ESTUDIANTES:

ANTHONY PULLA

KELLY PALTIN

ASIGNATURA:

VISION POR COMPUTADOR

TEMA:

CLASIFICACIÓN DE OBJETOS USANDO TÉCNICAS DE DEEP LEARNING E IDENTIFICACIÓN DE FORMAS EMPLEANDO EL DESCRIPTOR SHAPE SIGNATURE (TRANSFORMADA DE FOURIER)

DOCENTE:

ING. VLADIMIR ROBLES

FECHA:

19 DE JULIO DE 2024

PERIODO ACADÉMICO:

64

PARTE 1A

• Emplear una red YOLOv10 (como se ha visto en clase) o similar a la que se haya realizado un proceso de transfer learning para reconocer 5 tipos de objetos del Ecuador con imágenes captadas con la cámara del computador. Deberá realizar pruebas de rendimiento usando GPUs vs CPU (para ello puede emplear las computadoras del laboratorio de Cómputo 8). El entrenamiento y validación de resultados de la red puede hacerlo en Google Colab o en cualquier entorno de su preferencia. Por ejemplo, podría realizar el reconocimiento de especies endémicas (iguana marina, tortugas gigantes, rana enana de colorado, colibrí de Esmeraldas y Viscacha Ecuatoriana). Se sugiere emplear imágenes que se puedan encontrar en repositorios para realizar la tarea. No puede haber ningún grupo que tenga las mismas imágenes o código que otros grupos.

Dataset utilizado es uno propio, le llamamos “Dataset Ecuador”, en ella hicimos un repositorio de animales endémicos y únicos a nuestro Ecuador y estas conforman 5 clases:

0: cóndor andino

1: iguana marina

2: lobo marino

3: pinzón de Darwin

4: tortuga gigante de galápagos

Esta dataset contiene 50 imágenes por cada clase de animal, las imágenes las dividimos en 80% para entrenamiento y 20% para validación. Luego necesitábamos las anotaciones de cada una de esas imágenes respectivamente, con el uso de ‘labelImg’, en formato YOLO, creamos los labels de las imágenes y las dividimos tal igual como para las imágenes.

Código Implementado en Python en un Entorno Local en Jupyter Notebook para el acceso a la camera de la computadora y para evaluar el rendimiento del uso del CPU:

```
import cv2

cap = cv2.VideoCapture(0)

ret, frame = cap.read()

 image_path = 'captured_image.jpg'
```

```
cv2.imwrite(image_path, frame)
 cap.release()
cv2.destroyAllWindows()
print("Imagen capturada y guardada en 'captured_image.jpg'")

!pip install ultralytics onnx matplotlib pillow requests numpy
!pip install ultralytics
!pip install dill
!pip install supervision
!pip install onnx

import zipfile

zip_file_path = 'C:/Users/USUARIO_PC/Downloads/datasetecuador.zip'
extract_path = 'C:/Users/USUARIO_PC/Downloads/datasetecuador'

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("Archivo descomprimido con éxito")

import os

train_label_dir = 'C:/Users/USUARIO_PC/Downloads/datasetecuador/labels/train/'
val_label_dir = 'C:/Users/USUARIO_PC/Downloads/datasetecuador/labels/val/'
train_image_dir = 'C:/Users/USUARIO_PC/Downloads/datasetecuador/images/train/'
val_image_dir = 'C:/Users/USUARIO_PC/Downloads/datasetecuador/images/val/'

class_mapping = {
    15: 0,
    16: 1,
```

```
    17: 2,
    18: 3,
    19: 4
}

def check_and_correct_labels(label_dir):
    for label_file in os.listdir(label_dir):
        label_path = os.path.join(label_dir, label_file)
        if label_file.endswith('.txt'):
            with open(label_path, 'r') as f:
                lines = f.readlines()

            corrected_lines = []
            for line in lines:
                parts = line.strip().split()
                class_id = int(parts[0])
                if class_id in class_mapping:
                    new_class_id = class_mapping[class_id]
                    corrected_line = ' '.join([str(new_class_id)] + parts[1:]) + '\n'
                    corrected_lines.append(corrected_line)
                    print(f"Info: Changed class {class_id} to {new_class_id} in file
{label_file}.")
                else:
                    corrected_lines.append(line)

            with open(label_path, 'w') as f:
                f.writelines(corrected_lines)

check_and_correct_labels(train_label_dir)
check_and_correct_labels(val_label_dir)

data_config = """
```

```
train: C:/Users/USUARIO_PC/Downloads/datasetecuador/images/train
val: C:/Users/USUARIO_PC/Downloads/datasetecuador/images/val

nc: 5
names: ['condorandino', 'iguanamarina', 'lobomarino', 'pinzondarwin', 'tortugagigante']
"""

with open('C:/Users/USUARIO_PC/Downloads/datasetecuador/ecuador.yaml', 'w') as f:
    f.write(data_config)

import os

train_label_dir = 'C:/Users/USUARIO_PC/Downloads/datasetecuador/labels/train/'
val_label_dir = 'C:/Users/USUARIO_PC/Downloads/datasetecuador/labels/val/'

def check_and_correct_labels(label_dir):
    for label_file in os.listdir(label_dir):
        label_path = os.path.join(label_dir, label_file)
        if label_file.endswith('.txt'):
            with open(label_path, 'r') as f:
                lines = f.readlines()

            corrected_lines = []
            for line in lines:
                parts = line.strip().split()
                class_id = int(parts[0])
                if class_id in range(5):
                    corrected_lines.append(line)
                else:
                    print(f"Warning: Invalid class {class_id} in file {label_file}. Removing this
line.")
```

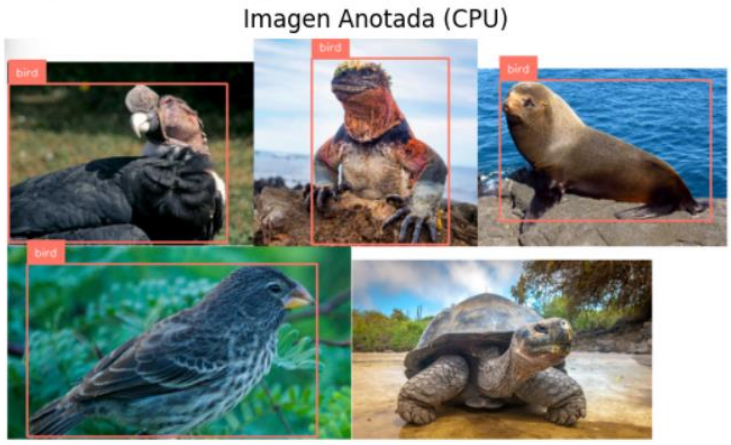
```
            with open(label_path, 'w') as f:
                f.writelines(corrected_lines)

check_and_correct_labels(train_label_dir)
check_and_correct_labels(val_label_dir)

from ultralytics import YOLO

import matplotlib.pyplot as pp
import supervision as sv
from io import BytesIO
from PIL import Image
import requests
import numpy as np
import onnx
model = YOLO('yolov10n.pt')
model.train(data='C:/Users/USUARIO_PC/Downloads/datasetecuador/ecuador.yaml',
epochs=50)

from ultralytics import YOLO
from PIL import Image
import torch
import time

model = YOLO('yolov10n.pt')

image = Image.open('captured_image.jpg')

%matplotlib inline
from ultralytics import YOLO
import matplotlib.pyplot as plt
from PIL import Image
```

```
import torch
import time
import numpy as np
import supervision as sv

device_cpu = torch.device('cpu')
print(f"Using device: {device_cpu}")

model = YOLO('yolov10n.pt').to(device_cpu)

image_path = 'captured_image.jpg'
img = Image.open(image_path).convert('RGB')
image = np.array(img)

start_time = time.time()
results_cpu = model(img)
end_time = time.time()
cpu_inference_time = end_time - start_time
print(f"Tiempo de inferencia con CPU: {cpu_inference_time:.4f} segundos")

results_cpu = results_cpu[0]
detections_cpu = sv.Detections.from_ultralytics(results_cpu)

box_annotator = sv.BoxAnnotator()
label_annotator = sv.LabelAnnotator()

annotated_image_cpu = box_annotator.annotate(scene=image,
detections=detections_cpu)
annotated_image_cpu = label_annotator.annotate(scene=annotated_image_cpu,
detections=detections_cpu)

plt.figure(figsize=(15, 7))
```

```
plt.subplot(1, 2, 2)
plt.imshow(annotated_image_cpu)
plt.axis('off')
plt.title('Imagen Anotada (CPU)')

plt.show()
```

