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

```

```
Mounted at /content/drive
```

```python
import os
import random
import numpy as np
import cv2
import xml.etree.ElementTree as ET
import chardet

# === 1. Descargar dataset desde Kaggle ===
import kagglehub
path = kagglehub.dataset_download("abhyudaya12/veri-vehicle-re-identification-dataset")
image_dir = os.path.join(path, "VeRi", "image_train")

# === 2. Leer etiquetas del archivo XML ===
label_path = os.path.join(path, "VeRi", "train_label.xml")
with open(label_path, "rb") as f:
    raw_data = f.read()
    encoding = chardet.detect(raw_data)['encoding']
xml_content = raw_data.decode(encoding)
root = ET.fromstring(xml_content)

# === 3. Crear lista de tuplas (imagen, tipo, color) ===
labeled_images = []
for item in root.findall('.//Item'):
    name = item.get('imageName')
    type_id = int(item.get('typeID'))
    color_id = int(item.get('colorID'))
    labeled_images.append((name, type_id, color_id))

# === 4. Seleccionar aleatoriamente 500 imágenes originales y 500 filtradas (125 por filtro) ===
random.shuffle(labeled_images)
originals = labeled_images[:500]
filtered_groups = {
    "mean": labeled_images[500:625],
    "median": labeled_images[625:750],
    "gaussian": labeled_images[750:875],
    "sharpened": labeled_images[875:1000]
}

# === 5. Funciones de imagen ===
def resize_image(img, size=(224, 224)):
    return cv2.resize(img, size)

def apply_filter(img, kind):
    if kind == "mean":
        return cv2.blur(img, (5, 5))
    elif kind == "median":
        return cv2.medianBlur(img, 5)
    elif kind == "gaussian":
        return cv2.GaussianBlur(img, (5, 5), 0)
    elif kind == "sharpened":
        kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
        return cv2.filter2D(img, -1, kernel)
    else:
        return img

# === 6. Construir dataset correctamente alineado ===
X = []
y_type = []
y_color = []
image_names = []

# Originales
for name, type_id, color_id in originals:
    img_path = os.path.join(image_dir, name)
    img = cv2.imread(img_path)
    if img is not None:
        img = resize_image(img)
        X.append(img)
        y_type.append(type_id)
        y_color.append(color_id)
```

```
        y_color.append(color_id)
        image_names.append(name)

# Filtradas
for kind, group in filtered_groups.items():
    for name, type_id, color_id in group:
        img_path = os.path.join(image_dir, name)
        img = cv2.imread(img_path)
        if img is not None:
            img = resize_image(img)
            img = apply_filter(img, kind)
            X.append(img)
            y_type.append(type_id)
            y_color.append(color_id)
            image_names.append(f"{name} [{kind}]")  # marcar que es una versión filtrada

# === 7. Guardar dataset final ===
X = np.array(X)
y_type = np.array(y_type)
y_color = np.array(y_color)
image_names = np.array(image_names)

np.savez_compressed("/content/drive/MyDrive/vehiculo_dataset_corregido.npz", X=X, y_type=y_type, y_color=y_color, image_names=image_names
```

```
import numpy as np

data = np.load("/content/drive/MyDrive/vehiculo_dataset_corregido.npz")
X = data["X"]
y_type = data["y_type"]
y_color = data["y_color"]

print("✅ Dataset cargado:", X.shape, y_type.shape, y_color.shape)
```

```
✅ Dataset cargado: (1000, 224, 224, 3) (1000,) (1000,)
```

```
# === 1. Cargar dataset ===
import numpy as np

data = np.load("/content/drive/MyDrive/vehiculo_dataset_mixto.npz", allow_pickle=True)
X = data["X"]
y_type = data["y_type"]
y_color = data["y_color"]

print("✅ Dataset cargado:", X.shape, y_type.shape, y_color.shape)

# === 2. Preprocesamiento ===
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# Normalizar imágenes
X = X.astype("float32") / 255.0

# Ajustar etiquetas para que empiecen desde 0
y_type_adjusted = y_type - 1
y_color_adjusted = y_color - 1

# One-hot encoding
num_type_classes = len(np.unique(y_type_adjusted))   # 9
num_color_classes = len(np.unique(y_color_adjusted)) # 10

y_type_cat = to_categorical(y_type_adjusted, num_classes=num_type_classes)
y_color_cat = to_categorical(y_color_adjusted, num_classes=num_color_classes)

# Generar índices para rastrear sincronización
indices_all = np.arange(len(X))

# División entrenamiento / validación + rastreo de índice original
X_train, X_val, y_type_train, y_type_val, y_color_train, y_color_val, idx_train, idx_val = train_test_split(
    X, y_type_cat, y_color_cat, indices_all, test_size=0.2, random_state=42
)


```

