

Unidad 2 - Modelos de visión y aplicaciones

UNR - TUIA - Procesamiento de Imágenes y Visión por Computadora

Docente teoría: Juan Pablo Manson

Docentes práctica: Lucas Bruge, Constantino Ferrucci

Temario

Técnicas y modelos de visión: Feature detection and matching. Motion Detection. Image classification. Object Detection. Object Segmentation. Panoptic Segmentation. Object Tracking. Face detection & Recognition. Facial expression recognition. Human Pose Estimation. Marker detection. Color Quantization. Image captioning. Image embeddings.

Código de los ejemplos:

Los diferentes métodos y modelos que veremos en este capítulo, tienen sus correspondientes ejemplos prácticos en el siguiente cuaderno Colab:

Google Colab

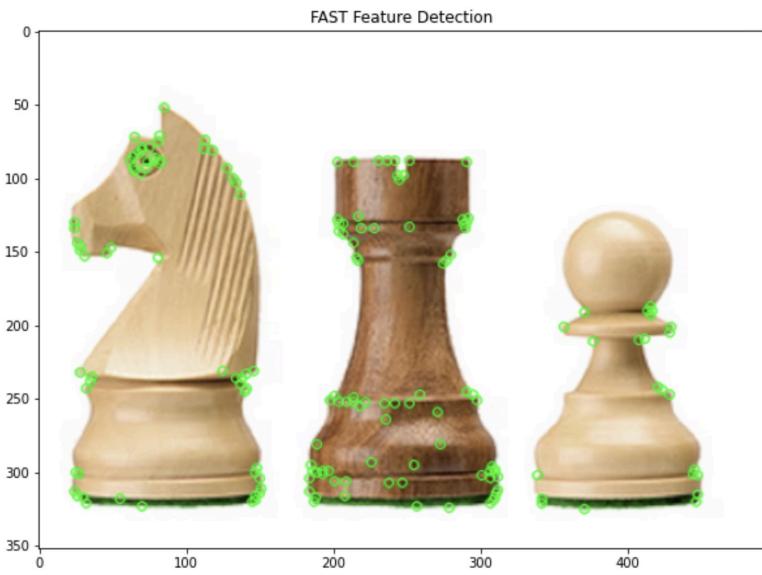
🔗 <https://colab.research.google.com/drive/1bbpmK-G-0f7MseD3LIZY-RiaxgIQKe9u?usp=sharing>



1. Feature detection and matching

Introducción y conceptos teóricos

En el campo de la visión por computadora, la "Detección y Coincidencia de Características" (Feature Detection and Matching) es un proceso fundamental, que involucra la identificación de puntos o áreas específicas en una imagen que son únicos y fácilmente reconocibles. Estas áreas o puntos únicos se conocen como "características" y suelen ser esquinas, bordes o manchas específicas en la imagen. La detección y coincidencia de características es esencial para muchas tareas de visión por computadora, como el seguimiento de objetos, el reconocimiento de imágenes, la reconstrucción 3D y la navegación de robots.



Ejemplo de detección de características, usando el algoritmo FAST.

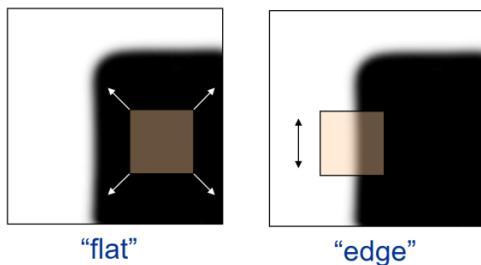
Clasificaciones de Características

En visión por computadora, la identificación de características distintivas en las imágenes es fundamental en tareas como el reconocimiento de objetos, seguimiento y reconstrucción 3D. Algunos de los tipos más importantes de características son los bordes (edges), esquinas (corners), manchas (blobs) y crestas (ridges). Estas características se utilizan para detectar patrones, formas y estructuras en imágenes. Aquí está una explicación de cada uno:

- **Técnicas de Detección:** La detección de crestas puede realizarse mediante algoritmos que buscan cambios máximos en la intensidad en una dirección específica, como el filtro de matriz de Hessian o el análisis de componentes principales locales (PCA).

1. Bordes (Edges)

- **Definición:** Un borde en una imagen es un conjunto de píxeles donde hay un cambio abrupto o una discontinuidad en la intensidad de la imagen o en los niveles de gris. Los bordes suelen indicar los límites entre diferentes regiones de la imagen.

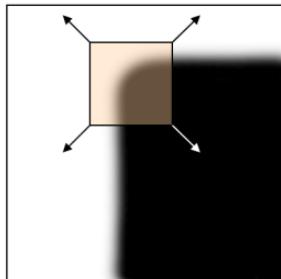


Flat (Plano): No hay cambios en ninguna dirección. Edge (Borde): No hay cambios en la dirección del borde.

- **Importancia:** La detección de bordes es fundamental en la visión por computadora porque los bordes forman el contorno básico de objetos y características en una imagen.
- **Técnicas de Detección:** Algoritmos comunes para la detección de bordes incluyen el Operador de Sobel, el Operador de Prewitt, el Detector de Canny, y el Operador de Laplace.

2. Esquinas (Corners)

- **Definición:** Una esquina se define como el punto donde se encuentran dos o más bordes, o donde hay un cambio significativo en la dirección del borde. Las esquinas son puntos de intensidad locales que se mantienen consistentes en diferentes vistas de la misma escena.



“corner”

En una esquina (corner), los cambios son significativos en todas las direcciones.

- **Importancia:** Las esquinas son puntos de interés clave porque son invariantes a la traslación, rotación y, hasta cierto punto, a los cambios de escala, lo que las hace útiles para el seguimiento y reconocimiento de objetos.
- **Técnicas de Detección:** Los detectores de esquinas incluyen el [Detector de Esquinas de Harris](#), el [Detector de Esquinas de Shi-Tomasi](#), y el detector [FAST](#).

Claude Artifact
Try out Artifacts created by Claude users

 **Claude**

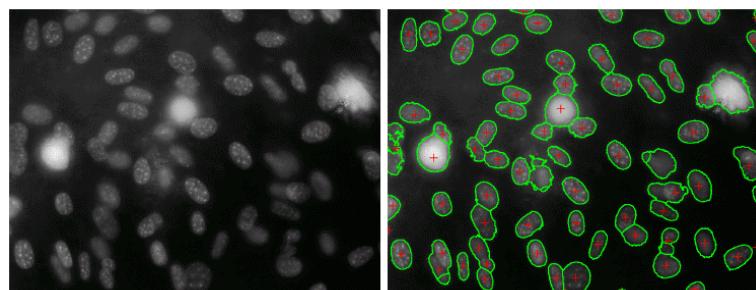


 <https://claude.site/artifacts/ada7a5c0-6afe-4fb7-bd57-77b22a25f29e?fullscreen=true>

Ejemplo interactivo para probar un detector de Harris

3. Manchas (Blobs)

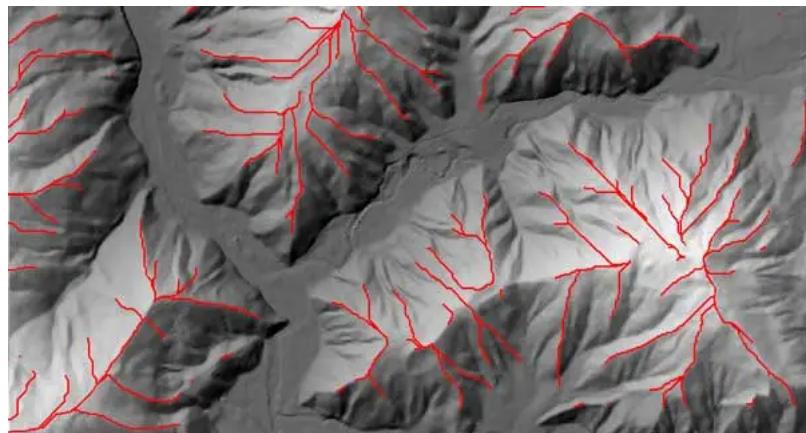
- **Definición:** Un blob es una región de la imagen que difiere en propiedades como la intensidad o el color en comparación con las áreas circundantes. Los blobs son típicamente grupos de píxeles que comparten alguna propiedad común.



- **Importancia:** La detección de blobs es útil para la extracción de características que representan objetos o partes de objetos en una imagen. Los blobs proporcionan información sobre la forma y el tamaño de las características.
- **Técnicas de Detección:** Algoritmos como el Detector de Blob de [Laplacian of Gaussian \(LoG\)](#), el Detector de Blob de [Difference of Gaussians \(DoG\)](#), y el detector de [blob de determinante de matriz Hessian](#) se utilizan para identificar blobs.

4. Crestas (Ridges)

- **Definición:** Las crestas son patrones lineales en una imagen que pueden representar curvas o líneas. Las crestas (ridges) se refieren a los máximos locales en una dirección específica dentro de una imagen o función (un máximo local es un punto donde el valor de la función es mayor que los valores en su vecindad inmediata en una dirección particular). En el caso de las crestas, se consideran máximos locales en la dirección perpendicular a la orientación de la cresta. A lo largo de la dirección de la cresta, los valores pueden mantenerse constantes o cambiar gradualmente. Para identificar una cresta, se buscan puntos que sean máximos locales en al menos una dirección, típicamente perpendicular a la orientación de la característica.



- **Importancia:** Las crestas son útiles en la detección de estructuras lineales en imágenes, como carreteras en imágenes de satélite, vasos sanguíneos en imágenes médicas, o arrugas en imágenes de rostros.
- **Técnicas de Detección:** El Operador de Canny, aunque es principalmente un detector de bordes, también puede usarse para identificar crestas. Utiliza la primera y segunda derivada de la imagen para encontrar máximos locales. El Detector de Harris, originalmente diseñado para detectar esquinas, pero también puede identificar crestas. La Transformada de Hough, es útil para detectar líneas y curvas que pueden corresponder a crestas. Otro métodos pueden utilizarse también, como Filtros de Gabor, Operador Laplaciano, wavelets entre otros.

Detección de Características

La detección de características implica identificar puntos de interés en la imagen. Estos puntos de interés son distintivos y pueden ser fácilmente localizados en diferentes vistas de la misma escena o objeto.



Las características más relevantes, son aquellas que permanecen invariantes ante las transformaciones. Una buena característica, presenta invarianza geométrica (traslación, rotación y escala) e invarianza fotométrica (brillo, exposición o contraste).

Algunos de los detectores de características más tradicionales son:

- **Harris Corner Detector:** Detecta esquinas en la imagen basándose en la respuesta local a cambios en diferentes direcciones. (**OpenCV:** https://docs.opencv.org/5.x/dc/d0d/tutorial_py_features_harris.html)
- **Shi-Tomasi Corner Detector:** Los autores modificaron la función de puntuación utilizada en la Detección de Esquinas de Harris para lograr una técnica de detección de esquinas mejorada. (**OpenCV:** https://docs.opencv.org/5.x/d4/d8c/tutorial_py_shi_tomasi.html)
- **FAST (Features from Accelerated Segment Test):** Detecta esquinas rápidamente y es útil en aplicaciones en tiempo real. (**OpenCV:** https://docs.opencv.org/5.x/df/d0c/tutorial_py_fast.html)
- **SIFT (Scale-Invariant Feature Transform):** Detecta y describe características locales en imágenes que son invariantes a la escala y la rotación. (**OpenCV:** https://docs.opencv.org/5.x/da/df5/tutorial_py_sift_intro.html)
- **SURF (Speeded Up Robust Features):** Similar a SIFT, pero más rápido, es útil en aplicaciones que requieren eficiencia en tiempo real. (**OpenCV:** https://docs.opencv.org/5.x/df/dd2/tutorial_py_surf_intro.html)
- **ORB (Oriented FAST and Rotated BRIEF):** Una alternativa eficiente a SIFT y SURF, que es invariante a la rotación y resistente a cambios de iluminación. (**OpenCV:** https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html)
- **Binary Robust Invariant Scalable Keypoints (BRISK):** BRISK se diseñó para ser un algoritmo eficiente y robusto para la detección y descripción de puntos clave en imágenes. Utiliza una técnica basada en la determinante de la matriz Hessiana para detectar puntos clave en la imagen. Estos puntos clave son esquinas o regiones de alta intensidad que son distintivas y repetibles. BRISK describe los puntos clave utilizando un descriptor binario compacto. Este descriptor se calcula mediante el muestreo de pares de píxeles alrededor del punto clave y la comparación de sus intensidades. El resultado se codifica como un vector binario, lo que lo hace computacionalmente eficiente y robusto al ruido y cambios de iluminación. BRISK utiliza un patrón de muestreo

circular para lograr la invarianza a la rotación, y una técnica de pirámide de imágenes para lograr la invarianza a la escala. BRISK está diseñado para ser muy eficiente en términos de tiempo de cómputo y uso de memoria.

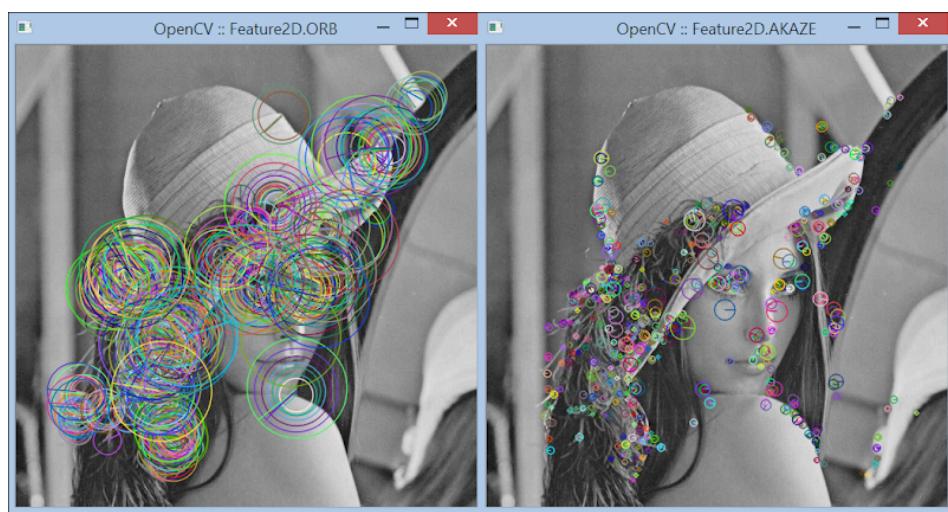
Los diferentes métodos de detección existentes, se especializan en la detección de uno o dos tipos de características. El siguiente es un resumen de varios métodos de detección y sus capacidades:

Feature detector	Edge	Corner	Blob	Ridge
Canny[3]	Si	No	No	Si
Sobel	Si	No	No	No
Harris & Stephens / Plessey[4]	Si	Si	No	No
SUSAN[5]	Si	Si	No	No
Shi & Tomasi[6]	No	Si	No	No
Level curve curvature[7]	No	Si	No	No
FAST[8]	No	Si	No	No
Laplacian of Gaussian[7]	No	Si	Si	No
Difference of Gaussians[9][10]	No	Si	Si	No
Determinant of Hessian[7]	No	Si	Si	No
Hessian strength feature measures[11][12]	No	Si	Si	No
MSER[13]	No	No	Si	No
Principal curvature ridges[14][15][16]	No	No	No	Si
Grey-level blobs[17]	No	No	Si	No

Fuente: [https://en.wikipedia.org/wiki/Feature_\(computer_vision\)](https://en.wikipedia.org/wiki/Feature_(computer_vision)).

Definiciones: Descriptores (descriptors) y Puntos Clave (keypoints)

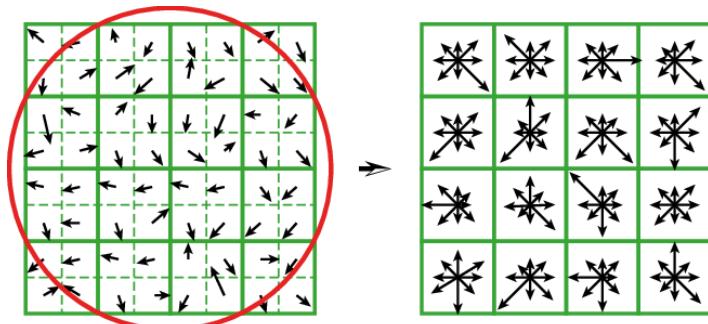
Si bien podemos encontrar en una imagen cientos de características, nos interesa trabajar con un conjunto de puntos más significativos, que llamamos *keypoints*. Estos son puntos específicos en la imagen resultan más representativos y son seleccionados generalmente porque tienen una estructura única en términos de su entorno, como esquinas, bordes, o regiones con texturas interesantes. Nos resultan útiles porque proporcionan una manera compacta y eficiente de describir una imagen. Los algoritmos como SIFT, SURF, ORB, AKAZE y FAST identifican *keypoints* basándose en criterios como la intensidad del contraste, la estabilidad y la repetibilidad.