Resultados:

```
Using device: cpu

0: 384x640 4 birds, 199.0ms
Speed: 4.0ms preprocess, 199.0ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)
Tiempo de inferencia con CPU: 0.2909 segundos
```



Código Implementado en Python en un Entorno de Google Colab para evaluar el rendimiento del uso del GPU:

```
!pip install ultralytics
!pip install dill
!pip install supervision
!pip install onnx
!pip install ultralytics onnx matplotlib pillow requests numpy

import zipfile

zip_file_path = '/content/datasetecuador.zip'
```

```
extract_path = '/content/datasetecuador/'

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

import os

train_label_dir = '/content/datasetecuador/datasetecuador/labels/train/'
val_label_dir = '/content/datasetecuador/datasetecuador/labels/val/'
train_image_dir = '/content/datasetecuador/datasetecuador/images/train/'
val_image_dir = '/content/datasetecuador/datasetecuador/images/val/'

class_mapping = {
    15: 0,
    16: 1,
    17: 2,
    18: 3,
    19: 4
}

def check_and_correct_labels(label_dir):
    for label_file in os.listdir(label_dir):
        label_path = os.path.join(label_dir, label_file)
        if label_file.endswith('.txt'):
            with open(label_path, 'r') as f:
                lines = f.readlines()

            corrected_lines = []
            for line in lines:
                parts = line.strip().split()
                class_id = int(parts[0])
```

```
            if class_id in class_mapping:
                new_class_id = class_mapping[class_id]
                corrected_line = ' '.join([str(new_class_id)] + parts[1:]) + '\n'
                corrected_lines.append(corrected_line)
                print(f"Info: Changed class {class_id} to {new_class_id} in file
{label_file}.")
            else:
                corrected_lines.append(line)

        with open(label_path, 'w') as f:
            f.writelines(corrected_lines)

check_and_correct_labels(train_label_dir)
check_and_correct_labels(val_label_dir)

data_config = """
train: /content/datasetecuador/datasetecuador/images/train
val: /content/datasetecuador/datasetecuador/images/val

nc: 5
names: ['condorandino', 'iguanamarina', 'lobomarino', 'pinzondarwin', 'tortugagigante']
"""

with open('/content/ecuador.yaml', 'w') as f:
    f.write(data_config)

import os

train_label_dir = '/content/datasetecuador/datasetecuador/labels/train/'
val_label_dir = '/content/datasetecuador/datasetecuador/labels/val/'

def check_and_correct_labels(label_dir):
```

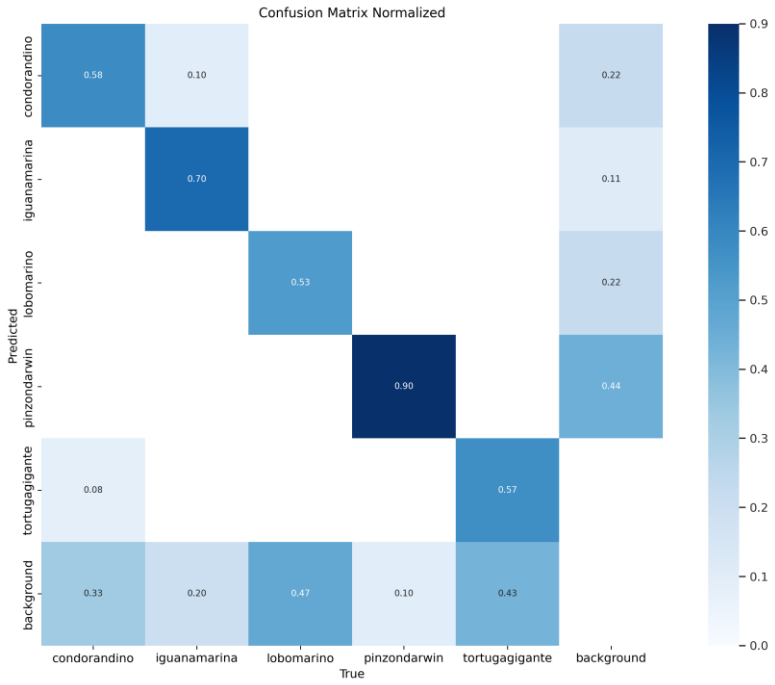
```
    for label_file in os.listdir(label_dir):
        label_path = os.path.join(label_dir, label_file)
        if label_file.endswith('.txt'):
            with open(label_path, 'r') as f:
                lines = f.readlines()

            corrected_lines = []
            for line in lines:
                parts = line.strip().split()
                class_id = int(parts[0])
                if class_id in range(5):
                    corrected_lines.append(line)
                else:
                    print(f"Warning: Invalid class {class_id} in file {label_file}. Removing this line.")

            with open(label_path, 'w') as f:
                f.writelines(corrected_lines)

check_and_correct_labels(train_label_dir)
check_and_correct_labels(val_label_dir)

from ultralytics import YOLO

import matplotlib.pyplot as pp
import supervision as sv
from io import BytesIO
from PIL import Image
import requests
import numpy as np
import onnx
model = YOLO('yolov10n.pt')
```

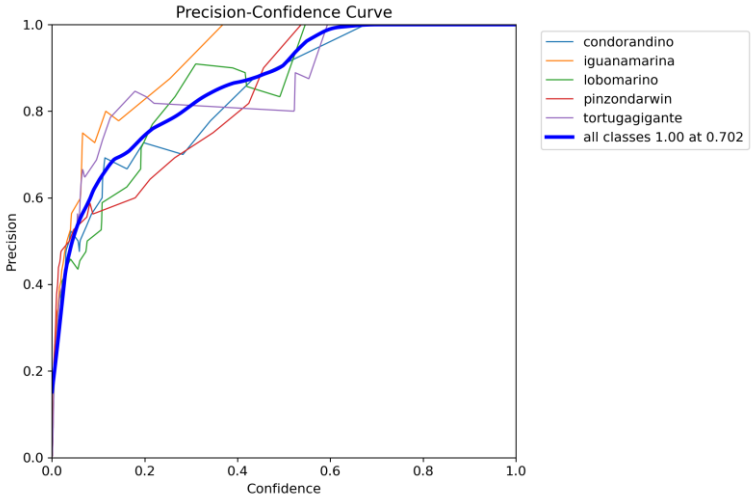
```
model.train(data='/content/ecuador.yaml', epochs=50)

from ultralytics import YOLO
import matplotlib.pyplot as plt
from PIL import Image
import torch
import time
import numpy as np
from io import BytesIO
import supervision as sv

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

model = YOLO('/content/yolov10n.pt').to(device)

image_path = '/content/captura_imagen.jpg'
img = Image.open(image_path).convert('RGB')
image = np.array(img)  # Convertir la imagen en una representación de NumPy

start_time = time.time()
results = model(img)[0]
end_time = time.time()
gpu_inference_time = end_time - start_time
print(f"Tiempo de inferencia con GPU: {gpu_inference_time:.4f} segundos")

detections = sv.Detections.from_ultralytics(results)

bounding_box_annotator = sv.BoundingBoxAnnotator()
label_annotator = sv.LabelAnnotator()
```

