Artículo **Revista de Tecnologías Computacionales**
Septiembre, 2019 Vol.3 No.11 7-16

Identificación de figuras geométricas en un sistema de visión basándose en el entrenamiento de una red neuronal artificial convolucional utilizando Python

Identification of geometric figures in a vision system based on the training of a convolutional artificial neural network using Python

AGUILERA-HERNÁNDEZ, Martha Isabel†*, VELASCO-MARIN Jorge Alan, ORTIZ-SALAZAR, Manuel y ORTIZ-SIMÓN, Jose Luis

Instituto Tecnológico de Nuevo Laredo

ID 1er Autor: *Martha Isabel, Aguilera-Hernández* / **ORC ID:** 0000-0001-8127-190X**, Researcher ID Thomson:** S-4724-2018, **CVU CONACYT ID:** 19115

ID 1er Coautor: *Jorge Alan Velasco-Marín* / **ORC ID:** 0000-0002-5649-9504

ID 2do Coautor: *Manuel, Ortiz-Salazar* / **ORC ID:** 0000-0002-1680-4025 **CVU CONACYT ID:** 558056

ID 3er Coautor: *Jose Luis, Ortiz-Simón* / **ORC ID:** 0000-0001-6548-3849, **Researcher ID Thomson:** S-7037-2018, **CVU CONACYT ID:** 289883

DOI: 10.35429/JOCT.2019.11.3.7.16

Recibido 23 de Julio, 2019; Aceptado 3 Septiembre, 2019

Resumen

Los proyectos de procesamiento de imágenes por medio de sistemas de visión, son un gran punto de apoyo didáctico en la carrera de mecatrónica, ya que tienen amplia aplicación en la industria en las líneas de proceso primordialmente para realizar trabajos de ensamble, inspección, selección y colocación de componentes. Uno de los métodos utilizados es la de aplicar redes neuronales artificiales para la identificación de las imágenes y un factor a analizar es la evaluación de las capacidades de aprendizaje de estas redes en identificación de figuras geométricas. En este artículo, se presenta el entrenamiento de una red neuronal artificial convolucional utilizando Python. Este tipo de trabajos está enfocado a unir proyectos basados en industria 4.0 que puedan contener opciones de enlace con sistemas de proceso basados en estas tecnologías En este trabajo, se realizó un sistema de visión basado en programación python y tiene su aportación en que se diseñaron librerías que pueden enlazarse con diferentes tipos de aplicaciones dentro de un proceso de manufactura

Red neuronal convolucional, sistema de visión, entrenamiento

Resumen

The image processing projects through vision systems, are a great didactic support point in the mechatronics career, since they have wide application in the industry in the process lines primarily to perform assembly, inspection, selection and component placement. One of the methods used is to apply artificial neural networks for the identification of images and a factor to analyze is the evaluation of the learning capacities of these networks in the identification of geometric figures. In this article, the training of a convolutional artificial neural network using Python is presented. This type of work is focused on joining projects based on industry 4.0 that may contain link options with process systems based on these technologies. In this work, a vision system based on python programming was made and has its contribution in the libraries that were designed and can be linked to different types of applications within a manufacturing process.

Convolutional neural network, vision system, training

Citación: AGUILERA-HERNÁNDEZ, Martha Isabel, VELASCO-MARIN Jorge Alan, ORTIZ-SALAZAR, Manuel y ORTIZ-SIMÓN, Jose Luis. Identificación de figuras geométricas en un sistema de visión basándose en el entrenamiento de una red neuronal artificial convolucional utilizando Python . Revista de Tecnologías Computacionales. 2019. 3-11: 7-16

* Correspondencia del Autor (Correo electrónico: maguilera@hotmail.com)
† Investigador contribuyendo como primer autor.

Introducción

Una red neuronal artificial, como su nombre lo dice, pretende emular por medio de un software y un sistema operativo, el comportamiento y la manera en que funcionan las neuronas del cerebro. Los cursos de ingeniería e informática involucran cada vez más temas y proyectos aplicando modelos de redes neuronales artificiales. Esto se debe a la continua búsqueda de mejores métodos para que los medios de procesamiento de datos puedan asemejarse más al funcionamiento del cerebro humano.

El entrenamiento de un sistema de visión se considera un buen proyecto introductorio hacia las redes neuronales artificiales, pues brinda la facilidad de comprender todos los aspectos necesarios que se tienen que considerar al momento en que algún usuario desee desarrollar su propio entrenamiento de red neuronal artificial para procesamiento de imágenes.

El programa debe de ser capaz de entrenar a la neurona, de manera que el modelo logre diferenciar y clasificar diferentes tipos de imágenes, las cuales estarán dadas por "clases". En el caso del presente artículo, se presenta un modelo capaz de diferenciar y de clasificar los dibujos elaborados a mano de triángulos, cuadrados y círculos en 3 diferentes clases.

El artículo se organiza de la siguiente forma: primeramente, los antecedentes para después enfocarse al desarrollo que describe la configuración de la red neuronal, y el procesamiento de imágenes.

