



®

Instituto Tecnológico de Culiacán

## Inteligencia Artificial Modelo de Plantas

**Unidad:** 4

**Maestro:** Zuriel Dathan Mora Felix

**Carrera:** Ingeniería en Sistemas  
Computacionales

**Integrantes:**

Barraza Leon Miguel Angel

De Los Santos Cuevas Daniel Alejandro

Culiacán, Sinaloa.

27/05/25

# Documentación Clasificador de Plantas

## Descripción General

Este proyecto entrena un modelo de red neuronal para clasificar imágenes en **30 clases diferentes** utilizando la técnica de **Transfer Learning**, aprovechando la arquitectura **MobileNetV2**, que ha sido previamente entrenada con el dataset ImageNet. El sistema final permite realizar predicciones en tiempo real usando la cámara del dispositivo.

## Estructura del Dataset

El conjunto de datos está organizado en 3 carpetas principales:

- Train\_Set\_Folder/ — Datos de entrenamiento (30 clases × 100 imágenes = 3000 imágenes)
- Validation\_Set\_Folder/ — Datos de validación (30 clases × 100 imágenes = 3000 imágenes)
- Test\_Set\_Folder/ — Datos de prueba (30 clases × 100 imágenes = 3000 imágenes)

Cada clase contiene **100 imágenes**.

## Arquitectura del Modelo

Se utiliza **MobileNetV2 es un modelo preentrenado**, lo que significa que:

- Ya fue entrenado por expertos en un **dataset gigantesco** llamado **ImageNet** (más de 1 millón de imágenes en 1000 clases).
- Ha aprendido a **extraer características visuales complejas**: bordes, formas, texturas, patrones... que son comunes a muchas imágenes, **incluso fuera de ImageNet**.

## Estructura del modelo final:

### Base MobileNetV2 (sin capa superior)

- `include_top=False`: excluye la capa de clasificación original.
- `weights='imagenet'`: utiliza pesos preentrenados.
- `trainable=False`: sus pesos no se ajustan durante el entrenamiento.

## **GlobalAveragePooling2D**

Reduce la salida convolucional a un vector, manteniendo la información espacial.

- **Dense(128, activation='relu')**  
Capa densa oculta para aprender combinaciones de características.
- **Dense(30, activation='softmax')**  
Capa de salida con activación softmax para clasificar entre las 30 clases.

## **Proceso de Entrenamiento**

- **Optimizador:** Adam
- **Función de pérdida:** categorical\_crossentropy
- **Métrica:** accuracy
- **Tamaño de imagen:** 224x224 píxeles
- **Batch size:** 32
- **Épocas:** 5

## **Aumento de datos:**

- Rotación aleatoria (rotation\_range=20)
- Zoom aleatorio (zoom\_range=0.2)
- Inversión horizontal (horizontal\_flip=True)
- Normalización (rescale=1./255)

## **Resultados (entrenamiento hasta la época 3)**

**Precisión en validación:** ~90%

**Pérdida en validación:** ~0.28

El modelo muestra una mejora constante en precisión y pérdida.

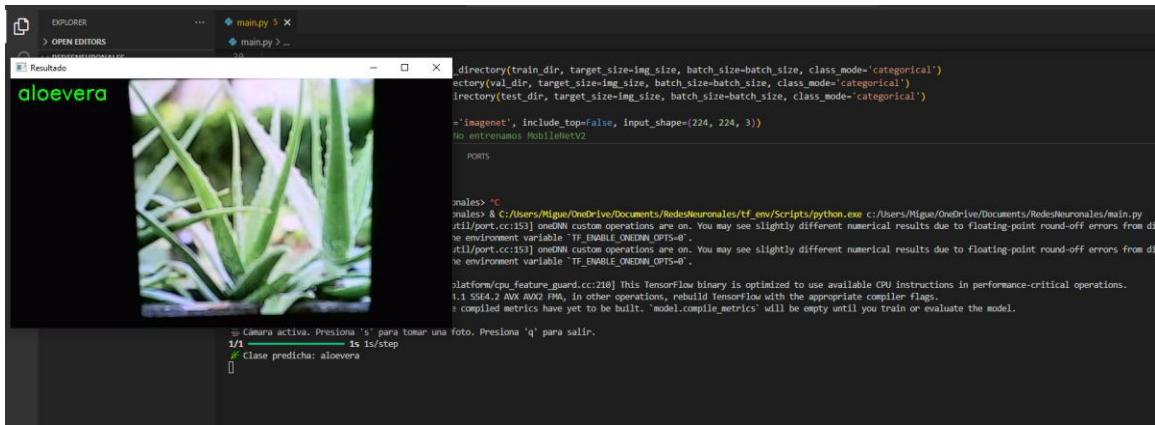
```

9:44 AM 11/24/2018 - [ipykernel_launcher.py:main:3]
Epoch 1/5
750/750 [=====] 0s 689ms/step - accuracy: 0.6497 - loss: 1.2565C:\Users\Wigue\OneDrive\Documents\RedesNeuronales\tf_env\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset
ing: Your 'PyDataset' class should call `super().__init__(**kwargs)` in its constructor. '**kwargs' can include 'workers', 'use_multiprocessing', 'max_queue_size'. Do not pass these arguments t
ignored.
    self._warn_if_super_not_called()
750/750 [=====] 568s 750ms/step - accuracy: 0.6499 - loss: 1.2558 - val_accuracy: 0.8686 - val_loss: 0.3965
Epoch 2/5
750/750 [=====] 552s 736ms/step - accuracy: 0.8804 - loss: 0.3773 - val_accuracy: 0.9007 - val_loss: 0.2823
Epoch 3/5
750/750 [=====] 555s 740ms/step - accuracy: 0.9074 - loss: 0.2776 - val_accuracy: 0.9122 - val_loss: 0.2591
Epoch 4/5
750/750 [=====] 544s 725ms/step - accuracy: 0.9271 - loss: 0.2208 - val_accuracy: 0.9294 - val_loss: 0.2244
Epoch 5/5
750/750 [=====] 568s 757ms/step - accuracy: 0.9386 - loss: 0.1842 - val_accuracy: 0.9208 - val_loss: 0.2253
WARNING:absl:You are saving your model as an HDF5 file via "model.save()" or "keras.saving.save_model(model)". This file format is considered legacy. We recommend using instead the native keras
'my_model.keras' or 'keras.saving.save_model(model, "my_model.keras")'.

