RRC (Red Rose Condition – Condición de Rosa Roja) (Febrero de 2022)

Johann S. Kwan A., estudiante de la Universidad Surcolombiana

Abstract

Context: It is said that a person can identify if a red rose is withered or not, but there have been cases that, due to mistrust of the customer to the seller, they come to ask "Isn't it damaged?", with what refers to if the red rose is withered or no longer in good condition.

Methods: There have been works that it is possible to measure the quality of roses and other projects have managed to design efficient artificial vision models. However, the works are oriented to the agricultural environment, given this, a model is needed that clarifies the characteristics to a client..

Results: This project categorizes whether a red rose is in “aceptable” condition, “marchita desde hace 1 a 4 días” or “marchita desde hace mucho”, being “mucho” more than 4 days, with an average accuracy percentage of 81.23%..

Conclusion: The results show a good precision, according to the rubric with which this project is evaluated, with this managing to satisfy the initial problem.

Keywords: red rose condition categorization, red rose condition classification model, detect withered red rose.

Resumen.

Contexto: Se dice que una persona puede identificar si una rosa roja esta marchita o no, pero se han dado casos que, por desconfianza del cliente al vendedor, se llega a preguntar “¿No esta dañada?”, con lo que se refiere a que si la rosa roja se encuentra marchita o ya no se ve en buenas condiciones.

Método: Han existido trabajos que es posible medir la calidad de las rosas y otros proyectos han logrado diseñar modelos visión artificial eficientes. No obstante, los trabajos son orientados al entorno agrícola, dado esto se necesita de un modelo que aclare las características a un cliente.

Resultados: Este proyecto categoriza si una rosa roja está en condición “aceptable”, “marchita desde hace 1 a 4 días” o “marchita desde hace mucho”, siendo “mucho” más de 4 días, con un porcentaje de precisión promedio del 81.23%.

Conclusión: Los resultados muestran una precisión buena, según la rúbrica con la cual se evalúa este proyecto, con esto logrando satisfacer la problemática inicial.

Palabras clave: categorización de la condición de una rosa roja, modelo de clasificación del estado de una rosa roja, detectar rosa roja marchita.

Estudiante de Ingeniería de Software de la Universidad Surcolombiana. Neiva, Colombia.
Contacto: u20191179607@usco.edu.co

I. INTRODUCCIÓN

Este documento muestra cómo se llevó a cabo el proyecto “RRC” para la materia de “Data Science”, que se necesitó, que se usó, como funciona, cuál es su propósito y como cumplió con esto.

II. PLANTEAMIENTO DEL PROBLEMA

Es común que un cliente al comprar rosas pregunte: "¿Están en buen estado?". Y dependiendo la situación del vendedor puede ser difícil contestar. Para responder de manera correcta a esta pregunta se puede desarrollar un programa o aplicación que permita saber cuántos días aproximadamente le quedan a la rosa roja, sin agua o con agua, para que se deje de ver presentable.

Hay entre 5 a 500 floristerías por cada municipio en Colombia aproximadamente y en todas se venden rosas rojas, e incluso se venden en más tipos de tiendas, es normal que los clientes quieran saber si las rosas rojas están en buen estado o si ya están marchitas. El problema de esto es que causa confusión y desconfianza al vendedor, en caso de que las rosas duren menos de lo que se creía.

¿Como se puede implementar un modelo de visión artificial a este problema?, para solucionar el problema es posible crear una aplicación que pueda decir si está en una buena condición o marchita una rosa roja, según la imagen que se capte de la rosa roja. Dado que mi madre es dueña de una floristería, conozco de la realidad de la problemática, podre tomar bastantes imágenes de guía algunas sin costo y otras con un mínimo costo, que servirán para alimentar la inteligencia de la aplicación y además al tener los resultados se podrá decidir si es de utilidad o no.

Objetivos específicos

- Hacer que la aplicación logre identificar el tiempo de vida presentable de una rosa roja.
- Ser sencilla de usar.
- Ser visiblemente agradable.

III. ESTADO DEL ARTE

- Diseño De Un Sistema De Visión Artificial Para El Análisis De Calidad Y Producción De Rosas: En este proyecto se busca evaluar la calidad de las rosas, me

sirvió de ayuda para mi proyecto dado que dentro de este se dan a tratar ciertos aspectos similares al mío, estos aspectos son: el uso de visión artificial, el análisis de una rosa, la identificación de partes no presentables de una rosa y la metodología usada por este proyecto me es de ayuda. [3]
Metodología: La metodología usada para este proyecto consistió en cinco partes estas fueron:
  - Documentación e indagación sobre las técnicas de visión artificial aplicadas al sector agrícola.
  - Evaluación de la viabilidad de la implementación en Matlab.
  - Diseño del sistema.
  - Desarrollo de los diferentes algoritmos.
  - Pruebas y corrección de errores.

  Porcentaje de precisión promedio: 67.1% [3]
- Diseño de un sistema de identificación y clasificación de flores: Este proyecto busca identificar distintos parámetros de las rosas, para poder clasificar el grado de calidad de las rosas. En este proyecto se tratan elementos similares al mío, cosas como la identificación de aspectos de una rosa con visión artificial, calificación de calidad de una rosa, siendo esto similar ya que se suele decir que una rosa marchita o una rosa con señales de estar próxima a marchitarse es un producto de baja calidad.[4][17][18]
  Metodología: En este proyecto propone el siguiente paso a paso para identificar y clasificar las rosas:
  - Adquisición y digitalización de la imagen.
  - Procesamiento.
  - Descripción.
  - Clasificación.
  - Toma de decisiones. [4]

  Porcentaje de precisión máximo alcanzado: 93%.
- Sistema de clasificación de rosas de la variedad explorer, usando visión por computadora: En el proyecto de sistema de clasificación se rosas según sus características, una de las características que se clasifica es la coloración, la cual lleve de forma correcta, por lo que fue útil para mi proyecto, dado que al cambiar de coloración las rosas pueden identificarse como marchitas o con signos de estar próximas a marchitarse. [5]
- Clasificación de imágenes (perros y gatos): Este código representa el sitio web, una vez que se crea y entrena el modelo de inteligencia artificial con Python, Tensorflow y demás utilidades, dado esto y que se utiliza un modelo matemático similar al que yo use, me fue útil, para basarme en este código. Porcentaje de precisión promedio: 67.1%. [2] [6]
- API_DeepLearning: En este código se implementa una red neuronal usando keras-tensorflow y se ejecuta en un servicio web de flask. [1]

.

IV. Marco teórico/conceptual

¿Qué es RRC?, Una aplicación que permite identificar aproximadamente cuanto tiempo de vida presentable le queda a una rosa, definiendo que una rosa es presentable cuando no se ve marchita por completo o por partes, precisando también que una rosa roja se empieza a ver marchita cuando ciertas partes de los pétalos se tornan de un tono café.