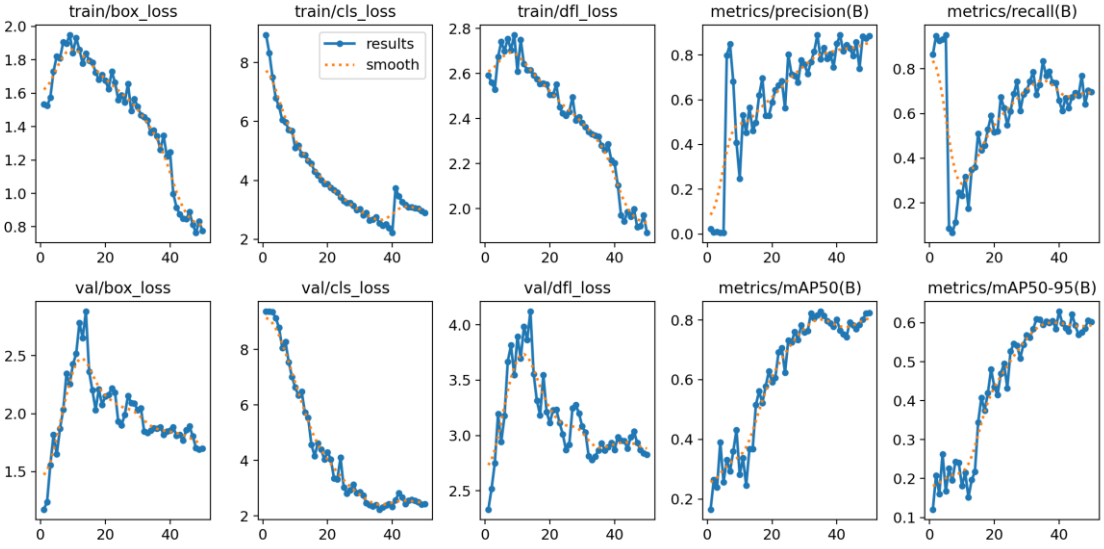
```
annotated_image = bounding_box_annotator.annotate(scene=image,
detections=detections)

annotated_image = label_annotator.annotate(scene=annotated_image,
detections=detections)


figure, axes = plt.subplots(nrows=1, ncols=2)

figure.set_size_inches(10, 7)


axes[0].imshow(image)

axes[0].axis('off')

axes[0].set_title('Imagen Original')


axes[1].imshow(annotated_image)

axes[1].axis('off')

axes[1].set_title('Imagen Anotada')


plt.show()
```

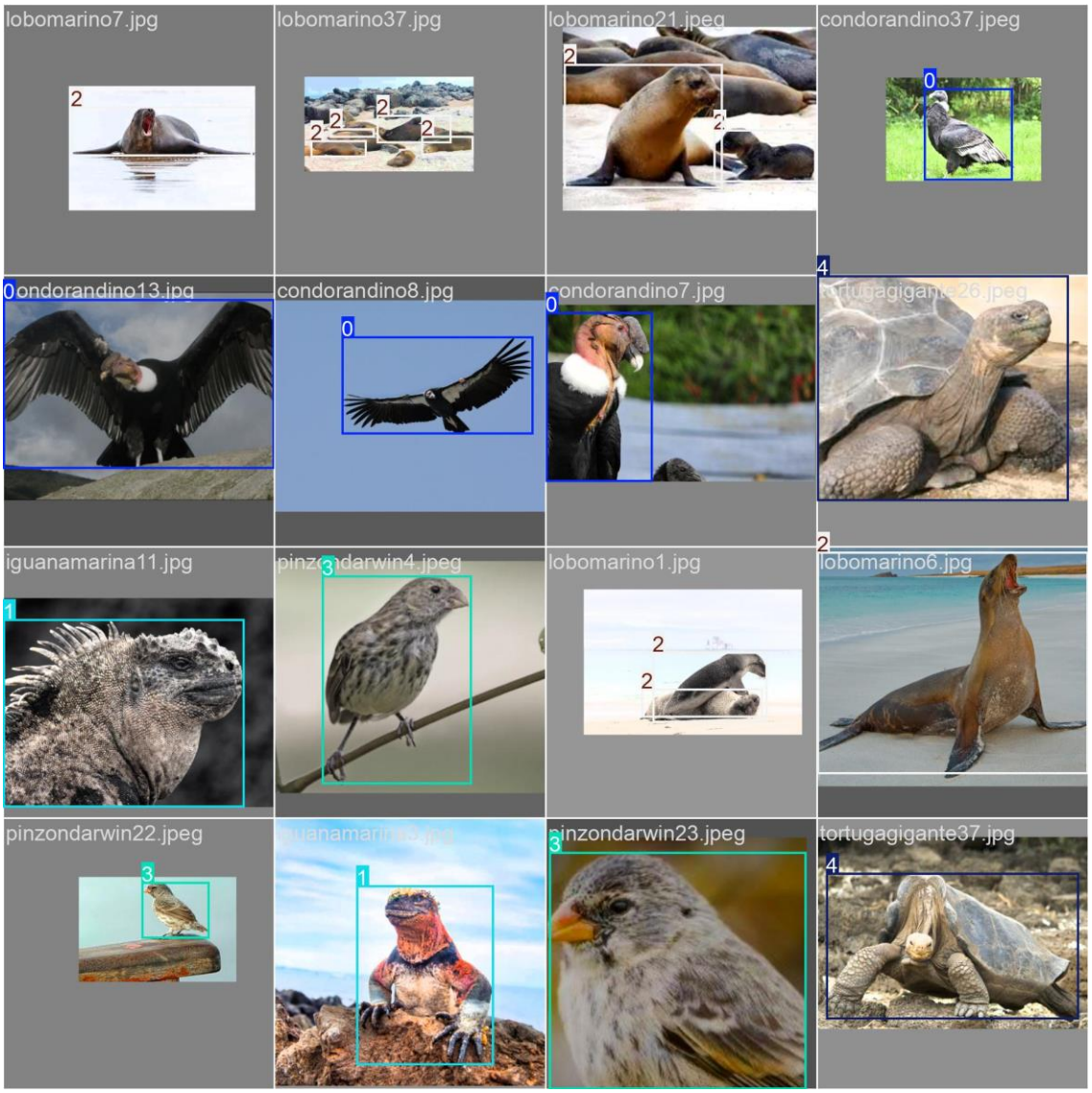
```
Using device: cuda

0: 384x640 4 birds, 9.4ms
Speed: 1.8ms preprocess, 9.4ms inference, 0.4ms postprocess per image at shape (1, 3, 384, 640)
Tiempo de inferencia con GPU: 0.1500 segundos
SupervisionWarnings: BoundingBoxAnnotator is deprecated: `BoundingBoxAnnotator` is deprecated and has been renamed to `BoxAnnotator`
```

Lo que podemos llegar a concluir es que se evidencia que las etiquetas del propio YOLOv10 ganan a las nuevas que queríamos implementar, lo cual es entendido por la cantidad de entrenamiento que tuvo cada clase, pero podemos evidenciar, dentro de la carpeta ‘runs’ que si se realiza los entrenamientos correctamente y como las validaciones realizan sus intentos. En el futuro sería de trabajar con un dataset mucho más grande con más imágenes posibles y trabajar sobre el GPU ya que sobre ella el entrenamiento en el YOLOv10 no pasa la hora (dependiendo el número de epochs).


Confusion Matrix Normalized
Predicted
True
condorandino
iguanamarina
lobomarino
pinzondarwin
tortugagigante
background
0.58
0.10
0.22
0.70
0.11
0.53
0.22
0.90
0.44
0.08
0.57
0.33
0.20
0.47
0.10
0.43
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Precision-Confidence Curve
condorandino
iguanamarina
lobomarino
pinzondarwin
tortugagigante
all classes 1.00 at 0.702
Precision
Confidence


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)



PARTE 1B

• Realizar pruebas de rendimiento de la red neuronal MobileNetv3 (como se ha visto en clase), comparando GPUs vs CPU (para ello debe emplear las computadoras del laboratorio de Cómputo 8). Para ello, deberá visual la cantidad de Frames por Segundo que se tiene en CPU frente a GPU.

