Universidad Autónoma Chapingo

Departamento de Mecánica Agrícola

Ingeniería Mecatrónica Agrícola

Proyecto Final

Asignatura:

Inteligencia Artificial

Nombre del profesor:

Luis Arturo Soriano Avendaño

Alumno:

Jym Emmanuel Cocotle Lara [1710451-3]

GRADO:	**GRUPO:**
7°	**7**

Chapingo, Texcoco Edo. México

Fecha de entrega: 11/12/2021

Índice

Introducción

La inteligencia artificial es una de las herramientas de desarrollo que ha sido utilizada en gran medida en la actualidad ya que el progreso tecnológico ha sido grande en los últimos años.

Rouhiainen define a la inteligencia artificial como “la habilidad de los ordenadores para hacer actividades que normalmente requieren inteligencia humana”. También se puede definir como la capacidad de las máquinas para usar algoritmos, aprender de los datos y utilizar lo aprendido en la toma de decisiones tal y como lo haría un ser humano (Rouhiainen, L. 2018).

Uno de los elementos de la inteligencia artificial son las redes neuronales artificiales (RNA), las cuales son modelos computacionales formados por neuronas. Estas redes aprenden de la experiencia, realizan una generalización de ejemplos previos a ejemplos nuevos y abstraen las características principales de una serie de datos.

A través de la exploración de las RNA se desarrolló un concepto conocido como “Deep Learning” o “Aprendizaje Profundo”, 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).

Dentro de las RNA existen las redes neuronales convolucionales (CNN por sus siglas en inglés), las cuales son un tipo de redes neuronales artificiales donde las “neuronas” corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico. Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales,

son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones (IntelDig. 2019).

Como se ha mencionado las CNN son utilizadas ampliamente en la visión por computadora o también en el modelado acústico para el reconocimiento automático de voz, siendo la primera aplicación, la de interés en este informe.

En este informe se busca emplear una red neuronal convolucional para el reconocimiento de chayotes.

Desarrollo

A lo largo de este informe se abordará primeramente las partes que componen a red neuronal utilizada para el reconocimiento de frutas o verduras (para este caso se clasificaran chayotes), dicha clasificación se llevará a cabo con una red neuronal convolucional la cual será programada en el lenguaje de Python. Posteriormente se mostrará el programa empleado, y los resultados obtenidos después del entrenamiento de la red neuronal.

Funcionamiento de la red neuronal convolucional.

Las redes neuronales convolucionales aprovechan el hecho de que la entrada consiste en imágenes y restringen la arquitectura de una manera más sensible. En particular, a diferencia de una red neuronal normal, las capas de una CNN tienen neuronas dispuestas en 3 dimensiones: ancho, alto, profundidad, con profundidad nos referimos a los canales de colores existentes, es decir, RGB tendría una profundidad de 3 y una escala de grises hace referencia a una profundidad de 1. Para este informe se hará uso de 1000 imágenes con dimensiones de 80x80x1, es decir, imágenes de 80 de alto por 80 de ancho y a escala de grises.

Base de datos

Las redes neuronales convolucionales están basadas en la idea de que las características se replican: como un objeto se mueve y aparece en diferentes píxeles, un detector de una característica que funciona en una parte de la imagen es probable que también vaya a funcionar en otra parte de la imagen.

El objetivo es construir varias copias del mismo detector de características en diferentes posiciones y para una misma posición tener muchos detectores de características, de manera que cada parte de la imagen pueda ser representada con características de diferentes tipos. Replicar a través de posición ayuda a reducir el número de parámetros a aprender (Pusiol, P. D. 2014).

Para poder entrenar a nuestra red neuronal es necesario tener un conjunto de datos tanto verdaderos como falsos (en este caso los datos son imágenes de chayotes), y dichos datos requieren una dimensionalidad de entrada constante. Por lo tanto, redujimos la muestra de las imágenes a una resolución fija de 80 × 80, esta resolución se logró con ayuda de un programa realizado en Python, esto para evitar que el entrenamiento de la CNN conllevara demasiado tiempo.

