Green Citrus Leave Identification

CHRISTIAN RODRIGUEZ, ESAU MAY, ERUBIEL TUN, EDWIN CAN

Compiled December 6, 2023

This paper addresses the widespread threat of citrus greening, also known as Huanglongbing (HLB), to the global citrus industry. Caused by the bacterium Candidatus Liberibacter asiaticus, this disease poses significant economic risks due to its detrimental impact on the health and productivity of citrus trees. Traditional diagnostic methods, based on visual inspection by specialists, are time-consuming and error-prone. Taking advantage of advances in machine learning, this study presents the design and implementation of a supervised learning model, an unsupervised learning model, and a reinforcement learning mode to accurately identify and classify citrus greening symptoms on leaves. The main objective is to distinguish between healthy foliage and leaves showing signs of citrus greening using a photographic dataset and be able to provide a reliable tool for early detection and mitigation, revolutionizing citrus disease management. By integrating machine learning algorithms and image processing techniques, the proposed methodology offers an efficient and proactive approach to monitor and address the plant-related problem of Huanglongbing (HLB) pest. Successful adoption of this tool has the potential to transform agricultural practices, providing growers and specialists with a more effective means of managing and combating the spread of citrus greening.

1. INTRODUCTION

This technical paper delves into the design and implementation of a supervised machine learning model, an unsupervised learning model and a reinforcement learning model; this with the objective of accurately identifying and classifying citrus greening leaves. Our goal is to develop a reliable tool to aid in the early diagnosis and mitigation of citrus greening through the use of state-of-the-art machine learning algorithms and image processing techniques. Successful adoption of this methodology has the potential to change citrus disease management techniques, providing growers and agricultural specialists with an efficient and proactive approach to monitor and address disease development.

2. MAIN PROBLEM

Like humans, plants tend to suffer from different diseases throughout their growth, within citrus there is a critical problem known as *Huanglongbing (HLB)*; this is a pest that severely attacks different citrus and has become the most devastating citrus disease worldwide and is caused mainly by the bacterium *Candidatus liberibacter asiaticus* although there are also other forms of the bacterium that also cause the disease; For example, this disease could be spread through the use of infected propagation material (buds or plant parts) and by an insect vector called *Diaphorina citri* which, by feeding on a diseased HLB plant, is able to acquire the bacterium and transmit it to other healthy plants when it feeds on them.

The main problem of this situation is that in order to avoid dispersion, constant monitoring is needed in large agricultural areas which is a complicated task to perform manually, so a solution proposed is to use tools and machine learning models to attack the different problems presented in such a way that the spread of the pest can be detected at an early stage and mitigate this problem in order to take precautions and avoid the spread to healthy plants and thus reduce the number of citrus affected.

3. PROPOSED SOLUTION

As previously mentioned, our proposal to address the Huanglongbing (HLB) pest detection problem consists in the application of machine learning tools. To achieve this, we first use supervised learning together with a dataset containing photographs of leaves affected by the greening problem, which is one of the main characteristics of HLB-contaminated species. The idea is to train a model capable of distinguishing between healthy leaves and leaves infected with greening. Since some samples can sometimes be difficult to identify with the naked eye, this model becomes a valuable tool to analyze leaves efficiently and determine the presence or absence of the pest; this approach is especially useful for leaves that are difficult to analyze visually. Regarding the implementation of unsupervised learning, we plan to use K-Means and PCA to group citrus leaf

images into two clusters, with the objective of identifying patterns that may indicate the presence of the "greening" disease. Additionally, as an integral part of our solution, we have implemented reinforcement learning. This method allows training an agent to classify the health of a plant, thus providing detailed analysis and effective control over samples affected by the pest. This complementary approach strengthens our overall strategy, enhancing the system's ability to efficiently identify and manage the presence of HLB in plants.

4. COMPONENT AND DATA USED

In order to train the models, it was first necessary to investigate and research more about this problem, until we finally found a dataset composed of images representing those samples that have been affected by the pests and some images representing healthy citrus leaf samples; to be more specific, this dataset we used was composed of 186 files labeled as *"greening"* and another 191 files labeled as *"healty"* so we consider it a good dataset to apply to our models because we have almost the same number of samples of both labels. This dataset is called Citrus Greening and can be accessed through the Kaggle platform, in our particular case we decided to work in this platform the Supervised Learning section, since the dataset was quite heavy, however it could be worked in a notebook in Kaggle without any problem, additionally, the part of unsupervised learning and reinsforcement were developed in a Colab notebook due to the facilities that this offers us to work collaboratively.