• Cada grupo deberá usar un computador distinto

• Deberá grabar un vídeo del resultado que se obtiene tanto en CPU como en GPU y mostrar la siguiente información:

◦ Uso de la memoria con el comando nvidia-smi

◦ Número de FPS (frames per second) en GPU vs CPU

◦ Uso de memoria RAM en GPU vs CPU

Codigo para el fps del video:

```
while (videoCapture.read(currentFrame))
```

```
{
if (!videoWriter.isOpened())
videoWriter.open(outputFilePath, videoCodec, videoCapture.get(cv::CAP_PROP_FPS),
currentFrame.size(), true);

cv::TickMeter time;
time.start();

analyzeFrame(currentFrame);

// Stop timer and calculate FPS
time.stop();
double fps = time.getFPS();

// Draw FPS on the frame
std::string fps = "FPS: " + std::to_string(fps).substr(0, 5);
cv::putText(currentFrame, fps, cv::Point(20, 30), cv::FONT_HERSHEY_SIMPLEX, 1,
cv::Scalar(255, 255, 255), 4);

videoWriter.write(currentFrame);
cv::imshow("Result Window", currentFrame);
if (cv::waitKey(1) >= 0) break;
}
```

Link al Video Demostrativo:

https://drive.google.com/file/d/1sFjpai5YuQ1dMNt4dZcmhh_7zAI1MwGJ/view?usp=sharing

PARTE 2

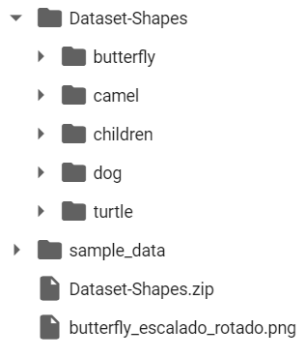
• Desarrollar un programa que obtener el descriptor Shape Signature del dataset de imágenes DatasetShapes.zip. Para ello deberá escoger 5 categorías diferentes y realizar las siguientes tareas:

1. Seleccionar 10 categorías de imágenes (ningún grupo puede tener las mismas imágenes) del corpus Dataset-Shapes.

2. Extraer el descriptor Shape Signature para todas las imágenes (tanto para las de entrenamiento como para las de pruebas). Estos descriptores deberán estar almacenados en archivos desde donde realizará la lectura y el análisis correspondiente.

3. Determinar el nivel de precisión para clasificar las imágenes de pruebas comparando el descriptor de cada imagen de entrenamiento con el descriptor de cada imagen de pruebas. Para ello, puede usar la distancia Euclídea (u otra medida de similitud que desee) vista en clase (en el tema de los momentos de HU).

En el corpus elegimos las categorías:

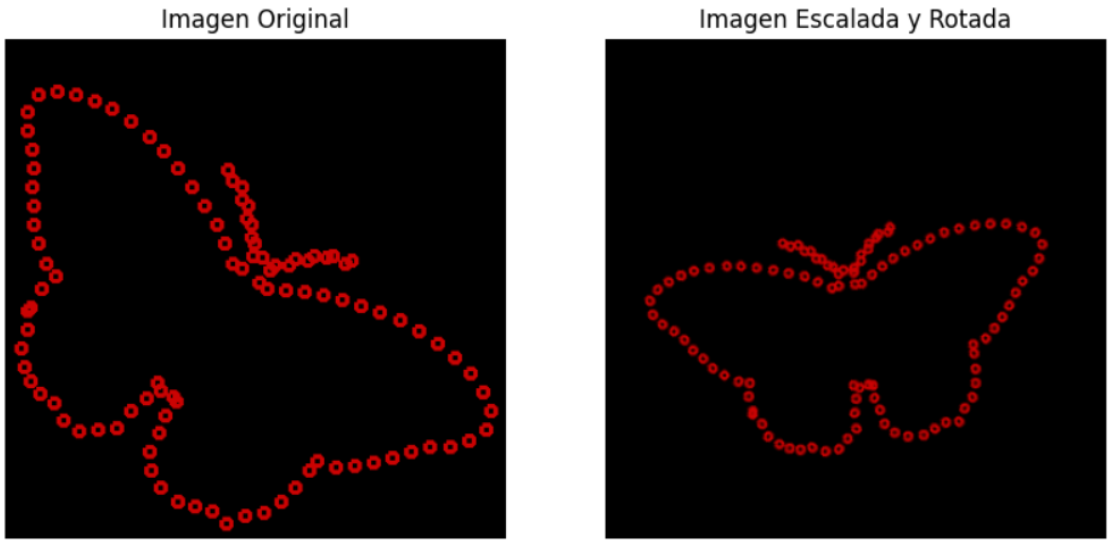


Código Implementado para la comparación entre figuras con el Shape Signature:

```
import cv2
from io import BytesIO
from PIL import Image
import requests
import numpy as np
import scipy as sp
from scipy.fftpack import fft, ifft
import zipfile
import os
import matplotlib.pyplot as plt
from io import BytesIO
import requests
zip_path = '/content/Dataset-Shapes.zip'
extract_dir = '/content'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

ruta_imagen_butterfly = '/content/Dataset-Shapes/butterfly/butterfly-1.png'
ruta_imagen_children = '/content/Dataset-Shapes/children/children-1.png'

def imagen_escalar_rotar(image_path, scale_factor=1.0, angle=0, save_path=None):
    image = cv2.imread(image_path)
    if image is None:
```

```
        raise ValueError("La imagen no se pudo cargar. Verifica la ruta y el nombre del archivo.")

    (h, w) = image.shape[:2]

    nuevo_w = int(w * scale_factor)
    nuevo_h = int(h * scale_factor)
    imagen_escalda = cv2.resize(image, (nuevo_w, nuevo_h))

    diagonal = int(np.sqrt(nuevo_w**2 + nuevo_h**2))
    nuevo_tamaño = (diagonal, diagonal)

    nuevo_centro = (diagonal // 2, diagonal // 2)

    translation_matrix = np.array([[1, 0, nuevo_centro[0] - nuevo_w // 2], [0, 1, nuevo_centro[1] - nuevo_h // 2]], dtype=np.float32)
    imagen_centrada = cv2.warpAffine(imagen_escalda, translation_matrix, nuevo_tamaño)

    M = cv2.getRotationMatrix2D(nuevo_centro, angle, 1.0)
    imagen_rotado = cv2.warpAffine(imagen_centrada, M, nuevo_tamaño)

    if guardar_ruta:
        cv2.imwrite(guardar_ruta, imagen_rotado)

    return imagen_rotado

guardar_ruta = '/content/butterfly_escalado_rotado.png'

imagen_escalar_rotar = imagen_escalar_rotar(ruta_imagen_butterfly, scale_factor=0.7, angle=45, save_path=guardar_ruta)

imagen_original = cv2.imread(ruta_imagen_butterfly)
```

```
imagen_original_rgb = cv2.cvtColor(imagen_original, cv2.COLOR_BGR2RGB)

imagen_escalado_rotado_rgb = cv2.cvtColor(imagen_escalar_rotar,
cv2.COLOR_BGR2RGB)


plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(imagen_original_rgb)

plt.title('Imagen Original')

plt.axis('off')


plt.subplot(1, 2, 2)

plt.imshow(imagen_escalado_rotado_rgb)

plt.title('Imagen Escalada y Rotada')

plt.axis('off')


plt.show()
```




```
ruta_butterfly_escalado_rotado = '/content/butterfly_escalado_rotado.png'


def contorno_centroide_figura(image_path, title_prefix):

    image = cv2.imread(image_path)
```

```
    if image is None:
        raise ValueError("La imagen no se pudo cargar. Verifica la ruta y el nombre del archivo.")

    imagen_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    gray = cv2.cvtColor(imagen_rgb, cv2.COLOR_RGB2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY_INV, 11, 2)

    contornos, jerarquia = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    area_min = 1000
    contornos_filtrados = [cnt for cnt in contornos if cv2.contourArea(cnt) > area_min]
    img_bordes = cv2.Canny(gray, 50, 150, apertureSize=3)

    img_contornos = np.zeros_like(imagen_rgb)
    if len(contornos_filtrados) > 0:
        max_contour = max(contornos_filtrados, key=cv2.contourArea)
        cv2.drawContours(img_contornos, [max_contour], -1, (0, 255, 0), 2)
        M = cv2.moments(max_contour)
        if M['m00'] != 0:
            cx = int(M['m10'] / M['m00'])
            cy = int(M['m01'] / M['m00'])
            cv2.circle(img_contornos, (cx, cy), 5, (0, 0, 255), -1)
            print(f"Centroide en: ({cx}, {cy})")
        else:
            print("No se pudo calcular el centroide correctamente (división por cero).")
    else:
        print("No se encontraron contornos.")

    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
```

```
    axes[0, 0].imshow(imagen_rgb)
    axes[0, 0].axis('off')
    axes[0, 0].set_title('Imagen Original')

    axes[0, 1].imshow(gray, cmap='gray')
    axes[0, 1].axis('off')
    axes[0, 1].set_title('Imagen en escala de grises')

    axes[1, 0].imshow(img_bordes, cmap='gray')
    axes[1, 0].axis('off')
    axes[1, 0].set_title('Bordes')

    axes[1, 1].imshow(img_contornos)
    axes[1, 1].axis('off')
    axes[1, 1].set_title('Contorno y Centroide')

    plt.show()

contorno_centroide_figura(ruta_imagen_butterfly, 'Butterfly')
contorno_centroide_figura(ruta_imagen_children, 'Children')
contorno_centroide_figura(ruta_butterfly_escalado_rotado, 'Butterfly Esclada y
Rotada')
```

```
Centroide en: (132, 185)
```