¿Cómo funcionaria RRC?, La aplicación necesita que se importe la imagen de una rosa roja de la cual se desea saber el tiempo de vida presentable, la imagen debe tener las mismas especificaciones que se usaron para tomar las fotos del dataset, después de importada la imagen, será analizada y dirá, por medio de texto, cuantos días de tiempo presentable le quedan a la rosa roja.

¿Dónde se aplica RRC?, sirve en el caso que una persona o personas deseen saber la condición de una rosa.

¿Qué significa que una rosa roja este marchita?, se sabe que una rosa roja o una planta común esta marchita cuando sus hojas o pétalos están quebradizos, si el tallo cambio de color verde a color negro, si el tallo perdió la flexibilidad y que la cabeza de la rosa, también llamada flor, este decaída.

¿Qué significa que una rosa este en buena condición o aceptable?, una rosa roja es aceptable o está en buenas condiciones cuando sus pétalos, hojas y tallo son flexibles, otro aspecto es que el tallo sea de color verde y que la flor o cabeza de la rosa este levantada. [9] [10] [11]

V. Modelo matemático

Se analizo la problemática si se opto por un sistema de clasificación, que dividiría la información en tres categorías, “aceptable”, “Marchita desde hace 1 a 4 días” y “Marchita hace mucho”, la primera categoría indicando que la rosa roja se encuentra en buena condición, la segunda indica que la rosa se empezó a marchitar desde hace uno a tres días y la última indica que la rosa se marchito desde hace más de 4 días y que además no ha sido proveída de agua durante un lapso de tiempo considerable.
El modelo matemático escogido fue la red neuronal convolucional, con tres neuronas de salida, siendo estas las tres categorías en las que se realizó la clasificación.
Dado que las imágenes se les ajusta su resolución a 224x224pixeles y con canales de color rgb, dio como resultado a 150528 las neuras de entrada. Se hizo uso de un modelo preentrenado llamado “MobileNet” en su versión “v2”, por lo que al representar los procesos que realiza el modelo se le denominan “Capas Ocultas”. [7] [12] [13] [19] [20] [21] [23]

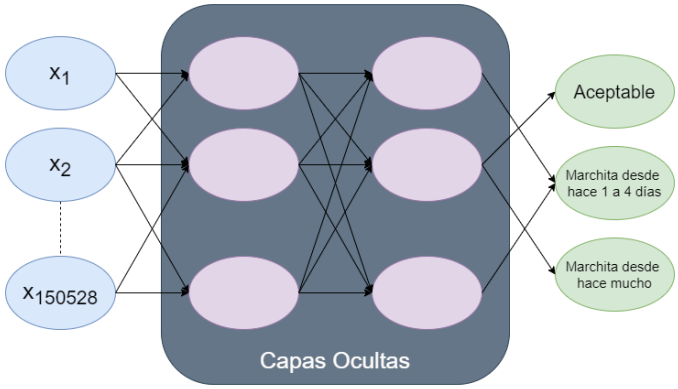


Fig. 1. Modelo matemático, representado gráficamente.

VI. Construcción del Dataset

Se uso la aplicación "Camara FV-5" para android, procurando que la cámara estuviese en la misma posición para todas las fotos, las fotos se tomaron entre las 10:30p.m. y 11:10p.m., con la iluminación dada por un aro de luz.
Se tomaron 3465 fotos a 10 rosas desde distintos ángulos, durante 6 días a 5 rosas que iniciaron en buenas condiciones y en el último día a 5 rosas que llevaban mucho tiempo marchitas y deshidratadas. Al tomar cada foto se rotaba la rosa roja, se acercaba y alejaba, con un fondo blanco y dos hojas de papen que hicieron como guía para la ubicación de las rosas.

Fig. 2. Ejemplo de construcción del dataset.

TABLA I
Distribución del Dataset

DISTRIBUCIÓN DEL DATASET			
Entrenamiento	**80%**	**Validación**	**20%**
2772		**693**	
3465			

VII. Creación del Modelo de Predicción

Para implementar los conocimientos del planteamiento del problema, marco teórico, modelo matemático y poder desarrollar un código capaz de clasificar el dataset construido, se hizo uso de la herramienta "Google Colaboratory", también denominado "Colab", el cual corre en la nube, fuera del sistema local la computadora personal que se usó, este corre con el lenguaje de programación Python.
El paso a paso para la implementación fue:

1. Se guarda y organiza el dataset dentro del servicio de "Google Drive".

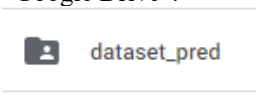


Fig. 3. Paso 1, código colab.

2. Se crea un nuevo código en "Colab", llamado "RRC".

Fig. 4. Paso 2, código colab.

3. Codificación dentro de "Colab":
   3.1. Se ejecuta un comando para que "Colab" pueda conectar a "Google Drive".

```
from google.colab import drive
drive.mount('/content/drive')
```
```
Mounted at /content/drive
```

Fig. 5. Paso 3.1, código colab.

   3.2. Se crea la carpeta "dataset" dentro de la carpeta principal.

```
!mkdir dataset
```

Fig. 6. Paso 3.2, código colab.

   3.3. Se crean las carpetas "aceptable", "marchita_dia-1-4" y "marchita_hace_mucho".

```
!mkdir dataset/aceptable
!mkdir dataset/marchita_dia_1-4
!mkdir dataset/marchita_hace_mucho
```

Fig. 7. Paso 3.3.1, código colab.

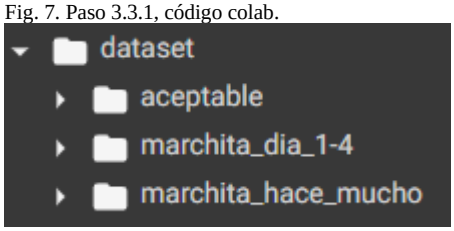


Fig. 8. Paso 3.3.2, código colab.

   3.4. Se importan las librerías "os" y "shutil".

```
import os
import shutil
```

Fig. 9. Paso 3.4, código colab.

   3.5. Ahora utilizando cierto código se pasa la información de las carpetas de "Google Drive" a las carpetas creadas dentro de "Colab".

```
#Copiar imagenes que subimos a carpetas del dataset
carpeta_fuente = '/content/drive/MyDrive/datasetmater/dataset_pred/falta_1_dia'
carpeta_destino = '/content/dataset/aceptable'

imagenes = os.listdir(carpeta_fuente)
for i, nombreimg in enumerate(imagenes):
   #Copia de la carpeta fuente a la destino
   shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino + '/' + nombreimg)
```

Fig. 10. Paso 3.5, código colab.

3.6. Solo para confirmar el paso de las imágenes, se ejecuta cierto código, el cual cuenta el número de archivos dentro de las carpetas creadas anteriormente en “Colab”.

```
!ls /content/dataset/aceptable | wc -l
```

```
1485
```

```
!ls /content/dataset/marchita_dia_1-4 | wc -l
```

```
1485
```

```
!ls /content/dataset/marchita_hace_mucho | wc -l
```

```
495
```

Fig. 11. Paso 3.6, código colab.

