J.L.J., Romero, G., Rivas, V.M., Merelo, J.J.: Distributed Evolutionary Computation using REST. arXiv preprint arXiv:1105.497 (2011)
5. Patterson, J., Gibson, A.: Deep learning: A practitioner's approach. O'Reilly Media, Inc. (2017)
6. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. IEEE transactions on evolutionary computation 2(5), 287–297 (1999)
7. Flynn, M.J.: Some computer organizations and their effectiveness. IEEE Transactions on Computers, pp. 94–96 (1972)
8. Kshemkalyani, A.D., Singhal, M.: Distributed computing: principles, algorithms, and systems. Cambridge University Press, Inc. (2011)
9. Deng, Li.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, pp. 141–142 (2012)
10. Tabik, S., Peralta, D., Herrera-Poyatos, A., Herrera, F.: A snapshot of image preprocessing for convolutional neural networks: case study of MNIST. International Journal of Computational Intelligence Systems, pp. 555–568 (2017)