Algoritmos Tesis

María José Bustamante - Paola Peralta Flores

Roboflow

Es una plataforma de visión artificial que permite a los usuarios crear modelos de visión artificial de forma más rápida y precisa mediante la provisión de mejores técnicas de recopilación de datos, preprocesamiento y formación de modelos. Permite a los usuarios cargar conjuntos de datos personalizados, dibujar anotaciones, modificar orientaciones de imágenes, cambiar el tamaño de imágenes, modificar el contraste de imágenes y realizar aumentos de datos. También se puede utilizar para entrenar modelos. Ha evolucionado al uso de técnicas de Machine Learning como:

1. Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN, etc.)
2. You Only Look Once (YOLO)
3. Single Shot MultiBox Detector (SSD)
4. Retina-Net

Model Predict Segmentation

La segmentación de imágenes, es la tarea de visión artificial de reconocer objetos en imágenes junto con su forma asociada. Es una extensión de la detección de objetos donde cada predicción también incluye una forma en lugar de solo un cuadro delimitador definido por un punto central, ancho y alto.

Con la segmentación, su aplicación puede determinar la cantidad de objetos en una imagen, las clasificaciones y su contorno.

Utilizamos un modelo de predicción para realizar las segmentaciones de la lengua de cada una de las imágenes con las que se va a trabajar.

```python
from roboflow import Roboflow

rf = Roboflow(api_key="ezWoyGKUgHzvpZ7l34C3")
project = rf.workspace().project("tongue-segmentation")
model = project.version(1).model

# infer on a local image
print(model.predict("lengua1.jpg").json())

# infer on an image hosted elsewhere
print(model.predict("lengua1.jpg").json())

# save an image annotated with your predictions
model.predict("lengua8.jpg").save("segmentar8.jpg")
```

Figure 1: Imagen de lengua segmentada

Yolo 8

El algoritmo 'You Only Look Once' (YOLO) logró reconocer objetos en imágenes/videos en tiempo real. Este éxito hizo popular el formato de anotación y, a medida que se desarrollaron nuevas variantes del algoritmo, sus respectivos formatos de anotación YOLO también ganaron popularidad.

```python
from ultralytics import YOLO

model = YOLO('yolov8n.pt')
model.predict(
    source='Tongue.v3i.yolov8',
    conf=0.25
)
```

En este caso utilizamos el yolov8n.pt para el entrenamiento y luego realiza las predicciones de reconocer la lengua, en cada una de las carcateríticas que necesitamos, en este caso son una lengua normal, fisurada y con bordes marcados por dientes.

```python
from ultralytics import YOLO

model = YOLO('/home/paola/Desktop/TEXTURA/runs/detect/train4/weights/best.pt')
model.predict(
    source='lengua8.jpg', save=True,
    conf=0.55
)
```

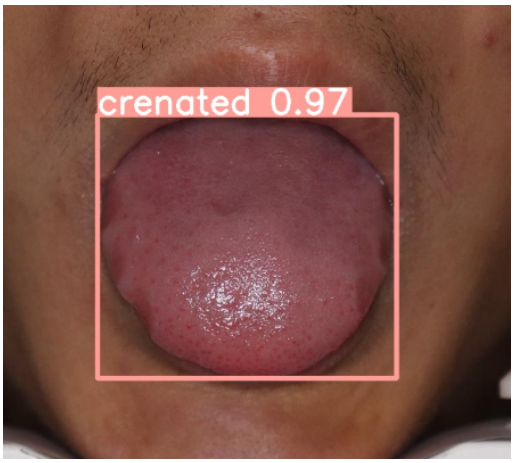

Figure 2: Lengua clasificada como bordes con dientes marcados

OpenCV

Contiene implementaciones de más de 2500 algoritmos. Además está disponible de forma gratuita para fines comerciales y académicos. Las posibilidades de análisis y tratamiento de imágenes con la biblioteca OpenCV son inmensas, desde detectar caras y clasificarlas según género hasta crear modelos de realidad aumentada o usar clasificadores para detectar objetos.

En este caso, usamos la librería de OpenCv para realizar el recorte de una imagen segmentada con anterioridad. En la imagen se muestra, como se utiliza esta librería con algunos de sus algorirtmos para lograr el recorte de nuestra imagen segmentada y dejarla solo con un fondo blanco.

```python
import cv2
import numpy as np

# Cargar la imagen
imagen = cv2.imread('segmentar8.jpg')

# Convertir la imagen a escala de grises
imagen_gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)

# Aplicar un umbral para obtener una imagen binaria del contorno azul
umbral_inf = np.array([100, 0, 0])  # BGR - límite inferior para el color azul
umbral_sup = np.array([255, 50, 50])  # BGR - límite superior para el color azul
mascara_azul = cv2.inRange(imagen, umbral_inf, umbral_sup)

# Encontrar los contornos en la imagen binaria
contornos, _ = cv2.findContours(mascara_azul, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Crear una máscara en blanco del tamaño de la imagen original
mascara_recorte = np.zeros_like(imagen_gris)

# Dibujar todos los contornos en la máscara
cv2.drawContours(mascara_recorte, contornos, -1, (255), thickness=cv2.FILLED)

# Aplicar la máscara al contorno original para recortar
imagen_recortada = cv2.bitwise_and(imagen, imagen, mask=mascara_recorte)

# Convertir los píxeles azules en blanco en la imagen recortada
imagen_recortada[np.where(mascara_azul == 255)] = [255, 255, 255]

# Crear una imagen en blanco del mismo tamaño que la imagen original
imagen_fondo_blanco = np.ones_like(imagen) * 255

# Aplicar la máscara inversa al fondo blanco para el contorno azul
mascara_contorno_azul = cv2.bitwise_not(mascara_recorte)
imagen_fondo_blanco = cv2.bitwise_and(imagen_fondo_blanco, imagen_fondo_blanco, mask=mascara_contorno_azul)

# Combinar la imagen recortada y el fondo blanco
imagen_recortada_final = cv2.add(imagen_recortada, imagen_fondo_blanco)

# Guardar la imagen resultante
cv2.imwrite('recortar8.jpg', imagen_recortada_final)

# Mostrar la imagen original y la imagen recortada
cv2.imshow('Imagen Original', imagen)
cv2.imshow('Imagen Recortada', imagen_recortada_final)
cv2.waitKey(1)
cv2.destroyAllWindows()
```