Antecedentes

Existen muchos tutoriales que muestran el funcionamiento de sistemas de visión entrenados por redes neuronales artificiales programadas en lenguajes *C, Python* y *MATLAB*. Visto desde el punto de vista didáctico, se consideran un apoyo para el estudiante para lograr diseños de modelos neuronales con mayor complejidad. El programa del modelo neuronal, debe utilizar un conjunto de conceptos y reglas que se generan de acuerdo al entorno de desarrollo y que al mismo tiempo representen resultados positivos y cercanos al del pensamiento humano.

Así, el modelo neuronal entrenado, es capaz de diferenciar y clasificar las imágenes dadas para su procesamiento, mediante una serie de procedimientos propuestos para alcanzar su objetivo. En este caso, la meta es clasificar tres clases de imágenes con un porcentaje de acierto por arriba del 90%.

En la actualidad existen muchos métodos capaces de realizar el procesamiento de información por medio de redes neuronales, y que presentan diversas librerías que permiten el manejo de todos los elementos que conforman a una red neuronal, entre las más populares están *TensorFlow* y *Keras*.

El primer modelo computacional de una red neuronal, fue formalmente establecido en 1943 por el neuropsicólogo Warren McCulloch, y el matemático Walter Pitts. En su trabajo, discuten la manera en la que las neuronas del cerebro funcionan. Su modelo matemático marca la creación de un campo de estudio nuevo. El incentivo principal del proyecto fue encontrar la manera en la que el cerebro opera; sin embargo, el desarrollo en aquel entonces estaba altamente limitado por la poca capacidad de procesamiento de los ordenadores de la época. Una década después, la capacidad de procesamiento de las computadoras logró ser suficiente para poder simular redes neuronales hipotéticas, y eventualmente los investigadores descubrieron que la relación de patrones y las capacidades de aprendizaje de una red neuronal podrían permitirles resolver una amplia variedad de problemas que serían demasiado complicados si se intentaran resolver por métodos y algoritmos convencionales. El primer modelo de red neuronal en resolver un problema de la vida real, fue desarrollado por Bernard Widrow y Marcian Hoff de Stanford en 1959. El nombre del proyecto fue *ADALINE*, y era capaz de reconocer patrones binarios, y consistía en la lectura de un flujo de bits de una línea telefónica, logrando entrenar al modelo para que fuera capaz de predecir el siguiente bit de dicho flujo, lo cual permitía eliminar ecos del flujo de datos. Como consecuencia, las redes neuronales artificiales probaron ser una poderosa herramienta capaz de resolver problemas complejos, como la percepción de datos, aprendizaje de conceptos, el desarrollo de habilidades motoras y reconocimiento de voz e imágenes.

ISSN 2523-6814

El área de las redes neuronales continuó con su auge hasta la década de 1990, pero siempre existió un problema constante con el desempeño, puesto que el desarrollo de redes neuronales con mayor robustez, podía tomar semanas, o incluso hasta meses en poder entrenarse, utilizando hardware antiguo. Hoy en día, los logros que han obtenido las redes neuronales son impresionantes, pues se han aplicado en un amplio rango de campos, por mencionar algunos ejemplos: se utilizan en la medicina para identificar estructuras cerebrales en imágenes de resonancias magnéticas, en geología para analizar volúmenes de datos 3D sísmicos, en energéticos para predecir las cargas de consumo eléctrico de la región, o en logística para el manejo de camiones.

Existen muchos proyectos realizados con anterioridad que marcan un antecedente importante para la culminación de este proyecto. Pues antes, la manera de procesar imágenes, era analizando cada pixel de manera individual, lo cual tomaba mucho tiempo y requería de mucho poder de procesamiento por parte de los ordenadores. Las redes neuronales han ido evolucionando para facilitar e incrementar la velocidad del procesamiento de imágenes, con el uso de diversos métodos, entre ellos destaca el método por convolución. En el 2018, Prakash realizó una publicación en un blog, en la cual demuestra la ventaja que tiene el método de convolución sobre los demás tipos de redes, cuyos fundamentos y ejemplos son aplicados en este proyecto. Otro proyecto fue publicado en el 2018, Olivetto realizó un estudio sobre el aprendizaje profundo para la identificación de objetos en robótica móvil, en el cual muestra la comparación y evaluación de dos redes neuronales artificiales de tipo convencional, aplicadas a un sistema de reconocimiento automatizado de objetos. Proyecto en el cual tuvieron que programar dos topologías de redes neuronales en *MATLAB*, las cuales fueron entrenadas con cuatro categorías de imágenes para obtener como resultados la cuantificación del tiempo de procesamiento, precisión, datos perdidos, y la taza de aprendizaje. Información que utilizaron para determinar cuál de las redes presenta los mayores beneficios en cuestiones de procesamiento y en resultados del proceso de clasificación.

Desarrollo del programa

Una red neuronal convolucional (RNC) se utiliza para procesar imágenes, y pueden aprender relaciones entrada-salida, donde la entrada es una imagen, siendo las aplicaciones más comunes:

- Detección y clasificación de objetos
- Clasificación de escenas
- Clasificación de imágenes en general

Una RNC cuenta con una capa de entrada, la cual se va a encargar de recibir todos los pixeles de la imagen. Por lo tanto, en el caso de tener una imagen de 10 x 10 pixeles, se tendrán 100 entradas para recibir la imagen. Posteriormente, la capa de entrada redirigirá toda la información a las capas ocultas, que es en donde se encuentra la información importante, y donde se llevan a cabo dos operaciones importantes, siendo la primera la agrupación (conocida como *pooling* en inglés).