```
✅ Dataset cargado: (1000, 224, 224, 3) (1000,) (1000,)
```

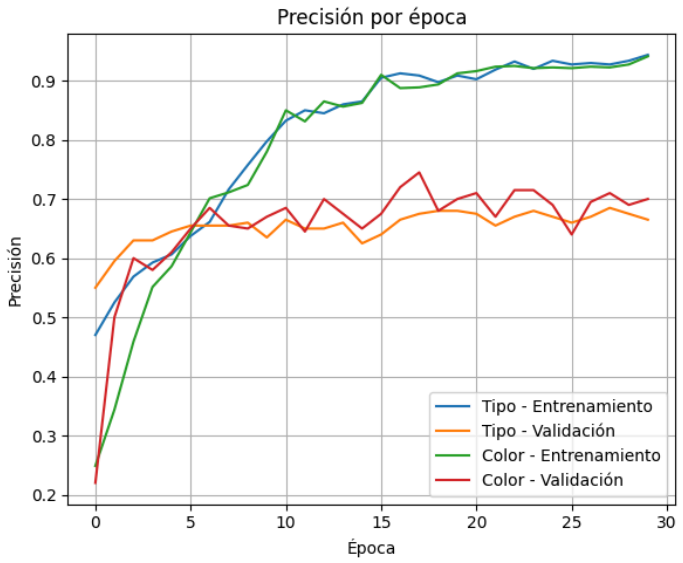
```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Entrada
input_img = Input(shape=(224, 224, 3))

# Bloques convolucionales
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D()(x)

# Capa densa común
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)

# Salida 1: tipo de vehículo (9 clases)
type_output = Dense(num_type_classes, activation='softmax', name='type_output')(x)

# Salida 2: color del vehículo (10 clases)
color_output = Dense(num_color_classes, activation='softmax', name='color_output')(x)

# Modelo final
model = Model(inputs=input_img, outputs=[type_output, color_output])

# Compilar
model.compile(
    optimizer='adam',
    loss={
        'type_output': 'categorical_crossentropy',
        'color_output': 'categorical_crossentropy'
    },
    metrics={
        'type_output': 'accuracy',
        'color_output': 'accuracy'
    }
)

```

```python
history = model.fit(
    X_train,
    {'type_output': y_type_train, 'color_output': y_color_train},
    validation_data=(X_val, {'type_output': y_type_val, 'color_output': y_color_val}),
    epochs=30,
    batch_size=32
)

```

```
Epoch 1/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 7s 130ms/step - color_output_accuracy: 0.2135 - color_output_loss: 2.4112 - loss: 4.6773 - type_output_acc
Epoch 2/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - color_output_accuracy: 0.3041 - color_output_loss: 1.9190 - loss: 3.5381 - type_output_accu
Epoch 3/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - color_output_accuracy: 0.4634 - color_output_loss: 1.5615 - loss: 3.0487 - type_output_accu
Epoch 4/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 70ms/step - color_output_accuracy: 0.5277 - color_output_loss: 1.3529 - loss: 2.7326 - type_output_accu
Epoch 5/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - color_output_accuracy: 0.5700 - color_output_loss: 1.2421 - loss: 2.5010 - type_output_accu
Epoch 6/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - color_output_accuracy: 0.6203 - color_output_loss: 1.0710 - loss: 2.1989 - type_output_accu
Epoch 7/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - color_output_accuracy: 0.6943 - color_output_loss: 0.8645 - loss: 1.8653 - type_output_accu
Epoch 8/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 68ms/step - color_output_accuracy: 0.7013 - color_output_loss: 0.8148 - loss: 1.6331 - type_output_accu
Epoch 9/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - color_output_accuracy: 0.7484 - color_output_loss: 0.6734 - loss: 1.3776 - type_output_accu
Epoch 10/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - color_output_accuracy: 0.7934 - color_output_loss: 0.5833 - loss: 1.1444 - type_output_accu
Epoch 11/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - color_output_accuracy: 0.8399 - color_output_loss: 0.4716 - loss: 0.9591 - type_output_accu
Epoch 12/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - color_output_accuracy: 0.8199 - color_output_loss: 0.4368 - loss: 0.9086 - type_output_accu
Epoch 13/30
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - color_output_accuracy: 0.8482 - color_output_loss: 0.3866 - loss: 0.8014 - type_output_accu
Epoch 14/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - color_output_accuracy: 0.8640 - color_output_loss: 0.3836 - loss: 0.7546 - type_output_accu
Epoch 15/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - color_output_accuracy: 0.8759 - color_output_loss: 0.3431 - loss: 0.6743 - type_output_accu
Epoch 16/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - color_output_accuracy: 0.9001 - color_output_loss: 0.2844 - loss: 0.5629 - type_output_accu
Epoch 17/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - color_output_accuracy: 0.9081 - color_output_loss: 0.2532 - loss: 0.5350 - type_output_accu
Epoch 18/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - color_output_accuracy: 0.8911 - color_output_loss: 0.2937 - loss: 0.5638 - type_output_accu
Epoch 19/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - color_output_accuracy: 0.9176 - color_output_loss: 0.2278 - loss: 0.4484 - type_output_accu
Epoch 20/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - color_output_accuracy: 0.9078 - color_output_loss: 0.2572 - loss: 0.5054 - type_output_accu
Epoch 21/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 64ms/step - color_output_accuracy: 0.9183 - color_output_loss: 0.2080 - loss: 0.4769 - type_output_accu
Epoch 22/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - color_output_accuracy: 0.9237 - color_output_loss: 0.1908 - loss: 0.4275 - type_output_accu
Epoch 23/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - color_output_accuracy: 0.9277 - color_output_loss: 0.2033 - loss: 0.4081 - type_output_accu
Epoch 24/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - color_output_accuracy: 0.9093 - color_output_loss: 0.2391 - loss: 0.4823 - type_output_accu
Epoch 25/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - color_output_accuracy: 0.9334 - color_output_loss: 0.2154 - loss: 0.3675 - type_output_accu
Epoch 26/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - color_output_accuracy: 0.9192 - color_output_loss: 0.1949 - loss: 0.3949 - type_output_accu
Epoch 27/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - color_output_accuracy: 0.9215 - color_output_loss: 0.1972 - loss: 0.4014 - type_output_accu
Epoch 28/30
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - color_output_accuracy: 0.9165 - color_output_loss: 0.2478 - loss: 0.4348 - type_output_accu
```