5. METODOLOGY USED

A. Supervised Learning

To apply the different types of Machine Learning, we first started with Supervised Learning, in this way we first imported the necessary libraries such as: numpy, matplotlib, tensorflow, among others; then we defined the directory where the images are located or rather the dataset to be used as shown in the image below.

```
In [1]:
# standard libraries
import numpy as np
import time
import PIL.Image as Image
import matplotlib.pylab as plt
import matplotlib.image as mpimg
%matplotlib inline
import datetime
from tqdm.keras import TqdmCallback
from skimage import transform
import requests

# tensorflow libraries
import tensorflow as tf
import tensorflow_hub as hub
```

Fig. 1. Libraries

After this, the data contained in the dataset was divided, so it was necessary to divide said data for training and testing. In the same way, some configurations were made such as the normalization of the data and the implementation of the efficentnet_b7_fv. variable that holds the URL pointing to a pre-trained model available on Kaggle. This refers to EfficientNet B7 Models used for extracting features from images.

```
# feature vector model
efficientnet_b7_fv = 'https://kaggle.com/models/tensorflow/efficientnet/frameworks/
feature_extractor_model = efficientnet_b7_fv

feature_extractor_layer = hub.KerasLayer(
    feature_extractor_model,
    input_shape=(img_width, img_height, 3),
    trainable=False)
```

Fig. 2. EfficietnetB7

Hub.KerasLayer: This function creates a Keras layer using TensorFlow. It provides a way to use pretrained models and modules. So, the KerasLayer allows you to use a model from the URL.

Input_shapes: Specifies the input shape expected by the models, define the width and the height of the input images.

Trainable = False, configuring trainable = false When a layer contains false, its weights become frozen and cannot be modified during training. This is frequently employed in transfer learning to add new layers for a given job while maintaining previously acquired features.

Subsequently, the model was configured to be trained, then, What happens here is that the model is configured for training, as we can see in the code we use an optimization algorithm called Adam that is responsible for adjusting the weights of the network to minimize the loss function, said loss function measures the difference between the predicted and true labels, finally, metrics are used to monitor during training and at the end print the accuracy of the model. After this, early stopping is configured as a regularization technique, this means that training will stop if the loss metric stops remembering after a certain number of epochs **patience=3**. In the same way, the number of training epochs is defined, that is, the times that the model will see the entire training data set.

```
[ ]:
# compile the model
model.compile(
  optimizer=tf.keras.optimizers.Adam(),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['acc'])

# early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

# Define Epochs
NUM_EPOCHS = 4
```

Fig. 3. Configuration for training

It is during this line of code where the model training is actually done; During this process the fit method is used providing the training and validation data sets, the number of epochs, and the stop early callback. Finally, the final precision of the model is calculated after training and is intended to show us said precision in percentage format.

```
In [7]:
# train the model
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=NUM_EPOCHS,
                    callbacks=[early_stopping, TqdmCallback(verbose=0)],verbose=0)

# view model accuracy
model_acc = '{:.2%}'.format(history.history['acc'][-1])
print(f"\n Model Accuracy Reached: {model_acc}")

100% 4/4 [01:26<00:00, 14.59s/epoch, loss=0.462, acc=0.872, val_loss=0.492, val_acc=0.807]

 Model Accuracy Reached: 87.22%
```

Fig. 4. Model Training

Finally, matplotlib is used to plot the training history (weight and loss) over the epochs, with the aim of being able to visualize how the model learns the training data set.