La segunda operación que se lleva a cabo dentro de las capas ocultas es la convolución. La convolución consiste en aplicar varios filtros a la imagen, de manera que se detecten ciertos patrones relevantes dentro de ella. En la convolución, cada pixel de salida es una combinación de los pixeles de entrada, en otras palabras, es una ventana deslizante en la que las máscaras representan la conectividad entre las capas sucesivas. Por lo tanto, cada convolución irá pasando por toda la imagen, y formará una nueva imagen de salida, la cual tendrá una altura y longitud más pequeña, pero incrementará en su profundidad.

Posteriormente, entra en acción la operación de agrupación, la cual es una operación en la cual se reduce aún más el tamaño de la imagen dada, pero dejando la profundidad intacta para que cuando pase a las siguientes capas, el procesamiento sea sobre una imagen más reducida, haciendo la tarea del procesamiento menos exigente en cuestión de recursos para los equipos computacionales.

Por lo tanto, los parámetros más importantes dentro de una capa convolucional, son el tamaño del filtro, la profundidad de la capa y el *stride* o paso entre pixeles.

ISSN 2523-6814

El tamaño del filtro es el tamaño de la ventana deslizante que se irá pasando por la imagen, el filtro procesará la imagen poco a poco, y está definido por una altura y longitud dadas.

La profundidad de la capa convolucional es el número de filtros que se aplicarán a la imagen. Cada filtro aplicado generará una nueva imagen, en la cual dé como resultado la detección de ciertos patrones deseados como bordes, sombras y contrastes.

La función del agrupamiento es reducir el tamaño de la imagen final, lo cual se requiere por dos razones:

- Para reducir el número de conexiones y que la red tenga mayor facilidad de procesar toda la información.

- Evitar un sobreajuste en el modelo.

Para utilizar aprendizaje máquina, una opción es el "*Keras"*. Con este software se pueden implementar redes neuronales. *Keras* está escrito en Python, un lenguaje predominante en el mundo del aprendizaje máquina.

Una función de activación de un nodo, define la salida de ese nodo, dada una entrada o varias entradas. No son lineales y son muy importantes puesto que afectan a la exactitud de los modelos neuronales. Las funciones de activación más populares son: Identidad, Paso Binario, Sigmoidal, Tangente hiperbólica, Arco tangente, Unidad Lineal Rectificada (ReLU), Leaky ReLU, Softmax.

Las funciones de activación juegan un papel muy importante al momento de decidir la exactitud deseada para el modelo

Para explicar la razón por la cual se escogió una CNN para el proyecto, se muestra la Figura 1. La estructura de una imagen es de una longitud, una altura y una profundidad (Figura 1). Por lo general, la profundidad de una imagen es de 3, ya que existen tres canales de colores en las imágenes: *RGB* (rojo, azul y verde). Si se le alimentara la información de la figura 1, a una red neuronal normal, ésta crearía una conexión con todos los pixeles de la imagen que se le den al sistema, con lo cual surgen dos problemas muy importantes.

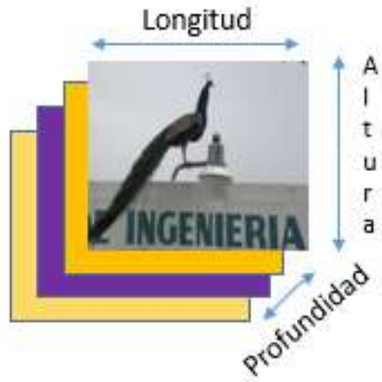

Figura 1 Parámetros de una imagen.

El primero es que el número de conexiones que se tiene entre capas es extremadamente grande, entonces en el caso de tener una imagen de 100×100 pixeles, tenemos que la primera capa oculta tiene 100 neuronas, esto daría como resultado 3,000,000 de operaciones, lo cual requiere mucho tiempo y poder de procesamiento. Este problema no existe en una red neuronal convolucional, por lo tanto, fue la elección apropiada para el desarrollo del proyecto.

Para poder entrenar y poner a prueba la red neuronal, se preparó una serie de imágenes que fueron alimentadas al programa para que éste pueda entrenarse. Se optó por pedirles a diferentes personas que hicieran 25 triángulos, círculos y rectángulos (representan las tres clases para el modelo) para que todo el material de prueba, entrenara al modelo de manera que este logre obtener una variedad más amplia en su reconocimiento de formas, puesto que estará entrenado con la caligrafía de diversas personas. Una vez dibujadas un total de 125 figuras de cada clase sobre hojas blancas (con un total de 375), es necesario escanear las hojas con la calidad más alta posible, por ejemplo 600 puntos por pulgada (DPI – *Dots Per Inch*), puesto que es necesaria para obtener la mejor resolución posible de las imágenes.

Después, es recomendable utilizar software de edición de imágenes, para recortar las figuras y guardarlas en formato *.jpeg*. Las imágenes deben de tener un nombre de archivo que puede variar según el tipo de imágenes a utilizar, en este caso fueron nombradas así: *circulo.1.jpeg*, *triangulo.1.jpeg*, *cuadrado.1.jpeg*, utilizando la siguiente nomenclatura para nombrar los archivos: *claseN.<<secuencia numérica>>.jpeg*, las imágenes serán etiquetadas en base a su nombre de archivo.