Imagen Original

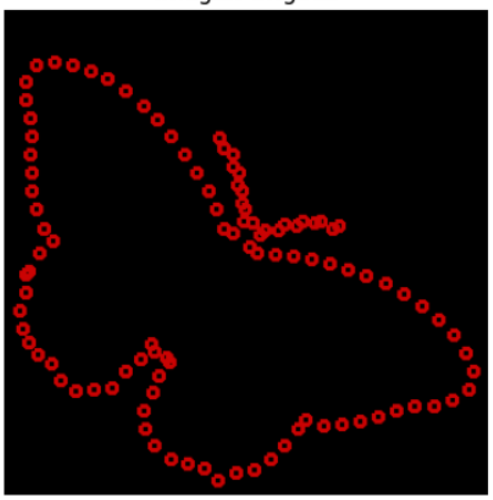
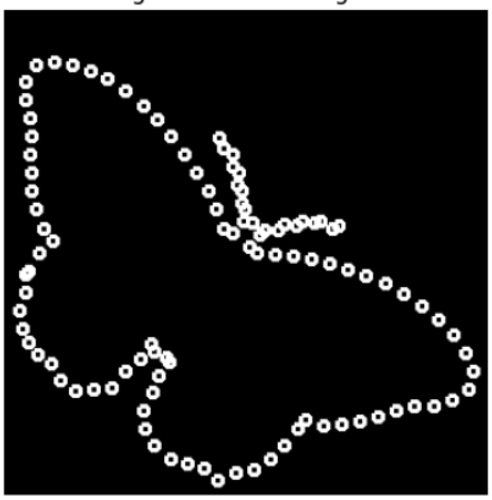
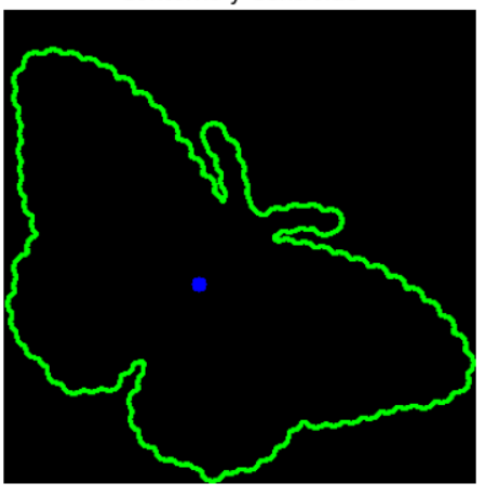


Imagen en escala de grises



Bordes

Contorno y Centroide



```
Centroide en: (60, 156)
```

Imagen Original

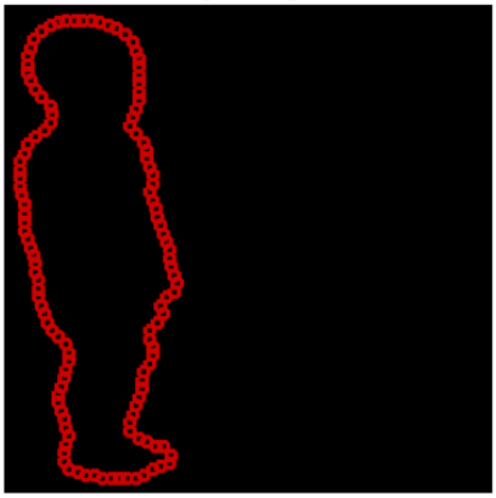
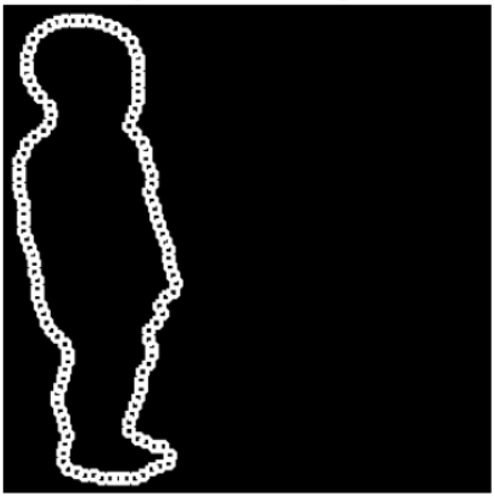
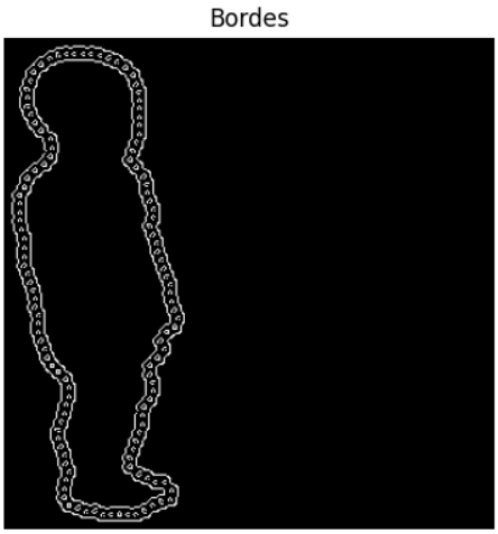
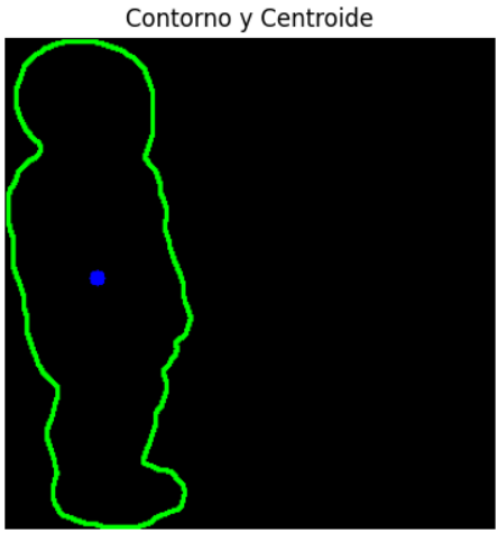