Las imágenes de los chayotes fueron tomadas con ayuda de la cámara de un celular, en total fueron tomadas 1000 imágenes, de las cuales, 500 eran de chayotes (verdaderas) y 500 donde no había chayotes (falsas).

La base de datos mencionada se encuentra en el siguiente enlace:

https://drive.google.com/file/d/1Usjlx9yRijn_2fiVLZLZnZD2FeMkeX54/view?usp=sharing

Convolución

El nombre de redes neuronales convoluciones tiene su origen en la operación que se realiza en algunas de sus capas. Se sustituye la operación de multiplicación matricial de entradas por pesos por la operación matemática de convolución.

La convolución consiste en filtrar una imagen usando una máscara, diferentes mascaras llevan a distintos resultados. En la convolución, cada píxel de salida es una combinación lineal de los pixeles de entrada por lo que las máscaras representan la conectividad entre las capas sucesivas (Loncomilla, P. 2016).

Si hablamos matemáticamente, una convolución es una operación aplicada a dos funciones con números reales como argumentos. La salida de esta operación es una tercera función que se interpreta como la modificada de una de las funciones de entrada (Aguado López, I. 2020). La primera convolución es capaz de detectar características primitivas como líneas o curvas y a medida que se van realizando más capas con las convoluciones, los mapas de características serán capaces de reconocer formas más complejas.

Es necesario tener en cuenta que la operación de convolución esencialmente realiza productos punto entre los filtros y las regiones locales de la entrada. Un patrón de implementación común de la capa convolucional es aprovechar este hecho y formular el pase directo de una capa convolucional como una matriz grande multiplicada.

Para la realización de la convolución se hace uso de dos funciones particulares, las cuales son la propagación hacia adelante y hacia atrás (forward propagation y backpropagation).

Propagación hacia adelante (Forward propagation)

El objetivo de la propagación hacia adelante es aproximar una función g. Para nuestro caso estamos asumiendo que existe una relación entre los datos que queremos clasificar y las categorías de las que disponemos, de tal manera que a cada entrada x le corresponda una categoría y. Esa relación la podemos representar por medio de una función g, tal que $g(x) = y$. Por lo que, x puede ser en este caso la matriz que representa la imagen de un chayote, mientras que y es el número asignado de la imagen (chayote = 1, otro=0).

Lo que se busca es poder clasificar adecuadamente a los chayotes, en otras palabras, que la red se comporte como g. Sin embargo, no se conoce a g completamente, por lo que, podemos considerar una muestra de información previamente rotulada (la base de datos creada), que es a lo que denominamos conjunto de entrenamiento, y entrenar la red para que aprenda a

clasificar correctamente ese conjunto, con la esperanza de que eso nos sirva para generalizar a otros datos.

Lo que se busca es aproximar g a través de f, con el conocimiento de una cantidad limitada de puntos de g. Luego a través del entrenamiento, lo que se espera es que f se comporte de manera aceptable para el resto de los puntos.

Propagación hacia atrás (backpropagation)

El objetivo del algoritmo de backpropagation es calcular la derivada parcial $\frac{\partial C}{\partial \theta}$, la cual será usada posteriormente en el algoritmo de optimización para actualizar los parámetros θ.

Para poder usar este algoritmo hay dos hipótesis que la función costo debe cumplir. La primera es que la función costo C se pueda escribir como un promedio de los costos individuales C_x, y la segunda es que el costo se pueda escribir como una función de las salidas de la red (Repetur, A. E. 2019).

La razón por la que requerimos de la primera condición es que el algoritmo de backpropagation nos permitirá calcular la derivada parcial de $\frac{\partial Cx}{\partial \theta}$ para cada ejemplo individual del conjunto de entrenamiento, mientras que en la segunda condición el algoritmo parte de la salida y propaga la información hacia atrás para obtener los gradientes.