En el gráfico, vemos la graficación de keypoints, en este caso, obtenidos a través de los algoritmos ORB y AKAZE

Los *descriptores* son vectores numéricos que describen de manera cuantitativa las características de la imagen en y alrededor de cada *keypoint*. Estos vectores capturan la apariencia de la imagen en los *keypoints*, como la forma, la textura, el color, la orientación de los bordes, etc. Los descriptores son cruciales para comparar keypoints entre diferentes imágenes. Permiten a los algoritmos de visión por computadora determinar si un *keypoint* en una imagen corresponde a un *keypoint* en otra imagen, facilitando tareas como el reconocimiento de objetos y el emparejamiento de imágenes. Los *descriptores* son construidos tomando en cuenta el entorno de cada *keypoint*. Por ejemplo, en el

descriptor SIFT, se calcula la orientación del gradiente y la magnitud del vector de cada pixel, en una región alrededor del *keypoint*.



Proceso de construcción del descriptor SIFT. (Izquierda) Se calculan y agrupan los gradientes locales en una cuadrícula de 16×16 alrededor del punto clave (se muestra como 8×8 aquí por simplicidad). (Derecha) Para cada celda de la cuadrícula superpuesta de 4×4 , se calcula un histograma ponderado de 8D de los gradientes. (Fuente:

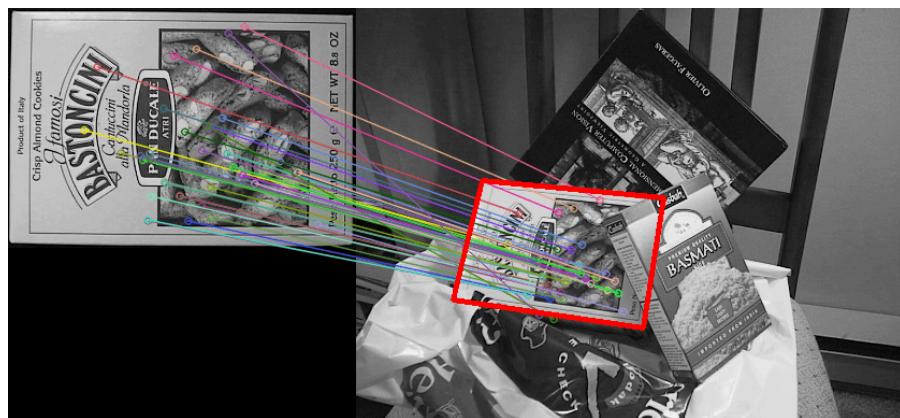
https://www.researchgate.net/publication/304185720_A_survey_on_compact_features_for_visual_content_analysis)

Los descriptores pueden subdividirse en 2 categorías: descriptores locales y descriptores globales.

- *Descriptor Local*: El objetivo es capturar la información en los alrededores más inmediatos del keypoint.
- *Descriptor Global*: Capturan la imagen como un todo. Esto los hace altamente poco fiables en escenarios del mundo real, ya que la más mínima transformación en una muestra de imagen puede hacer que el modelo falte.

Coincidencia de Características (Matching)

Una vez detectadas las características, el siguiente paso es la coincidencia de características, que implica encontrar características similares o correspondientes en diferentes imágenes. Esto se hace comparando las descripciones de las características (descriptores) entre las imágenes. Los descriptores proporcionan una representación única de la apariencia de la característica en la imagen, lo que facilita su identificación en otras imágenes.



La coincidencia de características, permite por ejemplo encontrar objetos en una imagen. Por ejemplo, podemos encontrar el objeto de la izquierda en la imagen de la derecha, aún estando en una posición diferente o parcialmente tapado por otro objeto.

Algunos métodos comunes de coincidencia de características incluyen:

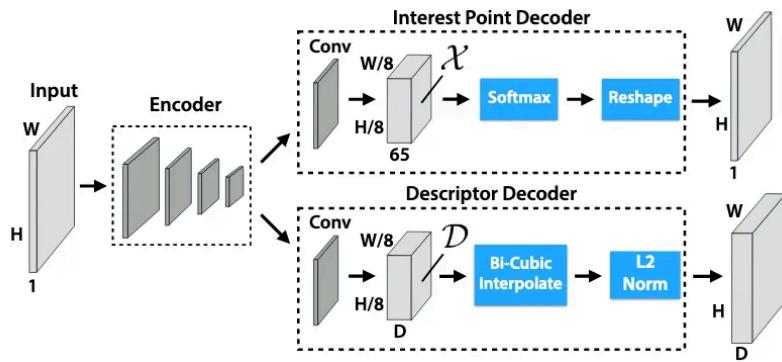
- **Coincidencia por Fuerza Bruta**: Compara cada descriptor en una imagen con todos los descriptores en la otra imagen para encontrar coincidencias.
- **FLANN (Fast Library for Approximate Nearest Neighbors)**: Encuentra coincidencias de forma más rápida y eficiente que la fuerza bruta, especialmente útil cuando el número de características es grande.

Técnicas de extracción de características mediante aprendizaje profundo

Los extractores de características tradicionales pueden ser reemplazados por una red neuronal convolucional (CNN), ya que las CNN tienen una fuerte capacidad para extraer características complejas que expresan la imagen con mucho

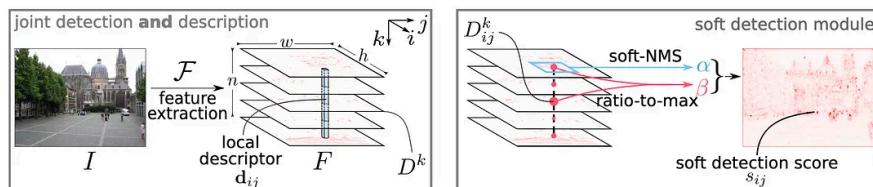
más detalle, aprenden las características específicas de la tarea y son mucho más eficientes. Se han realizado múltiples trabajos sobre esto. Algunos de ellos se enumeran a continuación:

- **SuperPoint: Self-Supervised Interest Point Detection and Description** ([paper](#)): Los autores sugieren una red neuronal completamente convolucional que calcula ubicaciones y descriptores de puntos de interés similares a SIFT en una sola pasada hacia adelante. Utiliza un codificador de estilo VGG (Very Deep Convolutional Network) para la extracción de características y luego dos decodificadores, uno para la detección de puntos y el otro para la descripción de puntos.



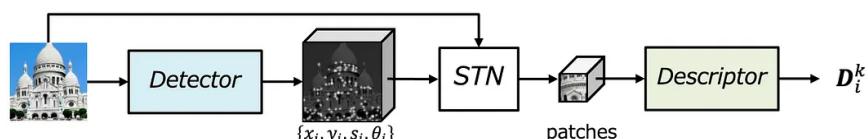
Arquitectura de SuperPoint (fuente: <https://arxiv.org/pdf/1712.07629.pdf>)

- **D2-Net: A Trainable CNN for Joint Description and Detection of Local Features** ([paper](#)): Los autores sugieren una única red neuronal convolucional que funciona tanto como descriptor de características densas como detector de características:



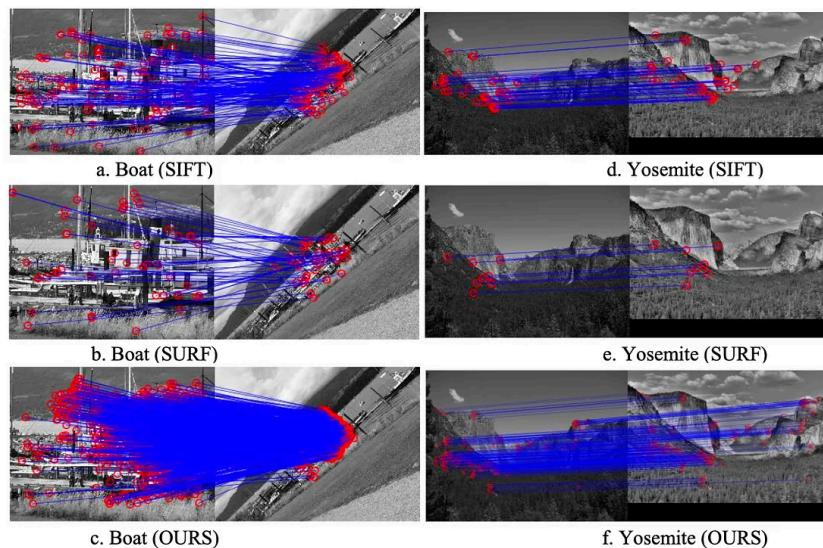
Detect and Describe D2 network (src: <https://arxiv.org/pdf/1905.03561.pdf>)

- **LF-Net: Learning Local Features from Images** ([paper](#)): LF-Net es una arquitectura de red neuronal profunda que aprende de manera integral la extracción de características locales en imágenes, abarcando la detección de puntos clave, estimación de escala y orientación, y descripción de características. El enfoque innovador utiliza un entrenamiento end-to-end y auto-supervisado, empleando pares de imágenes con pose relativa y mapas de profundidad conocidos, pero sin requerir anotaciones manuales. La clave del método es una estrategia de entrenamiento de dos ramas que permite la optimización a través de componentes no diferenciables, como la selección de puntos clave. Una rama procesa una imagen para generar puntos clave, mientras que la otra crea un "objetivo virtual" en la segunda imagen utilizando la geometría conocida. Este proceso iterativo, combinado con múltiples funciones de pérdida a nivel de imagen y parche, permite que LF-Net aprenda a extraer características locales robustas y distintivas, superando a métodos anteriores tanto aprendidos como diseñados manualmente en tareas de emparejamiento de características.



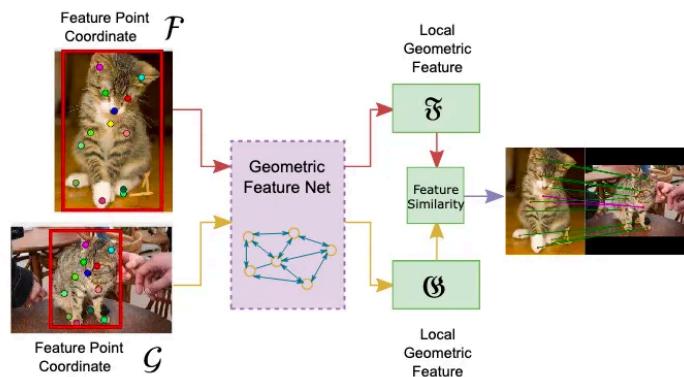
LF-Net (src: <https://papers.nips.cc/paper/7861-lf-net-learning-local-features-from-images.pdf>)

- **Image Feature Matching Based on Deep Learning** ([paper](#)): Adoptan un modelo de red neuronal convolucional (CNN) profunda, que se enfoca en parches de imagen, en el emparejamiento de puntos de características de imágenes:



Comparación con los métodos SIFT y SURF (fuente: <https://ieeexplore.ieee.org/abstract/document/8780936>)

- **Deep Graphical Feature Learning for the Feature Matching Problem** ([paper](#)): Sugieren utilizar una red neuronal gráfica para transformar las coordenadas de puntos característicos en características locales, lo que luego facilita el uso de un algoritmo de inferencia simple para el emparejamiento de características:



(fuente: https://openaccess.thecvf.com/content_ICCV_2019/papers/Zhang_Deep_Graphical_Feature_Learning_for_the_Feature_Matching_Pro

Ejemplos prácticos

Detección de Keypoints

Para encontrar e imprimir los keypoints de una imagen usando Python y OpenCV, podemos utilizar los algoritmos ORB (Oriented FAST and Rotated BRIEF) y SIFT (Scale-Invariant Feature Transform). A continuación vemos una comparativa de ambos métodos:

```

#!/usr/bin/python
# Importar bibliotecas
import cv2
from matplotlib import pyplot as plt

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

# Convertir la imagen a RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Inicializar ORB
orb = cv2.ORB_create()
kp = orb.detect(image_rgb, None)
kp, des = orb.compute(image_rgb, kp)

# Mostrar los keypoints
plt.imshow(image)
plt.scatter([kp[i].pt for i in range(len(kp))], [kp[i].pt for i in range(len(kp))], 10, 'red')
plt.show()

```

```

orb = cv2.ORB_create()
keypoints_orb = orb.detect(image, None)
image_with_orb_keypoints = cv2.drawKeypoints(
    image_rgb,
    keypoints_orb,
    None,
    color=(0, 255, 0), # Verde brillante en RGB
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS # Muestra círculos y orientación
)

# Inicializar SIFT
sift = cv2.SIFT_create()
keypoints_sift = sift.detect(image, None)
image_with_sift_keypoints = cv2.drawKeypoints(
    image_rgb,
    keypoints_sift,
    None,
    color=(255, 0, 0), # Rojo brillante en RGB
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS # Muestra círculos y orientación
)

# Usar matplotlib para mostrar las imágenes
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(image_with_orb_keypoints)
plt.title('ORB Keypoints')
plt.axis('off') # Oculta los ejes para una mejor visualización

plt.subplot(1, 2, 2)
plt.imshow(image_with_sift_keypoints)
plt.title('SIFT Keypoints')
plt.axis('off') # Oculta los ejes

plt.show()

```

Y el resultado será:



El método `cv2.drawKeypoints` permite graficar los keypoints de diferentes maneras, por ejemplo, se puede enriquecer la forma de crear el gráfico modificando los flags:

```

cv2.drawKeypoints(image_rgb, keypoints_sift, None, color=None,
                  flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

```

Con el flag `cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS` le indicamos a `drawKeypoints` que no solo dibuje la posición de cada punto clave (que sería solo un pequeño punto), sino que también dibuje el tamaño y la orientación del punto clave. Los keypoints pueden poseer un ángulo, y esto depende del método de detección utilizado. Esta información es útil para lograr la invarianza a la rotación en el procesamiento y análisis de imágenes.

Descriptoros y búsqueda de coincidencias

Varios algoritmos de OpenCV, permiten obtener *keypoints* con sus *descriptoros*, usando `detectAndCompute`. Eso nos permiten obtener los *keypoints* con información adicional, que nos permitirán encontrar coincidencias con los *keypoints* obtenidos en otra imagen. Veamos un ejemplo con dos fotos diferentes, donde queremos encontrar coincidencias:

```
!wget "https://mywowo.net/media/images/cache/barcellona_parc_guell_02_visita_jpg_1200_630_cover_85.jpg" -O sala
!wget "https://i.pinimg.com/originals/a4/57/c5/a457c577a58f3715ce1b53d2b76993c9.jpg" -O salamandra2.jpg

import cv2

# Cargar las imágenes
image1 = cv2.imread('salamandra1.jpg', cv2.IMREAD_GRAYSCALE) # Reemplaza con la ruta a tu imagen
image2 = cv2.imread('salamandra2.jpg', cv2.IMREAD_GRAYSCALE) # Reemplaza con la ruta a otra imagen

# Inicializar ORB
orb = cv2.ORB_create()

# Detectar keypoints y calcular descriptoros
keypoints1, descriptors1 = orb.detectAndCompute(image1, None)
keypoints2, descriptors2 = orb.detectAndCompute(image2, None)

# Crear un objeto BFMatcher para encontrar las coincidencias
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Encontrar coincidencias
matches = bf.match(descriptors1, descriptors2)

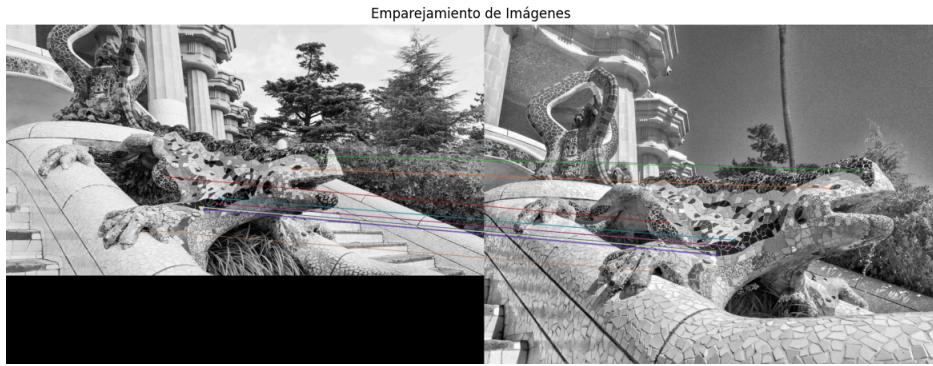
# Ordenar las coincidencias en orden de distancia (cuanto más baja, mejor)
matches = sorted(matches, key=lambda x:x.distance)

# Dibujar las primeras 15 coincidencias
matched_img = cv2.drawMatches(image1, keypoints1, image2, keypoints2, matches[:15], outImg=None, flags=2)

# Ajustar el tamaño de la figura de matplotlib
plt.figure(figsize=(15, 20))
plt.imshow(cv2.cvtColor(matched_img, cv2.COLOR_BGR2RGB))
plt.title('Emparejamiento de Imágenes')
plt.axis('off')
plt.show()
```

En este código, después de obtener los descriptoros para ambas imágenes, utilizamos un `BFMatcher` para encontrar las mejores coincidencias entre los descriptoros. Este es un método de coincidencias por "fuerza bruta".

Luego, dibujamos estas coincidencias en las imágenes para visualizar cómo los *keypoints* de una imagen corresponden a los de la otra. Esto es particularmente útil para entender cómo las características de una imagen coinciden con las de otra, lo cual es una tarea fundamental en muchas aplicaciones de visión por computadora. Con el ejemplo, obtendremos un resultado como el siguiente:



Allí vemos como se encontraron varias coincidencias en las imágenes, a pesar de ser fotos diferentes.