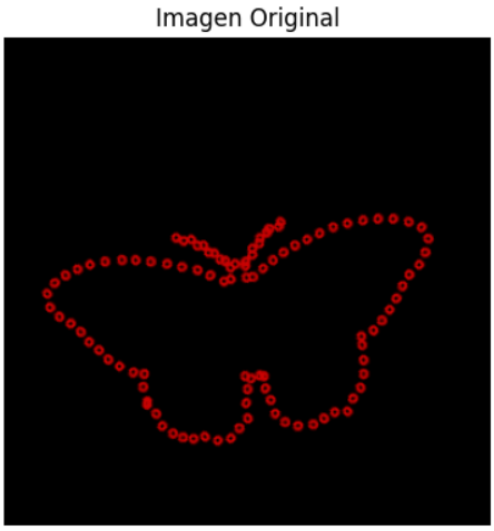
Imagen en escala de grises

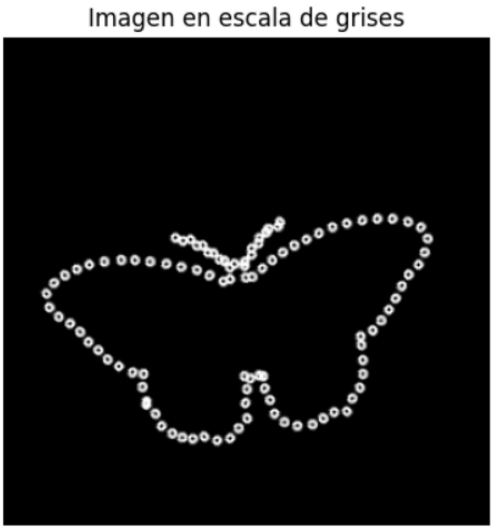


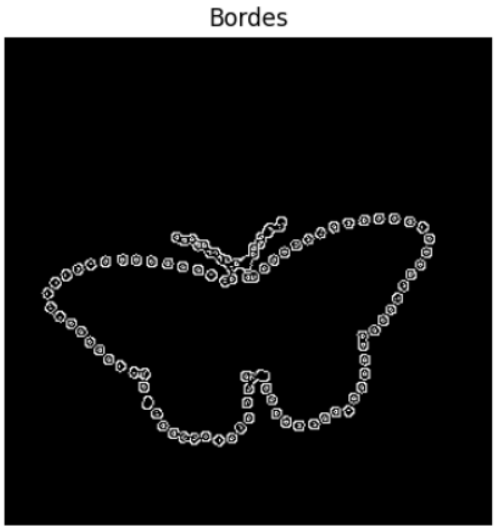


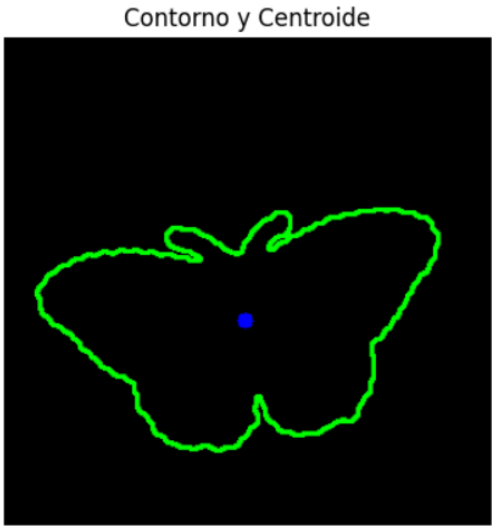



```
Centroide en: (156, 183)
```







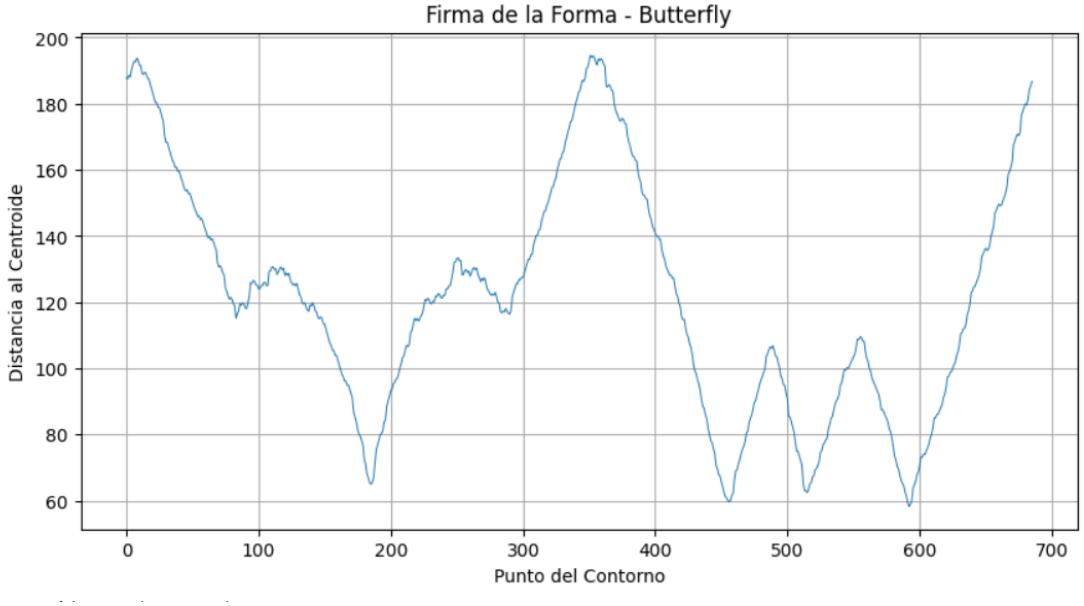



```
def obtener_distancia(contor, centroide):
    distancia = np.sqrt((contor[0][0] - centroide[0])**2 + (contor[0][1] - centroide[1])**2)
```

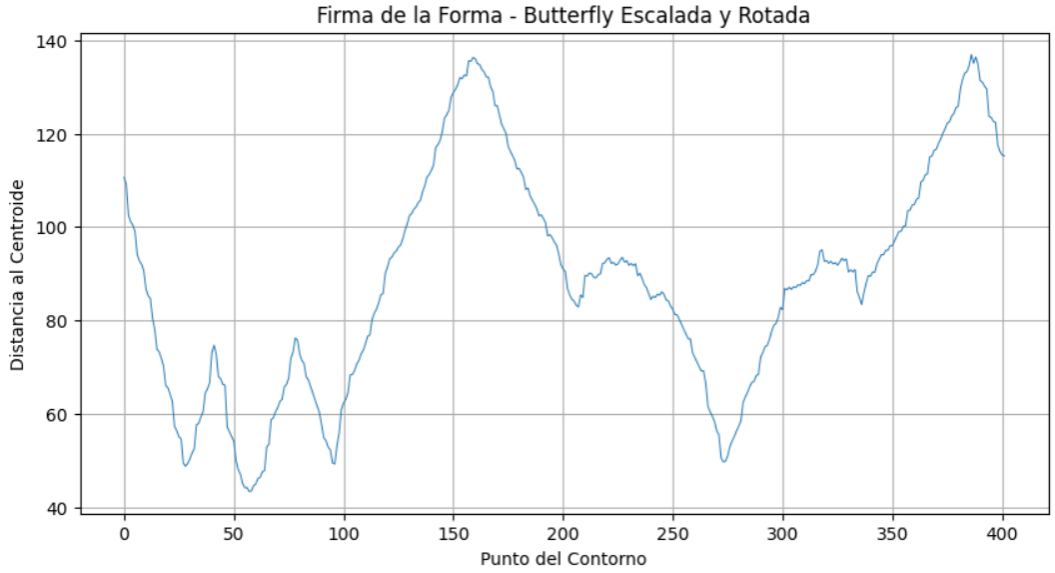
```
    return distancia


def firma_forma(image_path, title_prefix):
    image = cv2.imread(image_path)


    if image is None:
        raise ValueError("La imagen no se pudo cargar. Verifica la ruta y el nombre del archivo.")


    imagen_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    gray = cv2.cvtColor(imagen_rgb, cv2.COLOR_RGB2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)


    contornos, jerarquia = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)


    area_min = 1000
    contornos_filtrados = [cnt for cnt in contornos if cv2.contourArea(cnt) > area_min]


    img_bordes = cv2.Canny(gray, 50, 150, apertureSize=3)


    img_contornos = np.zeros_like(imagen_rgb)
    if len(contornos_filtrados) > 0:
        max_contour = max(contornos_filtrados, key=cv2.contourArea)
        cv2.drawContours(img_contornos, [max_contour], -1, (0, 255, 0), 2)
        M = cv2.moments(max_contour)
        if M['m00'] != 0:
            cx = int(M['m10'] / M['m00'])
            cy = int(M['m01'] / M['m00'])
            centroide = (cx, cy)
            cv2.circle(img_contornos, (cx, cy), 5, (0, 0, 255), -1)
```

```
            print(f"Centroide en: ({cx}, {cy})")

            distancias = [obtener_distancia(pt, centroide) for pt in max_contour]

            ix = np.arange(0, len(distancias), 1)
            plt.figure(figsize=(10, 5))
            plt.plot(ix, distancias, lw=0.7)
            plt.title(f'Firma de la Forma - {title_prefix}')
            plt.xlabel('Punto del Contorno')
            plt.ylabel('Distancia al Centroide')
            plt.grid(True)
            plt.show()
        else:
            print("No se pudo calcular el centroide correctamente (división por cero).")
    else:
        print("No se encontraron contornos.")
firma_forma(ruta_imagen_butterfly, 'Butterfly')
firma_forma(ruta_butterfly_escalado_rotado, 'Butterfly Escalada y Rotada')
```

```
Centroide en: (132, 185)
```



