Universidad Autónoma Chapingo

Departamento de Mecánica Agrícola

Ingeniería Mecatrónica Agrícola

Informe 2:
Redes Neuronales Profundas (Deep Learning)

Asignatura:

Inteligencia artificial

Nombre del profesor:

Luis Arturo Soriano Avendaño

Alumno:

Cocotle Lara Jym Emmanuel [1710451-3]

GRADO:	GRUPO:
7°	7

Chapingo, Texcoco Edo. México

Fecha de entrega: 05/12/2021

Índice

Introducción

Las Redes Neuronales Artificiales, RNA están inspiradas en las redes neuronales biológicas del cerebro humano. Están constituidas por elementos que se comportan de forma similar a la neurona biológica en sus funciones más comunes.

Las RNA aprenden de la experiencia, generalizan de ejemplos previos a ejemplos nuevos y abstraen las características principales de una serie de datos.

A principios de los años 80, se desarrolló un concepto que, más tarde se conocería como "Deep Learning" por medio del investigador Geoffrey Hinton, que ayudo a lanzar una revolución ambiciosa al explorar las RNA, en la universidad de Cambridge y la Universidad de Edimburgo, la cual consistía en imitar el celebro por medio del uso de hardware y software, para crear una forma más pura de inteligencia artificial.

Sin embargo, Hinton y sus colegas al trabajar con esta idea, en esa época las computadoras no tenían la capacidad computacional requerida para procesar las enormes colecciones de datos que requerían las RNA, por lo que su éxito fue limitado (García Navarro, B. 2015).

El aprendizaje profundo, son aquellas técnicas de aprendizaje automático que hacen uso de arquitecturas de redes neuronales. El concepto "profundo" viene referido al número de capas que poseen estas técnicas. Mientras que las redes neuronales convencionales posen una capa, las redes neuronales profundas contienen varias capas (Saez De La Pascua, A. 2019).

El aprendizaje profundo abarca algoritmos que funcionan en base a "un proceso por capas". El aprendizaje profundo simula el funcionamiento básico del cerebro, que se realiza a través de las neuronas. En el aprendizaje profundo, esas neuronas serían las capas.

Las grandes empresas tecnológicas están apostando por el desarrollo y la mejora de algoritmos de reconocimiento de voces, imágenes y textos. Google desarrolló con éxito redes neuronales que reconocen voces en teléfonos Android e imágenes en Google Plus. Facebook usa aprendizaje profundo para orientar los anuncios e identificar rostros y objetos en fotos y vídeos. Microsoft lo hace en proyectos de reconocimiento de voz.

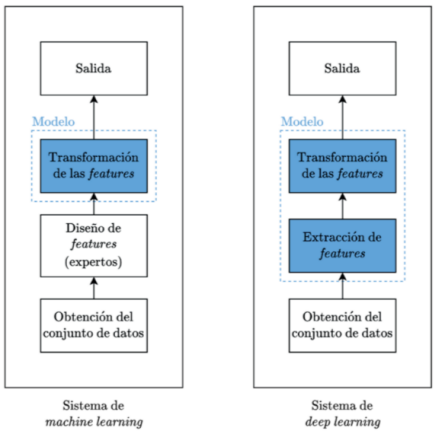
En este informe se presentan las características del aprendizaje profundo y se muestra un ejemplo de la implementación de un algoritmo de aprendizaje profundo para el reconocimiento de rostros.

Desarrollo

Se empezará hablando del aprendizaje profundo y las partes esenciales que la componen, para posteriormente realizar un programa en Python que muestre dichas partes para una mejor comprensión del tema. El programa en Python es un clasificador de rostros.

Aprendizaje profundo (Deep Learning)

El aprendizaje profundo, es una técnica de aprendizaje automático diseñada siguiendo el modelo biológico de red neuronal, el cual mejora su funcionamiento con la observación de nuevos datos y ajusta el aprendizaje a través de múltiples capas de redes neuronales.



1.- Comparación entre machine learning y deep learning.

Una de las características de las técnicas de aprendizaje profundo es que la extracción que se realiza de los datos originales suele generar una representación jerárquica distribuida: se extraen características de menor a mayor complejidad, de forma independiente, definiéndose las características de los niveles superiores en función de las características de los niveles inferiores.