Veamos ahora un método alternativo de emparejamiento (matching), basado en el método FLANN, que resulta más eficiente que un método de fuerza bruta, sobre todo cuando se deben analizar gran cantidad de descriptores:

```

!wget "https://raw.githubusercontent.com/jpmanson/tuia-unr/main/images/box.png" -O box.png
!wget "https://raw.githubusercontent.com/jpmanson/tuia-unr/main/images/box_in_scene.png" -O box_in_scene.png

import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

# Cargar imágenes en escala de grises
img1 = cv.imread('box.png', cv.IMREAD_GRAYSCALE)      # Imagen de consulta (queryImage)
img2 = cv.imread('box_in_scene.png', cv.IMREAD_GRAYSCALE) # Imagen de entrenamiento (trainImage)

# Iniciar el detector SIFT
sift = cv.SIFT_create()

# Encontrar los keypoints y descriptores con SIFT
kp1, des1 = sift.detectAndCompute(img1, None)           # Imagen de consulta (queryImage)
kp2, des2 = sift.detectAndCompute(img2, None)           # Imagen de entrenamiento (trainImage)

# Parámetros FLANN
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50) # o pasar un diccionario vacío

# Iniciar FLANN matcher
flann = cv.FlannBasedMatcher(index_params, search_params)

# Realizar emparejamientos kNN
matches = flann.knnMatch(des1, des2, k=2)

# Necesitamos dibujar solo los buenos emparejamientos, por lo que creamos una máscara
matchesMask = [[0, 0] for i in range(len(matches))]

# Test de proporción según el artículo de Lowe
# https://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/lowe_ijcv2004.pdf
for i, (m, n) in enumerate(matches):
    if m.distance < 0.7 * n.distance:
        matchesMask[i] = [1, 0]

# Parámetros de dibujo para los emparejamientos
draw_params = dict(matchColor=(0, 255, 0), # color de los emparejamientos
                   singlePointColor=(255, 0, 0), # color de los puntos individuales

```

```

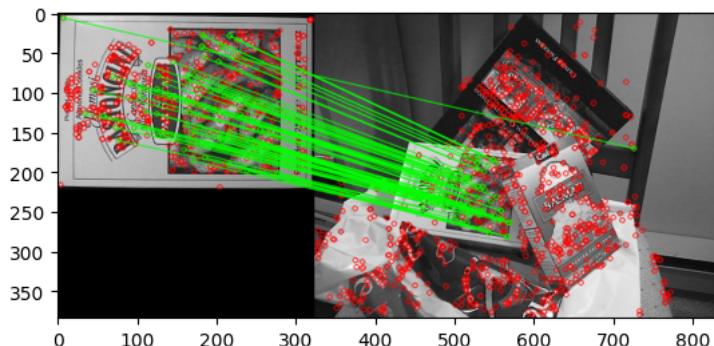
matchesMask=matchesMask, # máscara para seleccionar emparejamientos
flags=cv.DrawMatchesFlags_DEFAULT)

# Dibujar los emparejamientos kNN
img3 = cv.drawMatchesKnn(img1, kp1, img2, kp2, matches, None, **draw_params)

# Mostrar la imagen resultante
plt.imshow(img3), plt.show()

```

El resultado será el siguiente:



Aspectos fundamentales del código:

- Carga de Imágenes:** Las imágenes se cargan en escala de grises, lo cual es común en la detección de características, ya que muchos algoritmos operan mejor en una sola intensidad de color.
- Detector SIFT:** Se utiliza el detector y descriptor SIFT para encontrar *keypoints* y sus descriptores en ambas imágenes. SIFT es efectivo para detectar características que son invariantes a la escala y la rotación.
- Matcher FLANN:** Se usa FLANN (Fast Library for Approximate Nearest Neighbors) para encontrar emparejamientos entre los descriptores de las dos imágenes. FLANN es más rápido que los métodos de fuerza bruta, especialmente en grandes conjuntos de datos.

El ejemplo nos muestra como podemos encontrar un objeto de referencia, en otra imagen, aunque el objeto se encuentre en otra perspectiva y parcialmente oculto.

Más información:

<https://medium.com/@deepanshut041/introduction-to-feature-detection-and-matching-65e27179885d>

<http://csundergrad.science.uoit.ca/courses/cv-notes/notebooks/11-local-features.html>
Libro "Feature Extraction and Image Processing for Computer Vision" (Mark S. Nixon, Alberto S. Aguado).

Aplicaciones

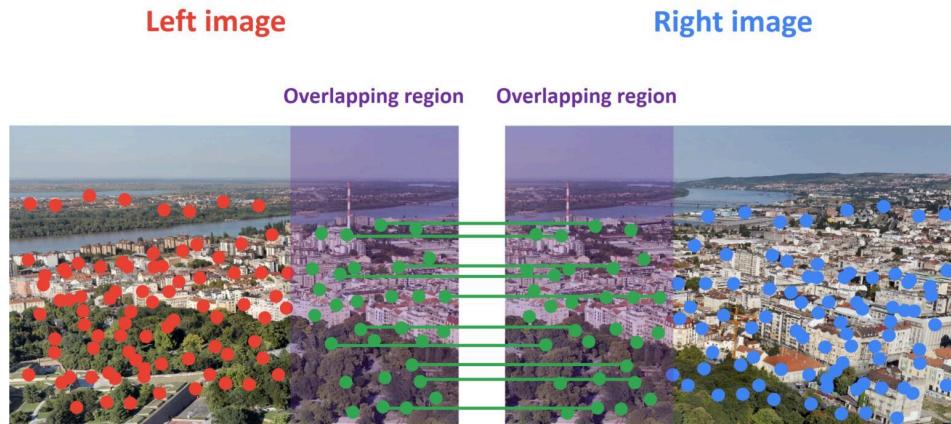
La extracción y el emparejamiento de características (feature extraction and matching) son procesos clave en la visión por computadora y tienen una amplia gama de aplicaciones importantes. Aquí enumeramos algunas de las más destacadas:

- Reconocimiento de Objetos:** Identificar y clasificar objetos dentro de una imagen. Es fundamental en áreas como la automatización industrial, la robótica, o seguridad.
- Emparejamiento de Imágenes:** Comparar y encontrar similitudes entre dos imágenes, utilizado en aplicaciones como la búsqueda de imágenes duplicadas o la construcción de panoramas.

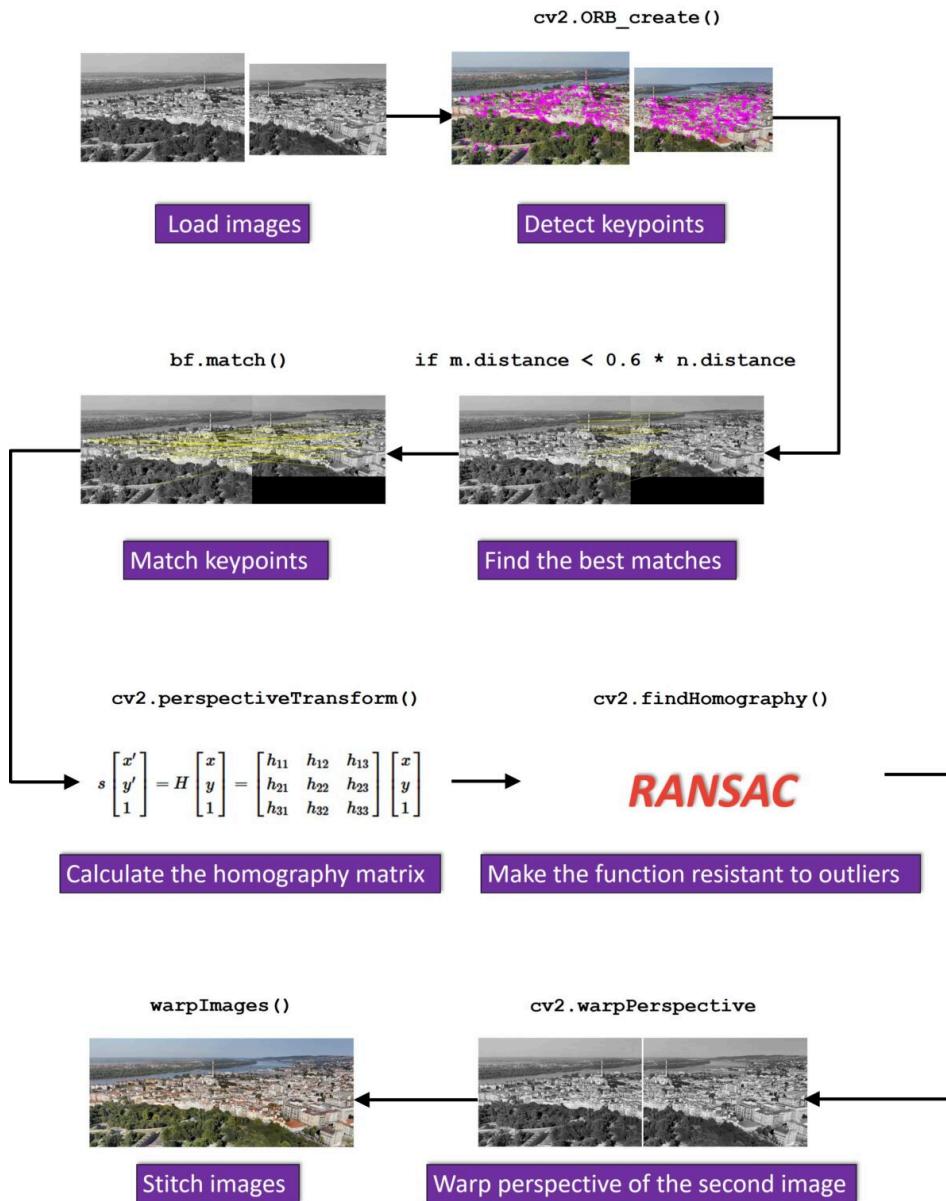
3. **Seguimiento de Objetos:** Seguir el movimiento de un objeto a lo largo del tiempo en una secuencia de video. Esencial para la vigilancia, el seguimiento deportivo y la interfaz de usuario basada en gestos.
4. **Navegación y Mapeo para Robots (SLAM):** Ayuda a los robots y vehículos autónomos a mapear su entorno y navegar de forma autónoma.
5. **Realidad Aumentada:** Superponer información virtual en el mundo real, donde el emparejamiento de características permite un posicionamiento preciso de los elementos virtuales.
6. **Reconocimiento Facial y Biométrico:** Identificar o verificar personas basándose en rasgos faciales u otras características biométricas, ampliamente utilizado en seguridad y aplicaciones móviles.
7. **Inspección Industrial:** Detectar defectos o realizar control de calidad en procesos de fabricación mediante la comparación de imágenes de productos.
8. **Análisis Médico de Imágenes:** Ayudar en el diagnóstico médico mediante la comparación y análisis de imágenes médicas, como radiografías o escaneos por MRI.

Panoramas

Una de las aplicaciones típicas es la creación de "panoramas" o el stitching de imágenes, que permiten solapar una o más imágenes, para crear una imagen más grande con la fusión de ellas. Veamos un ejemplo, de como el feature extraction y matching nos puede ayudar con la tarea:



Con las técnicas que hemos visto, podemos encontrar los keypoints en común de ambas imágenes. El pipeline para fusionar las dos imágenes, sería el siguiente:



Fuente: <https://datahacker.rs/005-how-to-create-a-panorama-image-using-opencv-with-python/>

Luego, el resultado final, es la fusión de las dos imágenes en una sola:

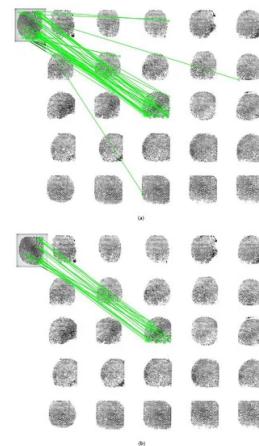
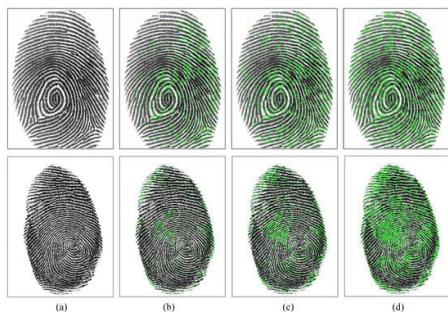


Si bien el ejemplo se vale de algunas técnicas de transformación de perspectiva de la imagen con OpenCV, para lograr una coincidencia (`perspectiveTransform()`, `findHomography()` y `warpPerspective()`), la aplicación es posible gracias a la extracción de

keypoints y descriptores de las imágenes originales.

Identificación biométrica

La extracción y el emparejamiento de características juegan un papel fundamental en la lectura y análisis de huellas digitales. Este proceso es una parte esencial de los sistemas de identificación biométrica, donde las huellas digitales se utilizan para identificar o verificar la identidad de un individuo. Las huellas digitales contienen patrones únicos formados por crestas y valles. Los sistemas de identificación de huellas digitales extraen características clave, conocidas como minutiae, que incluyen puntos donde las crestas terminan (terminaciones) y puntos donde las crestas se dividen (bifurcaciones). Estas minutiae son únicas para cada huella digital y son utilizadas como puntos de referencia para la identificación:



Fuente: Robust Fingerprint Minutiae Extraction and Matching Based on Improved SIFT Features (<https://www.mdpi.com/2076-3417/12/12/6122>)

Podemos ver un ejemplo de implementación de identificación biométrica en Python en el siguiente notebook:
https://github.com/vasantvohra/Fingerprint-Recognition/blob/master/Fingerprint_Recognition.ipynb

2. Motion Detection (detección de movimiento)

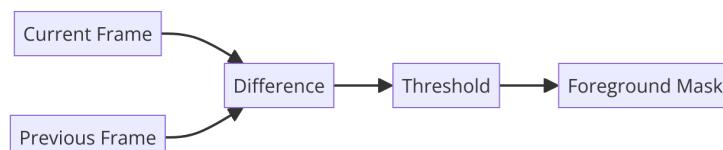
Introducción y conceptos teóricos

La Detección de Movimiento es el proceso de detectar objetos en movimiento dentro de una secuencia de video. En Visión por Computadora, es el proceso de detectar un cambio a nivel de píxeles a través de los cuadros del video. Podemos usarlo para descubrir nuevos objetos en el mundo real e incluso realizar detección de objetos agnóstica de clase, lo cual es extremadamente útil en análisis geoespacial, vigilancia, monitoreo de tráfico entre otros.

Existen diferentes métodos para la detección de movimiento, como por ejemplo Frame Difference, Optical Flow o Background Subtraction. A continuación veremos cada uno de ellos.

Frame Difference (Diferencia de fotograma)

La diferenciación de fotogramas o “cuadros” es simplemente sustraer la imagen actual de la anterior en la secuencia de video:



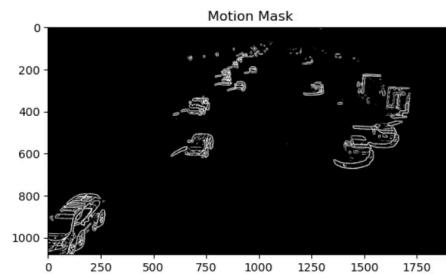
Un supuesto clave para este método es que la cámara debe estar fija; si la cámara se mueve, tendremos que emplear técnicas más avanzadas como la compensación de movimiento de la cámara para determinar con precisión qué objetos se están moviendo.

Veamos un ejemplo de lo que sucede al sustraer un cuadro de video (frame) del cuadro anterior:

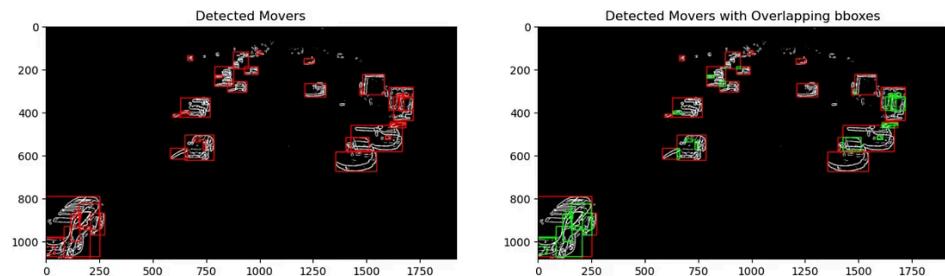


A la izquierda y centro tenemos las imágenes originales de video que vamos a restar, y a la derecha el resultado de la operación de sustracción. Como podemos ver, al realizar esta operación, se destacan los píxeles diferentes de las dos imágenes, tornándose más intensos aquellos que son más diferentes. Si los objetos estarían quietos en la imagen, no habría diferencia.