Lo que se busca con backpropagation, es calcular los errores δ^l para cada capa l, haciendo uso de la siguiente ecuación:

$$\delta_j^{\mathrm{L}} = \frac{\partial \mathrm{C}}{\partial \mathrm{z}_j^{\mathrm{L}}} = \sum_k \frac{\partial \mathrm{C}}{\partial \mathrm{a}_k^{\mathrm{L}}} \left(\frac{\partial \mathrm{a}_k^{\mathrm{L}}}{\partial \mathrm{z}_j^{\mathrm{L}}} \right) = \sum_k \frac{\partial \mathrm{C}}{\partial \mathrm{a}_k^{\mathrm{L}}} \left(\frac{\partial \sigma(\mathrm{z}_k^{\mathrm{L}})}{\partial \mathrm{z}_j^{\mathrm{L}}} \right) = \frac{\partial \mathrm{C}}{\partial \mathrm{a}_j^{\mathrm{L}}} \sigma'(\mathrm{z}_j^{\mathrm{L}})$$

Maxpool

Como hemos comentado, partiendo de una imagen de 80x80 tenemos una primera capa de entrada de 2500 neuronas. Si se llevara a cabo una nueva convolución a partir de esta capa, el número de neuronas de la próxima capa crecería exponencialmente implicando así un mayor procesamiento. Por lo que, para reducir el tamaño de la próxima capa se realiza el proceso de subsampling en el que se reduce el tamaño de las imágenes filtradas, pero donde prevalecen las características más importantes que detectó cada filtro.

Existen diferentes tipos de subsampling, pero el que se empleará en este informe es el maxpool, el cual consiste en dividir una matriz bidimensional en sub matrices más pequeñas de igual tamaño y generar como salida a una nueva matriz formada por los valores más grandes de cada sub matriz, generalmente las dimensiones más utilizadas para el maxpooling son 2x2, reduciendo así los datos a la mitad y por lo tanto reduciendo los costes computacionales del entrenamiento. El descenso es considerable y teóricamente almacenan la información más importante para detectar las características deseadas.

El punto de finalización de la función maxpool es tomar la última capa oculta a la que se le realizó el proceso de subsampling, que se dice que es tridimensional y se procesa para que deje de serlo pasando a ser una capa de neuronas tradicionales. Entonces a esta última se le

aplica la función de activación Softmax que conecta contra la capa de salida final que tendrá la cantidad de neuronas correspondientes con las clases que estamos clasificando (Artola Moreno, Á. 2019).

Softmax

La función softmax (función exponencial normalizada) es una generalización de la función sigmoidea y popularizada por las redes neuronales convolucionales. Se utiliza como función de activación de salida para la clasificación multiclase porque escala las entradas precedentes de un rango entre 0 y 1 y normaliza la capa de salida, de modo que la suma de todas las neuronas de salida sea igual a la unidad, dicho de otro modo, su trabajo es convertir variables independientes de rango casi infinito en probabilidades simples entre 0 y 1 (Fran Ramírez. 2020).

La ecuación que define la función softmax es la siguiente:

$$a2 = softmax(z2) = \frac{e^{z_i}}{e^{z_j}}$$

Donde j va desde 1 hasta K.