El objetivo de las técnicas de aprendizaje profundo consiste en encontrar unas características de alto nivel, de forma que el modelo pueda transformar estas características en la salida esperada, con más facilidad que empleando la representación original de los datos. Por ejemplo, en el caso de la clasificación de imágenes, las técnicas de aprendizaje profundo procesan la imagen de entrada para extraer características de bajo nivel (como colores y bordes), mientras que en niveles superiores se detectan características más complejas, de más alto nivel (como pueden ser las ruedas de un coche o la cara de una persona). Estas características de más alto nivel serán las utilizadas para generar la salida del modelo (Borrero, I. P., & Arias, M. E. G. 2021).

De forma general, en un modelo de aprendizaje profundo se distinguen dos etapas:

Etapa de extracción de características: consiste en crear la jerarquía de características a partir de los datos en bruto del problema.

Etapa de transformación de características: consiste en tomar las características de más alto nivel de la jerarquía y las utiliza para aplicarles una transformación que permita dar la salida esperada por el sistema.

La parte del entrenamiento de la neurona es la parte más genuina del aprendizaje profundo y podemos ver este proceso de aprendizaje en una red neuronal como un proceso iterativo de "ir y venir" por las capas de neuronas. El "ir" propagando hacia delante lo llamaremos forwardpropagation y el "venir" como retropropagación o backpropagation (Torres, J. 2018).

Forwardpropagation

Esta fase se da cuando se expone la red a los datos de entrenamiento y estos cruzan toda la red neuronal calcular sus predicciones (labels). Es decir, pasar los datos de entrada a través de la red de tal manera que todas las neuronas apliquen su transformación a la información que reciben de las neuronas de la capa anterior y la envíen a las neuronas de la capa siguiente. Cuando los datos hayan cruzado todas las capas, y todas sus neuronas han realizado sus cálculos, se llegará a la capa final con un resultado de predicción de la predicción para aquellos ejemplos de entrada.

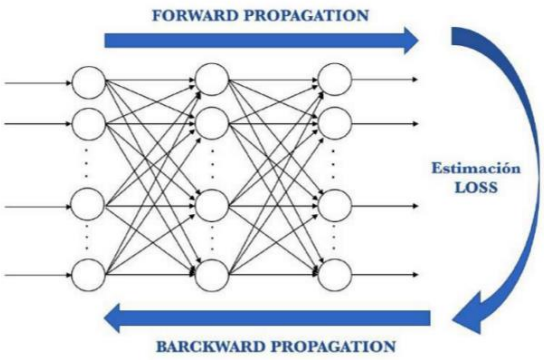
Después, se hace uso de una función para estimar el error (loss), para comparar y medir cuán bueno o malo fue nuestro resultado de la predicción en relación con el resultado correcto.

Lo que se busca es que el error sea cero, es decir, sin diferencia entre el valor estimado y el esperado. Por eso, cada vez que se entrena el modelo se irán ajustando los pesos de las interconexiones de las neuronas de manera automática hasta obtener buenas predicciones.

Backpropagation

Una vez que se tiene calculado el error, se propaga hacia atrás esta información. De ahí el nombre de retropropagación, en inglés backpropagation. Partiendo de la capa de salida, esa información de error se propaga hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo, las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.

Un posible diagrama para explicar lo anterior puede ser el siguiente:



2.- Proceso de aprendizaje profundo.

Ya que existen diferentes situaciones en la que se puede emplear una red neuronal profunda, es necesario conocer que tipo de red neuronal es factible para cada caso, por ejemplo:

- Para el procesamiento de texto, el análisis de sentimientos, el análisis y el reconocimiento de la entidad de nombre, utilizamos una red recurrente o red de tensor neuronal recursivo o RNTN;
- Para cualquier modelo de lenguaje que funcione a nivel de carácter, usamos la red recurrente.
- Para el reconocimiento de imágenes, utilizamos la red de creencias profundas DBN o red convolucional CNN.
- Para el reconocimiento de objetos, utilizamos una RNTN o una red convolucional.
- Para el reconocimiento de voz, utilizamos una red recurrente.