3.7. Se importa “tensorflow.keras”, “numpy” y “matplotlib”. [8]

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
```

Fig. 12. Paso 3.7, código colab.

3.8. Se crea el generador del dataset, para permitirle usar las imágenes a color y asignar “20%” del dataset para la validación.

```
datagen = ImageDataGenerator(
    rescale=1. / 255,
    validation_split=0.2 #20% para pruebas
)
```

Fig. 13. Paso 3.8.1, código colab.

Se generan los sets de entrenamiento y validación, también llamado pruebas y se reescalan las imágenes a una resolución de 224x224pixeles. [16]

```
data_gen_entrenamiento = datagen.flow_from_directory('/content/dataset', target_size=(224,224),batch_size=32, shuffle=True, subset='training')
data_gen_pruebas = datagen.flow_from_directory('/content/dataset', target_size=(224,224),batch_size=32, shuffle=True, subset='validation')
```

Fig. 14. Paso 3.8.2, código colab.

Para prueba que las imágenes se cargaron, se reescalaron y que solo quedaron 3 categorías, se muestran 10 imágenes aleatorias de las que se seleccionaron para entrenar.

```
for imagen, etiqueta in data_gen_entrenamiento:
  for i in range(10):
    plt.subplot(2,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(imagen[i])
  break
plt.show()
```

```
Found 2772 images belonging to 3 classes.
Found 693 images belonging to 3 classes.
```

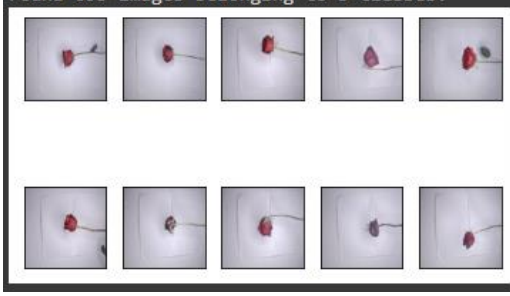


Fig. 15. Paso 3.8.3, código colab.

3.9. Se importa “tensorflow”, “tensorflow_hub” y se importa el modelo preentrenado “mobilenetv2”.[7][12]

```
import tensorflow as tf
import tensorflow_hub as hub

url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
mobilenetv2 = hub.KerasLayer(url, input_shape=(224,224,3))
```

Fig. 16. Paso 3.9, código colab.

3.10. Se congela el modelo preentrenado para que los entrenamientos nuevos sean sumados al nuevo modelo usado.

```
mobilenetv2.trainable = False
```

Fig. 17. Paso 3.10, código colab.

3.11. Se define “modelo”, se le asigna que sea para 3 categorías o clases. [8]

```
modelo = tf.keras.Sequential([
    mobilenetv2,
    tf.keras.layers.Dense(3, activation='softmax')
])
```

Fig. 18. Paso 3.11, código colab.

3.12. Se ejecuta “modelo.sumary()”, para saber cuántos parámetros se usaran para el entrenamiento.

```
modelo.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              2257984

 dense (Dense)               (None, 3)                 3843

=================================================================
Total params: 2,261,827
Trainable params: 3,843
Non-trainable params: 2,257,984
_________________________________________________________________
```

Fig. 19. Paso 3.12, código colab.

3.13. Se compila, de modo que de optimizador use “Adam” y se ajusta la perdida de datos para que use “categorical_crossentropy”. [14]

```
modelo.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Fig. 20. Paso 3.13, código colab.

3.14. A la hora de entrenar el modelo se realizaron 5 secuencias de epochs o épocas, en el siguiente orden:

- 5 epochs.

Fig. 21. Paso 3.14, 5 épocas, código colab.

- 10 epochs.

Fig. 22. Paso 3.14, 10 épocas, código colab.

- 20 epochs.

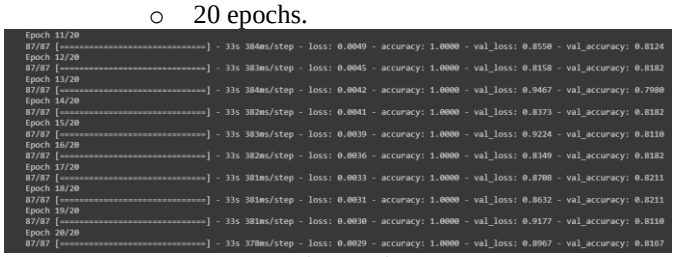


Fig. 23. Paso 3.14, 20 épocas, código colab.

- 40 epochs.

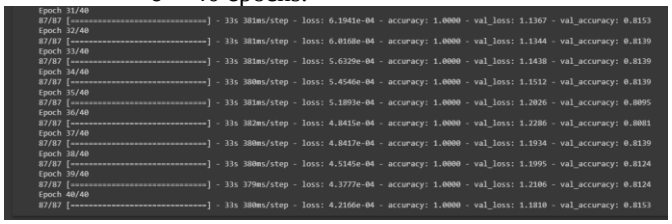


Fig. 24. Paso 3.14, 40 épocas, código colab.

- 60 epochs.

Fig. 25. Paso 3.14, 60 épocas, código colab.

Dando como resultado que la precisión del entrenamiento sea **100%** y la precisión de la validación o pruebas sea en promedio **81.23%**.

3.15. Para observar la precisión del modelo, se realizó una grafica por cada secuencia de épocas.

- 5 epochs.

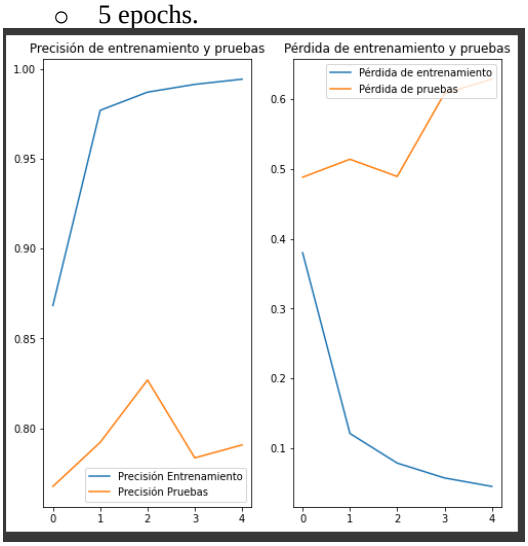


Fig. 26. Paso 3.15, graficas 5 épocas, código colab.

- 10 epochs.