El recorte de la imagen lo realizamos, ya que el siguiente paso en encontrar tanto el color dominante como una paleta de colores de la lengua y el área de labios o dientes puede interferir en estos resultados.

Figure 3: Lengua recortada con fondo blanco para poder relizar los siguientes pasos.

K - means

Es un algoritmo de clasificación no supervisada (clusterización) que agrupa objetos en k grupos basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster.

El algoritmo *k-means* resuelve un **problema de optimización**, siendo la función a optimizar (minimizar) la suma de las distancias cuadráticas de cada objeto al centroide de su cluster.

```python
import cv2
import numpy as np
from sklearn.cluster import KMeans

def segment_tongue(image_path, num_clusters=2):
    # Cargar la imagen
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Convertir la imagen al espacio de color Lab
    lab_image = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)

    # Obtener los canales L, a y b
    l_channel, a_channel, b_channel = cv2.split(lab_image)

    # Crear una matriz de características utilizando a_channel y b_channel
    feature_matrix = np.column_stack((a_channel.flatten(), b_channel.flatten()))

    # Aplicar K-means
    kmeans = KMeans(n_clusters=num_clusters, random_state=0)
    kmeans.fit(feature_matrix)

    # Obtener las etiquetas de píxeles
    labels = kmeans.labels_

    # Reshape las etiquetas para que coincidan con la forma de la imagen
    labels = np.reshape(labels, (image.shape[0], image.shape[1]))
```

```python
    # Crear una máscara para separar el cuerpo de la lengua y el revestimiento de la lengua
    tongue_mask = np.zeros_like(labels, dtype=np.uint8)
    tongue_mask[labels == 1] = 128  # Revestimiento de la lengua en gris
    tongue_mask[labels == 0] = 255  # Cuerpo de la lengua en blanco

    # Calcular el porcentaje del revestimiento de la lengua
    coating_pixels = np.count_nonzero(tongue_mask == 128)
    total_pixels = tongue_mask.shape[0] * tongue_mask.shape[1]
    coating_percentage = (coating_pixels / total_pixels) * 100

    # Imprimir el porcentaje del revestimiento de la lengua
    print("Porcentaje del revestimiento de la lengua: {:.2f}%".format(coating_percentage))

    # Aplicar la máscara a la imagen original
    segmented_image = cv2.bitwise_and(image, image, mask=tongue_mask)

    # Mostrar la imagen segmentada y la máscara
    cv2.imshow('Segmented Image', segmented_image)
    cv2.imshow('Tongue Mask', tongue_mask)
    cv2.imwrite('saburra7.jpg', tongue_mask)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

# Ejemplo de uso
segment_tongue('recortar7.jpg', num_clusters=2)
```

En este caso utilizamos el algoritmo para detectar la saburra de le lengua y encontrar su porcentaje, también antes de aplicar este algoritmo se tranformó la imagen a espacio de color lab.

Figure 4: Saburra detetctada en la lengua (blanco)

Fortalezas

Trabajar con algoritmos de inteligencia artificial tiene ventajas, como:

1. Automatización y eficiencia: Aumentar la eficiencia y la productividad. Realiza tareas repetitivas y tediosas de manera más rápida y precisa, liberando tiempo y recursos.
2. Toma de decisiones basada en datos: Analiza grandes volúmenes de datos y extrae información relevante para la toma de decisiones. Identifica patrones, tendencias y correlaciones para mejorar la precisión de las decisiones y optimizar los resultados.

3. Personalización y experiencia del usuario: Se adapta y personaliza según las preferencias y necesidades individuales.

Debilidades

La majyor debilidad que se presenta es el hecho de entender el fncionamiento de cada algoritmo para poder aplicarlo de anera correcta. A continuación se presentan otras debilidades comunes:

1. Dependencia de datos de calidad: Los algoritmos de IA requieren grandes cantidades de datos de alta calidad para entrenar y funcionar de manera efectiva. Si los datos utilizados están desactualizados, incompletos, sesgados o incorrectos, esto puede afectar la precisión y confiabilidad de los resultados obtenidos.
2. Falta de comprensión y explicabilidad: Algunos algoritmos de IA, como las redes neuronales profundas, pueden ser altamente complejos y difíciles de entender y explicar cómo llegan a sus resultados.
3. Falta de contexto y sentido común: Los algoritmos de IA pueden tener dificultades para comprender el contexto y aplicar el sentido común en ciertas situaciones. Pueden tomar decisiones basadas únicamente en los patrones y datos con los que fueron entrenados, sin considerar factores externos relevantes. Esto puede llevar a resultados inesperados o inapropiados en situaciones no previstas.
4. Privacidad y seguridad de los datos: El uso de algoritmos de IA implica la recopilación y el análisis de grandes cantidades de datos, lo que plantea preocupaciones sobre la privacidad y la seguridad. Si los datos almacenados o utilizados por los algoritmos no se protegen adecuadamente, pueden surgir riesgos de violaciones de datos y compromiso de la privacidad.