Algunos tipos de redes neuronales profundas son las siguientes:

Redes de Boltzman restringidas o Autoencoders (RBM)

RBM es el equivalente matemático de un traductor bidireccional. Un pase hacia adelante toma entradas y las traduce en un conjunto de números que codifica las entradas. Mientras tanto, una pasada hacia atrás toma este conjunto de números y los traduce de nuevo en entradas reconstruidas. Una red bien entrenada se desempeña de nuevo con un alto grado de precisión.

Redes de creencias profundas (DBN)

Las redes de creencias profundas (DBN) se forman combinando RBM e introduciendo un método de entrenamiento inteligente. En un DBN, cada RBM aprende toda la entrada. Un DBN funciona globalmente ajustando la entrada completa en sucesión a medida que el modelo mejora lentamente como la lente de una cámara enfocando una imagen lentamente.

Redes Generativas antagónicas (GAN)

Son redes neuronales profundas que comprenden dos redes, enfrentadas una con la otra, por lo tanto, el nombre de “antagónicas”. En una GAN, una red neuronal, conocida como el

generador, genera nuevas instancias de datos, mientras que la otra, el discriminador, las evalúa para determinar su autenticidad.

Redes neuronales recurrentes (RNN)

Son redes neuronales en las que los datos pueden fluir en cualquier dirección. Estas redes se utilizan para aplicaciones como el modelado de lenguaje o el procesamiento de lenguaje natural (NLP). Los RNN se denominan recurrentes ya que repiten la misma tarea para cada elemento de una secuencia, y la salida se basa en los cálculos anteriores.

Redes neuronales convolucionales (CNN)

Las CNN se utilizan ampliamente en la visión por computadora; Se han aplicado también en el modelado acústico para el reconocimiento automático de voz.

La idea detrás de las redes neuronales convolucionales es la idea de un “filtro en movimiento” que pasa a través de la imagen. Las CNN reducen drásticamente la cantidad de parámetros que deben ajustarse (IntelDig. 2019).