Para completar el proceso, se aplica un filtro de umbral (threshold), para obtener una máscara que destaca los puntos diferentes. Con una imagen binarizada, se facilita luego trabajar con otros filtros de morfología:



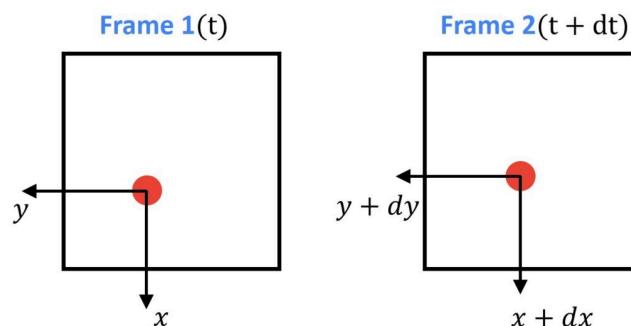
Vemos entonces que esta técnica puede ser útil para detectar objetos que se mueven en una escena, ya que podemos trabajar luego sobre la imagen de la diferencia con otras herramientas de procesamiento de imágenes:



Fuente: https://github.com/itberrios/CV_projects/blob/main/motion_detection/detection_with_frame_differencing.ipynb

Optical Flow (flujo óptico)

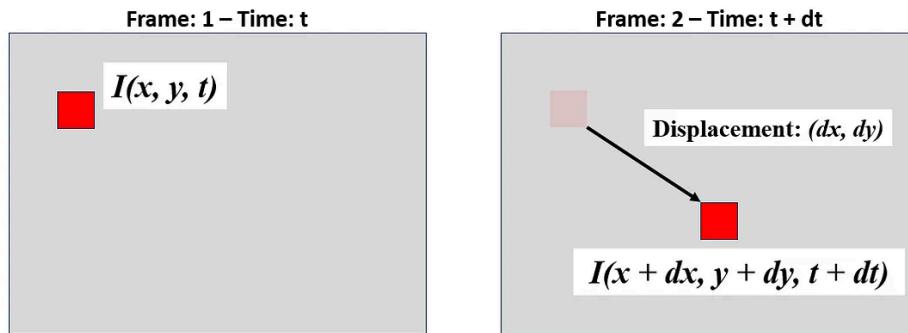
El flujo óptico es el movimiento de objetos entre cuadros consecutivos de una secuencia, causado por el movimiento relativo entre el objeto y la cámara.



Observemos un solo píxel en el primer cuadro, que se encuentra en la ubicación (x,y) . Dado que estamos procesando una secuencia de video, necesitamos introducir una tercera dimensión a esta función, que es el tiempo t . Por lo tanto, el valor de intensidad en este píxel particular ahora se puede denotar como $f(x,y,t)$.

Este píxel se desplaza en el segundo cuadro en la dirección x por una distancia dx ; en la dirección y por una distancia dy ; y en la dirección t por una distancia dt .

Horn y Schunck, propusieron un método para calcular el flujo óptico, basándose en la suposición de que la intensidad de un píxel permanece constante entre dos cuadros consecutivos de un video. Esto implica que, si analizamos dos cuadros consecutivos, podemos asumir que el brillo de los objetos no cambia significativamente entre ellos. El problema del flujo óptico puede expresarse como:



La ecuación implícita en la imagen se basa en la suposición de constancia de brillo, lo que significa que la intensidad de un píxel en la posición (x, y) en el tiempo t , denotada como $I(x, y, t)$, es igual a la intensidad del mismo píxel después de desplazarse a la nueva posición $(x + dx, y + dy)$ en el tiempo $t + dt$, es decir, $I(x + dx, y + dy, t + dt)$.

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

Para derivar una ecuación útil, se realiza una aproximación lineal (usando una expansión de Taylor) que lleva a la ecuación del flujo óptico:

$$I_x \cdot u + I_y \cdot v + I_t = 0$$

donde:

- I_x y I_y son las derivadas parciales de la intensidad de la imagen con respecto a x e y (gradientes espaciales),
- I_t es la derivada parcial de la intensidad con respecto al tiempo t (gradiente temporal),
- $u = dx/dt$ y $v = dy/dt$ son las componentes del flujo óptico (velocidades en las direcciones x e y).

Estimar el flujo óptico es una tarea difícil. Un problema fundamental es que la ecuación anterior tiene dos incógnitas (u y v), pero solo una ecuación por píxel, lo que se conoce como el "problema de apertura". Esto significa que no podemos determinar de manera única el desplazamiento de un píxel basándonos únicamente en la información local de intensidad. Para resolver este problema, Horn y Schunck introdujeron una suposición adicional de suavidad, asumiendo que el flujo óptico varía de manera suave en la imagen, lo que permite resolver el sistema mediante un enfoque de optimización global.

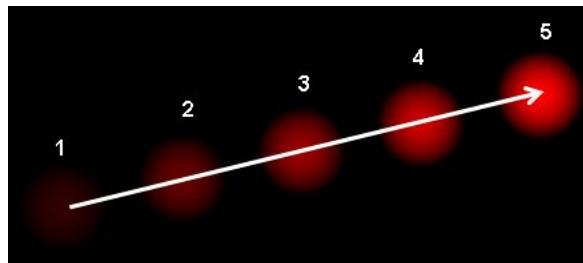
Otra suposición clave es que el desplazamiento de los píxeles (dx, dy) y la diferencia de tiempo dt entre los dos cuadros son lo suficientemente pequeños para que la aproximación lineal sea válida. Además, se asume que la intensidad o brillo de los píxeles permanece constante durante el movimiento. Si el brillo de un objeto cambia (por ejemplo, debido a cambios en la iluminación), el problema se vuelve mucho más complejo.



Los vectores u y v representan las **componentes del vector de movimiento** en los ejes horizontal y vertical, respectivamente, para cada píxel entre dos frames consecutivos.

Posición de la cámara

Considere un clip de cinco cuadros de una pelota moviéndose desde la parte inferior izquierda del campo de visión hasta la parte superior derecha. Las técnicas de estimación de movimiento pueden determinar que en un plano bidimensional la pelota se está moviendo hacia arriba y hacia la derecha, y vectores que describen este movimiento pueden extraerse de la secuencia de cuadros.



Fuente: http://en.wikipedia.org/wiki/Optical_flow

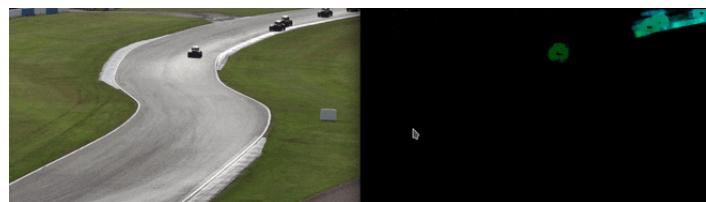
Para los fines de la compresión de video (por ejemplo, MPEG), la secuencia ahora está descrita tan bien como necesita estarlo. Sin embargo, en el campo de la visión artificial, la pregunta de si la pelota se está moviendo hacia la derecha o si el observador se está moviendo hacia la izquierda es información incognoscible pero crítica. Ni siquiera si un fondo estático y estampado estuviera presente en los cinco cuadros, podríamos afirmar con confianza que la pelota se estaba moviendo hacia la derecha, porque el patrón podría tener una distancia infinita al observador.

Flujo óptico denso y disperso

El flujo óptico disperso calcula el flujo óptico solo para una pequeña cantidad de píxeles de una imagen. Estos píxeles suelen ser puntos de interés, como bordes u objetos reconocibles. Los algoritmos de flujo óptico disperso son más simples y rápidos que los algoritmos de flujo óptico denso, pero también son menos precisos. El flujo óptico denso calcula el flujo óptico para cada píxel de una imagen. Esto proporciona una descripción completa del movimiento de la escena. Veamos ejemplos con imágenes:



El flujo óptico disperso proporciona los vectores de flujo de algunas "características interesantes" (digamos, algunos píxeles que representan los bordes o esquinas de un objeto) dentro del cuadro.



El flujo óptico denso ofrece los vectores de flujo de todo el cuadro (todos los píxeles) - hasta un vector de flujo por píxel. El flujo óptico denso tiene una mayor precisión a costa de ser lento/costoso computacionalmente.

Algoritmos de flujo óptico

Algunos de los algoritmos más importantes y ampliamente utilizados para calcular el flujo óptico son:

1. **Lucas-Kanade Method**: Un método basado en el supuesto de que el flujo es esencialmente constante en una ventana local pequeña de píxeles. Este es un enfoque disperso que calcula el flujo óptico solo para un conjunto de puntos de interés. (ver [ejemplo](#)).
2. **Horn-Schunck Method**: Un método global que introduce una constante de suavizado para imponer suavidad en el flujo óptico a través de todo el cuadro. Este enfoque es más adecuado para entender el movimiento en todas las áreas de la imagen.
3. **Farneback Method**: Este método calcula el flujo óptico utilizando una aproximación polinomial de las características de la imagen, lo que permite un cálculo del flujo óptico denso que da un vector de flujo para cada píxel. (ver [ejemplo](#)).
4. **SimpleFlow**: Un algoritmo que utiliza correspondencias de píxeles entre cuadros basadas en una búsqueda de ventana fija, enfocado en calcular un flujo óptico denso.

5. **DeepFlow**: Un algoritmo que combina la correspondencia de píxeles con una red neuronal profunda, proporcionando una estimación de flujo óptico denso y preciso.
6. **TV-L1**: Un enfoque basado en la minimización de una función de costo que incluye términos de suavizado y de fidelidad de datos, que es especialmente eficaz para capturar detalles finos y movimientos rápidos.
7. **Brox Method**: Este método utiliza técnicas basadas en el gradiente y el flujo óptico en varias escalas para mejorar la robustez y precisión.
8. **PCA-Flow y PCA-Layers**: Estos métodos utilizan el análisis de componentes principales (PCA) para reducir la dimensión del problema del flujo óptico y calcular el flujo de manera más eficiente.
9. **Flujo óptico profundo (Deep Optical Flow)**: Los métodos modernos emplean redes neuronales convolucionales profundas para predecir el flujo óptico. Ejemplos destacados incluyen FlowNet, FlowNet 2.0 y SpyNet, que aprenden a estimar el flujo óptico a partir de grandes cantidades de datos.
10. **RAFT**: (Recurrent All-Pairs Field Transforms) es uno de los métodos más recientes y avanzados para calcular el flujo óptico en secuencias de video. RAFT se destaca por su precisión y su capacidad para capturar movimientos complejos y finos. Utiliza una CNN con una arquitectura recurrente que itera para refinar progresivamente la estimación del flujo óptico.

Más información:

<https://github.com/antran89/awesome-optical-flow-algorithm>

Papers with Code - Optical Flow Estimation

Optical Flow Estimation is a computer vision task that involves computing the motion of objects in an image or a video sequence. The goal of optical flow estimation is to determine the movement of pixels or features in the image, which can be used for various applications such as object tracking,

<https://paperswithcode.com/task/optical-flow-estimation>

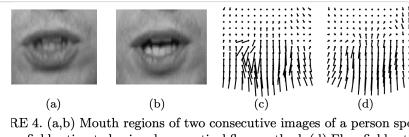


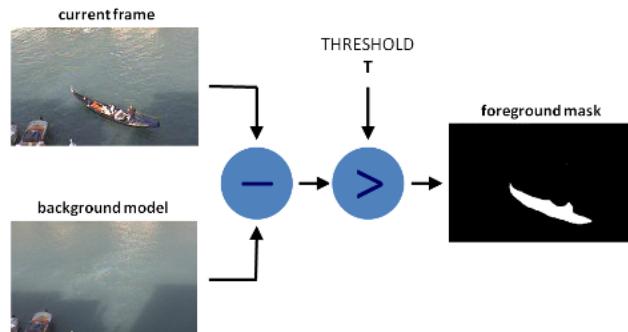
FIGURE 4. (a,b) Mouth regions of two consecutive images of a person with a learned flow field estimated using dense optical flow method. (d) Flow field estimated by the learned model with 6 basis flow fields. (After [16])

Background Subtraction (Sustracción de fondo)

La sustracción de fondo es una técnica ampliamente utilizada en visión por computadora y procesamiento de imágenes. Esta técnica tiene como objetivo detectar objetos en movimiento en una secuencia de fotogramas capturados por una cámara estática. Permite extraer el primer plano (objeto en movimiento) y el fondo (objeto estacionario) de una imagen para su posterior procesamiento, como el reconocimiento de objetos. Muchas aplicaciones, como las cámaras de seguridad, no necesitan toda la escena del video para generar una alerta, solo los movimientos.

La premisa principal de la sustracción de fondo es que el fondo siempre es estático, por lo que no se puede usar en escenarios con un entorno dinámico. La idea subyacente es restar el fondo del primer plano, ayudando a identificar los objetos en movimiento. Las técnicas de sustracción de fondo suelen devolver una máscara que se considera el primer plano en la secuencia relativa.

Una de las técnicas más comúnmente utilizadas es la diferenciación de fotogramas, que toma la diferencia absoluta de fotogramas consecutivos para detectar cambios de movimiento entre ellos. Despues de la diferenciación, se utiliza un umbral para seleccionar solo los cambios relevantes en fotogramas sucesivos. El proceso es el siguiente:



Por ejemplo, si necesitamos detectar automóviles en movimiento de una secuencia de video, la imagen resultante de la operación descrita anteriormente los extraería todos. Los resultados pueden mejorarse y limpiarse aún más aplicando operaciones morfológicas a la misma.

```
ap = cv2.VideoCapture(video)
ret, prev = cap.read()
prev = cv2.cvtColor(prev, cv2.COLOR_BGR2GRAY)
while(1):
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame_diff = cv2.absdiff(gray, prev)
    ret, thres = cv2.threshold(frame_diff, 35, 255, cv2.THRESH_BINARY)

    prev = gray.copy()
    cv2.imshow('original',frame)
    cv2.imshow('foregroundMask',thres)
```



Sustracción de fondo utilizando Frame Difference ([Fuente](#))

La diferencia de fotogramas es una técnica bastante básica y es sensible a su valor de umbral. Basándose en los mismos principios, se introdujeron algunas técnicas avanzadas. OpenCV ofrece muchos modelos de sustracción de fondo con mayor precisión para cumplir el mismo propósito. Los modelos de fondo disponibles en OpenCV consisten en dos pasos principales:

1. Inicialización del Fondo: Se genera un modelo inicial de fondo.
2. Actualización del Fondo: Posteriormente, este modelo se actualiza para adaptarse a posibles cambios en la escena. La única diferencia entre estos algoritmos es cómo eligen actualizar su información de fondo.

Un ejemplo es el modelo de mezclas gaussianas ([Gaussian mixture model](#)), un modelo probabilístico que asume que todos los puntos de datos se generan a partir de una mezcla de un número finito de distribuciones gaussianas con parámetros desconocidos. Considera que diferentes distribuciones representan los diversos colores del fondo y del primer plano. El peso de cada una de estas distribuciones en el modelo es proporcional al tiempo que cada color permanece en ese píxel. Por lo tanto, cuando el peso de la distribución de un píxel es bajo, ese píxel se clasifica como píxel de primer plano (es el que cambió más recientemente).

Además de esto, [OpenCV proporciona varios métodos de sustracción de fondo](#), como:

- [BackgroundSubtractorCNT](#)
- [BackgroundSubtractorGMG](#)
- [BackgroundSubtractorGSOC](#)
- [BackgroundSubtractorLSBP](#)
- [BackgroundSubtractorMOG](#)
- [BackgroundSubtractorKNN](#)

Estos modelos, se pueden utilizar muy fácilmente, por ejemplo:

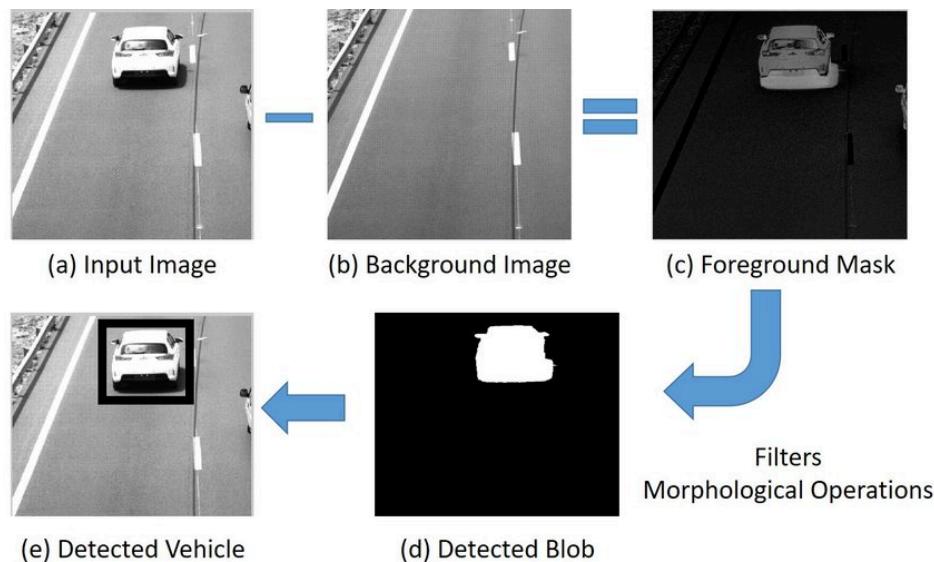
```
cap = cv2.VideoCapture(video)
backSub = cv.createBackgroundSubtractorMOG2(detectShadows=False)
while(1):
    ret, frame = cap.read()
    fgMask = backSub.apply(frame)
```

```

cv2.imshow('original',frame)
cv2.imshow('foregroundMask',fgMask)

```

Cada uno de estos métodos utiliza una estrategia diferente para actualizar su imagen de fondo y funciona bien en distintos escenarios. La sustracción de fondo es una forma fácil de extraer objetos del primer plano de secuencias de video, que luego pueden procesarse para extraer información significativa. Los resultados de estos métodos dependen de la secuencia de video, que pueden mejorarse con operaciones morfológicas.