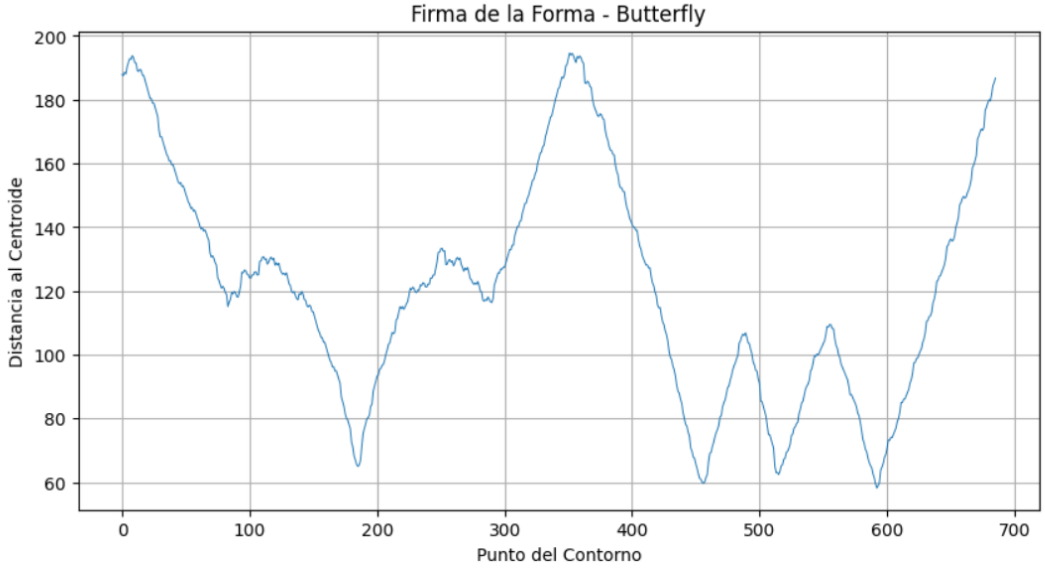

```
Centroide en: (156, 183)
```



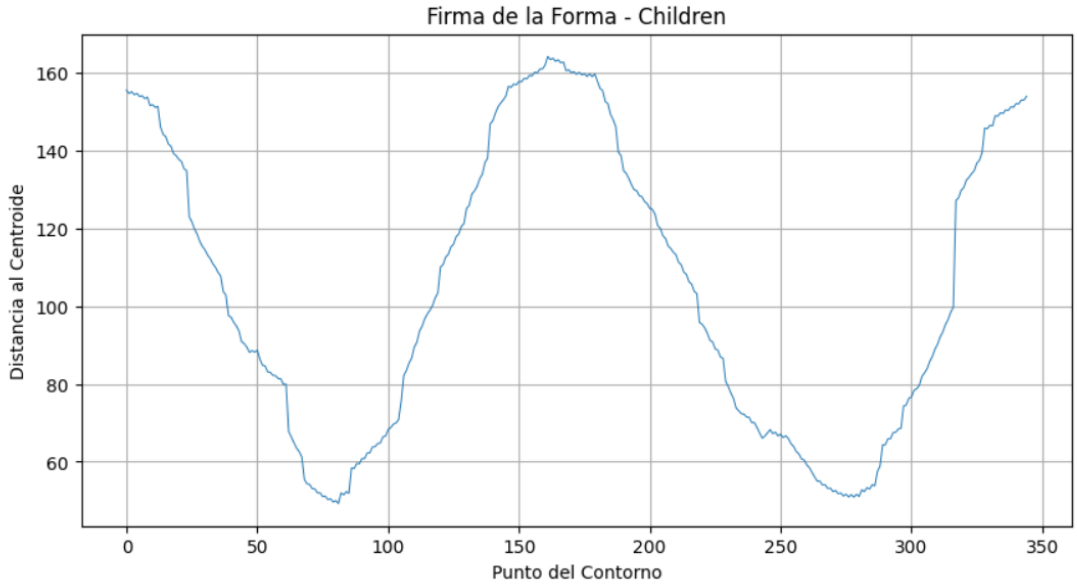
firma_forma(ruta_imagen_butterfly, 'Butterfly')

firma_forma(ruta_imagen_children, 'Children')

```
Centroide en: (132, 185)
```



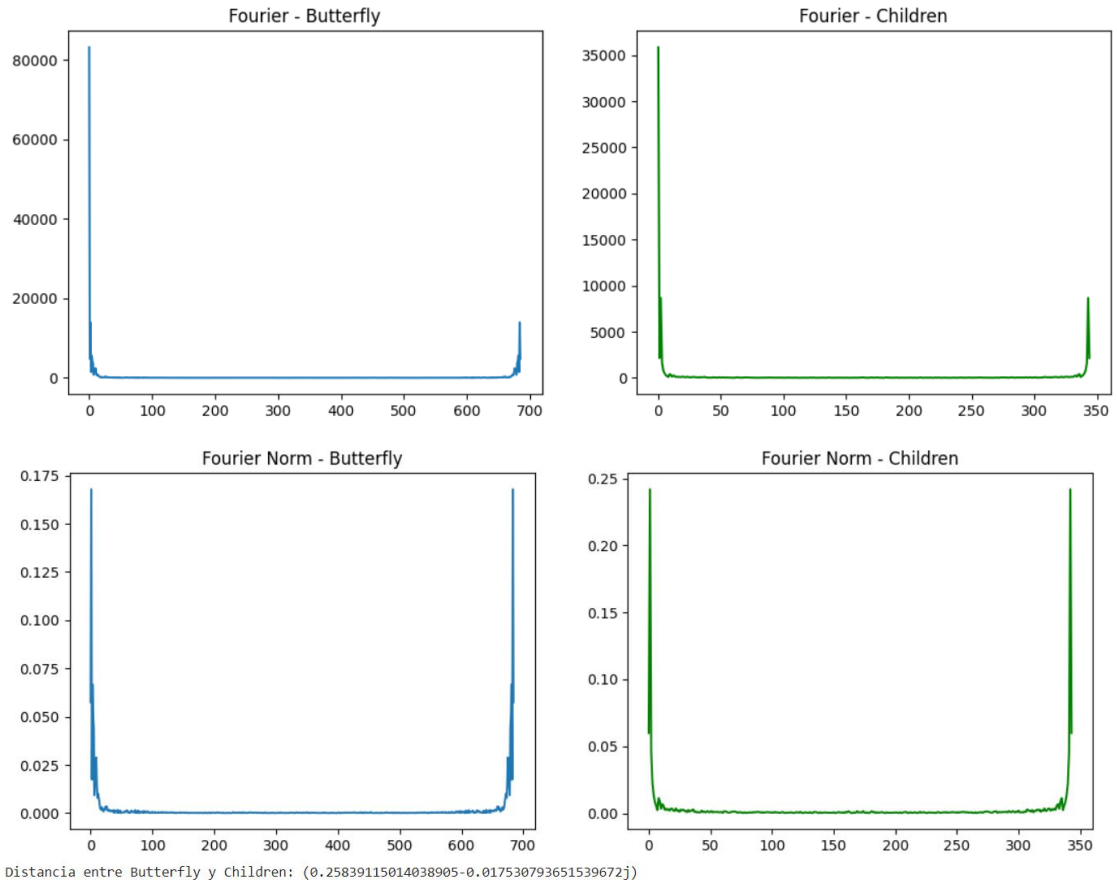

```
Centroide en: (60, 156)
```




```
def distancias_imgs(image_path):
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError("La imagen no se pudo cargar. Verifica la ruta y el nombre del archivo.")

    imagen_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    gray = cv2.cvtColor(imagen_rgb, cv2.COLOR_RGB2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
```

```
    contornos, jerarquia = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    area_min = 1000
    contornos_filtrados = [cnt for cnt in contornos if cv2.contourArea(cnt) > area_min]


    if len(contornos_filtrados) > 0:
        max_contour = max(contornos_filtrados, key=cv2.contourArea)
        M = cv2.moments(max_contour)
        if M['m00'] != 0:
            cx = int(M['m10'] / M['m00'])
            cy = int(M['m01'] / M['m00'])
            centroide = (cx, cy)
            distancias = [obtener_distancia(pt, centroide) for pt in max_contour]
            return distancias
        else:
            raise ValueError("No se pudo calcular el centroide correctamente (división por
cero).")
    else:
        raise ValueError("No se encontraron contornos.")


def distancia(dis1, dis2):
    dis = 0.0
    for i in range(min(len(dis1), len(dis2))):
        dis += (dis1[i] - dis2[i])**2
    return np.sqrt(dis)


distancias_butterfly = distancias_imgs(ruta_imagen_butterfly)
distancias_children = distancias_imgs(ruta_imagen_children)
distancias_escalado_butterfly = distancias_imgs(ruta_butterfly_escalado_rotado)


fourier_butterfly = fft(distancias_butterfly)
fourier_children = fft(distancias_children)
```