Programa

Uno de los algoritmos de aprendizaje profundo son las redes neuronales convolucionales, por lo que para la identificación de rostros se optó por este algoritmo, cuyo programa en Python es el siguiente.

```
1  # Universidad Autónoma Chapingo
2  # Ingeniería Mecatrónica Agrícola
3  # Jym Emmanuel Cocotle Lara 7° 7
4
5  # Importamos las librerías a ocupar
6  import os
7  import numpy as np
8  from numpy import asarray
9  import cv2
10 import matplotlib.pyplot as plt
11
12
13 # Directorio donde se encuentran las imagenes que se ocuparán para
entrenar y para probar a la CNN
14 Data_train = "Data/training"  # Imagenes para entrenar
15 Data_det = "Data/detection"  # Imagenes para probar"
16
17
18 # Convolución
19 class Convolucion:
20     # Filtros requeridos para la convolución
21     def __init__(self, num_filt, tam_filt):
```

```
 22         self.num_filt = num_filt
 23         self.tam_filt = tam_filt
 24         self.conv_filt = np.random.randn(num_filt, tam_filt,
tam_filt)/(tam_filt * tam_filt)
 25
 26     # Obtenemos los parches de ruta de las imagenes
 27     def image_region(self,image):
 28         height, width = image.shape
 29         self.image = image
 30         # Extraemos las medidas de los parches
 31         for j in range(height - self.tam_filt + 1):
 32             for k in range(width - self.tam_filt +1):
 33                 # Se crea una imagen a partir de los valores de los
parches.
 34                 image_patch =
image[j:(j+self.tam_filt),k:(k+self.tam_filt)]
 35                 yield image_patch, j, k
 36
 37     # Propagación hacia adelante
 38     def forward_prop(self,image):
 39         height, width = image.shape
 40         sal_conv = np.zeros((height - self.tam_filt + 1, width -
self.tam_filt+1, self.num_filt))
 41         for image_path, i,j in self.image_region(image):
 42             # Multiplicación de los filtros con el parche de la
imagen
 43             sal_conv[i,j]= np.sum(image_path * self.conv_filt,
axis=(1,2))
 44         return sal_conv
 45
 46     # Propagación hacia atrás
 47     def back_prop(self, dL_dout, learning_rate):
 48         dL_dF_params = np.zeros(self.conv_filt.shape)
 49         for image_patch, i,j in self.image_region(self.image):
 50             for k in range(self.num_filt):
 51                 dL_dF_params[k] += image_patch*dL_dout[i,j,k]
 52
 53     # Actualización de los parámetros del filtro
 54         self.conv_filt -= learning_rate*dL_dF_params
 55         return dL_dF_params
 56
 57
 58 # Max-Pooling
 59 class Max_Pool:
 60     # Definición del tamaño del filtro
 61     def __init__(self, tam_filt):
```

```
        self.tam_filt = tam_filt

    # Reducción del tamaño de la imagen
    def image_region(self, image):
        # Nueva altura
        new_height = image.shape[0] // self.tam_filt
        # Nuevo ancho
        new_width = image.shape[1] // self.tam_filt
        self.image = image
        # Extracción de los parches de la imagen nueva que se calcula en la función de convolución
        for i in range(new_height):
            for j in range(new_width):
                image_patch = image[(i*self.tam_filt):(i*self.tam_filt + self.tam_filt), (j*self.tam_filt):(j*self.tam_filt + self.tam_filt)]
                yield image_patch, i, j

    # Propagacion hacia adelante
    def forward_prop(self,image):
        height, width, num_filt = image.shape
        output = np.zeros((height // self.tam_filt, width // self.tam_filt, num_filt))

        for image_patch, i, j in self.image_region(image):
            output[i,j] = np.amax(image_patch, axis = (0,1))

        return output

    # Propagación hacia atrás
    def back_prop(self, dL_dout):
        dL_dmax_pool = np.zeros(self.image.shape)
        for image_patch, i, j in self.image_region(self.image):
            height, width, num_filt = image_patch.shape
            maximum_val = np.amax(image_patch,axis = (0,1))

            for i1 in range(height):
                for j1 in range(width):
                    for k1 in range(num_filt):
                        if image_patch[i1,j1,k1] == maximum_val[k1]:
                            dL_dmax_pool[i*self.tam_filt + i1, j*self.tam_filt + j1, k1]=dL_dout[i,j,k1]
            return dL_dmax_pool

# Softmax
class Softmax:
```

```
103
104     def __init__(self, input_node, softmax_node):
105         # Se inicializa el peso a partir de valores aleatorios
106         self.weight = np.random.randn(input_node,
softmax_node)/input_node
107         # Se crea un sesgo inicial a partir de una matriz de ceros
108         self.bias = np.zeros(softmax_node)
109
110     # Se multiplican los pesos por las bases
111     def forward_prop(self, image):
112         self.orig_im_shape = image.shape
113         image_modified = image.flatten()
114         self.modified_input = image_modified
115         output_val = np.dot(image_modified, self.weight) + self.bias
116         self.out = output_val
117         exp_out = np.exp(output_val)
118         # Transformación de la salida en las salidas probables dadas
119         return exp_out/np.sum(exp_out, axis=0)
120
121     # Propagación hacia atrás
122     def back_prop(self, dL_dout, learning_rate):
123         for i, grad in enumerate(dL_dout):
124             if grad ==0:
125                 continue
126             #
127             transformation_eq = np.exp(self.out)
128             S_total = np.sum(transformation_eq)
129
130             # Gradiente con especto a la salida Z
131             dy_dz = -transformation_eq[i]*transformation_eq /
(S_total **2)
132             dy_dz[i] = transformation_eq[i]*(S_total -
transformation_eq[i]) / (S_total **2)
133
134             # Gradiente del total con pesos y entradas
135             dz_dw =  self.modified_input
136             dz_db = 1
137             dz_d_inp = self.weight
138
139             # Gradiente total contra el perdido
140             dL_dz = grad * dy_dz
141
142             # Gradiente de pérdida contra pesos, bases y entradas
143             dL_dw = dz_dw[np.newaxis].T @ dL_dz[np.newaxis]
144             dL_db = dL_dz * dz_db
145             dL_d_inp = dz_d_inp @ dL_dz
```

```
146
147       # Actualizamos los pesos y las bases
148         self.weight -= learning_rate *dL_dw
149         self.bias -= learning_rate * dL_db
150
151         return dL_d_inp.reshape(self.orig_im_shape)
152
153
154 def dataset_function(directorio,ancho,alto):
155     dataset = []
156     label = ["real","no_real"]
157     for categoria in label:
158         rostro = label.index(categoria)
159         ruta_imagen = os.path.join(directorio,categoria)
160
161         for file_name in os.listdir(ruta_imagen):
162             img =
cv2.imread(os.path.join(ruta_imagen,file_name),cv2.IMREAD_GRAYSCALE)
163             img = cv2.resize(img, (ancho,alto))
164             dataset.append([img,rostro])
165     X = []
166     Y = []
167     for caracteristicas,label in dataset:
168         X.append(caracteristicas)
169         Y.append(label)
170
171     X = np.array(X).reshape(-1,ancho,alto)
172     Y = np.array(Y)
173
174     # Retornamos los valores X,Y
175     return X,Y
176
177
178 (X_train,y_train) = dataset_function(Data_train, 80,80) #resize
179
180 (X_test,y_test) = dataset_function(Data_det, 80,80)
181
182 # imás del entrenamiento (número de imágenes que tengamos)
183 train_images = X_train[0:1400]
184 train_labels = y_train[0:1400]
185
186 # imágenes de la prueba (número de imágenes que tengamos)
187 test_images = X_test[0:20]
188 test_labels = y_test[0:20]
189
190
```

