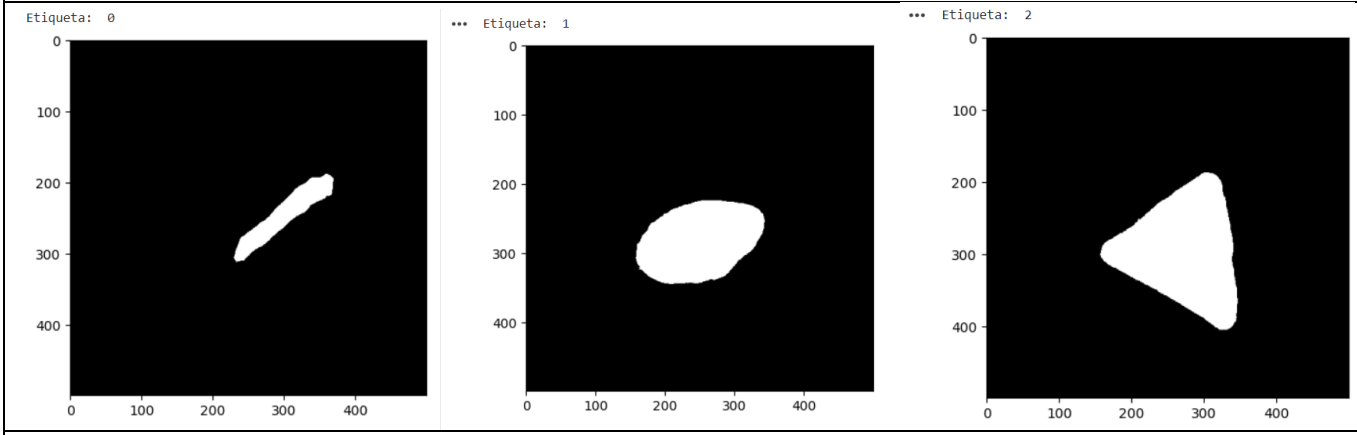
Reconocimiento de patrones – Reto 2

Resumen

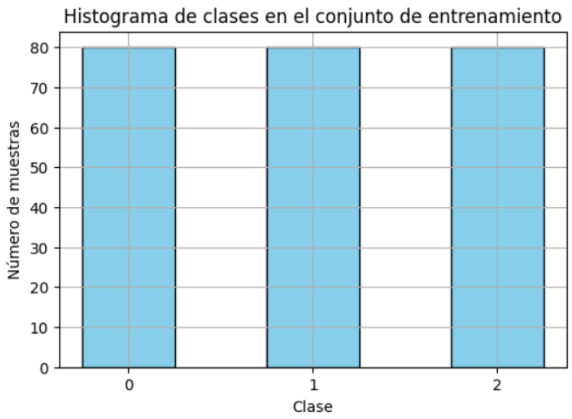
El Reto 2 consiste en crear un sistema de clasificación de imágenes para reconocer tres tipos de aperitivos: Doritos, Papitas y Cheese Tris. Solo se permitió el uso de características geométricas como el área, perímetro, dimensiones, circularidad, elipticidad, centros de masa y momentos de Hu el desarrollo del ejercicio. Como método de preprocesamiento, las imágenes se convirtieron en imágenes binarias mediante un filtro (Saturation Hue > 75) antes de utilizar las características geométricas para entrenar un modelo de clasificación KNN.