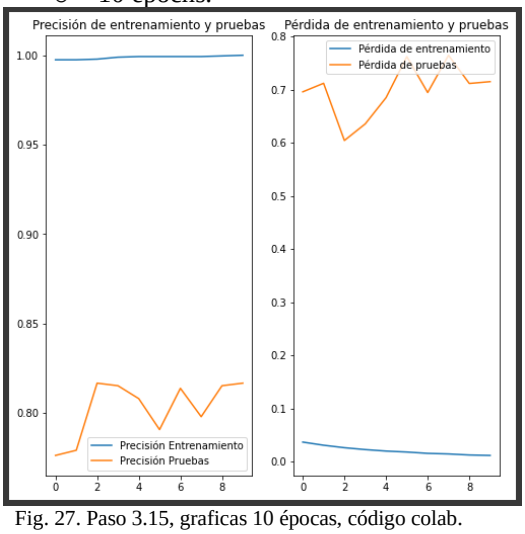


Fig. 27. Paso 3.15, graficas 10 épocas, código colab.

- 20 epochs.

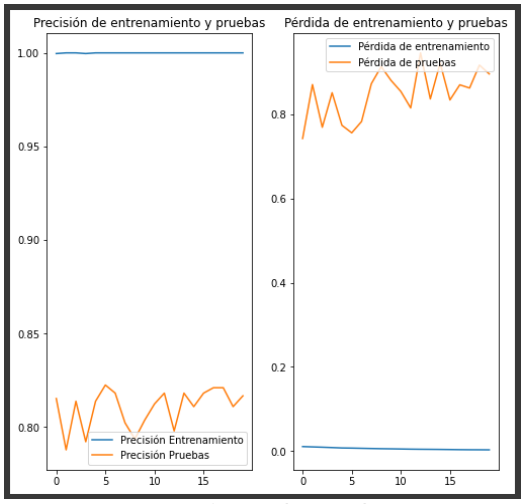


Fig. 28. Paso 3.15, graficas 20 épocas, código colab.

- 40 epochs.

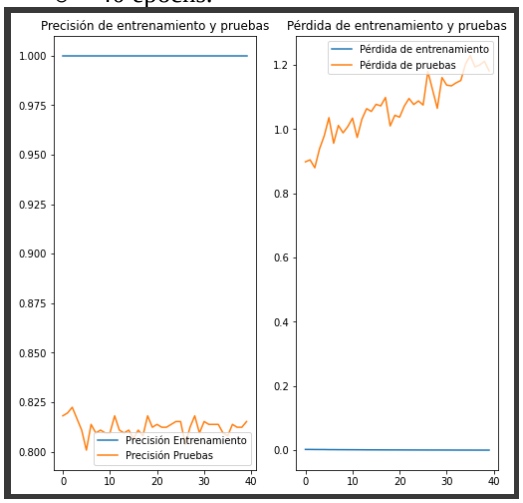


Fig. 29. Paso 3.15, graficas 40 épocas, código colab.

- 60 epochs.

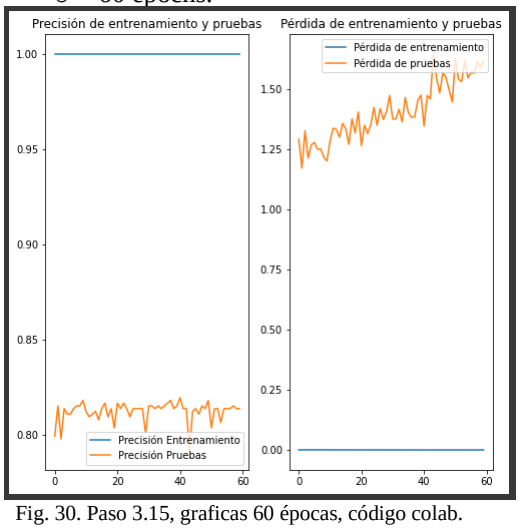


Fig. 30. Paso 3.15, graficas 60 épocas, código colab.

3.16. Se guarda el modelo con formato ".h5". [15] [24]

```
from keras.models import load_model
modelo.save('model_Rosas.h5')
```

Fig. 31. Paso 3.16, código colab.

3.17. Para poder probar el modelo de manera rápida, se puede cargar una imagen desde un url.

```
#Categorizar una imagen de internet
from PIL import Image
import requests
from io import BytesIO
import cv2

def categorizar(url):
  respuesta = requests.get(url)
  img = Image.open(BytesIO(respuesta.content))
  img = np.array(img).astype(float)/255

  img = cv2.resize(img, (224,224))
  prediccion = modelo.predict(img.reshape(-1, 224, 224, 3))
  return np.argmax(prediccion[0], axis=-1)
```

```
#0 = aceptable, 1 = marchita_dia_1-4, 2 = marchita_hace_mucho
url = ''

prediccion = categorizar (url)
print(prediccion)

if prediccion == 0:
  print("aceptable")
elif prediccion == 1:
  print("marchita entre 1 a 4 dias")
else:
  print("marchita hace mucho")
```

Fig. 32. Paso 3.17, código colab.

VIII. SIMULACIÓN DE RESULTADOS DE PREDICCIÓN

Para probar el funcionamiento del modelo, se realizaron 6 pruebas con distintas imágenes, estos fueron los resultados:

1.

Fig. 33. Simulación desde resultados, ImagenPrueba_1.jpg.

```
#Simulacion 1
#0 = aceptable, 1 = marchita_dia_1-4, 2 = marchita_hace_mucho

url = 'https://i.ibb.co/b1tZjtN/Imagen-Prueba-1.jpg'
prediccion = categorizar (url)
print(prediccion)

if prediccion == 0:
  print("aceptable")
elif prediccion == 1:
  print("marchita entre 1 a 4 dias")
else:
  print("marchita hace mucho")
```

```
0
aceptable
```

Fig. 34. Simulación desde resultados, simulación 1. [19]

2.

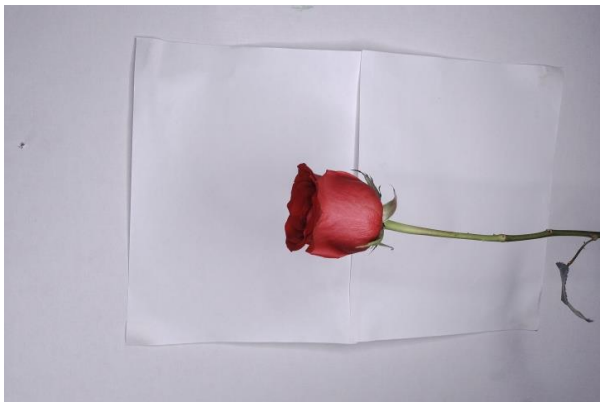


Fig. 35. Simulación desde resultados, ImagenPrueba_2.jpg.

```
#Simulacion 2
#0 = aceptable, 1 = marchita_dia_1-4, 2 = marchita_hace_mucho

url = 'https://i.ibb.co/YXMjBVB/Imagen-Prueba-2.jpg'
prediccion = categorizar (url)
print(prediccion)

if prediccion == 0:
  print("aceptable")
elif prediccion == 1:
  print("marchita entre 1 a 4 dias")
else:
  print("marchita hace mucho")
```

```
0
aceptable
```

Fig. 36. Simulación desde resultados, simulación 2. [19]

3.