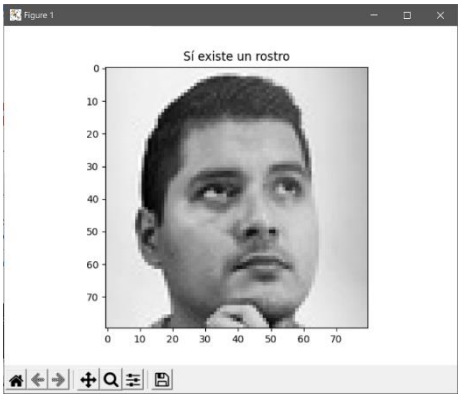
```
num_test = len(test_images)
num_train = len(train_images)

# 8 filtros de 3x3
conv = Convolucion(8,3)
# Operación de encontrar el número mayor
pool = Max_Pool(2)
# (80 - tamaño de filtro + 1/maxpool)*, numero de imagenes a
compactar
softmax = Softmax(39*39*8,5)



def cnn_forward_prop(image, label):
    #  alimentamos a la imagen con la operación de convolución hacia
adelante
    out_p = conv.forward_prop((image /255) - 0.5)
    # le pasamos el parámetro a la operación de max pool
    out_p = pool.forward_prop(out_p)
    # nuevamente le pasamos el parámetro a la función de softmax
    out_p = softmax.forward_prop(out_p)

    # calculamos la pérdida de la entropía y la precisión
    cross_ent_loss = -np.log(out_p[label])
    accuracy_eval = 1 if np.argmax(out_p) == label else 0


    return out_p, cross_ent_loss, accuracy_eval

# vamos a entrenar a la CNN a través de la propagación hacia atrás
# obtenemos los resultados de la salida y alimentamos hacia atrás el
error a las capas anteriores
def training_cnn(image, label, learn_rate = 0.005):
    # Se calcula la salida, la pérdida y la precisión
    out, loss, acc = cnn_forward_prop(image, label)

    # Se calcula el gradiente inicial
    gradient = np.zeros(10)
    gradient[label] = -1 / out[label]

    # propagación hacia atrás
    # le pasamos el valor del gradiente inicial a las funciones
    grad_back = softmax.back_prop(gradient, learn_rate)
    grad_back = pool.back_prop(grad_back)
    grad_back = conv.back_prop(grad_back, learn_rate)