Programa

Creación de la base de datos

```
# Universidad Autónoma Chapingo
# Departamento de Ingeniería Mecánica Agrícola
# Ingeniería Mecatrónica Agrícola
# Jym Emmanuel Cocotle Lara
# 7° 7
# ------Proyecto Final------
# ------Base de datos-------

# Librerías utilizadas
import numpy as np
from PIL import Image
import os

# Creamos arreglos para almacenar las imagenes y las etiquetas
categorias = []
# Ubicación de las imágenes
categorias = os.listdir('D:/Desktop/Data/Data2')
tam_img = 80


def class_img(categorias,tam_img1,tam_img2):
    # Contadores
    x = 0
    y = 0
    labels = []
    imagenes = []
    for direccion in categorias:
```

```
        # Lee todas las imágenes de la carpetas en la ubicación
        for imagen in os.listdir('D:/Desktop/Data/Data2/'+ direccion):
            # Redimensionamiento de la imágen
            img2 = Image.open('D:/Desktop/Data/Data2/' + direccion + '/'
+ imagen).resize((tam_img,tam_img))
            # Convertimos en arreglo a la imagen redimensionada
            img2 = np.array(img2)
            imagenes.append(img2)
            # Agregamos el identificador a la imagen
            labels.append(x)
            if y == 500:
                break
            y += 1
        y = 0
        if x == 1:
            break
        x += 1

    # Convertimos en arreglo las etiquetas
    labels = np.array(labels)
    imagenes = np.array(imagenes)
    # Extraemos los valores de cada píxel (0-255)
    imagenes = imagenes[:,:,:,0]
    archivo = open("D:/Desktop/Data/Chayotes.csv","w")
    total_img = np.size(imagenes[:,0,0])

    for j in range(total_img):
        # Colocación de identificador
        archivo.write(str(labels[j]))
        archivo.write(",")
        for k in range(tam_img): #
            for l in range(tam_img):
                # Convertimos a escala 0 - 1
                pixels = imagenes[j,k,l]
                archivo.write(str(pixels))
                if k < (tam_img-1) or l < (tam_img-1):
                    # Separación de cada valor
                    archivo.write(",")
        archivo.write("\n")

    print("Archivo ordenado")
    archivo.close()

class_img(categorias, tam_img,tam_img)
```