ISSN 2523-6814

Una vez terminadas todas las imágenes, se deben de separar en dos juegos, uno de entrenamiento y uno de prueba o validación. Para esto, se crearon dos carpetas, una con el nombre *train* y otra con el nombre "test". La carpeta *train*, debe de contener al menos100 imágenes de cada clase, siendo un total de 300 imágenes de entrenamiento, y la carpeta *test* debe de tener al menos 25 imágenes de cada clase, siendo un total de 75 imágenes de prueba.

Las imágenes pueden ser de cualquier clase deseada, desde vehículos como: carros, aviones o motocicletas, o especies de diferentes de animales, incluso pueden estar mezcladas y pueden ser descargadas desde google, o capturadas por una cámara.

Tomando en cuenta las ventajas para el procesamiento de imágenes que posee una RNC, es posible crear un clasificador de imágenes utilizando *Python* y *Tensorflow*. En las Figuras 2 y 3, se muestran los bloques de código que llevan a cabo el entrenamiento de la red neuronal artificial.

```
import cv2
import numpy as np
import os
from random import shuffle
from tqdm import tqdm
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline

train_data = 'C:/Users/jorge/Pictures/0/train'
test_data = 'C:/Users/jorge/Pictures/0/test'

def one_hot_label(img):
    label = img.split('.')[0]
    if label == 'circle':
        ohl = np.array([1,0,0])
    if label == 'cuadrado':
        ohl = np.array([0,1,0])
    elif label == 'triangle':
        ohl = np.array([0,0,1])
    return ohl
def train_data_with_label():
    train_images = []
    for i in tqdm(os.listdir(train_data)):
        path = os.path.join(train_data, i)
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (64,64))
        train_images.append([np.array(img), one_hot_label(i)])
    shuffle(train_images)
    return train_images

def test_data_with_label():
    test_images = []
    for i in tqdm(os.listdir(test_data)):
        path = os.path.join(test_data, i)
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (64,64))
        test_images.append([np.array(img), one_hot_label(i)])
    return test_images
```

a)

b]

c)

Figura 2 Parte 1 - Clasificador de figuras con activación softmax.

Es necesario importar las librerías necesarias de *opencv*, *numpy*, *os*, *tensorflow* y *matplotlib* como se muestra en a) lo cual permitirá utilizar comandos correspondientes a cada librería en el programa. Después se declaró la dirección de las imágenes a utilizar para prueba y entrenamiento tal y como se muestra en b). Es una buena práctica, que la carpeta que contiene las imágenes, no se encuentre dentro de muchas subcarpetas, para mantener la dirección de ubicación corta.

En c), la función *one_hot_label*, se encargará de agregar etiquetas a todas las imágenes utilizadas, basándose en el nombre de la imagen. En este ejercicio, el nombre utilizado para una clase de las imágenes es: *circle.1.jpeg*. Después se separa el nombre de la etiqueta utilizando un ".", y basado en el primer elemento, se pudo etiquetar la información de la imagen. Cuando se utiliza el método de *one hot encoding*, por ejemplo: en caso de tener 5 clases, entonces solo una clase tendrá el valor 1 y las demás serán cero. En este caso, se tienen tres clases ["circle", "cuadra", "triangulo"], por lo tanto, todas las imágenes de círculos, tendrán como etiqueta el arreglo [1,0,0] y todas las imágenes de cuadrados tendrán el arreglo [0,1,0], y así sucesivamente con clases extra.

Se definió la función de entrenamiento por medio de etiquetas con la función *train_data_with_label*, se asignó la dirección de las imágenes de entrenamiento utilizando el comando *string train_data*, que ya fue previamente declarado. Se asignó al programa el valor *path*, que es la trayectoria a seguir para obtener la información de entrenamiento, mientras que "*img*" es la lectura de las imágenes encontradas en la trayectoria, a las cuales se les aplicará un filtro de escala de grises utilizando *cv2.IMREAD_GRAYSCALE*. Después, se redimensionaron las imágenes para que tengan un tamaño de 64×64, por lo que no es necesario buscar modificar el tamaño de todas las imágenes que se deseen ocupar para el entrenamiento o prueba del programa. La función de *shuffle* y *return* van a "barajear" las imágenes, para que, de esta manera, el programa retorne las imágenes ordenadas aleatoriamente.

El segmento *test_data_with_label* se encarga de convertir la información de las imágenes en un arreglo *numpy* de tamaño 64×64. Es por ello, que las imágenes que se descarguen de internet o que sean creadas, pueden tener resoluciones diferentes y por ello es que se debe de redimensionar cada imagen a 64×64, el cual es un tamaño completamente arbitrario, puesto que se podría utilizar 16×16 ó 128×128, asegurándose de que la imagen mantenga por lo menos un poco de información que pueda ser relevante para la identificación de características deseadas.