```

```
234     return loss, acc
235
236 # Se entrena a la CNN de acuerdo con el número de épocas que se
quieran
237
238 for epocas in range(5):
239     print("Epoca numero: %d "% (epocas +1))
240
241     # 1400 imágenes se dividen en parches y cada parche tiene 100
imágenes
242     shuffle_data = np.random.permutation(len(train_images))
243     train_images = train_images[shuffle_data]
244     train_labels = train_labels[shuffle_data]
245
246     loss = 0.0
247     num_correct = 0
248
249     for i, (im, label) in enumerate(zip(train_images, train_labels)):
250         #
251         if i % 100 == 0:
252             # Se muestra en la consola el número de paso en el que se
encuentra el entrenamiento
253             # de acuerdo co la época en la que se encuentra
254             print('No. pasos: %d/%d Pérdida promedio: %.3f
Precisión: %d%%' %(i+1, num_train,loss/100, num_correct))
255             loss = 0
256             num_correct = 0
257
258         # Se calcula el entrenamiento de la red llamando a la función
antes creada
259         l1, accu = training_cnn(im, label)
260         loss += l1
261         num_correct +=accu
262
263 # Puesta a prueba de la CNN
264 print("Puesta a prueba...")
265 loss = 0 # Variable para mostrar la perdida
266 num_correct = 0 # Variable conocer el número de imagenes correctas
267 n = 0 # Identificador del número de imagen
268
269
270
271 for i,(im, label) in enumerate(zip(test_images, test_labels)):
272     __, l1, accu = cnn_forward_prop(im, label)
273     loss += l1
274     num_correct += accu
```

```
275
276     if (accu == 1):
277         plt.imshow(X_test[n],cmap="gray")
278         plt.title(f"Sí existe un rostro")
279         plt.show()
280
281     else:
282         plt.imshow(X_test[n],cmap="gray")
283         plt.title(f"No existe un rostro")
284         plt.show()
285
286     n+=1
287
288 # Cálculo de la precisión final
289 precision = (num_correct / num_test)*100
290 # cálculo de la perdida final
291 perdida = loss /num_test
292 # Se visualiza la perdida y la precisión
293 print("Precisión:", precision)
294 print("\nPérdida: ", perdida)
295
```

Resultados

Para poder entrenar a la red neuronal se hizo uso de una base de datos que se encuentra en internet cuyo enlace es el siguiente, https://www.kaggle.com/ciplab/real-and-fake-face-detection, y una vez que se ejecuto el programa, se obtuvieron los siguientes resultados.