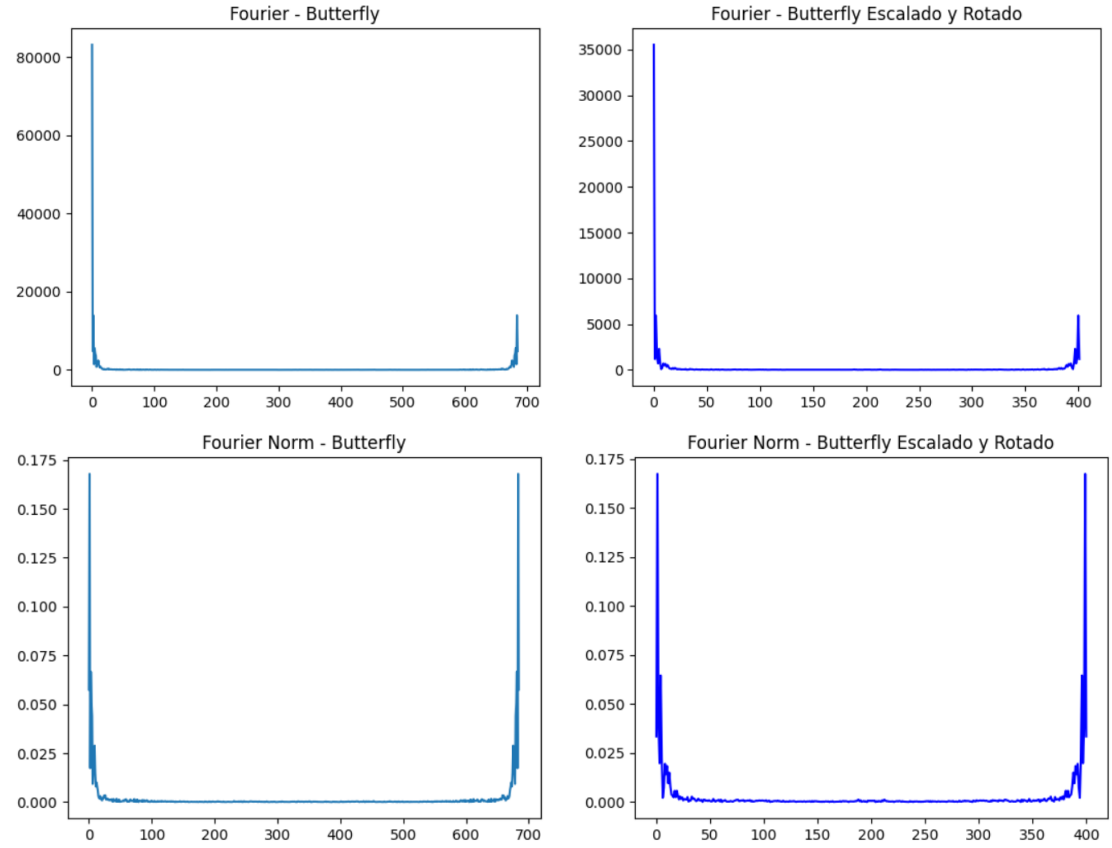
```
fourier_escalado_butterfly = fft(distancias_escalado_butterfly)

fourier_butterfly_N = fourier_butterfly / fourier_butterfly[0]
fourier_children_N = fourier_children / fourier_children[0]
fourier_escalado_butterfly_N = fourier_scaled_butterfly / fourier_escalado_butterfly[0]

figure, axes = plt.subplots(nrows=2, ncols=2)
figure.set_size_inches(13, 10)

axes[0, 0].plot(np.abs(fourier_butterfly))
axes[0, 0].set_title('Fourier - Butterfly')

axes[0, 1].plot(np.abs(fourier_children), color='green')
axes[0, 1].set_title('Fourier - Children')

axes[1, 0].plot(np.abs(fourier_butterfly_N[1:]))
axes[1, 0].set_title('Fourier Norm - Butterfly')

axes[1, 1].plot(np.abs(fourier_children_N[1:]), color='green')
axes[1, 1].set_title('Fourier Norm - Children')

plt.show()

distancia_butterfly_children = distancia(fourier_butterfly_N, fourier_children_N)
print('Distancia entre Butterfly y Children:', distancia_butterfly_children)
```




```
figure, axes = plt.subplots(nrows=2, ncols=2)
figure.set_size_inches(13, 10)

axes[0, 0].plot(np.abs(fourier_butterfly))
axes[0, 0].set_title('Fourier - Butterfly')

axes[0, 1].plot(np.abs(fourier_escalado_butterfly), color='blue')
axes[0, 1].set_title('Fourier - Butterfly Escalado y Rotado')

axes[1, 0].plot(np.abs(fourier_butterfly_N[1:]))
axes[1, 0].set_title('Fourier Norm - Butterfly')

axes[1, 1].plot(np.abs(fourier_escalado_butterfly_N[1:]), color='blue')
axes[1, 1].set_title('Fourier Norm - Butterfly Escalado y Rotado')

plt.show()
```

distancia_butterfly_escalado_butterfly = distancia(fourier_butterfly_N, fourier_escalado_butterfly_N)

print('Distancia entre Butterfly y Butterfly Escalada y Rotada:', distancia_butterfly_escalado_butterfly)




```
Distancia entre Butterfly y Butterfly Escalada y Rotada: (0.13877240190812423-0.2463066800145596j)
```

Gracias a las funcionalidades y teorema de Shape Signatures, podemos observar y demostrar matemáticamente las figuras de una imagen, podemos compararlos y ver sus similaridades y diferencias entre diferentes figuras y entre una misma figura alterada al escalarlo y rotarlo. A partir del cálculo de los contornos y centroide, nos basamos en esos valores para sacar la firma de las figuras y entender en que partes del recorrido del contorno, desde el centroide, existen las variaciones. Las imágenes de Butterfly y Butterfly escalado y rotado son exactamente similares, mientras que entre Butterfly y Children son bastante distintos. Finalmente, con la distancia euclidiana, mediante el cálculo de la transformada rápida de Fourier de las dos señales, entre Butterfly y Butterfly escalado y rotado tienen un valor de distancia mínima en comparación de entre Butterfly y Children tienen un valor mas alto.

Referencias:

- HumanSignal. (2022). labelImg: LabelImg is now part of the Label Studio community. The popular image annotation tool created by Tzutalin is no longer actively being developed, but you can check out Label Studio, the open source data labeling tool for images, text, hypertext, audio, video and time-series data. Recuperado de https://github.com/HumanSignal/labelImg
- THU-MIG. (2024). Yolov10: YOLOv10: Real-Time End-to-End Object Detection. Recuperado de https://github.com/THU-MIG/yolov10