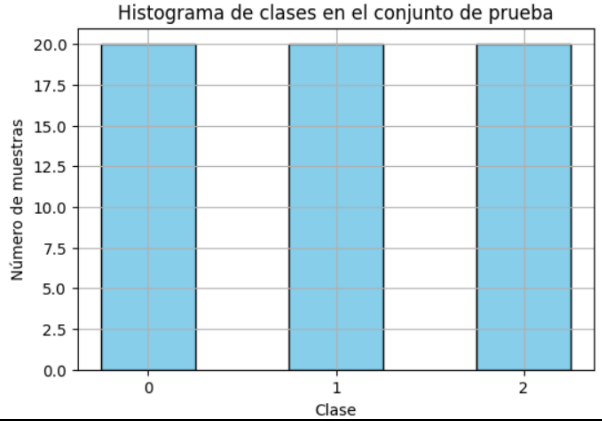
Pantallazos del trabajo

Filtrado y etiquetado de las imágenes



Reporte de datos designados para entrenamiento y testeo





Precisión del Modelo

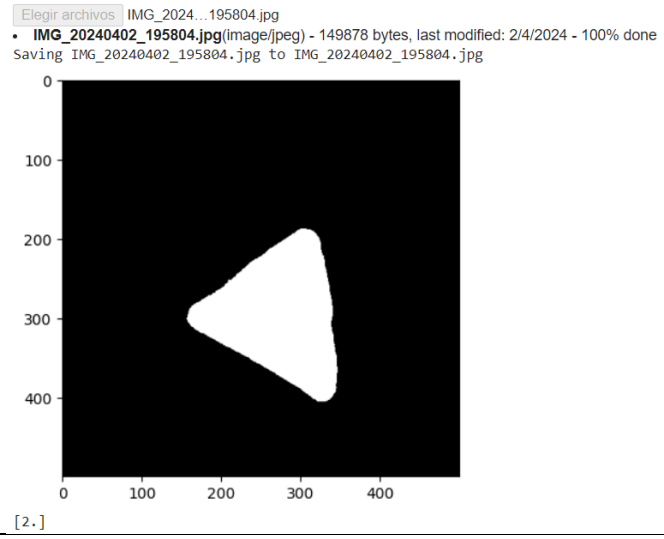
```python
#Métricas de evaluación
from sklearn.metrics import accuracy_score
Y_pred = Modelo_knn.predict(X_test_normalized)

# Calcular la precisión del modelo
accuracy = accuracy_score(Y_test, Y_pred)
print("Precisión del modelo KNN:", accuracy)
```

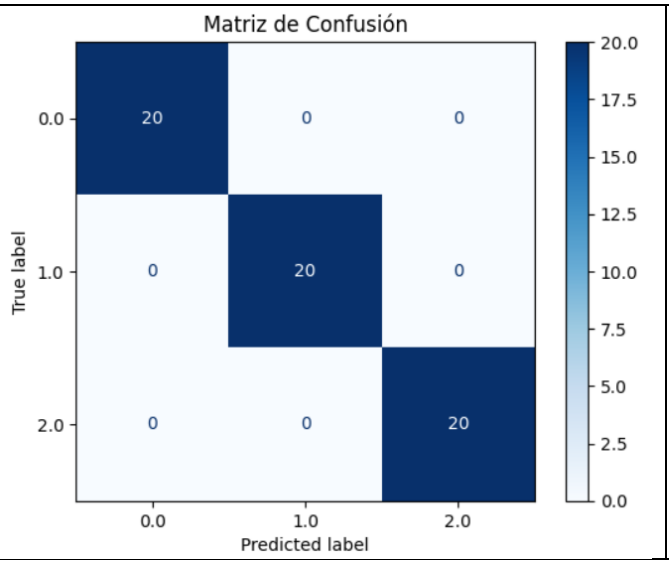
```
Precisión del modelo KNN: 1.0
```

Imagen para la evaluación del Modelo

Predicción exitosa (Etiqueta 2 = Dorito)



Métricas de desempeño




```
==================

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        20
         1.0       1.00      1.00      1.00        20
         2.0       1.00      1.00      1.00        20

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60

==================
```

Código

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

import matplotlib.pyplot as plt
import cv2
from google.colab import output
import numpy as np
import os
import time

Ruta = "/content/drive/MyDrive/Semestre 8/ReconocimientoDePatrones/Reto 2/Dataset"

imagenes = []
etiquetas = []

clases = sorted(os.listdir(Ruta))
```

```
for i, label in enumerate(clases):
  carpeta_clase = os.path.join(Ruta, label)
  for archivo in os.listdir(carpeta_clase):
    Ruta_img = os.path.join(carpeta_clase, archivo)

    output.clear()
    Img=cv2.imread(Ruta_img)
    ImgHsv = cv2.cvtColor(Img,cv2.COLOR_BGR2HSV)
    Saturation = ImgHsv[:,:,1]
    Bin = Saturation > 75