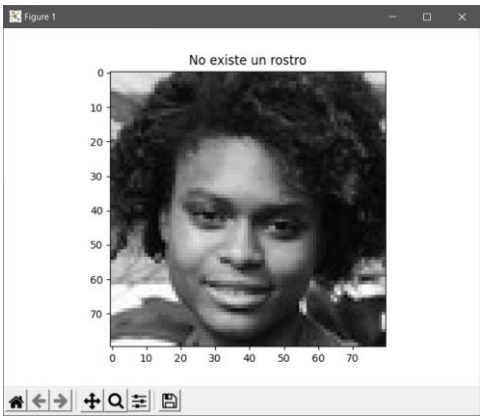


Figure 1
No existe un rostro
0 10 20 30 40 50 60 70

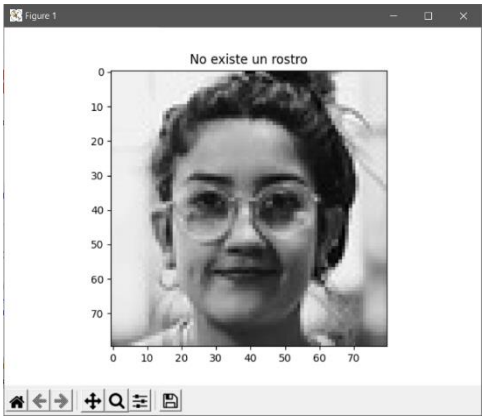


Figure 1
No existe un rostro
0 10 20 30 40 50 60 70

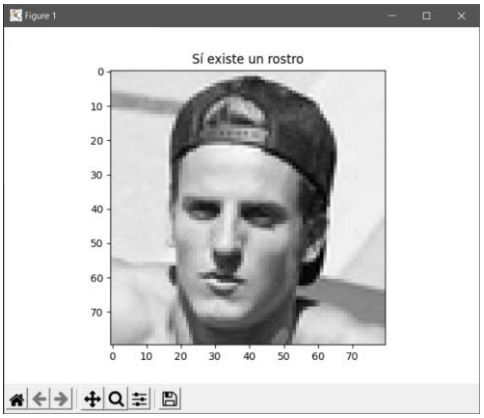


Figure 1
Sí existe un rostro
0 10 20 30 40 50 60 70

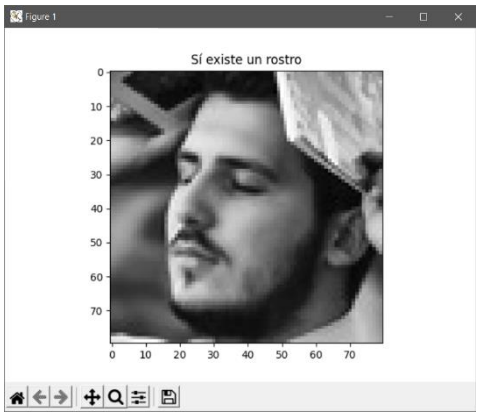


Figure 1
Sí existe un rostro
0 10 20 30 40 50 60 70

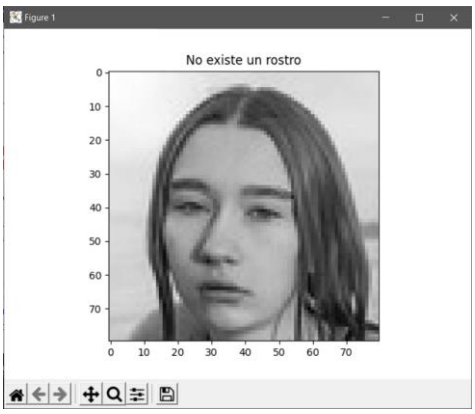


Figure 1
No existe un rostro
0 10 20 30 40 50 60 70

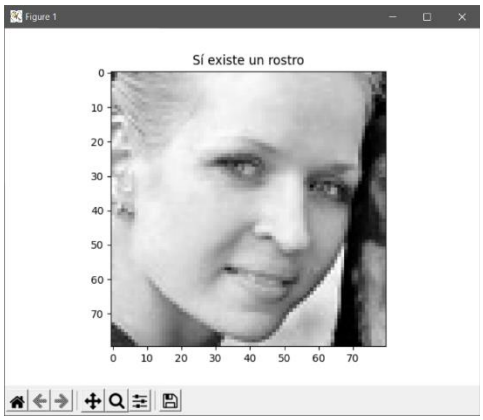


Figure 1
Sí existe un rostro
0 10 20 30 40 50 60 70

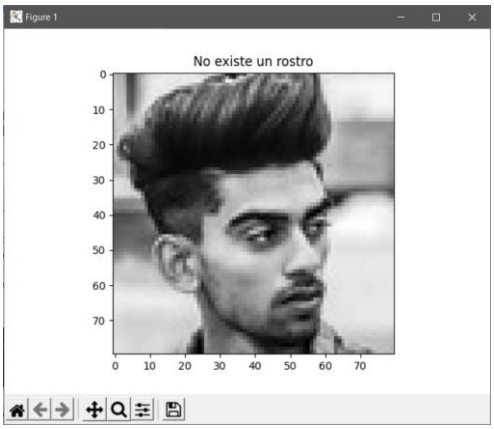


Figure 1
No existe un rostro

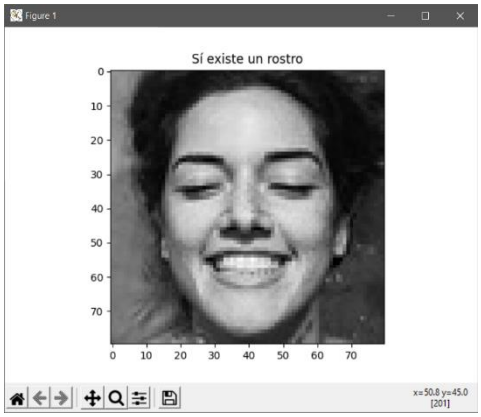


Figure 1
Sí existe un rostro
x=50.8 y=45.0
[201]