Red Neuronal Covolucional

```
# Universidad Autónoma Chapingo
# Departamento de Ingeniería Mecánica Agrícola
# Ingeniería Mecatrónica Agrícola
# Jym Emmanuel Cocotle Lara
# 7° 7
```

```
# ------Proyecto Final------
# ------CNN-----------------

# Librerías utilizadas
import pandas as pd
import numpy as np
import cv2 as cv

# Creamos una clase para una capa de convolución con filtros de 3x3
class Convolucion_3x3:
    def __init__(self, num_filtros):
        self.num_filtros = num_filtros
        # Dividimos por 9 para reducir la varianza de nuestros valores
iniciales
        self.filters = np.random.randn(num_filtros, 3, 3) / 9

    # Generamos todas las regiones de imagen de 3x3 posibles.
    # La imagen es una matriz numérica de dos dimensiones.
    def itera_reg(self, image):
        h, w = image.shape

        for i in range(h - 2):
          for j in range(w - 2):
            im_region = image[i:(i + 3), j:(j + 3)]
            yield im_region, i, j

    # Realizamos un paso hacia adelante de la capa de convolución usando
los
    # datos de entrada.
    # Donde la entrada es una matriz numérica de 2 dimensiones
    def forward_prop(self, input):
        self.last_input = input

        h, w = input.shape
        output = np.zeros((h - 2, w - 2, self.num_filtros))

        for im_region, i, j in self.itera_reg(input):
          output[i, j] = np.sum(im_region * self.filters, axis=(1, 2))

        # Se devuelve una matriz numérica de 3 dimensiones (h, w,
num_filtros).
        return output

    # Se realiza una propagación hacia atrás de la capa de convolución
    # Donde d_L_d_out es el gradiente de pérdida para las salidas de esta
capa.
    # y alfa es un flotante.
    def backprop(self, d_L_d_out, alfa):
        d_L_d_filtros = np.zeros(self.filters.shape)

        for im_region, i, j in self.itera_reg(self.last_input):
          for f in range(self.num_filtros):
            d_L_d_filtros[f] += d_L_d_out[i, j, f] * im_region
```

```
        # Se actualizan los filtros
        self.filters -= alfa * d_L_d_filtros

        # No se devuleve ningún valor ya que usamos una convolución de
3x3
        # como la primera capa en nuestra CNN.
        return None

# Creamos una clase denominada Maxpool 2, donde 2 hace referencia a 2x2
que
# son las dimensiones que se tomaran para esta acción.
# Es decir, recorreremos cada una de nuestras 1000 imágenes de
características
# obtenidas anteriormente de 80x80 px de izquierda-derecha, arriba-abajo
# pero en vez de tomar de a 1 pixel, tomaremos de 2 en 2
# e iremos preservando el valor más alto
class MaxPool_2:

    def itera_reg(self, image):
        h, w, _ = image.shape
        new_h = h // 2
        new_w = w // 2

        for i in range(new_h):
          for j in range(new_w):
            im_region = image[(i * 2):(i * 2 + 2), (j * 2):(j * 2 + 2)]
            yield im_region, i, j


    # Realizamos un pase hacia adelante de la capa maxpool usando los
datos de entrada.
    def forward_prop(self, input):
        # La entrada es una matriz de 3 dimensiones (h, w, número de
filtros)
        self.last_input = input

        h, w, num_filtros = input.shape
        output = np.zeros((h // 2, w // 2, num_filtros))

        for im_region, i, j in self.itera_reg(input):
            output[i, j] = np.amax(im_region, axis=(0, 1))

        # Retornamos una matriz de 3 dimensiones (h/2,w/2,número de
filtros)
        return output

    # Realizamos un paso hacia atras de la capa maxpool.
    def backprop(self, d_L_d_out):
        d_L_d_input = np.zeros(self.last_input.shape)

        for im_region, i, j in self.itera_reg(self.last_input):
            h, w, f = im_region.shape
```

```
            amax = np.amax(im_region, axis=(0, 1))

        for i2 in range(h):
            for j2 in range(w):
                for f2 in range(f):
                    # Si este píxel era el valor máximo. copiamos su
gradiente
                    if im_region[i2, j2, f2] == amax[f2]:
                        d_L_d_input[i * 2 + i2, j * 2 + j2, f2] =
d_L_d_out[i, j, f2]
        return d_L_d_input

# Creamos la clase Softmax
class Softmax:

    def __init__(self, input_len, nodos):
        self.weights = np.random.randn(input_len, nodos) / input_len
        self.biases = np.zeros(nodos)

    def forward_prop(self, input):

        self.last_input_shape = input.shape
        input = input.flatten()
        self.last_input = input
        input_len, nodos = self.weights.shape
        totals = np.dot(input, self.weights) + self.biases
        self.last_totals = totals

        exp = np.exp(totals)
        # Retornamos una matriz de una dimensión que contiene los valores
de
        # probabilidad respectivos.
        return exp / np.sum(exp, axis=0)

    def backprop(self, d_L_d_out, alfa):

        for i, gradient in enumerate(d_L_d_out):
            if gradient == 0:
                continue

            # Valores exponenciales
            t_exp = np.exp(self.last_totals)

            # Suma de todos los valores exponenciales
            S = np.sum(t_exp)

            # Gradiente de cada valor de salida out[i] contra los totales
            d_out_d_t = -t_exp[i] * t_exp / (S ** 2)
            d_out_d_t[i] = t_exp[i] * (S - t_exp[i]) / (S ** 2)

            # Gradiente de los valores totales contra ponderaciones,
sesgos y entradas
            d_t_d_w = self.last_input
```

```
            d_t_d_b = 1
            d_t_d_inputs = self.weights

            # Gradiente de las pérdidas contra los valores totales
            d_L_d_t = gradient * d_out_d_t

            # Gradiente de las pérdidas contra ponderaciones, sesgos y
entradas
            d_L_d_w = d_t_d_w[np.newaxis].T @ d_L_d_t[np.newaxis]
            d_L_d_b = d_L_d_t * d_t_d_b
            d_L_d_inputs = d_t_d_inputs @ d_L_d_t

            # Actualización de las ponderaciones y sesgos
            self.weights -= alfa * d_L_d_w
            self.biases -= alfa * d_L_d_b

            # Devolvemos el gradiente de pérdida para las entradas de
esta capa
            return d_L_d_inputs.reshape(self.last_input_shape)

# Número total de imágenes (recordando que es un array y la posición "0"
cuenta)
total_img = 999
# Resolución de las imágenes utilizadas
resol = 80
# Ubicación del archivo
data = pd.read_csv('D:/Desktop/Data/Chayotes.csv')
data2 = np.array(data)
m, n = data2.shape

data3 = data.drop(data.columns[[0]], axis='columns')
data4 = np.array(data3)

train_images1 = data4.reshape(total_img,resol,resol)
train_images = np.uint8(train_images1)

data_train = data2[0:m].T
train_labels1 = data_train[0]
train_labels = np.uint8(train_labels1)

conv = Convolucion_3x3(8) # 8 filtros
pool = MaxPool_2()
softmax = Softmax(39*39*8, 5)


# Creamos una función forward propagation (propagación hacia adelante)
# Completa una propagación hacia adelante de la CNN y calculamos la
precisión y
# pérdida de entropía cruzada
def forward_prop(image, label):
    # Transformamos la imagen de [0, 255] a [-0,5, 0,5] para que sea más
fácil
    # trabajar con ella.
```