ISSN 2523-6814

Posteriormente las imágenes fueron convertidas a escala de grises aún después de su redimensionamiento. En el caso de este proyecto, en el cual se pretende identificar figuras geométricas, no es necesario mantener un espectro amplio de colores, ya que no es necesario clasificar imágenes por colores, o buscar elementos que sean de un color en específico. Por lo que la conversión a escala de grises es perfecta para esta aplicación. En esta función se convirtió la imagen en un arreglo de pixeles y se agregó una *one hot label* a cada una. La segunda función de *test_data_with_label*, también hará lo mismo para el set de datos, con la diferencia de que estas imágenes no serán "barajeadas". La razón para barajear el juego de imágenes de entrenamiento es porque se debe de asegurar que el modelo obtenga imágenes de ambas clases en cada lote. Si las imágenes no se "barajean", el modelo neuronal podría terminar alimentando solamente de imágenes de círculos en el primer lote, o con puros cuadrados en otro lote.

La intención es acercarse al resultado iterativamente para que el modelo calcule el error basándose en la predicción y en la etiqueta de cada lote, por lo que, si la información de alimentación no es revuelta, entonces el aprendizaje del modelo no será ideal porque una sola clase dictará la dirección de la trayectoria del aprendizaje, y si se deja así, la optimización del programa sólo reconocerá una sola clase y no será capaz de recuperarse, aun cuando se le presenten otras clases posteriormente.

Una vez preparados los datos de entrenamiento y de prueba, el siguiente paso es construir el modelo con las librerías de *Keras* indicadas en d). En d) se muestra la importación del modelo Sequential, junto con las capas y sus optimizadores desde Keras. Después, en e), se carga la información a las variables *training_images* para las imágenes de entrenamiento y a *testing_images* las imágenes de prueba.

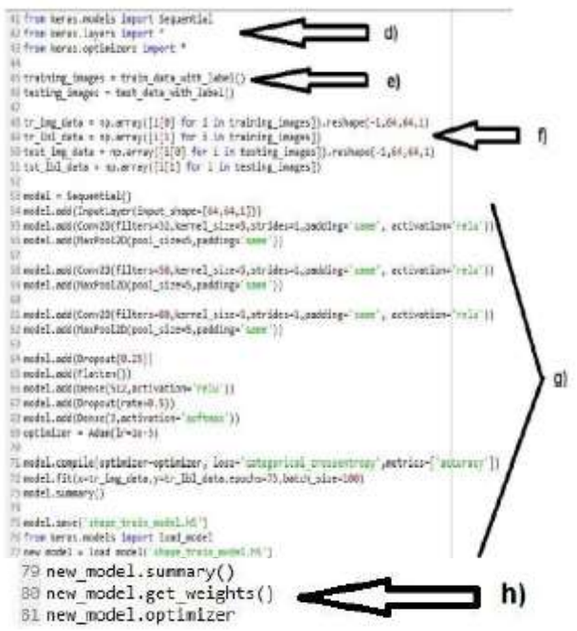

Figura 3 Parte 2 - Clasificador de figuras con activación softmax.

En f), las imágenes serán procesadas en un tamaño de 64×64 y posteriormente pasarán a ser "aplanadas" para que puedan ser enviadas a través de la capa convolucional, por lo que se redimensionaron las imágenes con *reshape*, a un tamaño −1×64×64×1, donde el valor 1 representa el código de color en escala de grises. Posteriormente, en g), se utilizaron 3 capas de convolución con un paso unitario, un *padding* igual y una activación ReLU (la cual puede ser cambiada para encontrar diferentes comportamientos en el programa). El conteo del filtro incrementa por cada capa, esto es debido a que la capa inicial representa a las características de alto nivel de la imagen y las capas con mayor profundidad representarán características más detalladas y por lo tanto suelen tener un mayor número de filtro. La cuenta del filtro es un número arbitrario y se puede manipular para ver los distintos comportamientos del modelo.

Después de las tres capas de convolución, existe una capa de expulsado (*dropout*), que evita que surja un problema de sobreajuste. Una vez que la imagen pasó a través de las capas de convolución, tiene que ser aplanada de nuevo para poder ser alimentada a las capas interconectadas, a esto se le conoce como una capa densa en *Keras*, y aquí todas las neuronas de la primera capa están conectadas con las neuronas de la segunda capa.

ISSN 2523-6814

Se tienen dos capas densas y la primera tendrá 512 neuronas y una activación ReLU, la cuenta de las neuronas puede ser asignada por el usuario, por lo que es un valor arbitrario. La segunda capa densa tiene tres neuronas, ya que solamente se clasificarán tres clases, usualmente el número de neuronas en la capa de salida es igual al número de clases a ser clasificadas, y esto aplica para este caso. La última capa se configuró con una activación softmax, la cual calculará las probabilidades de cada clase sobre todas las clases posibles y la suma de las probabilidades siempre será 1. La entrada será clasificada en cualquiera de las clases objetivo finales, basado en el valor de probabilidad más alto en softmax. Por ejemplo, si la salida de softmax es [0.7, 0.3, 0.4], la imagen de entrada es un círculo.

Se utilizó el optimizador Adam con una cros-entropía categórica: *categorical_crossentropy* que será la función de pérdida con una taza de aprendizaje de 0.001.

Se entrenó el modelo por 75 épocas (el modelo ajustará su valor de parámetros por cada época, para minimizar la pérdida) obteniendo así una exactitud esperada de aproximadamente 100% (1.0000 en el programa). Posteriormente, en h), se genera un resumen de las características que se autogenerarán al entrenarse el modelo.