Figure 1
No existe un rostro

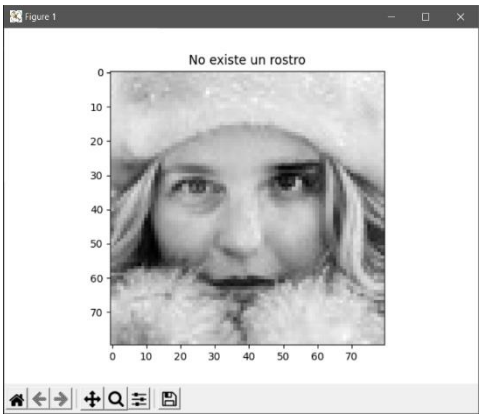


Figure 1
No existe un rostro

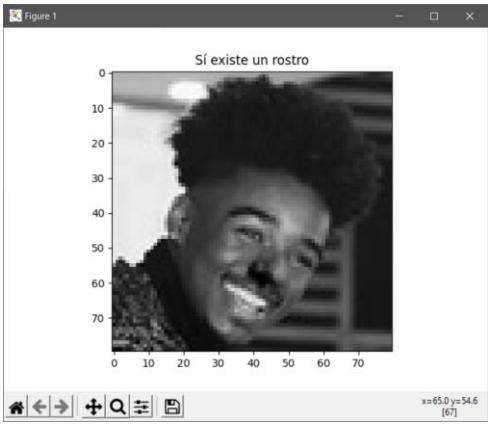
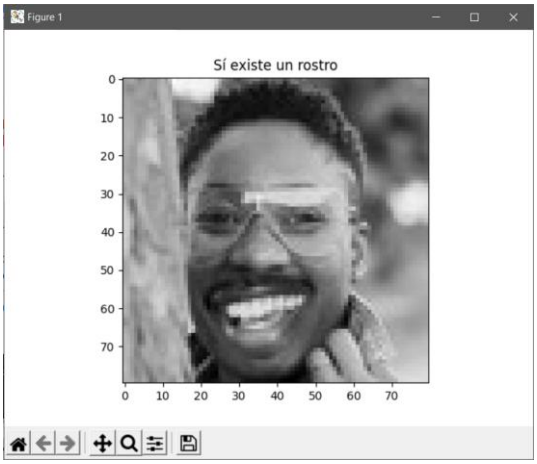
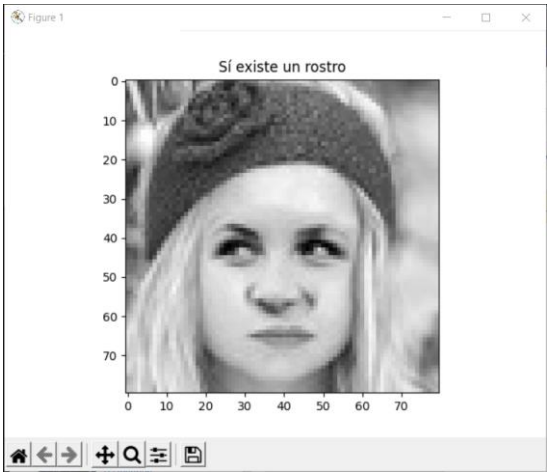


Figure 1
Sí existe un rostro
x=65.0 y=54.6
[67]


Figure 1
No existe un rostro





El programa tardo aproximadamente media hora en ejecutarse y se obtuvo una precisión del 55 % con una pérdida de 0.8019.

Conclusión

El aprendizaje profundo cada vez va tomando mayor importancia, y esto lo podemos ver en las grandes compañías existentes, las cuales han gastado millones de dólares en estos años para el desarrollo de este campo mediante sus propias investigaciones.

Una de las ventajas del aprendizaje profundo es que este puede ser aplicado a diferentes áreas las cuales anteriormente solo eran llevadas a cabo por los seres humanos, con lo cual las posibilidades de crecimiento y evolución del aprendizaje profundo son infinitas.

Bibliografía

García Navarro, B. (2015). Implementación de técnicas de deep learning.

Torres, J. (2018). DEEP LEARNING Introducción práctica con Keras. Lulu. com.

Borrero, I. P., & Arias, M. E. G. (2021). DEEP LEARNING (Vol. 19). Servicio de Publicaciones de la Universidad de Huelva.

Restrepo Arteaga, G. J. P. (2015). Aplicación del aprendizaje profundo (deep learning) al procesamiento de señales digitales (Bachelor's thesis, Universidad Autónoma de Occidente).

Saez De La Pascua, A. (2019). Deep learning para el reconocimiento facial de emociones básicas (Bachelor's thesis, Universitat Politècnica de Catalunya).

IntelDig. (2019). Redes neuronales profundas - Tipos y Características. Código Fuente. Consultado el 5 de Diciembre del 2021, de https://www.codigofuente.org/redes-neuronales-profundas-tipos-caracteristicas/