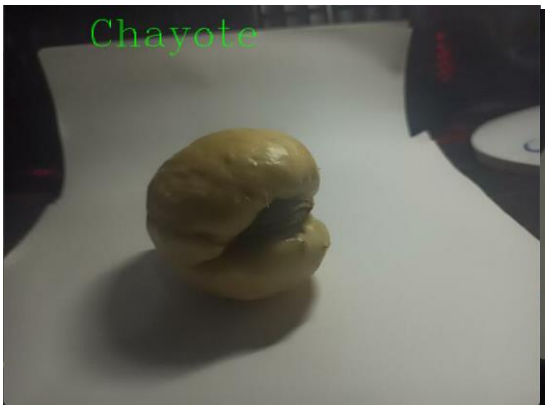
```
    out = conv.forward_prop((image / 255) - 0.5)
    out = pool.forward_prop(out)
    out = softmax.forward_prop(out)

    # Calculamos la precisión (acc) y la pérdida (loss)
    loss = -np.log(out[label])
    acc = 1 if np.argmax(out) == label else 0

    return out, loss, acc

# Creamos una función para realizar un paso de entrenamiento donde:
# im = la imagen expresada como una matriz de 2 dimensiones
# label = el número que identifica la imagen (1 = chayote, 0 = no hay
chayote)
# alfa = la tasa de aprendizaje (0.005)
def train(im, label, alfa=.005):
    # propagación hacia adelante
    out, loss, acc = forward_prop(im, label)

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

    # Backpropagation (propagación hacia atras)
    gradient = softmax.backprop(gradient, alfa)
    gradient = pool.backprop(gradient)
    gradient = conv.backprop(gradient, alfa)

    # Retornamos los valores de pérdida y precisión
    return loss, acc

print('--- Iniciando la CNN ---')

# Entrenamiento de la CNN con 3 épocas
for epoca in range(3):
    # Mostramos la época en la que nos encontramos
    print('--- Epoca %d ---' % (epoca + 1))

    # Mezclamos los datos de entrenamiento
    permutation = np.random.permutation(len(train_images))
    train_images = train_images[permutation]
    train_labels = train_labels[permutation]

    loss = 0.0
    num_correct = 0

    for i, (im, label) in enumerate(zip(train_images, train_labels)):
        if i % 100 == 0:
            print(
                '[Paso %d] 100 pasos dados| pérdida %.3f | Exactitud:
%d%%' %
                (i + 1, loss / 100, num_correct)
                )
```