Al correr el programa, la red neuronal convolucional comienza a entrenar al sistema de visión. En las figuras 4 y 5, se muestra la predicción del tiempo estimado de operación, el número de la época, la función de pérdida, el porcentaje de exactitud y el tiempo real transcurrido, la pérdida real, y la exactitud real. El incremento en la exactitud de la predicción, que indica que el modelo está siendo mejorado con cada época y calcula la relación de acierto-error para poder mostrar ese valor.

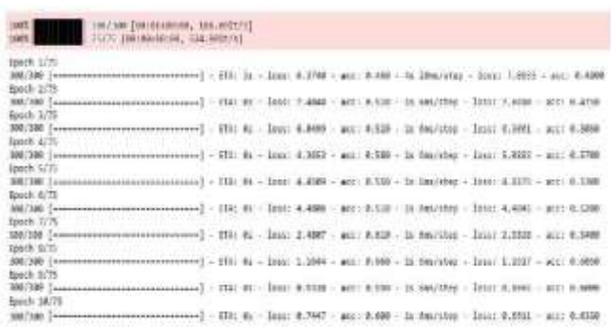

Figura 4 Parte 1 – Entrenamiento de red neuronal convolucional, con activación softmax

En la figura 3, se muestra el incremento del valor de la exactitud calculada, el cual asciende con cada época.

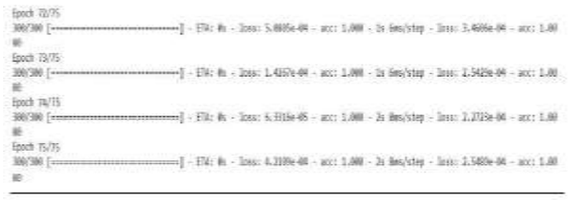

Figura 4 Parte 2 – Entrenamiendo de red neuronal convolucional con activación softmax.

En la figura 4, se puede observar que el entrenamiento alcanza una exactitud calculada de 100% (representado como: *acc: 1.00*), por lo que el paso final es poner a prueba el modelo neuronal, y para poder visualizar los resultados de la corrida del modelo y de su procesamiento de imágenes, es necesario decirle al programa que grafique las imágenes procesadas y que agregue la leyenda de la predicción que hizo a cada imagen, utilizando el código de la figura 5. En i) se establece que "*fig*" sea la gráfica de las figuras, y se le dicta el número de elementos a procesar con *testing_images[10:75]*, lo cual indica que se procesarán 65 imágenes (de la imagen diez, a la 75). En j), se ajusta el tamaño de cada fila y columna de la gráfica, se redimensionan los arreglos de las imágenes y se guarda la información de predicción del modelo en *model_out*.

```
#Graficación de figuras procesadas.                          i)
fig=plt.figure(figsize=(14,14))

for cnt, data in enumerate(testing_images[10:75]):

#Incremento de gráfica.                                      j)
    y = fig.add_subplot(8,9, cnt+1)
    img = data[0]
    data = img.reshape(1,64, 64,1)
    model_out = model.predict([data])

#Asignación de pesos, según la predicción.                   k)
    if np.argmax(model_out) == 1:
        str_label='Triangulo'
    elif np.argmax(model_out) == 2:
        str_label='Cuadrado'
    elif np.argmax(model_out) == 0:
        str_label='Circulo'

#Graficación de resultados y etiquetas.                      l)
    y.imshow(img, cmap='gray')
    plt.title(str_label)
    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
```

Figura 5 Procesamiento de imágenes de prueba utilizando activación softmax.

Posteriormente, en k), se deben asignar valores a cada clase, de la manera: [1,0,0]. Para ello, hay que asignar un peso a cada clase, el cual debe de ser almacenado en *model_out*.

Por lo tanto, el modelo puede diferenciar entre tres predicciones diferentes, según la proximidad del valor de *model_out*, con el valor asignado. En el caso en que el procesamiento de la imagen arroje una predicción de [1, 0, 0], el programa predecirá un círculo, como se muestra en la tabla 1.

Clase	Peso	Resultado de predicción
Círculos	0	[1,0,0]
Triángulos	1	[0,1,0]
Cuadrados	2	[0,0,1]

Tabla 1 Tabla de valores y pesos asignados

Finalmente, después de la asignación de pesos y valores, el modelo se puede poner a prueba. El tiempo de la ejecución del procesamiento puede variar según la capacidad de procesamiento de cada computadora. El resultado de las predicciones hechas por el modelo se muestra en la figura 6.

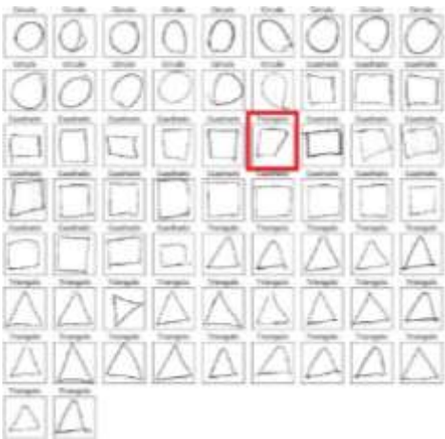

Figura 6 Predicciones de modelo con activación SoftMax.