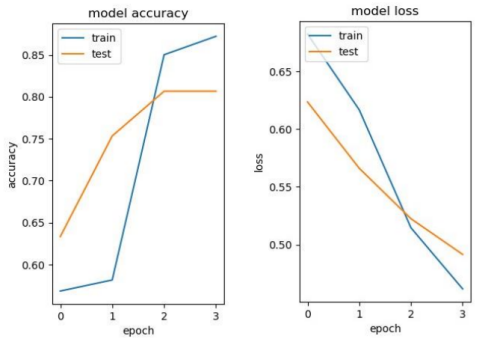

Fig. 5. Model Training

B. Unsupervised Learning

For the Unsupervised Learning case we employed the K-Means algorithm for the detection of the citrus disease called citrus greening. We first download a citrus disease related dataset from Kaggle. Then, the necessary configuration is performed to access this data in Google Colab to then load and resize citrus leaf images from two different folders, one for leaves affected by greening and one for healthy leaves. The images are loaded, resized to 64x64 pixels and stored in separate numpy arrays for the two classes.

```
In [6]:
import os
import cv2
import numpy as np
import random
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

In [7]:
def load_images(folder):
    images = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        img = cv2.resize(img, (64, 64))  # Resize the image to a fixed size
        images.append(img)
    return images
```

Fig. 6. load_images

Subsequently, flattening and normalization of the images was applied. The images were flattened to convert them into one-dimensional vectors and then combined into a single data set. The pixel values are normalized by dividing by 255. Now the next step was to use the K-Means algorithm with two clusters (one for greening and one for healthy leaves) on the normalized and flattened data in which points are assigned to one of the two clusters based on the similarity of their features.

```
In [22]:
# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
clusters = kmeans.fit_predict(all_images_flatten)
```

Fig. 7. K-Means

Finally, Principal Component Analysis (PCA) was applied to reduce the dimensionality of the data to two dimensions and visualizes the clustering results using a scatter plot. Each point in the graph represents an image, and the colors indicate to which cluster it belongs according to the K-Means algorithm.

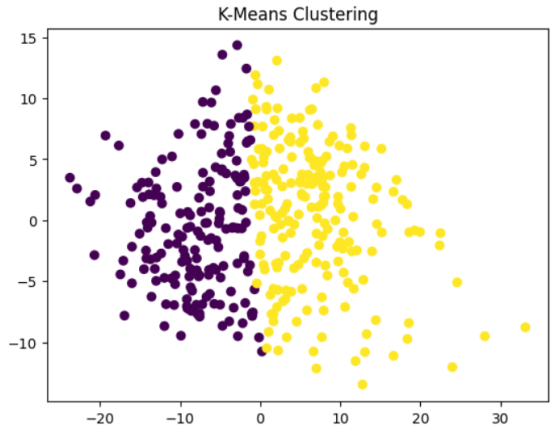

Fig. 8. Graph of the result

C. Reinforcement Learning

The code implements a reinforcement learning (Q-learning) agent for classifying images of citrus plants as "greening" or "healthy." In the first lines, the Kaggle environment is set up to download the necessary dataset. A directory is created to store the Kaggle API key, the key is copied into that directory, and the dataset is downloaded and extracted.

Next, the necessary libraries are imported, and functions are defined to load and process the images. Images are loaded from specific folders, resized, and assigned a label (0 for "greening" and 1 for "healthy"). Images and labels are combined into separate sets for "greening" and "healthy" plants.

```
greening_path = '/content/dataset/field/greening'
healthy_path = '/content/dataset/field/healthy'

def load_images(folder, label):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        img = cv2.resize(img, (64, 64))  # Resize the image to a fixed size
        images.append(img)
        labels.append(label)
    return images, labels

greening_images, greening_labels = load_images(greening_path, label=0)
healthy_images, healthy_labels = load_images(healthy_path, label=1)
```

Fig. 9. Load dataset

Afterward, the data is split into training and testing sets using the 'train_test_split' function from scikit-learn. Images are normalized by dividing pixel values by 255 to ensure they are in the range [0, 1].

```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(all_images, all_labels, test_size=0.2, random_state=42)
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))
```

```
301
76
301
76
```

```
# Normalizing pixel values to be between 0 and 1
X_train, X_test = X_train / 255.0, X_test / 255.0
```

Fig. 10. Split data

A convolutional neural network (CNN) model is defined using Keras' Sequential API. This model consists of convolutional layers with max-pooling and dense layers. The model is compiled with the Adam optimizer and the mean squared error (MSE) loss function, typical in Q-learning problems.

```
# Defining a CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(2, activation='linear'))  # 2 output neurons for the two actions (greening or healthy)
```

```
model.compile(optimizer='adam', loss='mse')  # Mean Squared Error Loss for Q-Learning
```