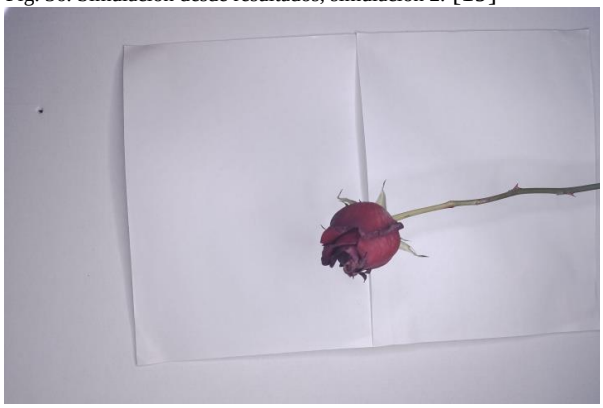


Fig. 37. Simulación desde resultados, ImagenPrueba_3.jpg.

```
#Simulacion 3
#0 = aceptable, 1 = marchita_dia_1-4, 2 = marchita_hace_mucho

url = 'https://i.ibb.co/tLQCJyQ/Imagen-Prueba-3.jpg'
prediccion = categorizar (url)
print(prediccion)

if prediccion == 0:
  print("aceptable")
elif prediccion == 1:
  print("marchita entre 1 a 4 dias")
else:
  print("marchita hace mucho")
```

```
1
marchita entre 1 a 4 dias
```

Fig. 38. Simulación desde resultados, simulación 3. [19]

4.

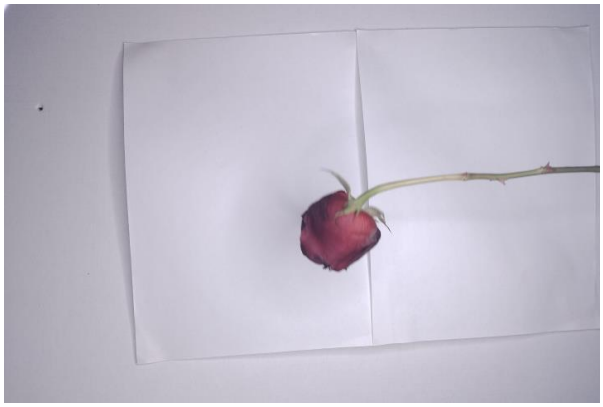


Fig. 39. Simulación desde resultados, ImagenPrueba_4.jpg.

```
#Simulacion 4
#0 = aceptable, 1 = marchita_dia_1-4, 2 = marchita_hace_mucho

url = 'https://i.ibb.co/kXz7gYW/Imagen-Prueba-4.jpg'
prediccion = categorizar (url)
print(prediccion)

if prediccion == 0:
  print("aceptable")
elif prediccion == 1:
  print("marchita entre 1 a 4 dias")
else:
  print("marchita hace mucho")
```

```
1
marchita entre 1 a 4 dias
```

Fig. 40. Simulación desde resultados, simulación 4. [19]

5.

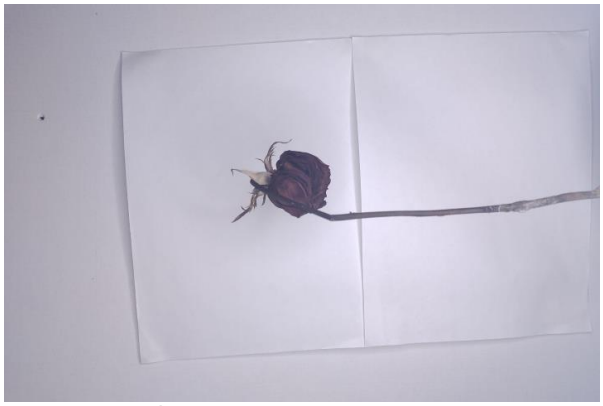


Fig. 41. Simulación desde resultados, ImagenPrueba_5.jpg.

```
#Simulacion 5
#0 = aceptable, 1 = marchita_dia_1-4, 2 = marchita_hace_mucho

url = 'https://i.ibb.co/njyJd9J/Imagen-Prueba-5.jpg'
prediccion = categorizar (url)
print(prediccion)

if prediccion == 0:
  print("aceptable")
elif prediccion == 1:
  print("marchita entre 1 a 4 dias")
else:
  print("marchita hace mucho")
```

```
1
marchita entre 1 a 4 dias
```

Fig. 42. Simulación desde resultados, simulación 5. [19]

6.

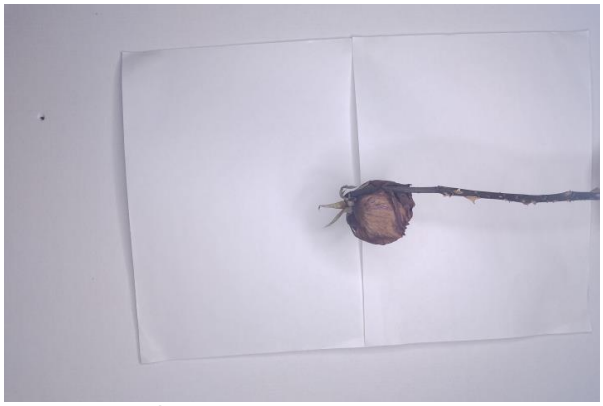


Fig. 43. Simulación desde resultados, ImagenPrueba_6.jpg.

```
#Simulacion 6
#0 = aceptable, 1 = marchita_dia_1-4, 2 = marchita_hace_mucho

url = 'https://i.ibb.co/jJqGhtN/Imagen-Prueba-6.jpg'
prediccion = categorizar (url)
print(prediccion)

if prediccion == 0:
  print("aceptable")
elif prediccion == 1:
  print("marchita entre 1 a 4 dias")
else:
  print("marchita hace mucho")
```

```
2
marchita hace mucho
```

Fig. 44. Simulación desde resultados, simulación 6. [19]

IX. Prueba de modelo en Colab(sin servicio web y sin base de datos)

Se realizo una prueba de cargar el modelo y que siguiese funcionando correctamente.

```
from keras.applications.imagenet_utils import preprocess_input
from keras.models import load_model

import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import cv2
import matplotlib.pyplot as plt
from PIL import Image
import requests
from io import BytesIO

width_shape = 224
height_shape = 224

names = ['aceptable','marchita_dia_1-4','marchita_hace_mucho']

path = 'models/model_Rosas.h5'
modelo = tf.keras.models.load_model((path),custom_objects={'KerasLayer':hub.KerasLayer})
print("Modelo cargado exitosamente")

imaget_path = "ImagenPrueba_1.jpg"
imaget=cv2.resize(cv2.imread(imaget_path), (width_shape, height_shape), interpolation = cv2.INTER_AREA)

xt = np.asarray(imaget)
xt=preprocess_input(xt)
xt = np.expand_dims(xt,axis=0)

print("Predicción")
preds = modelo.predict(xt)

print("Predicción:", names[np.argmax(preds)])
plt.imshow(cv2.cvtColor(np.asarray(imaget),cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

```
Modelo cargado exitosamente
Predicción
Predicción: marchita_hace_mucho
```

X. Prueba de Predicción con Servicio Web y Base de datos

Para implementar el servicio web se llevaron a acabo los siguientes pasos:

1. Crear carpetas necesarias y subir los archivos.