Ejemplos prácticos

Frame Difference

La siguiente función está diseñada para resaltar las diferencias entre dos fotogramas consecutivos de un video. La metodología empleada en esta función incluye la conversión de imágenes a escala de grises, la obtención de la diferencia absoluta entre fotogramas y su posterior normalización. Esto permite una comparación efectiva y destacada de los cambios en el contenido del video. Además, se utiliza umbralización para enfatizar áreas con cambios significativos, facilitando la identificación visual de las diferencias:

```

# Función actualizada para realizar diferencia de fotogramas con normalización:
def process_frame_difference(new_image, prev_image, **kwargs):
    # Convertir las imágenes a escala de grises
    new_gray = cv2.cvtColor(new_image, cv2.COLOR_RGB2GRAY)
    prev_gray = cv2.cvtColor(prev_image, cv2.COLOR_RGB2GRAY)

    # Calcular la diferencia absoluta entre los fotogramas actual y anterior
    frame_diff = cv2.absdiff(new_gray, prev_gray)

    # Normalizar la imagen de diferencia
    norm_diff = cv2.normalize(frame_diff, None, 0, 255, cv2.NORM_MINMAX)

    # Umbralizar la imagen para resaltar las diferencias
    _, thresh = cv2.threshold(norm_diff, 30, 255, cv2.THRESH_BINARY)

    # Convertir la imagen umbralizada a color para mantener la consistencia con el video original
    thresh_color = cv2.cvtColor(thresh, cv2.COLOR_GRAY2RGB)

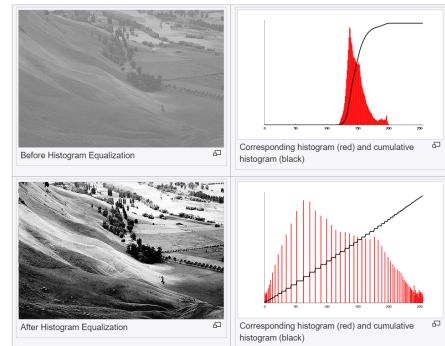
    return thresh_color

```

`cv2.absdiff` es una función de la librería OpenCV que se utiliza para calcular la diferencia absoluta entre dos imágenes o matrices. La operación se realiza píxel por píxel, restando los valores de intensidad correspondientes en las dos imágenes y tomando el valor absoluto del resultado. (ver [Ejemplos](#))

Luego, realizamos una normalización y umbralizado (thresholding):

La normalización ajusta la intensidad de los píxeles de una imagen para que el rango de sus valores se extienda sobre un rango específico, típicamente de 0 a 255 para imágenes de 8 bits. La normalización es útil para mejorar la visibilidad de las características importantes y para estandarizar las imágenes antes de realizar comparaciones o análisis posteriores.



https://en.wikipedia.org/wiki/Histogram_equalization

El umbralizado, convierte una imagen en escala de grises a una imagen binaria, donde los píxeles se asignan a uno de dos colores (generalmente negro o blanco) basándose en si su intensidad es mayor o menor que un valor umbral especificado. Esta binarización se realiza con `cv2.threshold` y es ampliamente utilizada para separar objetos de interés del fondo o para resaltar áreas específicas de una imagen.

El resultado de correr todo el procesamiento, será el siguiente:



Ahora haremos un ejemplo más completo, incorporando una operación morfológica (`cv2.dilate`) que ayuda a dilatar y cerrar los objetos de la imagen, y luego `cv2.findContours`, para hallar los contornos para graficar los rectángulos de los objetos:

```
# Función actualizada para detectar movimientos y dibujar cuadros delimitadores:  
def process_frame_difference_full(new_image, prev_image, **kwargs):  
    # Convertir las imágenes a escala de grises  
    new_gray = cv2.cvtColor(new_image, cv2.COLOR_RGB2GRAY)  
    prev_gray = cv2.cvtColor(prev_image, cv2.COLOR_RGB2GRAY)  
  
    # Calcular la diferencia absoluta entre los fotogramas actual y anterior  
    frame_diff = cv2.absdiff(new_gray, prev_gray)  
  
    # Normalizar la imagen de diferencia  
    norm_diff = cv2.normalize(frame_diff, None, 0, 255, cv2.NORM_MINMAX)  
  
    # Umbralizar la imagen para resaltar las diferencias
```

```

_, thresh = cv2.threshold(norm_diff, 30, 255, cv2.THRESH_BINARY)

# Dilatar la imagen umbralizada para mejorar la detección de contornos
kernel = np.ones((5,5),np.uint8)
dilated = cv2.dilate(thresh, kernel, iterations = 1)

# Convertir la imagen dilatada a formato adecuado para findContours
dilated = dilated.astype(np.uint8)

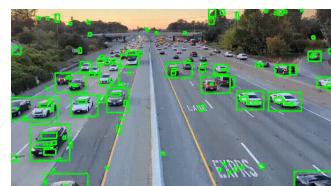
# Encontrar contornos en la imagen dilatada
contours, _ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Dibujar cuadros delimitadores alrededor de los contornos
if kwargs.get('draw_mode', 0) == 0:
    result_image = draw_contours(new_image, contours)
elif kwargs.get('draw_mode', 0) == 1:
    result_image = draw_contours(thresh, contours)

return result_image

```

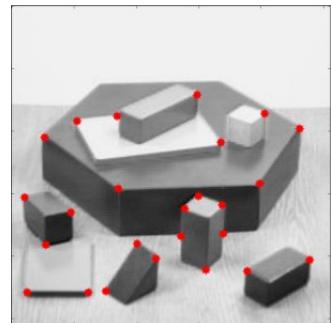
Esta función de procesamiento, nos permite obtener los siguientes resultados:



Optical Flow

El código proporcionado a continuación, implementa un proceso para rastrear el movimiento entre dos imágenes utilizando los métodos de Shi-Tomasi y Lucas-Kanade en el análisis de flujo óptico **disperso**.

El método de [Shi-Tomasi \(paper\)](#), utilizado para la detección de características, identifica puntos clave en la primera imagen que son buenos para seguir. Estos puntos suelen ser esquinas o regiones distintivas que se destacan bien entre un fotograma y otro. Esta detección es fundamental para el seguimiento eficaz del movimiento.



https://docs.opencv.org/4.x/d4/d8c/tutorial_py_shi_tomasi.html

Una vez identificados estos puntos, se utiliza el [método de Lucas-Kanade](#) para el seguimiento de flujo óptico. Este método calcula el movimiento de los puntos detectados por Shi-Tomasi entre el fotograma actual y el anterior. A través de este enfoque, se puede estimar el movimiento de partes específicas de la imagen, permitiendo un seguimiento detallado y preciso.

```

def process_sparse_optical_flow(new_image, prev_image):
    # Preparamos las imágenes de trabajo
    new_gray = cv2.cvtColor(new_image, cv2.COLOR_BGR2GRAY)
    prev_gray_image = cv2.cvtColor(prev_image, cv2.COLOR_BGR2GRAY)

```

```

# Verificar si ya se han detectado las características de Shi-Tomasi
if not hasattr(process_sparse_optical_flow, "shi_tomasi_done"):
    # Definir parámetros para la detección de esquinas de Shi-Tomasi
    feature_params = dict(maxCorners=300, qualityLevel=0.2, minDistance=2, blockSize=7)
    # Detectar puntos característicos en la imagen
    process_sparse_optical_flow.prev_points = cv2.goodFeaturesToTrack(new_gray, mask=None, **feature_params)
    # Crear una máscara para dibujar el flujo óptico
    process_sparse_optical_flow.mask = np.zeros_like(new_image)
    # Marcar que se ha completado la detección de Shi-Tomasi
    process_sparse_optical_flow.shi_tomasi_done = True

# Continuar si se ha completado la detección de Shi-Tomasi
if process_sparse_optical_flow.shi_tomasi_done:
    prev_points = process_sparse_optical_flow.prev_points
    mask = process_sparse_optical_flow.mask

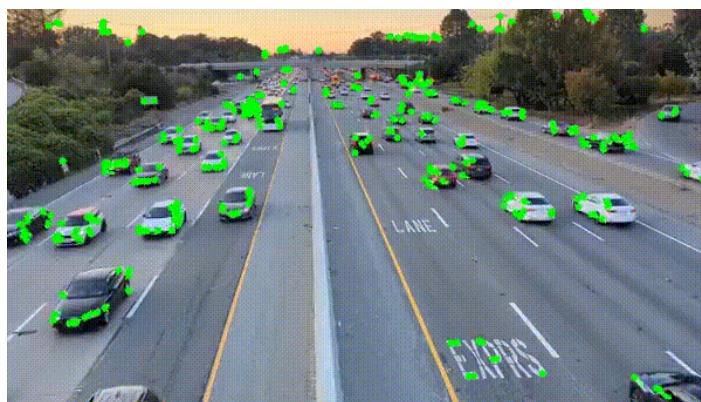
# Parámetros para el flujo óptico de Lucas-Kanade
lk_params = dict(winSize=(15, 15), maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Calcular el flujo óptico de Lucas-Kanade
new_points, status, error = cv2.calcOpticalFlowPyrLK(prev_gray_image, new_gray, prev_points, None, **lk_params)
# Filtrar puntos buenos
good_old = prev_points[status == 1]
good_new = new_points[status == 1]
color = (0, 255, 0) # Color para el dibujo
# Dibujar el movimiento (flujo óptico)
for i, (new, old) in enumerate(zip(good_new, good_old)):
    a, b = new.astype(int).ravel()
    c, d = old.astype(int).ravel()
    mask = cv2.line(mask, (a, b), (c, d), color, 2)
    new_image = cv2.circle(new_image, (a, b), 3, color, -1)

# Combinar la imagen actual con las líneas de flujo óptico dibujadas
output = cv2.add(new_image, mask)
# Actualizar puntos para el siguiente cuadro
process_sparse_optical_flow.prev_points = good_new.reshape(-1, 1, 2)
return output

```

El resultado de aplicar esa función de procesamiento, será el siguiente:



El siguiente código implementa el análisis de flujo óptico **denso** utilizando el [método de Farneback](#), un enfoque popular y efectivo en la visión por computadora. El método de Farneback se destaca por su capacidad para calcular un mapa

de flujo óptico que cubre cada píxel de la imagen, proporcionando una visión detallada del movimiento entre dos cuadros consecutivos.

El proceso comienza convirtiendo las imágenes a escala de grises para simplificar el cálculo del flujo óptico. Luego, se utiliza `cv2.calcOpticalFlowFarneback` para estimar el flujo óptico entre el cuadro actual y el anterior. Este flujo se representa a través de la magnitud y el ángulo de los vectores en cada punto de la imagen, lo que permite visualizar el movimiento en la escena.

El resultado se visualiza en una imagen en color, donde el tono y el valor representan la dirección y la magnitud del movimiento, respectivamente. Este tipo de análisis es útil en aplicaciones como la vigilancia de video, la navegación robótica y el análisis de video en general.

```
# Función para procesar el flujo óptico denso
def process_dense_optical_flow(new_image, prev_image):
    # Convierte la nueva imagen a escala de grises
    gray = cv2.cvtColor(new_image, cv2.COLOR_BGR2GRAY)

    # Inicialización en el primer frame
    if not hasattr(process_dense_optical_flow, "init_done"):
        process_dense_optical_flow.prev_gray = cv2.cvtColor(new_image, cv2.COLOR_BGR2GRAY)
        process_dense_optical_flow.mask = np.zeros_like(new_image)
        process_dense_optical_flow.mask[:, :, 1] = 255 # Saturación completa en HSV
        process_dense_optical_flow.init_done = True

    # Si ya está inicializado, usa los valores previos
    if process_dense_optical_flow.init_done:
        prev_gray = process_dense_optical_flow.prev_gray
        mask = process_dense_optical_flow.mask

    # Calcula el flujo óptico denso con Farneback
    flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)
    # Computa magnitud y ángulo de los vectores 2D
    magnitude, angle = cv2.cartToPolar(flow[:, :, 0], flow[:, :, 1])
    # Establece el tono (hue) según la dirección del flujo óptico
    mask[:, :, 0] = angle * 180 / np.pi / 2
    # Establece el valor (value) según la magnitud, normalizado a 0-255
    mask[:, :, 2] = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX)
    # Convierte de HSV a BGR para visualización
    rgb = cv2.cvtColor(mask, cv2.COLOR_HSV2BGR)
    # Actualiza la imagen previa para el siguiente frame
    process_dense_optical_flow.prev_gray = gray.copy() # Corrección del error tipográfico
    return rgb
```

El resultado de aplicar esa función de procesamiento, será el siguiente:



Más ejemplos prácticos, podemos encontrar en los siguientes links:
Ejemplos online de Lucas-Kanade y Farneback:

https://docs.opencv.org/5.x/db/d7f/tutorial_js_lucas_kanade.html

<https://sandipanweb.wordpress.com/2018/02/25/implementing-lucas-kanade-optical-flow-algorithm-in-python/>

<https://learnopencv.com/optical-flow-in-opencv/>

<https://www.sefidian.com/2019/12/16/a-tutorial-on-motion-estimation-with-optical-flow-with-python-implementation/>

Una alternativa interesante para aplicar diversos modelos de Optical Flow basados en CNN, es la librería [ptlflow](#) (<https://ptlflow.readthedocs.io/> - <https://github.com/hmorimitsu/ptlflow>). Veamos un ejemplo de aplicación de RAFT:

```
!pip install ptlflow
!wget https://raw.githubusercontent.com/jpmanson/tuia-unr/main/images/frame_0016.png -O frame_0016.png
!wget https://raw.githubusercontent.com/jpmanson/tuia-unr/main/images/frame_0017.png -O frame_0017.png
import cv2 as cv
import ptlflow
from ptlflow.utils import flow_utils
from ptlflow.utils.io_adapter import IOAdapter
from google.colab.patches import cv2_imshow

# Obtener un modelo de flujo óptico. Como ejemplo, usaremos RAFT Small
# con pesos preentrenados en el dataset FlyingThings3D
model = ptlflow.get_model('raft_small', pretrained_ckpt='things')

# Cargar las dos imágenes
img1 = cv.imread('frame_0016.png')
img2 = cv.imread('frame_0017.png')

# Obtener un modelo inicializado de PTLFlow
model = ptlflow.get_model('raft_small', 'things')
model.eval()

# IOAdapter es un helper para transformar las dos imágenes al formato de entrada aceptado por los modelos PTLFlow
io_adapter = IOAdapter(model, img1.shape[:2])
inputs = io_adapter.prepare_inputs([img1, img2])

# Procesar las entradas para obtener las predicciones del modelo
predictions = model(inputs)

# Visualizar el flujo óptico predicho
flow = predictions['flows'][0, 0] # Eliminar las dimensiones de lote y secuencia
flow = flow.permute(1, 2, 0) # cambiar de formato CHW a HWC
flow = flow.detach().numpy()
flow_viz = flow_utils.flow_to_rgb(flow) # Representar el flujo en colores RGB
flow_viz = cv.cvtColor(flow_viz, cv.COLOR_BGR2RGB) # OpenCV utiliza el formato BGR
cv2_imshow(flow_viz)

# Guardar la imagen resultante
cv.imwrite('raft_output.png', flow_viz)
```

El ejemplo anterior nos permite aplicar un modelo pre-entrenado de RAFT sobre dos cuadros de imagen, extraídos de un video. Aquí vemos como el resultado de aplicar el modelo sobre dos imágenes de entrada:



La imagen obtenida (derecha), refleja con una imagen de calor, la intensidad de movimiento de las diferentes partes de la entrada. En violeta, se muestra el brazo que tiene el mayor desplazamiento de un cuadro a otro.