Fig. 11. CNN

Then, the 'PlantEnvironment' class is defined to represent the reinforcement learning environment. This class has methods to reset the environment, get the current state, and perform a step with a given action.

```
# Defining the reinforcement learning environment
class PlantEnvironment:
    def __init__(self, images, labels):
        self.images = images
        self.labels = labels
        self.current_index = 0
        self.done = False

    def reset(self):
        self.current_index = 0
        self.done = False
        return self.get_state()

    def get_state(self):
        return self.images[self.current_index], self.labels[self.current_index]

    def step(self, action):
        self.current_index += 1
        if self.current_index == (len(self.images) - 1):
            self.done = True
        next_state = self.get_state()
        reward = 1 if action == next_state[1] else -1  # Reward for correct action, penalty for incorrect action
        return next_state, reward, self.done
```

```
# Creating the reinforcement learning environment
env = PlantEnvironment(images=X_train, labels=y_train)
```

Fig. 12. Learning Environment

The 'PlantAgent' class represents the Q-learning agent and has methods to choose an action based on the current state and update Q-values based on the observed reward.

```
# Define the Q-Learning agent
class PlantAgent:
    def __init__(self, model, action_space, learning_rate=0.001, discount_factor=0.9, epsilon=0.1):
        self.model = model
        self.action_space = action_space
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.epsilon = epsilon

    def choose_action(self, state):
        if np.random.uniform(0, 1) < self.epsilon:
            return random.choice(self.action_space)  # Exploratory action
        else:
            q_values = self.model.predict(np.expand_dims(state[0], axis=0))[0]
            return np.argmax(q_values)  # Exploitative action

    def update_q_values(self, state, action, reward, next_state):
        current_q_values = self.model.predict(np.expand_dims(state[0], axis=0))[0]
        next_q_values = self.model.predict(np.expand_dims(next_state[0], axis=0))[0]
        td_target = reward + self.discount_factor * np.max(next_q_values)
        td_error = td_target - current_q_values[action]
        current_q_values[action] += self.learning_rate * td_error
        return current_q_values
```

```
# Creating the Q-Learning agent
agent = PlantAgent(model=model, action_space=[0, 1])
```

Fig. 13. Q-Learning

Next, an instance of the agent is created, and training occurs for a specified number of episodes. The training loop involves selecting actions, updating Q-values, and accumulating rewards.

```
# Training
num_episodes = 50

# Initialize lists to store rewards during training
training_rewards = []

for episode in range(num_episodes):
    state = env.reset()
    done = False
    episode_reward = 0

    while not done:
        action = agent.choose_action(state)
        next_state, reward, done = env.step(action)
        q_values = agent.update_q_values(state, action, reward, next_state)
        state = next_state
        episode_reward += reward

    # Store the reward for this episode
    training_rewards.append(episode_reward)

print("Training complete.")
```