Como se puede observar en la figura 6, de una población de 65 imágenes, el modelo solamente tuvo un error de predicción en el caso de un cuadrado que confundió con un triángulo (se muestra con un contorno rojo).

La razón de esto es que primero, la red fue entrenada solamente con 300 imágenes, lo cual es una buena cantidad de imágenes para el entrenamiento del modelo, lo cierto es que entre mayor sea la cantidad de imágenes de entrenamiento, el programa será mucho más robusto y tendrá mejor capacidad de predicción.

La segunda razón por la que el programa falló en predecir correctamente algunas imágenes, es porque las imágenes dadas varían demasiado por el hecho de haber sido dibujadas a mano, la convolución solamente extrae partes de la imagen para crear una más pequeña, por lo que si las partes extraídas de la imagen son características parecidas a las de otra clase, el programa puede tener errar en su predicción.

En la Tabla 2, se puede apreciar el desglose de los resultados obtenidos de la figura 6 y se puede observar el porcentaje de exactitud y de error:

$$\frac{100\%}{65\ imágenes} = 1.54$$

Población total de imágenes procesadas	65
Predicciones acertadas	64
Predicciones erróneas	1
% de exactitud	98.46%
% de error	1.54%

Tabla 2 Resultados de ejecución de modelo con activación softmax.

Por lo que al multiplicar el valor de porcentaje unitario por el número de predicciones acertadas nos da como resultado un porcentaje de exactitud de:

$$1.54 * 64 = 98.46\%$$

Por lo tanto, el modelo neuronal, tiene un porcentaje de acierto mayor al 95% esperado en el establecimiento de los objetivos.

Es importante mencionar, que el modelo propuesto puede sufrir una gran variedad de ajustes, sobre todo en los métodos de activación, que podrían incrementar o disminuir el porcentaje de exactitud del programa.

Resultados

La función de activación tiene un impacto muy importante en el entrenamiento del modelo, ya que afecta directamente a la exactitud del mismo.

ISSN 2523-6814

Para demostrar cuál de los tres métodos de activación: LeakyReLU, Sigmoidal y SoftMax tiene un mayor porcentaje de exactitud, se utilizó el mismo programa del apartado 3.3, pero aplicando un cambio en el bloque del programa que se muestra en la figura 2, parte g), línea 68, dónde en vez de softmax, se escribirá sigmoidal para la segunda prueba y se debe de volver a cambiar por leakyrelu para la tercera prueba. Los dos nuevos casos, utilizarán las mismas 65 imágenes que se procesaron en el caso con activación softmax.

En las figuras 7-9, se muestran los resultados de cada método de activación.

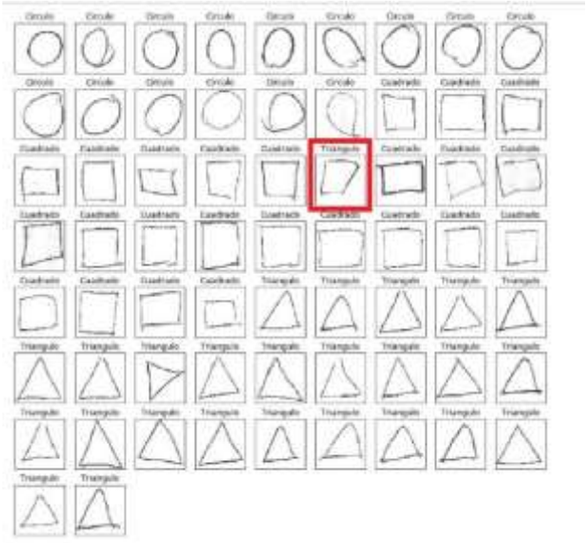

Figura 7 Predicciones de modelo con activación softmax.

En la figura 7, se puede observar que el modelo obtuvo un solo error en sus predicciones.

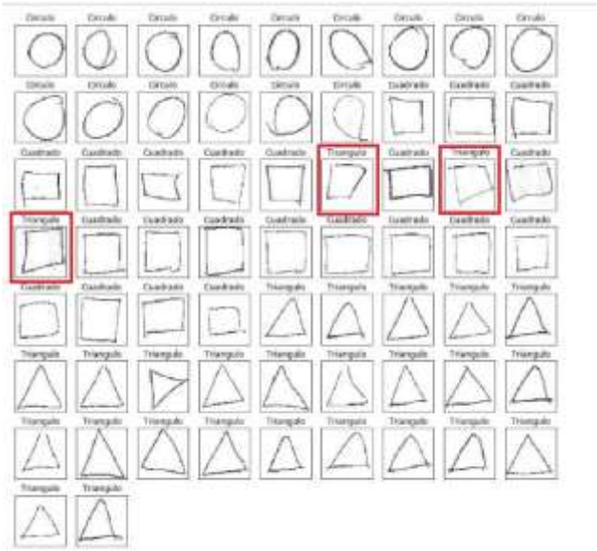

Figura 8 Predicciones de modelo con activación sigmoidal

En la figura 8, se puede observar que el modelo obtuvo tres errores en sus predicciones.

