# **Documentación del Modelo de Clasificación de Productos (Usando IA)**

## **1️. Objetivo**

Crear un modelo de inteligencia artificial capaz de clasificar imágenes de productos electrónicos y de oficina en tres estados: **nuevo**, **usado** y **mal\_estado**, para integrarlo posteriormente en la app de inventario.

## **2️. Preparación de Datos**

### **2.1 Recolección de imágenes**

* Se recolectaron imágenes de productos como:  
  + Monitores
  + Computadores de torre
  + Celulares
  + Impresoras
  + Toners de impresoras
* Cada imagen se clasificó manualmente en una de las tres clases: **nuevo, usado, mal\_estado**.

### **2.2 Organización de carpetas**

Estructura del dataset\_inicial:

dataset\_ready/

├─ train/

│ ├─ nuevo/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

│ ├─ usado/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

│ └─ mal\_estado/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

├─ val/

│ ├─ nuevo/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

│ ├─ usado/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

│ └─ mal\_estado/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

└─ test/

│ ├─ nuevo/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

│ ├─ usado/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

│ └─ mal\_estado/

│ │ ├─ impresora/

│ │ ├─ monitor/

│ │ ├─ smartphone/

│ │ ├─ toner/

│ │ └─ torre/

* Cada carpeta contiene las imágenes correspondientes a la clase.
* Se recomienda un número mínimo de 100 imágenes por clase para un entrenamiento efectivo.

### **2.3 Data Augmentation**

Para mejorar la generalización del modelo:

* Rotación aleatoria
* Zoom
* Cambio de brillo
* Recorte o recorte aleatorio (cropping)

Se implementa usando ImageDataGenerator de Keras.

**(Para el desarrollo de los puntos 1 y 2 se utilizaron script para poder automatizar y agilizar el desarrollo de estas. Scripts: descarga\_dataset.py y preparar\_dataset.py)**

## **3️. Creación del Modelo**

### **3.1 Modelo base**

Se utilizó **MobileNetV2** como base del modelo por ser una red ligera y eficiente, preentrenada con **ImageNet**, lo que permite que el modelo ya tenga conocimiento general de imágenes antes de entrenar con tus datos específicos.

from tensorflow.keras.applications import MobileNetV2  
base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

* weights='imagenet': usa pesos preentrenados.
* include\_top=False: descartamos la capa de clasificación original, porque añadiremos nuestra propia "cabeza".
* input\_shape=(224, 224, 3): todas las imágenes se redimensionan a 224x224 con 3 canales de color (RGB).

### **3.2 Cabeza del modelo**

Se añadieron **capas densas** sobre MobileNetV2 para transformar las características extraídas en predicciones de tus 3 clases: nuevo, usado y mal\_estado.

from tensorflow.keras import layers, Model

x = layers.GlobalAveragePooling2D()(base\_model.output)

x = layers.Dense(128, activation='relu')(x)

output = layers.Dense(3, activation='softmax')(x)

model = Model(inputs=base\_model.input, outputs=output)

* GlobalAveragePooling2D(): convierte los mapas de características 2D en un vector 1D, reduciendo el número de parámetros.
* Dense(128, activation='relu'): capa intermedia para aprender combinaciones de características.
* Dense(3, activation='softmax'): capa final que genera probabilidades para cada clase.

### **3.3 Compilación**

Antes de entrenar, se define cómo el modelo aprenderá:

model.compile(

optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy']

)

* **Optimizador:** adam (ajusta automáticamente las tasas de aprendizaje).
* **Función de pérdida:** categorical\_crossentropy (adecuada para clasificación múltiple).
* **Métrica:** accuracy (precisión del modelo durante entrenamiento y validación).

## **4️. Preparación del Dataset para Entrenamiento**

Se usan **generadores de imágenes** de Keras para alimentar el modelo en lotes, aplicando **preprocesamiento** y **data augmentation**:

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=20,

zoom\_range=0.2,

brightness\_range=[0.8,1.2],

validation\_split=0.2

)

train\_generator = train\_datagen.flow\_from\_directory(

'dataset',

target\_size=(224,224),

batch\_size=32,

class\_mode='categorical',

subset='training'

)

val\_generator = train\_datagen.flow\_from\_directory(

'dataset',

target\_size=(224,224),

batch\_size=32,

class\_mode='categorical',

subset='validation'

)

* rescale=1./255: normaliza los pixeles entre 0 y 1.
* rotation\_range, zoom\_range, brightness\_range: aumentan la variedad de datos para evitar sobreajuste.
* validation\_split=0.2: 20% de los datos se usan para validar el modelo.

## **5️. Entrenamiento del Modelo**

history = model.fit(train\_generator, validation\_data=val\_generator, epochs=10)

* Se entrena durante 10 epochs.
* history guarda información sobre pérdida y precisión durante el entrenamiento.

## **6️. Guardar el Modelo**

model.save('modelo\_final.h5')

print("✅ Modelo entrenado y guardado como modelo\_final.h5")

* El archivo modelo\_final.h5 contiene todo el modelo entrenado y listo para ser usado en la app.

## **7️. Uso del Modelo para Predicciones**

from tensorflow.keras.models import load\_model

import numpy as np

from tensorflow.keras.preprocessing import image

model = load\_model("modelo\_final.h5")

img\_path = "ruta\_a\_la\_imagen.jpg"

img = image.load\_img(img\_path, target\_size=(224, 224))

img\_array = image.img\_to\_array(img) / 255.0

img\_array = np.expand\_dims(img\_array, axis=0)

pred = model.predict(img\_array)

clases = ["nuevo", "usado", "mal\_estado"]

print("Predicción:", clases[np.argmax(pred)])

* Se carga el modelo ya entrenado.
* La imagen se redimensiona y normaliza igual que durante el entrenamiento.
* np.argmax(pred) devuelve la clase con mayor probabilidad.

## **8️. Integración en la App**

* Para apps móviles (Ionic React, Flutter, etc.), se recomienda convertir el modelo a **TensorFlow Lite**:

import tensorflow as tf

converter = tf.lite.TFLiteConverter.from\_keras\_model(model)

tflite\_model = converter.convert()

with open('modelo\_final.tflite', 'wb') as f:

f.write(tflite\_model)

* Esto permite ejecutar la IA **directamente en el dispositivo** sin depender de un servidor.

## **9️. Notas finales**

* Asegúrate de mantener las mismas dimensiones y normalización de imágenes en tu app.
* Para mejorar el modelo en el futuro:  
  + Agregar más imágenes por clase
  + Ajustar hiperparámetros (learning rate, número de capas)
  + Aplicar técnicas avanzadas de data augmentation