Background Subtraction

La siguiente función permite elegir entre varios métodos de sustracción de fondo, como CNT, GMG, GSOC, LSBP, KNN, y MOG. Su principal aplicación es separar objetos en movimiento del fondo en secuencias de video. Además, incluye operaciones de umbralización, erosión, dilatación y detección de contornos para mejorar la segmentación y visualización de los objetos detectados:

```
def process_back_sub(new_image, prev_image, **kwargs):
    # Inicializar el sustractor de fondo si aún no se ha hecho
    if not hasattr(process_back_sub, 'init_done'):
        # Obtener el método de sustracción de fondo especificado
        method = kwargs.get('method', 'MOG')
        match method:
            case 'CNT':
                process_back_sub.back_method = cv2.bgsegm.createBackgroundSubtractorCNT()
            case 'GMG':
                process_back_sub.back_method = cv2.bgsegm.createBackgroundSubtractorGMG()
            case 'GSOC':
                process_back_sub.back_method = cv2.bgsegm.createBackgroundSubtractorGSOC()
            case 'LSBP':
                process_back_sub.back_method = cv2.bgsegm.createBackgroundSubtractorLSBP()
            case 'KNN':
                process_back_sub.back_method = cv2.createBackgroundSubtractorKNN()
            case 'MOG' | 'MOG2':
                process_back_sub.back_method = cv2.createBackgroundSubtractorMOG2(detectShadows=kwargs.get('detect_shadows', False))
            case _:
                print(f'Método {method} no implementado')
        process_back_sub.init_done = True

    # Aplicar el método de sustracción de fondo seleccionado a la nueva imagen
    back_method = process_back_sub.back_method
    foreground_mask = back_method.apply(new_image)

    # Umbralizar la máscara para resaltar los objetos en movimiento
    _, thresh = cv2.threshold(foreground_mask, kwargs.get('threshold_val', 30), 255, cv2.THRESH_BINARY)

    # Aplicar erosión y dilatación para mejorar la segmentación
    kernel_size = kwargs.get('kernel_size', 5)
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    thresh = cv2.erode(thresh, kernel, iterations=kwargs.get('erode_iterations', 1))
    thresh = cv2.dilate(thresh, kernel, iterations=kwargs.get('dilate_iterations', 1))

    # Encontrar contornos
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    # Determinar qué imagen se usará para dibujar los contornos
    match result_frame:
        case 'mask':
            result_image = cv2.cvtColor(foreground_mask, cv2.COLOR_GRAY2BGR)
        case 'morph':
            result_image = cv2.cvtColor(thresh, cv2.COLOR_GRAY2BGR)
```

```

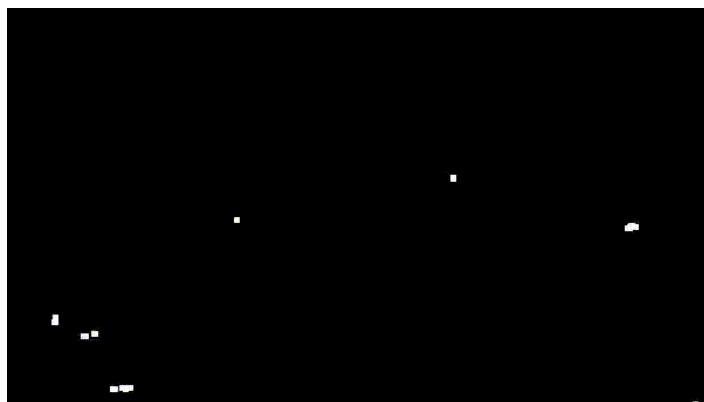
result_image = cv2.cvtColor(thresh, cv2.COLOR_GRAY2BGR)
case _.: # original
    result_image = new_image

# Dibujar cuadros delimitadores (bounding boxes)
show_boxes = kwargs.get('show_boxes', True)
if show_boxes:
    for contour in contours:
        if cv2.contourArea(contour) > kwargs.get('min_area', 100):
            x, y, w, h = cv2.boundingRect(contour)
            result_image = cv2.rectangle(result_image, (x, y), (x+w, y+h), (0, 255, 0), 2)

return result_image

```

El resultado de aplicar esa función de procesamiento usando el método CNT, será el siguiente:



Aplicaciones

Las técnicas de **Frame Difference** y **Background Subtraction** poseen muchas aplicaciones prácticas en el campo del procesamiento de imágenes:

- **Detección de Movimiento:** Identifica cambios entre fotogramas consecutivos, útil en vigilancia y seguridad.
- **Análisis de Video:** Permite analizar cambios dinámicos en videos, como en deportes o actividades al aire libre.
- **Vigilancia de Seguridad:** Detecta y rastrea objetos en movimiento en entornos estáticos, como en cámaras de seguridad.
- **Análisis de Tráfico:** Identifica y cuenta vehículos en carreteras o intersecciones.
- **Edición de Video:** Para efectos especiales, como en la técnica de pantalla verde.

Optical Flow (flujo óptico)

Las técnicas de flujo óptico tienen una variedad de usos como por ejemplo:

- **Compresión de video:** El flujo óptico permite predecir el movimiento de píxeles entre cuadros, reduciendo la cantidad de datos necesarios para describir un video.
- **Seguimiento de objetos:** Es posible rastrear objetos en movimiento en videos, útil en aplicaciones como vigilancia y vehículos autónomos.
- **Estabilización de imagen:** Utilizando el flujo óptico para detectar el movimiento de la cámara, se pueden estabilizar imágenes o videos.
- **Comprendión de escenas:** El flujo óptico ayuda a extraer información sobre la organización y movimiento de objetos en una escena.
- **Realidad aumentada:** Estimar el movimiento de la cámara y objetos reales mediante flujo óptico mejora la experiencia de realidad aumentada.

- **Interacción humano-computadora:** Juegos o sistemas interactivos pueden operarse con flujo óptico y gestos o movimientos manuales.
- **Robótica:** El flujo óptico se utiliza para estimar la velocidad de un robot y su entorno, útil para mapeo, localización y navegación.

Un ejemplo de aplicación, es la medición sin contacto de vibraciones de máquinas en entornos industriales. Por medio de optical flow es posible medir las vibraciones con una cámara, analizando la imagen de video de una máquina en funcionamiento:



Evaluación de un sistema de medición de vibraciones, basado en Optical Flow. ([Fuente](#))

3. Image classification (Clasificación de imágenes)

Introducción y conceptos teóricos

Los modelos de clasificación de imágenes son algoritmos de aprendizaje automático o inteligencia artificial diseñados para identificar y clasificar objetos o características dentro de imágenes digitales. Estos modelos son una parte fundamental de la visión por computadora, una rama de la inteligencia artificial que se ocupa de cómo las computadoras pueden ganar un alto nivel de comprensión a partir de contenido visual digital.

En términos simples, un modelo de clasificación de imágenes toma una imagen como entrada y predice la clase o categoría a la que pertenece esa imagen.



Ejemplo de un modelo clasificador de razas de gatos. Como salida, vemos las probabilidades de cada clasificación realizada por el modelo.

La evolución de las redes de clasificación de imágenes ha sido un viaje notable marcado por avances significativos en el campo de la inteligencia artificial y la visión por computadora.

Aquí hay un breve historial de los hitos clave en el desarrollo de las redes de clasificación de imágenes:

1. **Redes Neuronales (Años 50 — 80):** El concepto de redes neuronales artificiales (ANN) se remonta a los años 50, pero las limitaciones computacionales obstaculizaron su aplicación práctica. El perceptrón, una arquitectura básica de red neuronal, fue propuesto por Frank Rosenblatt a finales de los años 50. Sin embargo, las limitaciones de los perceptrones de una sola capa para tareas complejas llevaron a un declive en el interés por las redes neuronales a finales de los años 60.
2. **Retropropagación (Años 80 — 90):** En los años 80, se "redescubrió" el algoritmo de retropropagación, lo que permitió el entrenamiento eficiente de redes neuronales multicapa. Este desarrollo reavivó el interés en las redes neuronales. Sin embargo, desafíos como los gradientes desvanecientes y el sobreajuste limitaron su éxito en la clasificación de imágenes.

3. **LeNet-5 (1998)**: LeNet-5 de [Yann LeCun](#), introducido en 1998, fue una arquitectura pionera de red neuronal convolucional (CNN) diseñada para el reconocimiento de dígitos escritos a mano.

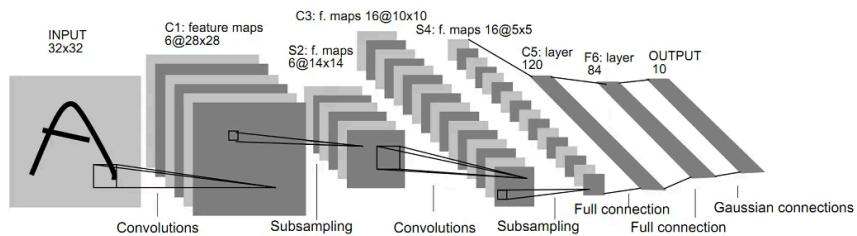


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Contenía capas convolucionales y de agrupamiento, componentes esenciales de las redes modernas de clasificación de imágenes. En este link, podemos ver un ejemplo de cómo entrenar a LeNet-5:

Google Colab

 <https://colab.research.google.com/drive/1hhI8-5buUkm2QAirUUKM51UHou1m4Dxz?usp=sharing>

4. **Resurgimiento del Aprendizaje Profundo (Años 2010)**: El momento decisivo para la clasificación de imágenes llegó a mediados de los años 2010 con el advenimiento del aprendizaje profundo y la disponibilidad de conjuntos de datos más grandes y GPUs más potentes.
5. **AlexNet (2012)**: AlexNet, desarrollada por Alex Krizhevsky y otros, ganó el Desafío de Reconocimiento Visual a Gran Escala de [ImageNet](#) (ILSVRC) en 2012. Fue una arquitectura de CNN profunda con múltiples capas convolucionales y demostró el potencial del aprendizaje profundo para tareas de clasificación de imágenes. En este modelo, en lugar de las funciones de activación Sigmoid o Tanh (usadas en LeNet), AlexNet utilizó la Unidad Lineal Rectificada ([ReLU](#)). Esto aceleró significativamente el entrenamiento y ayudó a mitigar el problema del desvanecimiento del gradiente (vanishing gradient). Además, introdujo la técnica de regularización "[Dropout](#)" en las capas totalmente conectadas. Durante el entrenamiento, algunas neuronas se "apagan" aleatoriamente, lo que obliga a la red a aprender representaciones más robustas y previene el sobreajuste (overfitting).
- Fue una de las primeras redes en demostrar la viabilidad y la enorme ventaja de entrenar redes neuronales profundas utilizando GPUs. Como dato adicional, el paper cita: *"Entrenamos la red durante aproximadamente 90 épocas sobre el conjunto de entrenamiento de 1.2 millones de imágenes, lo que tardó entre cinco y seis días en dos GPUs NVIDIA GTX 580 de 3GB."*
6. **VGGNet (2014)**: La arquitectura VGGNet, desarrollada por el Grupo de Geometría Visual de la Universidad de Oxford, demostró que arquitecturas más profundas mejoran la precisión. Ellos concluyeron que las redes más profundas, con filtros de convolución pequeños superan significativamente a arquitecturas más superficiales como AlexNet, especialmente en tareas de reconocimiento de imágenes a gran escala. Los autores atribuyen esta mejora a la capacidad de la red más profunda para aprender características más complejas en varios niveles de abstracción, lo que lleva a un mejor rendimiento tanto en clasificación como en localización de imágenes. Este hallazgo marcó un avance significativo en el campo del aprendizaje profundo y el reconocimiento de imágenes. Su simplicidad y estructura uniforme la convirtieron en un referente para la exploración de la profundidad de las redes.
7. **GoogLeNet (2014)**: GoogLeNet, también conocida como Inception v1, introdujo el concepto de módulos de inception, que permitieron un uso eficiente de la computación y llevaron a redes con mayor profundidad y amplitud. Esta arquitectura fue mejor en la utilización de recursos computacionales.
8. **ResNet (2015)**: Las Redes Residuales, o ResNets, introdujeron la idea de conexiones residuales, permitiendo entrenar redes muy profundas con éxito al abordar el problema del gradiente desvaneciente. El Problema de Degradación: Se observó que al hacer las redes "simplemente" más profundas (apilando más capas como en VGG), el rendimiento (precisión) primero se saturaba y luego comenzaba a degradarse rápidamente, tanto en el conjunto de entrenamiento como en el de validación. Esto no se debía principalmente al sobreajuste (overfitting),

sino a que la optimización de redes tan profundas se volvía muy difícil. Era contraintuitivo que añadir más capas pudiera empeorar el resultado.

La gran innovación de ResNet fue el **bloque residual con conexiones de salto**, que reformuló el aprendizaje dentro de bloques de capas para facilitar la optimización y permitir el entrenamiento efectivo de redes neuronales extremadamente profundas.

9. **DenseNet (2017)**: DenseNet introdujo patrones de conectividad densa, permitiendo la reutilización de características y promoviendo el flujo de gradientes. Esta arquitectura demostró una eficiencia de entrenamiento y precisión mejoradas en tareas de clasificación de imágenes.

Aprendizaje por Transferencia y Modelos Preentrenados (Años 2010): El concepto de aprendizaje por transferencia se popularizó, donde las redes preentrenadas en grandes conjuntos de datos (por ejemplo, ImageNet) se ajustaron para tareas específicas. Este enfoque redujo significativamente la necesidad de conjuntos de datos masivos y aceleró el desarrollo de modelos.

10. **EfficientNet (2019)**: A medida que las redes profundas aumentaron en tamaño, se volvieron computacionalmente costosas. EfficientNet, introducido en 2019, propuso una arquitectura escalable que logró un rendimiento de vanguardia con menos parámetros, haciéndola más factible para diversas aplicaciones.

11. **Transformers in Vision (Años 2020)**: Los transformers, inicialmente diseñados para el procesamiento del lenguaje natural, se adaptaron para tareas de visión por computadora, incluyendo la clasificación de imágenes. Los Transformers de Visión (ViTs) y modelos híbridos como combinaciones de CNN-Transformador surgieron como nuevos contendientes.

Más información:

<https://stanford.edu/~shervine/blog/evolution-image-classification-explained>

<https://towardsdatascience.com/10-papers-you-should-read-to-understand-image-classification-in-the-deep-learning-era-4b9d792f45a7>

Libro "Deep Learning for Vision Systems" (Mohamed Elgendi, 2020) - Cap. 5

Ejemplo práctico

Veamos un ejemplo en Python, para aplicar el concepto de clasificación de imágenes:

```
# Requisitos:  
# pip install tensorflow pillow  
  
import tensorflow as tf  
from tensorflow.keras.applications import EfficientNetB0  
from tensorflow.keras.applications.efficientnet import preprocess_input, decode_predictions  
from tensorflow.keras.preprocessing import image  
from PIL import Image  
import numpy as np  
  
# Cargar el modelo preentrenado EfficientNetB0  
model = EfficientNetB0(weights='imagenet')  
  
def classify_image(img_path):  
    # Cargar y preparar la imagen  
    img = Image.open(img_path)  
    img = img.resize((224, 224)) # Tamaño de entrada para EfficientNetB0  
  
    # Convertir la imagen a un array y procesarla  
    img_array = image.img_to_array(img)  
    img_array = np.expand_dims(img_array, axis=0)  
    img_array = preprocess_input(img_array)
```

```

# Realizar la predicción con el modelo
predictions = model.predict(img_array)

# Decodificar y retornar las predicciones
return decode_predictions(predictions, top=3)[0]

# Prueba con una imagen local
img_path = 'labrador.jpg' # Reemplaza con la ruta de tu imagen
predictions = classify_image(img_path)

print('\n')

# Mostrar las predicciones
for i, (imagenet_id, label, score) in enumerate(predictions):
    print(f'{i + 1}: {label} ({score:.2f})')

```

En este script, se carga un modelo EfficientNetB0 pre-entrenado con pesos de un modelo entrenado con el dataset ImageNet. Luego, el modelo se usa para clasificar una imagen, que se carga, se procesa y se pasa al modelo para obtener predicciones. Las predicciones se decodifican en etiquetas humanamente legibles y se imprimen con sus respectivas probabilidades:

```

1: Labrador_retriever (0.47)
2: golden_retriever (0.19)
3: Rhodesian_ridgeback (0.04)

```



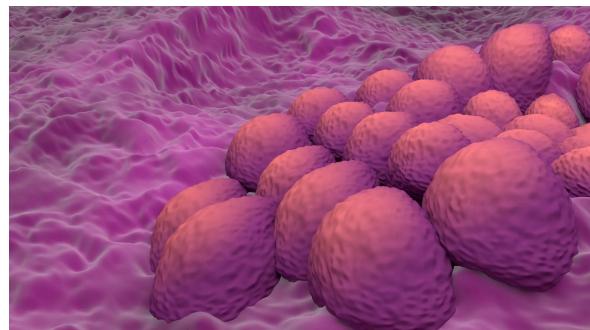
La salida de la red de clasificación es un vector numérico cuya longitud es igual al número de clases. Estos números pueden ser:

- **Logits/Scores:** Salidas brutas, no normalizadas, que indican la "evidencia" para cada clase.
- **Probabilidades:** Salidas normalizadas (usualmente mediante Softmax o Sigmoid) que representan la probabilidad estimada de pertenencia a cada clase. En este caso, las salidas dan una suma del valor 1.

Finalmente, para obtener la **predicción** concreta del modelo, normalmente se elige la clase que tiene el logit o la probabilidad más alta en el vector de salida.

Aplicaciones

Los modelos de clasificación de imágenes son herramientas poderosas en el campo de la inteligencia artificial y tienen una amplia gama de aplicaciones en diversos sectores. Por ejemplo, en el ámbito de la salud, la clasificación de imágenes se utiliza para analizar imágenes médicas como radiografías, resonancias magnéticas (MRI) y tomografías computarizadas (CT) para ayudar en el diagnóstico de enfermedades. Por ejemplo, pueden identificar tumores, fracturas óseas, o anomalías en los tejidos.



Diagnóstico de cáncer con un modelo de clasificación:
<https://es.wired.com/articulos/una-inteligencia-artificial-diagnostica-este-cancer-con-un-99-de-precision-un-avance-clave-para-la-salud-femenina>

Los modelos de clasificación de imágenes se utilizan en plataformas en línea para clasificar y recomendar contenido visual, como películas, series o productos, basándose en las preferencias visuales de los usuarios.

En redes sociales, ayudan a filtrar y clasificar imágenes inapropiadas o que violan las políticas de la plataforma. También se utilizan para organizar grandes volúmenes de contenido visual subido por los usuarios.