Fig. 14. Training

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
Total reward on the test set: -15 [-15]
```

Fig. 15. Rewards

Finally, the trained agent is evaluated on the test set, and the total reward is calculated. Matplotlib is used to visualize the learning curve, showing total rewards across training episodes.

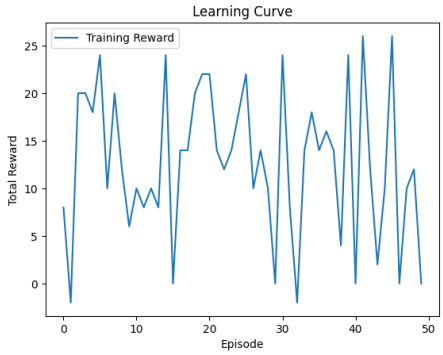

Fig. 16. Learning Curve

6. EVALUATION METHOD USED

A. Unsupervised Learning

A.1. Preparing and loading data:

- Two folders are used to load the images: one has greening-related photographs, and the other has healthy-related images.
- Images are shrunk to 64 by 64 pixels, which is a fixed size.
- After being transformed into NumPy arrays, images are flattened.

A.2. K-Means Grouping:

- One array is created by concatenating the flattened images together.
- Normalized pixel values fall within the interval [0, 1].
- Using `n_clusters=2`, K-Means clustering divides the photos into two clusters according to how similar their pixel values are.

A.3. Principal Component Analysis, or PCA:

- To show the clustered data, dimensionality reduction is achieved by PCA.
- There are two primary components used to construct the reduced data.

A.4. Illustration:

- Using the reduced data from PCA, a scatter plot is created, with the points colored according to the K-Means groupings.
- The scatter plot makes it easier to see how effectively K-Means has categorized the photographs according to their pixel values.

B. Reinforcement Learning

B.1. Training Assessment:

- Monitoring Training Reward Systems: In the training phase, the agent's rewards in every episode are documented. These awards show how well the agency is doing in relation to its training.

B.2. Evaluation of Test Sets:

- Assessment of Test Data: An independent test set is used to gauge the performance of the trained agent. Acting in accordance with its learned policy, the agent engages with the test environment. On the test set, the total reward earned during this interaction is computed.

C. Supervised Learning

C.1. Model Training and Validation:

- `Model.fit` is used to train the model over a predetermined number of epochs.
- Training performance of the model is assessed using validation data.

C.2. Metrics for Model Evaluation:

- The evaluation metric is accuracy. This is measured (`model_acc` variable) during model training and printed after training is finished.
- Plotting and visualizing the model's accuracy and loss across epochs on training and validation datasets is done using training history (`history`).

C.3. Early Termination:

- The `EarlyStopping` callback is used to halt the training process if the loss measure does not improve after a predetermined number of epochs (`patience=3`) to prevent overfitting.

C.4. Forecast for a Test Picture:

- The same preprocessing procedures used for the training data are used to obtain a test image from a URL and get it ready for prediction.
- The test image's class (`model.predict`) is predicted using the trained model.
- For visual inspection, the test image and the anticipated class are presented together.

7. CONSLUSION AND FUTURES STEPS

In conclusion for this project, there were successfully implemented three distinct approaches for plant health monitoring. The supervised learning model effectively categorizes plants as healthy or unhealthy based on labeled data. This foundational step is critical for early detection and targeted treatment of plant diseases. Moving forward, enhancing the supervised model involves refining and expanding the dataset to encompass a broader range of plant species and environmental conditions. Additionally, exploring advanced deep learning architectures can capture more complex relationships inherent in plant health. In the realm of unsupervised learning, our approach focuses on classifying different types of plants into healthy or greening categories without relying on labeled data. This technique provides valuable insights into plant variations and aids in identifying patterns that contribute to the overall health assessment. Future steps involve delving deeper into feature engineering to uncover additional attributes influencing plant health. Further exploration of various clustering algorithms promises to refine our understanding of plant type classification.

The reinforcement learning component introduces an autonomous decision-making agent trained to classify plants as healthy or not healthy. Leveraging Q-learning, the agent exhibits promise in adapting and learning over time. To optimize its performance, future steps include fine-tuning hyperparameters and exposing the agent to more complex scenarios, such as varied environmental conditions. Gradual exposure to intricate situations aims to enhance the agent's adaptability and decision-making capabilities.

In summary, the integration of supervised learning for labeled data, unsupervised learning for pattern recognition, and reinforcement learning for autonomous decision-making forms a comprehensive framework for plant health monitoring. Continuous collaboration between these models opens avenues for creating a robust system capable of adapting to evolving challenges in plant health. The project's success lies in its potential to contribute to more sophisticated and adaptable plant health monitoring systems in the future.

You can see the code here: https://github.com/Edwin9977/predictor_ML_from_scratch_and_sklearn/tree/main/final_ml_project

REFERENCES

1. Current situation, "Situación Actual y Perspectivas del Manejo del HLB de los Cítricos," Org.mx. [Online]. Available: https://www.scielo.org.mx/pdf/rmfi/v32n2/2007-8080-rmfi-32-02-00108.pdf. [Accessed: 06-Dec-2023].
2. "HLB: Cómo reconocer los síntomas de la enfermedad más grave de los cítricos," Argentina.gob.ar, 01-Sep-2021. [Online]. Available: https://www.argentina.gob.ar/noticias/hlb-como-reconocer-los-sintomas-de-la-enfermedad-mas-grave-de-los-citricos. [Accessed: 06-Dec-2023].
3. M. Marques, "Citrus Greening." 14-Jul-2022