```
!mkdir models
!mkdir static
!mkdir templates

!mkdir static/css
!mkdir static/js
```

2. Instalar la librería de “flask-ngrok” y “pygrok”, las cuales sirven para crear una conexión a un servicio que hace un túnel desde el host local de “Colab” aun host temporal generado en la web.

```
!pip install flask-ngrok
```

```
!pip install pyngrok --quiet
```

```
from pyngrok import ngrok
```

```
from flask_ngrok import run_with_ngrok
```

3. Se carga la ficha de autenticación de ngrok.

```
!ngrok authtoken 25PC5a0BW7mU4bCLAFB3snSxKE7_4xG5EnEhmNoVHhMQ9K5WW
```

```
NGROK_AUTH_TOKEN = "25PC5a0BW7mU4bCLAFB3snSxKE7_4xG5EnEhmNoVHhMQ9K5WW"
```

4. Se cargan las librerías, se ajusta el tamaño de las imágenes, se decide que la aplicación usara ngrok, se carga el modelo, se define el modelo de predicción, se define que desde la pagina principal se puedan usar los métodos “GET” y “POST”. Se define que en la subpágina “/predict” se pueda usar “GET” y “POST”, también se categoriza las imágenes que sean cargadas en la página web y por último se envía la predicción.

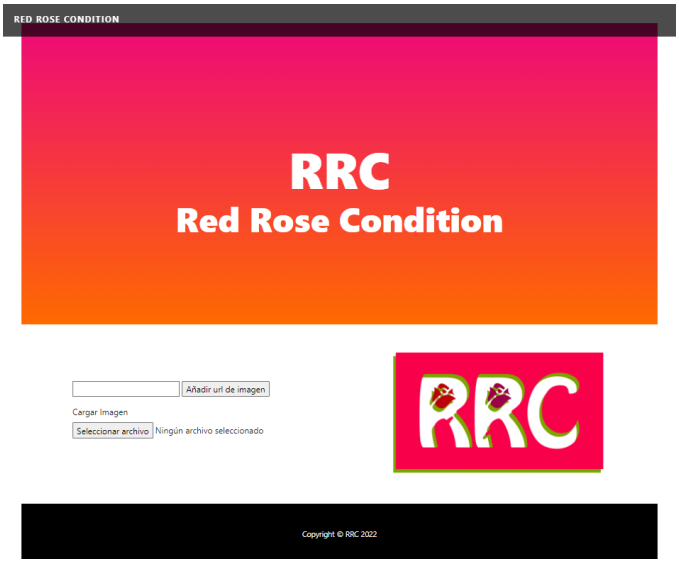
```python
from werkzeug.utils import secure_filename
import flask
import flask
from flask_ngrok import run_with_ngrok
from flask import Flask, request, render_template
import os
import numpy as np
import cv2
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import psycopg2

width_shape = 224
height_shape = 224

#names = ['aceptable','marchita_dia_1-4','marchita_hace_mucho']
names = ['0','1','2']

app = Flask(__name__)
run_with_ngrok(app)

path = 'models/model_Rosas.h5'
modelo = tf.keras.models.load_model((path),custom_objects={'KerasLayer':hub.KerasLayer})
print("Modelo cargado exitosamente")
```

```python
def model_predict(img_path, model):
    img=cv2.resize(cv2.imread(img_path),
                   (width_shape, height_shape),
                   interpolation = cv2.INTER_AREA)
    x=np.asarray(img)
    x=preprocess_input(x)
    x = np.expand_dims(x,axis=0)

    preds = model.predict(x)
    return preds

@app.route('/', methods=['GET', 'POST'])
def index():
    return render_template('index.php')

@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        f = request.files['file']

        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)

        preds = model_predict(file_path, model)

        print('PREDICCIÓN', names[np.argmax(preds)])

        result = str(names[np.argmax(preds)])
        return result
    return None

app.run()
```

5. Al correr la aplicación mostrara el enlace que creo ngrok, por medio de este se ingresara a la pagina principal.

```
Modelo cargado exitosamente
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
INFO:werkzeug: * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://427a-34-75-157-152.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

6. Al entrar al link se ve la pagina principal.



XI. Prueba de Predicción con Base de datos

Se realizo un sistema con predicción sin conexión a la base de datos, a causa de que la prueba anterior no dio frutos, por problemas técnicos, no ligados al modelo de predicción.
Para la elaboración de estos pasos se inicio con importar las librerías, ajustar el tamaño de las imágenes, asignar nombres a las categorías, cargar el modelo y ajustar lo valores de las imágenes.

```python
from werkzeug.utils import secure_filename
import os
import numpy as np
import cv2
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import psycopg2

width_shape = 224
height_shape = 224

#names = ['aceptable','marchita_dia_1-4','marchita_hace_mucho']
names = ['0','1','2']

path = 'models/model_Rosas.h5'
modelo = tf.keras.models.load_model((path),custom_objects={'KerasLayer':hub.KerasLayer})
print("Modelo cargado exitosamente")

def model_predict(img_path, model):
    img=cv2.resize(cv2.imread(img_path),(width_shape, height_shape),interpolation = cv2.INTER_AREA)
    x=np.asarray(img)
    x=preprocess_input(x)
    x = np.expand_dims(x,axis=0)

    preds = model.predict(x)
    return preds
```

Despues se hace la conexión a la base de datos, la cual fue creada por medio de “ElephantSQL”, un servicio de hosting de base de datos, administrada por “PgAdmin” , seguido a la conexión se hace un SELECT de la tabla imágenes, la cual cuenta con unicamente con una columna que contiene enlaces de la imágenes con las que se probara el modelo.

```
try:
  connection=psycopg2.connect(
      host='rosie.db.elephantsql.com',
      user='tnvfomkj',
      password='y22QIO3M0vRcv0ydEvak3JTIY-MhrmyL',
      database='tnvfomkj'
  )
  print("Conexión exitosa")
  cursor=connection.cursor()
  cursor.execute("SELECT * FROM imagen")
  rows=cursor.fetchone()
  for row in rows:
    print(row)
except Exception as ex:
    print(ex)
```

A continuación se definira como se categorizaran las imágenes y de donde se hara la predicción.

```
from PIL import Image
import requests
from io import BytesIO
import cv2

def categorizar(url):
  respuesta = requests.get(url)
  img = Image.open(BytesIO(respuesta.content))
  img = np.array(img).astype(float)/255

  img = cv2.resize(img, (224,224))
  prediccion = modelo.predict(img.reshape(-1, 224, 224, 3))
  return np.argmax(prediccion[0], axis=-1)
```