    Bin = cv2.resize(np.uint8(Bin), (500,500))

    imagenes.append(np.array(Bin))
    etiquetas.append(i)
```

```
import time
import matplotlib.pyplot as plt
import cv2
from google.colab import output
import numpy as np

X = np.zeros((len(imagenes),15)) # Vector de características geométricas
Y = np.zeros((len(imagenes),1)) # Vector de etiquetas, aunque ya se tiene, le pongo otro
nombre
for k in range(len(imagenes)):
  output.clear()

  Y[k] = etiquetas[k]
  Bin=imagenes[k]
  plt.imshow(Bin,vmin='0',vmax='1',cmap = 'gray')
  print('Etiqueta: ',Y[k])
  plt.show()
  time.sleep(0.5)

  # Encontrar los contornos de la imagen
  contours, _ = cv2.findContours(np.uint8(Bin), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

  if len(contours) > 0:
    # Tomar solo el contorno más grande (puede haber varios)
    largest_contour = max(contours, key=cv2.contourArea)

    # Calcular área
    area = cv2.contourArea(largest_contour)

    # Calcular perímetro
    perimeter = cv2.arcLength(largest_contour, closed=True)

    # Calcular el centro de masa
    M = cv2.moments(largest_contour)
```

```
    center_x = int(M['m10'] / M['m00'])
    center_y = int(M['m01'] / M['m00'])

    # Calcular la circularidad
    circularity = (4 * np.pi * area) / (perimeter ** 2)

    # Calcular la elipticidad
    _, (major_axis, minor_axis), _ = cv2.fitEllipse(largest_contour)
    ellipticity = major_axis / minor_axis

    #Calcular los momentos de Hu
    Hu_moments = np.transpose(cv2.HuMoments(M)) # Siete valores

    #Agregando alto y ancho
    P1,P2,Ancho,Alto = cv2.boundingRect(np.uint8(Bin))

    # Almacenando resultados
    X[k,0] = area
    X[k,1] = perimeter
    X[k,2] = ellipticity
    X[k,3] = center_x
    X[k,4] = center_y
    X[k,5] = circularity
    X[k,6:13] = Hu_moments
    X[k,13] = Alto
    X[k,14] = Ancho
```

```
from sklearn.model_selection import StratifiedShuffleSplit
Indices = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in Indices.split(X, Y):
    X_train, X_test = X[train_index,:], X[test_index,:]
    Y_train, Y_test = Y[train_index], Y[test_index]

# Dibujar el histograma del vector test de salida
hist_train, bins = np.histogram(Y_train, bins=[0,1,2,3])
plt.figure(figsize=(6, 4))
plt.bar(bins[:-1], hist_train, width=0.5, color='skyblue', edgecolor='black')
plt.xlabel('Clase')
plt.ylabel('Número de muestras')
plt.title('Histograma de clases en el conjunto de entrenamiento')
plt.xticks(bins[:-1])
plt.grid(True)
plt.show()

# Dibujar el histograma del vector test de salida
hist_test, bins = np.histogram(Y_test, bins=[0,1,2,3])
plt.figure(figsize=(6, 4))
plt.bar(bins[:-1], hist_test, width=0.5, color='skyblue', edgecolor='black')
plt.xlabel('Clase')
plt.ylabel('Número de muestras')
```

```
plt.title('Histograma de clases en el conjunto de prueba')
plt.xticks(bins[:-1])
plt.grid(True)
plt.show()

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier as KNN
# Inicializar el clasificador KNN
Modelo_knn = KNN(n_neighbors=3)
# Entrenar el clasificador KNN con el conjunto de entrenamiento
Modelo_knn.fit(X_train_normalized, Y_train)