```

## Usa del modelo

Se activa la webcam con OpenCV (cv2.VideoCapture). Al presionar 's' se toma una foto, se clasifica la imagen y se muestra en pantalla con la clase predicha.



## Guardado y reutilización del modelo

Una vez entrenado, el modelo se guarda como modelo\_clasificacion\_transfer.h5.

## Código

```

import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
img_to_array
from tensorflow.keras.models import load_model
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model

# ===== CONFIGURACIÓN =====
ruta_dataset = 'dataset'
train_dir = os.path.join(ruta_dataset, 'Train_Set_Folder')
val_dir = os.path.join(ruta_dataset, 'Validation_Set_Folder')

```

```

test_dir = os.path.join(ruta_dataset, 'Test_Set_Folder')
img_size = (224, 224)
batch_size = 32
modelo_path = 'modelo_clasificacion_transfer.h5'

# ====== CARGAR O ENTRENAR MODELO ======
if os.path.exists(modelo_path):
    print("Cargando modelo preentrenado...")
    model = load_model(modelo_path)
else:
    print("Entrenando modelo...")

    train_gen = ImageDataGenerator(rescale=1./255, rotation_range=20,
zoom_range=0.2, horizontal_flip=True)
    val_gen = ImageDataGenerator(rescale=1./255)
    test_gen = ImageDataGenerator(rescale=1./255)

    train_data = train_gen.flow_from_directory(train_dir,
target_size=img_size, batch_size=batch_size, class_mode='categorical')
    val_data = val_gen.flow_from_directory(val_dir, target_size=img_size,
batch_size=batch_size, class_mode='categorical')
    test_data = test_gen.flow_from_directory(test_dir,
target_size=img_size, batch_size=batch_size, class_mode='categorical')

    base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
    base_model.trainable = False # No entrenamos MobileNetV2

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    predictions = Dense(train_data.num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

    model.fit(train_data, validation_data=val_data, epochs=5)
    model.save(modelo_path)
    print(f"Modelo guardado en {modelo_path}")

# ====== CÁMARA Y PREDICCIÓN ======
train_gen = ImageDataGenerator(rescale=1./255)
train_data = train_gen.flow_from_directory(train_dir,
target_size=img_size, batch_size=batch_size, class_mode='categorical')

```

```
class_indices = train_data.class_indices
clases = [k for k, v in sorted(class_indices.items(), key=lambda item: item[1])]

cap = cv2.VideoCapture(0)
print("🎥 Cámara activa. Presiona 's' para tomar una foto. Presiona 'q' para salir.")

while True:
    ret, frame = cap.read()
    if not ret:
        break

    cv2.imshow('Webcam', frame)
    key = cv2.waitKey(1)

    if key == ord('s'):
        frame_resized = cv2.resize(frame, img_size)
        img_array = img_to_array(frame_resized) / 255.0
        img_array = np.expand_dims(img_array, axis=0)

        prediction = model.predict(img_array)
        pred_index = np.argmax(prediction)
        pred_class = clases[pred_index]
        print(f"📸 Clase predicha: {pred_class}")

        cv2.putText(frame, f"{pred_class}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0), 2)
        cv2.imshow('Resultado', frame)
        cv2.waitKey(2000)

    elif key == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```