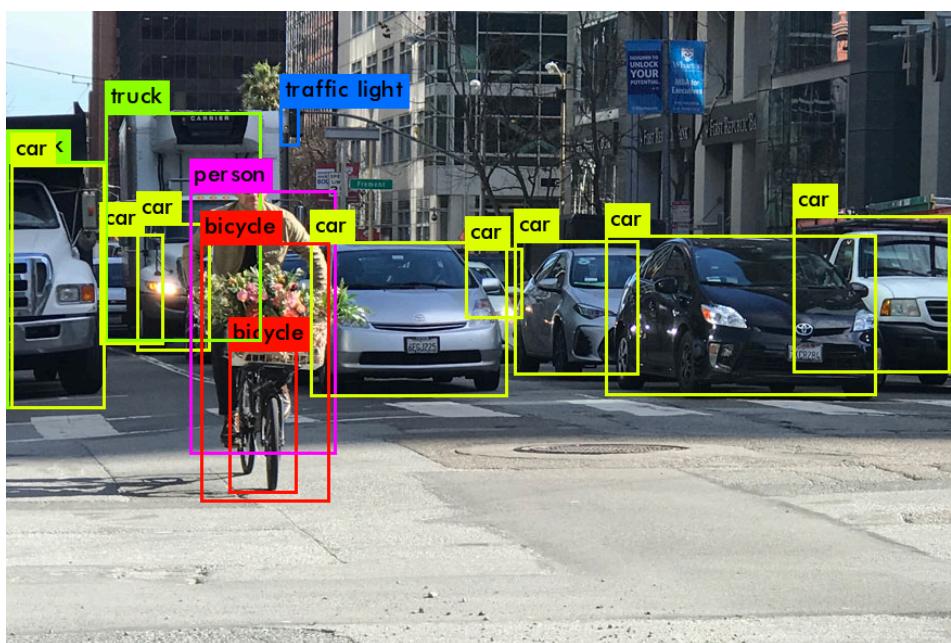
Aunque la detección de objetos, que implica identificar y localizar múltiples objetos dentro de una imagen, representa una evolución de los modelos de clasificación de imágenes, los modelos de clasificación siguen siendo importantes y relevantes por varias razones:

1. **Simplicidad y Eficiencia:** Los modelos de clasificación son generalmente más simples y requieren menos recursos computacionales en comparación con los modelos de detección de objetos. Esto los hace más adecuados para aplicaciones donde los recursos son limitados, como en dispositivos móviles o en entornos de computación en el borde.
2. **Aplicaciones Específicas de Clasificación:** Hay muchas aplicaciones donde el objetivo principal es determinar la presencia de una categoría de objeto o característica en una imagen completa, sin necesidad de localizar el objeto. Ejemplos incluyen la clasificación de imágenes médicas para diagnóstico, análisis de imágenes satelitales para estudios geológicos o ambientales, y clasificación de productos en e-commerce.
3. **Rapidez en el Procesamiento:** La clasificación de imágenes suele ser más rápida que la detección de objetos, lo que es crucial en aplicaciones que requieren respuestas en tiempo real o procesamiento de grandes volúmenes de imágenes.
4. **Base para Otros Modelos y Técnicas:** Los modelos de clasificación a menudo sirven como base para modelos más complejos, incluyendo los de detección de objetos. Comprender y mejorar los modelos de clasificación contribuye al desarrollo de técnicas más avanzadas en el campo del aprendizaje profundo y la visión por computadora.

4. Object Detection

Introducción y conceptos teóricos

La detección de objetos es una de las técnicas más importantes en el campo de la visión por computadora, que se enfoca en la identificación y localización de entidades dentro de imágenes digitales. Esta disciplina combina aspectos de la clasificación de imágenes y la localización para identificar múltiples objetos en imágenes y determinar su posición exacta a través de un "cuadro delimitador" o bounding box. La detección de objetos tiene aplicaciones amplias y significativas, incluyendo, pero no limitado a, sistemas de vigilancia, navegación de vehículos autónomos, y análisis de imágenes médicas.



La Detección de Objetos es una tarea que permite identificar y localizar varios objetos, como por ejemplo, automóviles, ciclistas y semáforos, dentro de una imagen. Esto se logra definiendo regiones rectangulares usando coordenadas (x_{min} , y_{min} , x_{max} , y_{max}) y asociándolas con un vector de clasificación y probabilidad (p_1, p_2, \dots, p_n). La Detección de Objetos supera la clasificación de imágenes en importancia práctica, ya que permite ubicar objetos en la imagen, para su posterior análisis, modificación o clasificación. Técnicas como la Difusión Estable (Stable Diffusion) y el

Intercambio de Caras (deepfake) aprovechan la detección de objetos para manipular y reemplazar objetos o rostros en imágenes. Los desafíos surgen cuando múltiples objetos de la misma clase, como peatones, se superponen, lo que lleva al uso de NMS (Supresión de No Máximos) como una solución común, aunque están surgiendo métodos alternativos.

Los métodos de detección de objetos han evolucionado considerablemente con el avance de las redes neuronales profundas. Inicialmente, las técnicas tradicionales, como los detectores basados en características manuales (por ejemplo, HOG, VJ detector) eran populares, pero enfrentaban limitaciones en términos de precisión y eficiencia computacional.

Con el surgimiento de las redes neuronales convolucionales (CNN), la detección de objetos ha experimentado mejoras significativas en términos de rendimiento y precisión.

Al comienzo del diagrama a continuación, podremos ver que las CNN han estado en desarrollo desde la década de 1980. En 1998, LeCun et al. introdujeron LeNet-5, una arquitectura significativa de CNN para el reconocimiento de dígitos.

Este conjunto de datos, conocido como MNIST, se originó en los años 90 y desde entonces se ha convertido en un estándar popular para evaluar algoritmos de aprendizaje automático y aprendizaje profundo. Más tarde, hubo solicitudes para la detección de objetos, pero en ese momento no había métodos efectivos suficientes para esto.

El primer método significativo fue el de Viola-Jones o cascadas Haar, que era rápido y fácil de usar en las PC de la época, proporcionando una velocidad aceptable de varios cuadros por segundo (FPS).

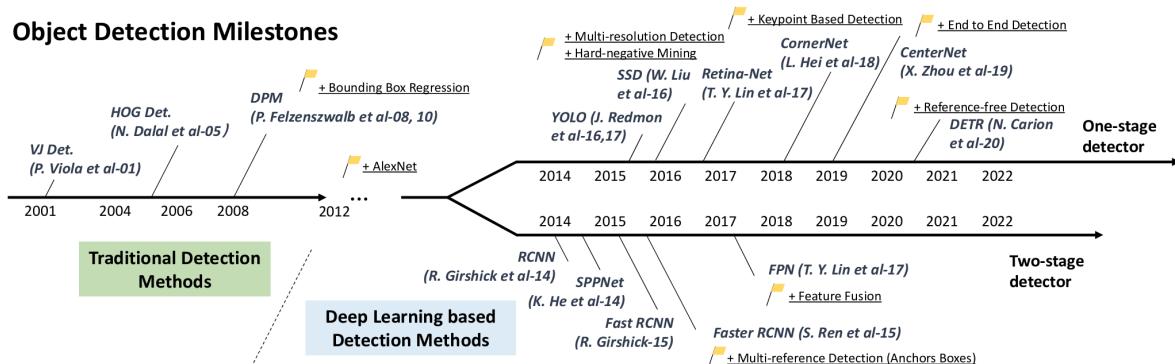
Unos años después, se introdujo el método Detector HOG como alternativa a Viola-Jones, centrándose principalmente en capturar formas y contornos de objetos.

Posteriormente, entraron en juego métodos que usaban Modelos de Partes Deformables (DPM), que durante mucho tiempo ocuparon posiciones líderes en clasificaciones de precisión de detección de objetos.

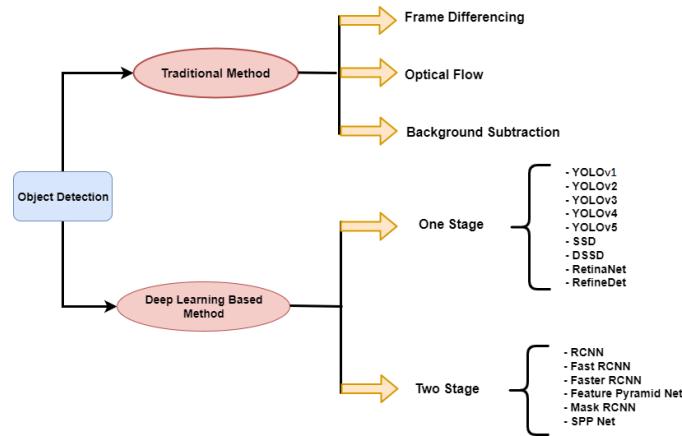
En 2012, aparecieron las primeras redes neuronales profundas grandes, incluyendo AlexNet. Aunque lentas y computacionalmente intensivas, las arquitecturas de AlexNet y modelos subsiguientes como MobileNet se volvieron más óptimas.

Estos modelos proporcionaron características representativas de imágenes de alta calidad que pueden describir el contexto y detectar una amplia gama de objetos.

Uno de los aspectos más importantes de estos métodos es su naturaleza end-to-end (de extremo a extremo), donde la imagen de entrada pasa por una secuencia de operaciones diferenciadas, lo que permite un procesamiento holístico dentro de una sola arquitectura.

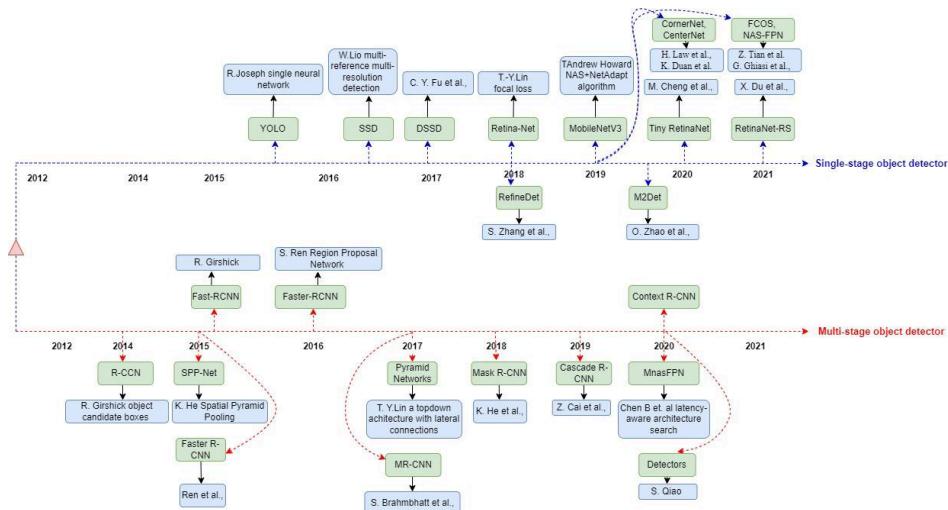


Los algoritmos modernos de detección de objetos se pueden clasificar principalmente en dos categorías: algoritmos de dos etapas y de una etapa. Los algoritmos de dos etapas, como R-CNN y sus variantes (Fast R-CNN, Faster R-CNN), primero generan propuestas de regiones que pueden contener objetos y luego clasifican y refinan la localización de estos objetos. Por otro lado, los algoritmos de una etapa, como YOLO y SSD, realizan la clasificación y localización en un solo paso, lo que los hace más rápidos aunque a veces menos precisos que los métodos de dos etapas.



Clasificaciones de los métodos de detección de objetos tradicionales y los más modernos basados en Deep Learning

Más recientemente, los transformers, conocidos por su éxito en el procesamiento del lenguaje natural, han encontrado aplicaciones en la detección de objetos, ofreciendo enfoques innovadores para manejar las tareas de visión por computadora. DETR (DEtection TRansformer) es un ejemplo notable que utiliza un mecanismo de atención para codificar relaciones globales en las imágenes, lo que representa un cambio significativo respecto a los enfoques basados exclusivamente en CNN.



Mapa taxonómico de la historia del desarrollo de la detección general de objetos en orden cronológico de la era Deep Learning, incluyendo detectores de una etapa y de dos etapas ([Object Detection Performance: A Comparative Study](#))

La detección de objetos sigue siendo un campo de investigación activo, con desafíos continuos como la detección de objetos pequeños, el manejo de imágenes en diferentes escalas y condiciones de iluminación, y la mejora de la eficiencia computacional para aplicaciones en tiempo real. La integración de técnicas avanzadas de aprendizaje profundo y la exploración de nuevas arquitecturas de redes neuronales continúan impulsando el progreso en este campo vital de la visión por computadora.

Algoritmos de dos etapas

Los algoritmos de dos etapas en la detección de objetos son una clase de técnicas en la visión por computadora que se centran en la identificación y localización precisa de objetos dentro de imágenes. Como su nombre indica, estos algoritmos funcionan en dos fases distintas: la generación de propuestas de regiones y la clasificación y refinamiento de estas propuestas. Estos algoritmos son conocidos por su alta precisión y son comúnmente utilizados en aplicaciones donde la exactitud es crítica. A continuación, se describen los aspectos clave de los algoritmos de dos etapas:

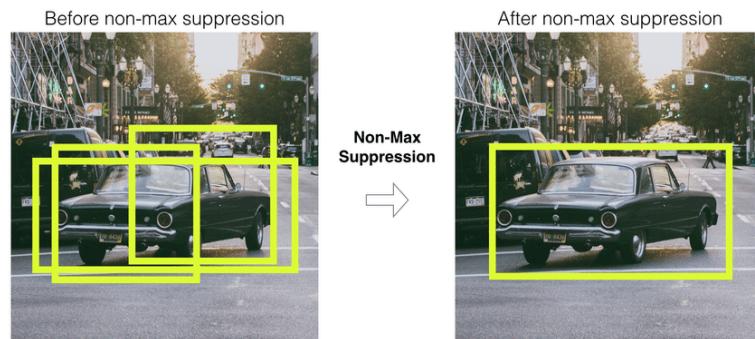
1. Generación de Propuestas de Regiones:

- La primera etapa de estos algoritmos implica identificar regiones de interés potencial dentro de una imagen. Estas regiones, conocidas como propuestas de regiones, son áreas que posiblemente contengan objetos.

- Métodos como Selective Search (utilizado en R-CNN) o Region Proposal Networks (RPN, utilizado en Faster R-CNN) se emplean para generar estas propuestas. Estas técnicas varían en su enfoque: mientras que Selective Search es un método basado en reglas y heurísticas, RPN es una red neuronal que propone regiones basadas en características aprendidas.

2. Clasificación y Refinamiento de las Propuestas:

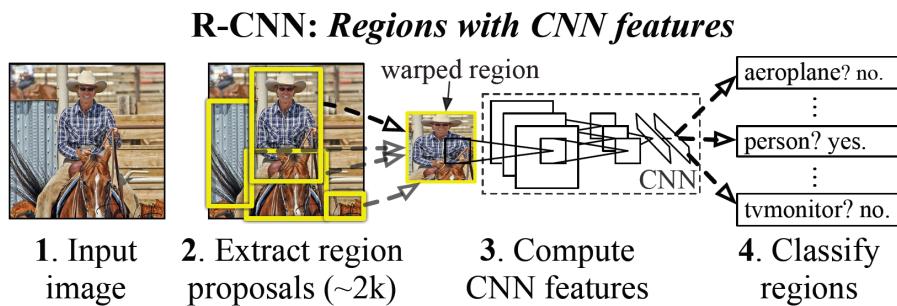
- En la segunda etapa, cada propuesta de región generada se procesa para determinar si contiene un objeto y, de ser así, qué categoría de objeto es.
- Las regiones propuestas se alimentan a una red neuronal convolucional que extrae características detalladas para la clasificación y el ajuste preciso de los cuadros delimitadores. Esta fase implica tanto la clasificación del objeto como la regresión del cuadro delimitador para refinar la posición y el tamaño del objeto detectado.
- Este paso también puede incluir procesos adicionales como el uso de la técnica Non-Maximum Suppression (NMS) para eliminar múltiples detecciones redundantes del mismo objeto.



Ejemplos de algoritmos de Dos Etapas:

1. R-CNN (Region-based Convolutional Neural Networks):

- R-CNN fue uno de los primeros en utilizar CNN para la detección de objetos. Utiliza la búsqueda selectiva para generar propuestas de regiones y luego aplica una CNN para clasificar cada propuesta.



2. Fast R-CNN:

- Mejora la eficiencia de R-CNN al compartir cálculos entre las propuestas de región. Utiliza una técnica llamada ROI (Region of Interest) pooling para extraer características para múltiples regiones de un único paso de características convolucionales.

3. Faster R-CNN:

- Introduce las Redes de Propuestas de Regiones (RPN), que reemplazan la búsqueda selectiva con una red neuronal para generar propuestas de regiones, integrando así completamente la generación de propuestas en el proceso de aprendizaje. Este avance significó una mejora considerable en velocidad y precisión.