```python
import matplotlib.pyplot as plt

# === 1. Accuracy ===
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['type_output_accuracy'], label='Tipo - Entrenamiento')
plt.plot(history.history['val_type_output_accuracy'], label='Tipo - Validación')
plt.plot(history.history['color_output_accuracy'], label='Color - Entrenamiento')
plt.plot(history.history['val_color_output_accuracy'], label='Color - Validación')
plt.title('Precisión por época')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)

# === 2. Loss ===
plt.subplot(1, 2, 2)
plt.plot(history.history['type_output_loss'], label='Tipo - Entrenamiento')
plt.plot(history.history['val_type_output_loss'], label='Tipo - Validación')
plt.plot(history.history['color_output_loss'], label='Color - Entrenamiento')
plt.plot(history.history['val_color_output_loss'], label='Color - Validación')
plt.title('Pérdida por época')
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

```




```

# ✅ Descargar y preparar dataset
path = kagglehub.dataset_download("abhyudaya12/veri-vehicle-re-identification-dataset")
image_dir = os.path.join(path, "VeRi", "image_train")
label_path = os.path.join(path, "VeRi", "train_label.xml")

with open(label_path, "rb") as f:
    raw_data = f.read()
    encoding = chardet.detect(raw_data)['encoding']
xml_content = raw_data.decode(encoding)
root = ET.fromstring(xml_content)

# ✅ Diccionarios oficiales
type_labels = {
    1: 'Sedan', 2: 'SUV', 3: 'Van', 4: 'Hatchback', 5: 'MPV',
    6: 'Pickup', 7: 'Bus', 8: 'Truck', 9: 'Estate'
}
color_labels = {
    1: 'Yellow', 2: 'Orange', 3: 'Green', 4: 'Gray', 5: 'Red',
    6: 'Blue', 7: 'White', 8: 'Golden', 9: 'Brown', 10: 'Black'
}

# ✅ Lista (nombre, tipo, color)
items = []
for item in root.findall('.//Item'):
    name = item.get('imageName')
    type_id = int(item.get('typeID'))
    color_id = int(item.get('colorID'))
    items.append((name, type_id, color_id))

# ✅ Seleccionar imágenes para prueba visual
sample = random.sample(items, 1)

plt.figure(figsize=(20, 8))
for i, (name, type_id, color_id) in enumerate(sample):
    img_path = os.path.join(image_dir, name)
    img = cv2.imread(img_path)
    if img is None:
        continue
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, (224, 224))
    input_img = np.expand_dims(img_resized / 255.0, axis=0)

    # 🔍 Predicción
    pred_type_prob, pred_color_prob = model.predict(input_img, verbose=0)
    pred_type_id = np.argmax(pred_type_prob[0]) + 1  # sumamos 1 porque restamos antes
    pred_color_id = np.argmax(pred_color_prob[0]) + 1

    # 🖼️ Visualización
    plt.subplot(2, 5, i + 1)
```

```
    plt.imshow(img_rgb)
    plt.axis('off')
    plt.title(
        f"{name}\n"
        f"🟢 Real: {type_labels[type_id]}, {color_labels[color_id]}\n"
        f"🔵 Pred: {type_labels[pred_type_id]}, {color_labels[pred_color_id]}",
        fontsize=10
    )

plt.tight_layout()
plt.show()


```

```
<ipython-input-52-52d0a16886c3>:59: UserWarning: Glyph 128994 (\N{LARGE GREEN CIRCLE}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
<ipython-input-52-52d0a16886c3>:59: UserWarning: Glyph 128309 (\N{LARGE BLUE CIRCLE}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
```