#Métricas de evaluación
from sklearn.metrics import accuracy_score
Y_pred = Modelo_knn.predict(X_test_normalized)
# Calcular la precisión del modelo
accuracy = accuracy_score(Y_test, Y_pred)
print("Precisión del modelo KNN:", accuracy)

import joblib

joblib.dump(Modelo_knn, '/content/drive/MyDrive/Semestre 8/ReconocimientoDePatrones/Reto
2/Modelo_knn.pkl')

from google.colab import files
import cv2
import numpy as np

# Cargar el modelo desde el archivo
Modelo_entrenado = joblib.load('/content/drive/MyDrive/Semestre
8/ReconocimientoDePatrones/Reto 2/Modelo_knn.pkl')

# Cargar archivos desde el sistema local
uploaded = files.upload()

# Obtener el nombre del archivo cargado
Nombre_archivo = list(uploaded.keys())[0]

Imagen=cv2.imread(Nombre_archivo)

#Procesando la imagen
ImgHsv = cv2.cvtColor(Imagen,cv2.COLOR_BGR2HSV)
Saturation = ImgHsv[:,:,1]
Bin = Saturation > 75
Bin = cv2.resize(np.uint8(Bin), (500,500))

plt.imshow(Bin,vmin='0',vmax='1',cmap = 'gray')
```

```
plt.show()

#Midiendo el área de las letras (Igual que en el entrenamiento)
X_new = np.zeros((1,15))

# Encontrar los contornos de la imagen
contours, _ = cv2.findContours(np.uint8(Bin), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

if len(contours) > 0:
  # Tomar solo el contorno más grande (puede haber varios)
  largest_contour = max(contours, key=cv2.contourArea)

  # Calcular área
  area = cv2.contourArea(largest_contour)

  # Calcular perímetro
  perimeter = cv2.arcLength(largest_contour, closed=True)

  # Calcular la elipticidad
  _, (major_axis, minor_axis), _ = cv2.fitEllipse(largest_contour)
  ellipticity = major_axis / minor_axis

  # Calcular el centro de masa
  M = cv2.moments(largest_contour)
  center_x = int(M['m10'] / M['m00'])
  center_y = int(M['m01'] / M['m00'])

  # Calcular la circularidad
  circularity = (4 * np.pi * area) / (perimeter ** 2)

  #Calcular los momentos de Hu
  Hu_moments = np.transpose(cv2.HuMoments(M)) # Siete valores

  #Agregando alto y ancho
  P1,P2,Ancho,Alto = cv2.boundingRect(np.uint8(Bin))

  # Almacenando resultados
  X_new[0,0] = area
  X_new[0,1] = perimeter
  X_new[0,2] = ellipticity
  X_new[0,3] = center_x
  X_new[0,4] = center_y
  X_new[0,5] = circularity
  X_new[0,6:13] = Hu_moments
  X_new[0,13] = Alto
  X_new[0,14] = Ancho

  #Ojo, se debe normalizar
  X_new_normalized = scaler.transform(X_new)
```

```
print(Modelo_entrenado.predict(X_new_normalized))

from sklearn.model_selection import StratifiedShuffleSplit
Indices = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in Indices.split(X, Y):
    X_train, X_test = X[train_index,:], X[test_index,:]
    Y_train, Y_test = Y[train_index], Y[test_index]

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)

#Implementando un clasificador por comparación (KNN)
from sklearn.neighbors import KNeighborsClassifier as KNN

# Inicializar el clasificador KNN
Modelo_knn = KNN(n_neighbors=3)

# Entrenar el clasificador KNN con el conjunto de entrenamiento
Modelo_knn.fit(X_train_normalized, Y_train)

Y_pred = Modelo_knn.predict(X_test_normalized)

import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
import seaborn as sns

 # Calcular métricas
accuracy = Modelo_knn.score(X_test_normalized, Y_pred)
cm = confusion_matrix(Y_test, Y_pred)
report = classification_report(Y_test, Y_pred)

# Mostrar resultados
print(f"Accuracy: {accuracy:.2f}")
print('==================')
print(' ')

# Visualizar la matriz de confusión
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=Modelo_knn.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Matriz de Confusión')
plt.show()
print('==================')

print(' ')
print("Classification Report:")
print(report)
print('==================')
```