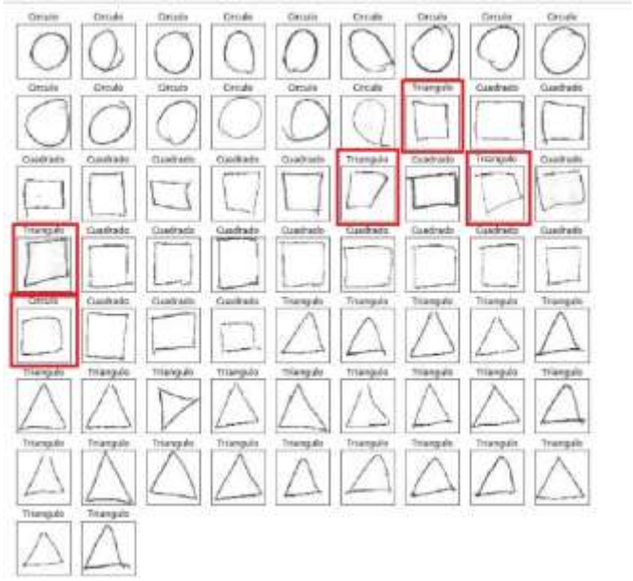

Figura 9 Predicciones de modelo con activación LeakyReLU.

En la figura 9, se puede observar que el modelo obtuvo cinco errores en sus predicciones.

En la Tabla 3 se muestran los resultados obtenidos, y el porcentaje de acierto, el cual se puede calcular utilizando el mismo método que se utiliza para la Tabla 2.

Función de activación	***SoftMax***	***Sigmoidal***	***Leaky ReLU***
Población total de imágenes procesadas	65	65	65
Predicciones acertadas	64	62	60
Predicciones erróneas	1	3	5
% de error	1.54%	4.61%	7.69%
% de exactitud	98.46%	95.38%	92.31 %

Tabla 3 Tabla comparativa de resultados finales, de las diferentes funciones de activación.

Es importante recalcar que la mejor función de activación para las capas escondidas de la red neuronal aritificial, es la ReLU, ya que evita y rectifica los problemas de gradiente descendiente que puedan aparecer en el entrenamiento de la red neuronal convolucional.

ISSN 2523-6814

Por último, utilizando los datos de la Tabla 3, se demuestra que la función de activación con mayor porcentaje de exactitud es la función softmax, seguida por la función sigmoidal y llegando en último lugar, la función LeakyReLU, sin embargo, esto no significa que las últimas dos funciones sean inútiles o que nunca deban de utilizarse. En caso de que el modelo sufra de neuronas muertas durante el entrenamiento, se debe de pensar en cambiar la función de activación por sigmoidal, LeakyReLU, o alguna otra función.

En conclusión, la función de activación recomendada como primera opción para el entrenamiento de cualquier modelo neuronal en estas condiciones es la función de activación softmax.

Conclusiones

En conclusión, la función de activación recomendada como primera opción para el entrenamiento de para reconocimiento de figuras geométricas es la función de activación softmax. A pesar de que los resultados obtenidos de este programa no llegaron a un porcentaje mayor o igual a 99%, de igual manera son altamente satisfactorios, puesto que solo se utilizaron 300 imágenes para entrenar la red neuronal convolucional.

Así mismo, se logró implementar una red neuronal que puede aplicarse para reconocimiento de figuras en un proceso de manufactura. Ya que la red de aprendizaje multicapa pudo clasificar elementos de tres clases diferentes y predecir las imágenes de prueba con un porcentaje de exactitud superior al 95%.

Por lo tanto el proyecto brindó resultados con el cual se pretende abrir la puerta a las generaciones futuras para que puedan mejorar sus modelos neuronales e implementarlos en cualquier aplicación. Existen muchas maneras de realizar una red neuronal convolucional, por lo que se reconoce que pueden existir muchas mejoras para incrementar la eficiencia del modelo. Ante todo, este proyecto es muy afable para introducir a cualquier principiante al mundo del aprendizaje máquina y al lenguaje de programación de *Python*.

Referencias

Velasco, J.. (2019). Implementación de Sistema de Visión Basado en Red Neuronal Artificial y Sistema Difuso en un Robot Móvil. Instituto Tecnológico de Nuevo Laredo: ITNL.

McCulloh, Warren S. and Pitts, Walter, M. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics.* , (5), 115-133,

Francium Tech. (Noviembre 21, 2018). *Image Clasifier Tensorflow*. . Francium Tech Recuperado de https://blog.francium.tech/build-your-own-image-classifier-with-tensorflow-and-keras-dc147a15e38e

Olivetto Rendón, A. (2018). Aprendizaje profundo para la identificación de objetos. *Lla Mecatrónica en México,* (7), p.1.

Diego Calvo. (5 de Diciembre del 2018). *Neural-network*. Recuperado de http://www.diegocalvo.es/definicion-de-red-neuronal/

Researchgate. (6 de Diciembre del 2018). *Convolution operator*. Recuperado de https://www.researchgate.net/figure/Sketch-of-a-basic-convolution-operator-in-the-CNN-architecture_fig1_326988108

[]. (2018 septiembre 21). Convolutional Networks. [Video]. Recuperado de https://www.youtube.com/watch?v=ns2L2T6wvAY

Data science. (7 de Diciembre del 2018). *Activation_functions*. Recuperado de https://towardsdatascience.com/activation-functions-in-neural-networks-58115cda9c96

ISSN 2523-6814