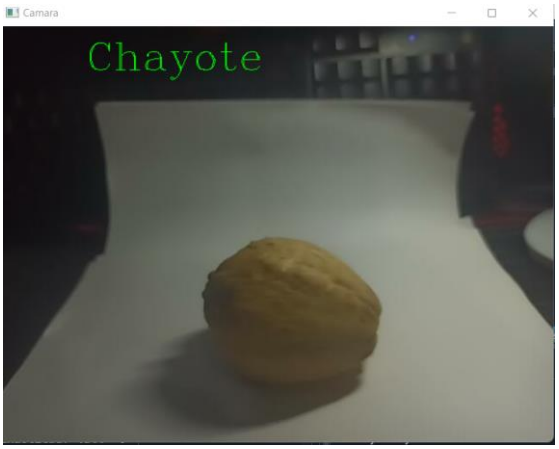
```
            loss = 0
            num_correct = 0

        l, acc = train(im, label)
        loss += l
        num_correct += acc

# Inicializamos el video
video=cv.VideoCapture(1)

if not video.isOpened():
    print("No se puede abrir la camara")
    exit()

# Bucle de la cámara
while True:
    # Captura trama a trama
    ret,frame = video.read()
    # Escala de grises
    escala_grises = cv.cvtColor(frame,cv.COLOR_BGR2GRAY)

    if not ret:
        print("No se puede recibir el frame, el video ha terminado")
        break

    fotog = np.array(cv.resize(escala_grises, (80,80))).reshape(-1,80,80)
    test_images = fotog
    test_labels = np.array([0])

    loss = 0
    num_correct = 0
    n = 0

    for i,(im, label) in enumerate(zip(test_images, test_labels)):
        _, l, acc = forward_prop(im, label)
        loss += l
        num_correct += acc

        if acc == 1:
            print("Si hay Chayote")
            objt ="Chayote"
            x_imagen = 100
            color = (0,255,0)
        else:
            print("No hay Chayote")
            objt = "No hay"
            x_imagen = 50
            color = (0,0,255)

    n+=1
    teste = len(test_images)
    escala_grises =
cv.putText(frame,objt,(x_imagen,50),cv.FONT_HERSHEY_COMPLEX, 1.5,color,1)
```

```
cv.imshow("Camara",escala_grises)

if cv.waitKey(1)==ord('q'):
    break

print('pérdida: ',loss/teste)
print('Prediccion: ',num_correct)
```

Resultados

El entrenamiento finalizó después de 3 épocas (aproximadamente 20 minutos) con lo cual se obtuvo los siguientes resultados:





Conclusión

Una vez que se entrenó a la CNN con las 1000 imágenes se pudo hacer el reconocimiento de objetos (en este caso chayotes), sin embargo, dicho reconocimiento no es del todo eficiente ya que en algunas ocasiones el programa detectaba chayotes donde no los había, esta precisión puede ser aumentada si aumentamos el número de épocas, sin embargo, se tiene que tener en cuenta que si se realizan muchas épocas el tiempo para que la CNN sea entrenada

aumentara en gran medida, y además, podría existir el problema que la red neuronal solo pueda identificar las imágenes con las que se entrenó, por lo que no tendríamos un reconocimiento general si no uno más especializado.

Debido a que el entrenamiento se realizó a partir de una base de datos de un archivo de Excel, el tiempo entre épocas disminuyó considerablemente a comparación de proyectos anteriores en los cuales se ingresaban las imágenes sin ser procesadas.

Otra opción para mejorar la eficacia de nuestra red neuronal convolucional es obtener mejores fotografías de chayotes, con una mejor iluminación para que los contornos se puedan diferenciar más fácilmente.

Bibliografía

Rouhiainen, L. (2018). Inteligencia artificial. Madrid: Alienta Editorial.

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 de https://www.codigofuente.org/redes-neuronalesprofundas-tipos-caracteristicas/

Fran Ramírez. (2020). Las matemáticas del Machine Learning: Funciones de activación. Consultado el 15 diciembre 2021, de https://empresas.blogthinkbig.com/las-matematicas-del-machine-learning-funciones-de-activacion/

Pusiol, P. D. (2014). Redes convolucionales en comprensión de escenas (Bachelor's thesis)

Loncomilla, P. (2016). Deep learning: Redes convolucionales. Consultado de https://ccc.inaoep.mx/pgomez/deep/presentations

Aguado López, I. (2020). Deep Learning: Redes Neuronales Convolucionales en R.

Artola Moreno, Á. (2019). Clasificación de imágenes usando redes neuronales convolucionales en Python.

Repetur, A. E. (2019). Redes Neuronales Artificiales (Doctoral dissertation, Universidad Nacional del Centro de la Provincia de Buenos Aires).