Y para finalizar se muestra el resultado de la predicción.

```
#0 = aceptable, 1 = marchita_dia_1-4, 2 = marchita_hace_mucho

url = row
prediccion = categorizar (url)
print(prediccion)

if prediccion == 0:
  print("aceptable")
elif prediccion == 1:
  print("marchita entre 1 a 4 dias")
else:
  print("marchita hace mucho")
```

XII. Resultados y discusión

Para analizar los resultados dados por la simulación, se reviso uno por uno, siendo los siguientes puntos correspondientes al orden en el que se dieron en la simulación de resultados:

1. Se cargo una imagen con una rosa en estado “aceptable” y el modelo dio una respuesta **correcta**.
2. Se cargo una imagen con una rosa en estado “aceptable” y el modelo dio una respuesta **correcta**.
3. Se cargo una imagen con una rosa en estado “marchita entre 1 a 4 días” y el modelo dio una respuesta **correcta**.
4. Se cargo una imagen con una rosa en estado “marchita entre 1 a 4 días” y el modelo dio una respuesta **correcta**.
5. Se cargo una imagen con una rosa en estado “marchita hace mucho” y el modelo dio una respuesta **incorrecta**.
6. Se cargo una imagen con una rosa en estado “marchita hace mucho” y el modelo dio una respuesta **correcta**.

Se probo con dos imágenes de cada categoría y se observó que de las 6 simulaciones solamente una imprimió la predicción incorrecta.

Análisis por prueba:

- Sin servicio web y sin base de datos: se puede notar bastante simple, pero sin inconvenientes, fácil de hacer y los resultados fueron correctos.
- Con servicio web y base de datos: en este caso no se pudo desarrollar de manera correcta, se intentó lo máximo posible, pero no se consiguió el objetivo deseado, al ser todo en la nube se tuvo obtener nuevos conocimientos, se logro conectar Google Colab con un servicio de hosting de paginas web temporales (ngrok), diseñado para desarrolladores, pero hubo inconvenientes al tratar de cargar información desde la pagina web al Google Colab y a la base de datos de ElephantSQL, estos errores se pueden observar en consola los errores no solucionados.

```
INFO:werkzeug:127.0.0.1 - - [24/Feb/2022 06:52:15] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [24/Feb/2022 06:52:15] "GET /static/css/main.css HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [24/Feb/2022 06:52:16] "GET /js/scripts.js HTTP/1.1" 404 -
INFO:werkzeug:127.0.0.1 - - [24/Feb/2022 06:52:16] "GET /static/js/main.js HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [24/Feb/2022 06:52:16] "GET /js/scripts.js HTTP/1.1" 404 -
INFO:werkzeug:127.0.0.1 - - [24/Feb/2022 06:52:16] "GET /favicon.ico HTTP/1.1" 404 -
```

- Sin servicio web y con base de datos, no tiene los mejores apartados graficos, la usabilidad no es la mejor, pero consigue el objetivo, predice mayormente con resultados correctos con las tres diferentes categorias.

XIII. Conclusiones

En este documento se mostró cómo se llevó a cabo el proyecto “RRC” para la materia de “Data Science”, que se necesitó, que se usó, como funciona, cuál es su propósito y como cumplió con esto. Este proyecto categoriza si una rosa roja está en condición “aceptable”, “marchita desde hace 1 a 4 días” o “marchita desde hace mucho”, siendo “mucho” más de 4 días, con un porcentaje de precisión promedio del **81.23%.**
Los resultados muestran una precisión buena, según la rúbrica con la cual se evalúa este proyecto, con esto logrando satisfacer la problemática inicial. Puede que en “Resultados y discusión” haya habido una respuesta incorrecta pero no le quita el valor al proyecto y va de acuerdo con la precisión conseguida.

XIV. Trabajos Futuros

El proyecto puede ser complementado de varios modos como: incrementando el dataset, creando nuevas categorías y desarrollando un servicio web o un API, que funcione correctamente, para que el proyecto tenga una mejor usabilidad en la utilización del modelo.

Este proyecto será usado para complementar el proyecto del mismo nombre que será entrega en la materia “Visión Artificial” y quizás en alguna otra materia aun no prevista.

XV. Referencias

Las referencia con color verde, fueron las principales referencia para el estado del arte, las de color rojo son referencias en ingles y las de color negro son referencias comunes en español.

[1] [1] D. R. Luna, «API_DeepLearning,» 15 Enero 2021. [En línea]. Available: https://github.com/DavidReveloLuna/API_DeepLearning. [Último acceso: 21 Febrero 2022].

[2] [2] R. T. «Clasificación de imágenes (perros y gatos),» 13 Septiembre 2021. [En línea]. Available: https://github.com/ringa-tech/clasificador-perros-gatos. [Último acceso: 21 Febrero 2022].

[3] [3] O. A. M. Amaya, «Diseño De Un Sistema De Visión Artificial Para El Análisis De Calidad Y Producción De Rosas,» 2019. [En línea]. Available: http://repository.pedagogica.edu.co/bitstream/handle/20.500.12209/12074/TE-24132.pdf?sequence=1. [Último acceso: 21 Febrero 2022].

[4] [4] P. A. C. Aguilar, «Diseño de un sistema de identificación y clasificación de flores,» 2017. [En línea]. Available: https://ciencia.lasalle.edu.co/cgi/viewcontent.cgi?article=1000&context=ing_automatizacion. [Último acceso: 21 Febrero 2022].

[5] [5] B. Y. Alexis, C. G. Jonathan, A. B. Ángela, Y. V. Marco y L. R. Samuel, «Sistema de clasificación de rosas de la variedad explorer usando visión por computadora,» 2019. [En línea]. Available: https://investigacion.utmachala.edu.ec/proceedings/index.php/utmach/article/download/442/382/799. [Último acceso: 21 Febrero 2022].

[6] [6] R. T. «Clasificación de imágenes (perros y gatos),» 13 Septiembre 2021. [En línea]. Available: https://colab.research.google.com/drive/1DQc8a-WOTctenvoy5T0lWUXn47EuteCy?usp=sharing. [Último acceso: 21 Febrero 2022].

[7] [7] M. Sandler, «tf2-preview/mobilenet_v2/feature_vector,» TensorFlowHub, 13 Enenro 2019. [En línea]. Available: https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4. [Último acceso: 21 Febrero 2022].

[8] [8] «Keras,» Keras, [En línea]. Available: https://keras.io/api/models/model_saving_apis/#save_model-function. [Último acceso: 21 Febrero 2022].

[9] [9] «Red Leaves On Roses: What To Do For Red Leaves On A Rose Bush,» Gardening know how, 26 Julio 2021. [En línea]. Available: https://www.gardeningknowhow.com/ornamental/flowers/roses/red-leaves-on-roses.htm. [Último acceso: 21 Febrero 2022].

[10] [10] C. Boeckmann, «Roses,» Almanac, 2019. [En línea]. Available: https://www.almanac.com/plant/roses. [Último acceso: 21 Febrero 2022].

[11] [11] Jackson & Perkins, «Rose Anatomy 101,» Jackson & Perkins, 11 Septiembre 2021. [En línea]. Available: https://www.jacksonandperkins.com/blog/rose-blogs/rose-anatomy/b/rose-anatomy/. [Último acceso: 21 Febrero 2022].

[12] [12] A. Howard, «MobileNetV2: Inverted Residuals and Linear Bottleneck,» Arvix, 13 Enero 2018. [En línea]. Available: https://arxiv.org/abs/1801.04381. [Último acceso: 21 Febrero 2022].

[13] [13] M. Sanjeevi, «Chapter 8 .0: Convolutional neural networks for deep learning,» Medium, 15 Octubre 2018. [En línea]. Available: https://medium.com/deep-math-machine-learning-ai/chapter-8-0-convolutional-neural-networks-for-deep-learning-364971e34ab2. [Último acceso: 21 Febrero 2022].

[14] [14] A. Kumar, «Keras – Categorical Cross Entropy Loss Function,» Vitalflux, 28 Octubre 2020. [En línea]. Available: https://vitalflux.com/keras-categorical-cross-entropy-loss-function/. [Último acceso: 21 Febrero 2022].

[15] [15] A. Khaled, «Unknown layer: KerasLayer when i try to load_model,» Assem Khaled, 16 Mayo 2020. [En línea]. Available: https://stackoverflow.com/questions/61814614/unknown-layer-keraslayer-when-i-try-to-load-model. [Último acceso: 21 Febrero 2022].

[16] [16] itdxer, «What is batch size in neural network?,» itdxer, 5 Abril 2019. [En línea]. Available: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network. [Último acceso: 21 Febrero 2022].

[17] [17] M. Koziarkiewicz, «Rose Rosette Disease diagnosis through Deep Learning,» softwaremill, 3 Junio 2020. [En línea]. Available: https://blog.softwaremill.com/rose-rosette-disease-diagnosis-through-deep-learning-155ec3b45a6b. [Último acceso: 21 Febrero 2022].

[18] [18] Ó. M. d. l. F. Sanz, «Google Colab: Python y Machine Learning en la nube,» adictosaltrabajo, 4 Junio 2019. [En línea]. Available: https://www.adictosaltrabajo.com/2019/06/04/google-colab-python-y-machine-learning-en-la-nube/. [Último acceso: 21 Febrero 2022].

[19] [19] imgbb, «imgbb,» imgbb, 22 Enero 2022. [En línea]. Available: https://es.imgbb.com/. [Último acceso: 21 Febrero 2022].

[20] [20] J. Barrios, «Redes Neuronales Convolucionales,» Juan Barrios, 2022. [En línea]. Available: https://www.juanbarrios.com/redes-neurales-convolucionales/. [Último acceso: 21 Febrero 2022].

[21] [21] M. Padriñan, «Matemáticas de las redes neuronales convolucionales,» ichi.pro, 2021. [En línea]. Available: https://ichi.pro/es/matematicas-de-las-redes-neuronales-convolucionales-30697178023872. [Último acceso: 21 Febrero 2022].

[22] [22] J. Cuevas, «Diseño de una red neuronal convolucional para la clasificación de señales,» 2019. [En línea]. Available: https://www.ecorfan.org/proceedings/Proceedings_Ciencias_de_la_Tierra_Fisica_y_Matematicas_TI/Proceedings_Ciencias_de_la_Tierra_Fisica_y_Matematicas_TI_4.pdf. [Último acceso: 21 Febrero 2022].

[23] [23] L. D. Solutions, «Deep Learning, redes neuronales y visión artificial,» LIS Data Solutions, 2021. [En línea]. Available: https://www.lisdatasolutions.com/blog/deep-learning-redes-neuronales-y-vision-artificial/. [Último acceso: 21 Febrero 2022].

[24] [24] TensorFlow, «Guardando y Serializando Modelos con TensorFlow Keras,» TensorFlow Core, 05 11 2021. [En línea]. Available: https://www.tensorflow.org/guide/keras/save_and_serialize#registering_the_custom_object. [Último acceso: 21 Febrero 2022].

[25] [25] D. Magic, «github,» Data Magic, 24 Febrero 2021. [En línea]. Available: https://github.com/datamagic2020/deploy-ML-model-with-webapp. [Último acceso: 21 Febrero 2022].

[26] [26] M. Rivas, «GitHub,» Marcos Rivas, 10 Septiembre 2017. [En línea]. Available: https://github.com/marcosrivasr/Sitio-Web-CSS3. [Último acceso: 21 Febrero 2022].

[27] [27] O. Garcia, «GitHub,» Oscar Garcia, 21 Enero 2022. [En línea]. Available: https://github.com/UskoKruM. [Último acceso: 21 Febrero 2022].

[28] [28] S. Muria, «Youtube,» Shak Muria, 7 Julio 2021. [En línea]. Available: https://www.youtube.com/watch?v=62n_MhpHfoU. [Último acceso: 21 Febrero 2022].

[29] [29] Vida MRR, «YouTube,» Vida MRR, 12 Abril 2018. [En línea]. Available: https://www.youtube.com/watch?v=GNfTYvCVjSc. [Último acceso: 21 Febrero 2022].

[30] [30] Hot Examples, «Hot Examples,» Hot Examples, [En línea]. Available: https://python.hotexamples.com/es/examples/OpenLegLib.utils/-/fetch/python-fetch-function-examples.html. [Último acceso: 21 Febrero 2022].

[31] [31] w3resource, «w3resource,» w3resource, 19 Octubre 2021. [En línea]. Available: https://www.w3resource.com/PostgreSQL/PostgreSQL-insert.php. [Último acceso: 21 Febrero 2022].

[32] [32] My PHP.net, «php,» My PHP.net, [En línea]. Available: https://www.php.net/manual/es/function.pg-insert.php. [Último acceso: 21 Febrero 2022].

[33] [33] informaticapc.com, «informaticapc.com,» informaticapc.com, [En línea]. Available: https://informaticapc.com/tutorial-php/bases-de-datos-postgresql.php. [Último acceso: 21 Febrero 2022].

[34] [34] Vishal, «PYnative,» PYnative, 9 Marzo 2021. [En línea]. Available: https://pynative.com/python-cursor-fetchall-fetchmany-fetchone-to-read-rows-from-table/. [Último acceso: 21 Febrero 2022].

[35] startbootstrap, «startbootstrap,» startbootstrap, 21 Septiembre 2013. [En línea]. Available: https://startbootstrap.com/theme/one-page-wonder. [Último acceso: 21 Febrero 2022].