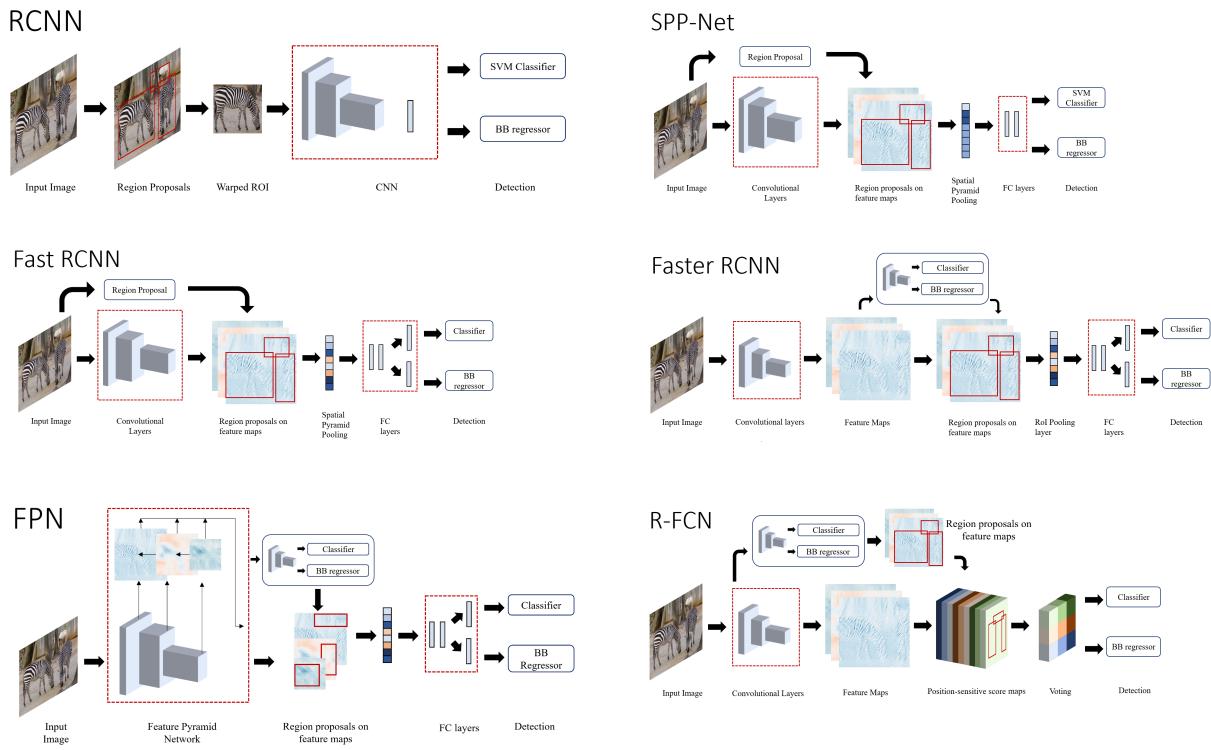


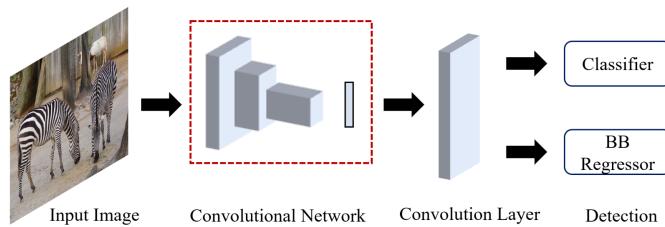
Ilustración de la arquitectura interna de diferentes detectores de objetos de dos etapas (Fuente: A Survey of Modern Deep Learning based Object Detection Models <https://arxiv.org/pdf/2104.11892.pdf>)

Algoritmos de una etapa

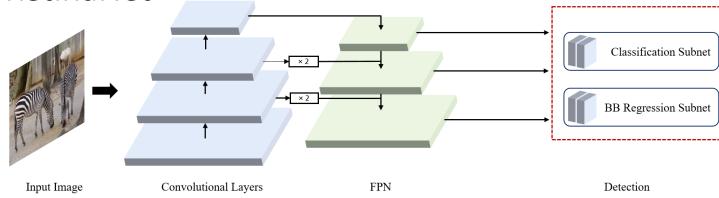
A diferencia de los algoritmos de detección de dos etapas, que primero proponen regiones candidatas y luego clasifican estas regiones, los algoritmos de una sola etapa realizan ambas tareas en un solo paso. Esto los hace generalmente más rápidos, aunque a menudo con una ligera disminución en la precisión en comparación con los métodos de dos etapas. Los ejemplos más conocidos de detectores de una sola etapa incluyen YOLO (You Only Look Once), y SSD (Single Shot Multibox Detector). El funcionamiento de estos modelos, es el siguiente:

- Entrada de Imagen:** Al igual que otros métodos de detección de objetos, los algoritmos de una sola etapa toman una imagen como entrada. Esta imagen pasa a través de una serie de capas convolucionales para extraer características.
- Extracción de Características:** Utilizan una red backbone (como ResNet, MobileNet, etc.) para extraer características de la imagen. A diferencia de los métodos de dos etapas, no hay una etapa separada para proponer regiones de interés.
- Predicción Simultánea de Clases y Localización:** Sobre la salida de la red de características, los algoritmos aplican una serie de filtros para detectar objetos. Estos filtros son responsables de predecir las clases y las ubicaciones de los objetos en la imagen. En esencia, dividen la imagen en una cuadrícula y, para cada celda de la cuadrícula, predicen las cajas delimitadoras (bounding boxes) y las probabilidades de clase para los objetos que podrían estar centrados en esa celda.
- Cajas Delimitadoras y Clasificación:** Cada celda de la cuadrícula produce múltiples predicciones de bounding boxes, cada una con su propio puntaje de confianza (que indica la probabilidad de que haya un objeto) y un conjunto de coordenadas (que definen la posición y el tamaño del objeto en la imagen). Además, se asigna una clasificación de clase a cada bounding box.
- Supresión de No Máximos (NMS):** Dado que cada celda puede predecir múltiples cajas y puede haber superposiciones significativas entre estas cajas, se aplica un proceso llamado Supresión de No Máximos para eliminar las cajas redundantes. NMS mantiene solo las cajas con los puntajes de confianza más altos y descarta las demás que tienen una superposición significativa con ellas.
- Salida:** El resultado final es un conjunto de bounding boxes, cada una con una etiqueta de clase y un puntaje de confianza, que indican la ubicación y el tipo de objetos detectados en la imagen.

YOLO



RetinaNet



SSD

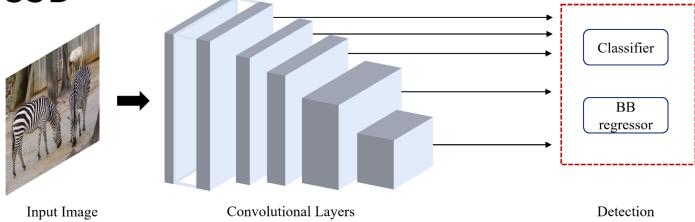
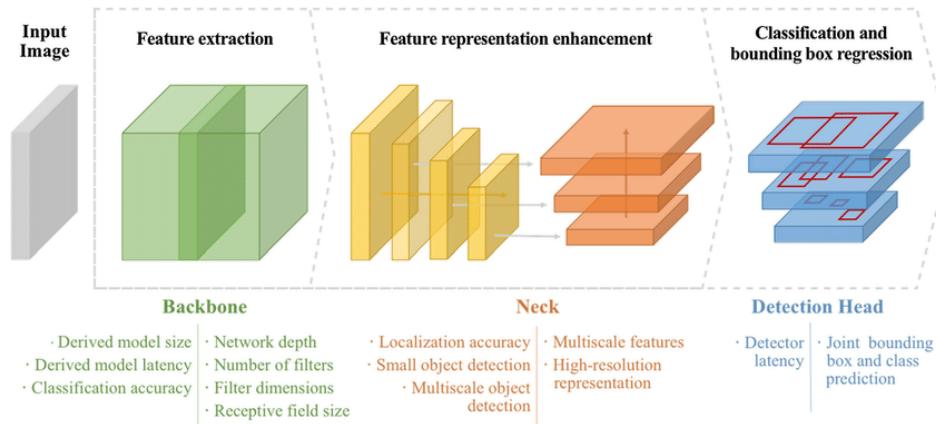


Diagrama de diferentes modelos de una sola etapa (Fuente: A Survey of Modern Deep Learning based Object Detection Models
<https://arxiv.org/pdf/2104.11892.pdf>)

Arquitectura de los modelos One Stage

En los modelos de detección de objetos en visión por computadora, especialmente aquellos basados en redes neuronales convolucionales (CNN), los términos "backbone", "neck", y "head" son comunes. Cada uno de estos términos se refiere a una parte específica de la arquitectura de la red, que tiene una función distinta en el proceso de detección de objetos:

1. **Backbone:** Esta es la base de la red neuronal y generalmente se encarga de la extracción de características. Es una serie de capas convolucionales que procesan la imagen de entrada. El backbone toma la imagen cruda y, a través de varias capas convolucionales y de pooling, produce un conjunto de mapas de características a alto nivel. Los modelos de backbone populares incluyen ResNet, VGG, Inception y MobileNet. Estos modelos suelen ser pre-entrenados en grandes conjuntos de datos como ImageNet para el reconocimiento de imágenes (clasificación), y luego se adaptan para tareas específicas como la detección de objetos.
2. **Neck:** El "cuello" de la red es una serie de capas que se encuentran entre el backbone y el head. Su función principal es procesar y refinar las características extraídas por el backbone antes de que pasen al head. El neck puede incluir operaciones como la normalización de características, la atención espacial, o la fusión de características de diferentes escalas. Un ejemplo común de neck es la Red de Pirámide de Características (FPN, por sus siglas en inglés), utilizada en modelos como Faster R-CNN y RetinaNet, que combina información de diferentes niveles de la jerarquía de la red para mejorar la detección de objetos de diferentes tamaños.
3. **Head:** La "cabeza" de la red es la parte que realiza tareas específicas como la clasificación y la localización de objetos. Después de que las características han sido extraídas por el backbone y procesadas por el neck, el head utiliza esta información para predecir clases de objetos y sus ubicaciones en la imagen. El head puede ser diseñado para diferentes tareas, como la detección de cajas delimitadoras (bounding boxes) en la detección de objetos o la segmentación de instancias en la segmentación semántica. Dependiendo de la tarea, el head puede tener diferentes arquitecturas y métodos de decodificación.



Más información sobre detección de objetos:

<https://www.jeremyjordan.me/object-detection-one-stage/>

<https://namrata-thakur893.medium.com/a-detailed-introduction-to-two-stage-object-detectors-d4ba0c06b14e>

<https://polukhin.tech/2023/06/19/how-object-detection-evolved>

Fusión de modelos de detección de objetos con lenguaje

Recientemente empiezan a aparecer modelos avanzados capaces de reconocer objetos en imágenes a partir de un prompt, indicando cuáles son los objetos de interés en la imagen.

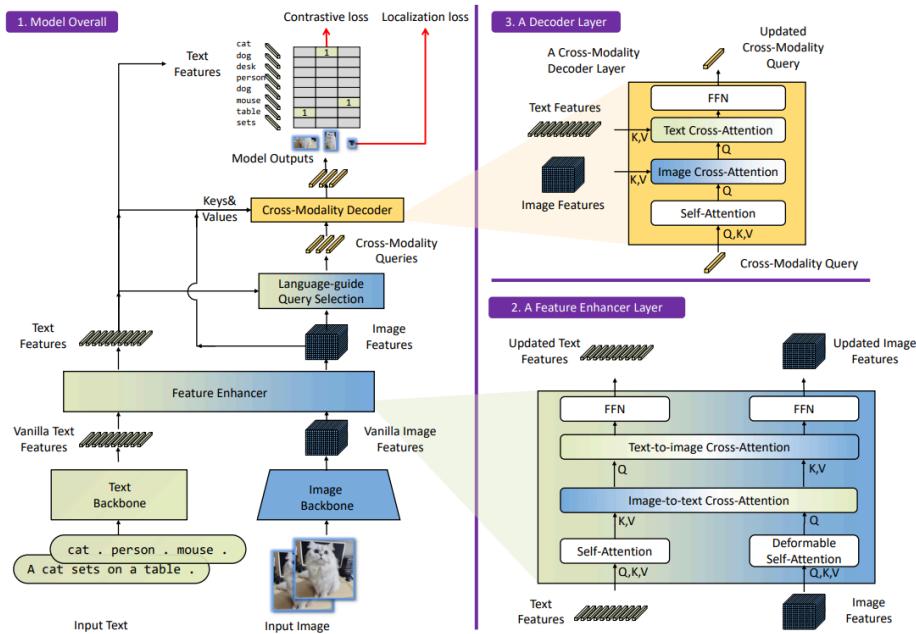
Grounding DINO

Papers:

[Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection](#)

[Grounding DINO 1.5: Advance the "Edge" of Open-Set Object Detection](#)

Grounding DINO es un modelo avanzado de detección de objetos de código abierto desarrollado por [IDEA Research](#), que integra información lingüística para mejorar la generalización en escenarios diversos y complejos. Se basa en el modelo Transformer DINO, y añade funcionalidades de fusión de modalidades visión-lenguaje en varias fases del modelo, permitiendo la detección de objetos utilizando descripciones textuales libres, lo que se conoce como detección de objetos de conjunto abierto (open-set object detection).



El marco de Grounding DINO: Una capa de mejora de características y una capa decodificadora en el bloque 1, bloque 2 y bloque 3, respectivamente.

GroundingDINO

Highlight

- Open-Set Detection. Detect everything with language!
- High Performance. COCO zero-shot 52.5 AP (training without COCO data!). COCO fine-tune 63.0

[➡ https://www.youtube.com/watch?v=wxWDt5UiwY8](https://www.youtube.com/watch?v=wxWDt5UiwY8)

Collaboration with Stable Diffusion

Inputs	Detection Results	Edited Images

Detection Prompt: Face
Detection Prompt: modify, face, .bv, first, detecting, a, face

Características principales de Grounding DINO

1. Fusión de Modalidades Visión-Lenguaje:

- **Enhancer de Características:** Integra atención cruzada entre texto e imagen para mejorar las representaciones características.
- **Selección de Consultas Guiada por Lenguaje:** Utiliza el texto para guiar la selección de consultas durante la detección.
- **Decodificador de Modalidad Cruzada:** Mejora la representación de consultas con atención cruzada entre imagen y texto.

2. Capacidades de Detección de Conjunto Abierto:

- Redefine la detección de objetos como una tarea de grounding de frases, facilitando el pre-entrenamiento a gran escala utilizando tanto conjuntos de datos de detección como de grounding.
- Permite la detección y localización de objetos basándose en descripciones textuales específicas dadas por el usuario, superando las limitaciones de los detectores de conjuntos cerrados que solo pueden identificar categorías predefinidas.

3. Resultados Empíricos:

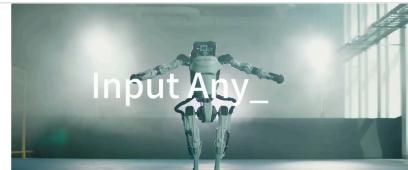
- Grounding DINO 1.5 Pro logra un AP de 54.3 en el benchmark de detección de COCO y un AP de 55.7 en el benchmark de transferencia de cero disparos de LVIS-minival, estableciendo nuevos récords para la detección de objetos de conjunto abierto.
- Grounding DINO ha demostrado una generalización impresionante y escalabilidad en diferentes escenarios y entradas textuales, con visualizaciones que muestran una precisa localización de objetos según descripciones textuales complejas.

Aplicaciones y Evaluaciones

- **Detección de Expresiones Referenciales (REC):** Grounding DINO extiende la evaluación de detección de conjunto abierto a los conjuntos de datos de comprensión de expresiones referenciales, lo que permite evaluar el rendimiento del modelo con entradas de texto libre.
- **Edición de Imágenes:** Permite la colaboración con modelos generativos como Stable Diffusion para la edición de imágenes basada en descripciones textuales, facilitando aplicaciones como la modificación del fondo de una imagen o la detección y modificación de objetos específicos.

Grounding DINO 1.5
IDEA Research's Most Capable Open-World Object Detection Model Series.

 <https://www.youtube.com/watch?v=v7SWCjQfTmA>



Más información:

<https://github.com/IDEA-Research/GroundingDINO>

<https://github.com/IDEA-Research/Grounding-DINO-1.5-API>

Google Colab

 https://colab.research.google.com/drive/1ezLi_lRur0LXFEOnek9Rlke9QxepvAnZ?usp=sharing



DeepDataSpace | Unleashing the Power of Cutting-Edge Computer Vision Technology

Discover DeepDataSpace, the official website of the CVR Center, dedicated to research in the field of CV algorithms. With its powerful and advanced algorithms, DeepDataSpace offers a range of exceptional products, including annotation platforms, algorithm platforms, and open-source computer vision tools. Experience the forefront of innovation and unleash the full potential of computer vision with DeepDataSpace. DeepDataSpace是CVR中心权威在线平

 https://deepdataspace.com/playground/grounding_dino



El **Grounding** se refiere a la tarea de establecer una conexión o correspondencia directa entre elementos del lenguaje (como palabras, frases o descripciones) y sus referentes visuales específicos en una imagen (como objetos o regiones). Permite ir más allá de la simple detección de categorías predefinidas (ej., "gato"). Permite localizar objetos basados en descripciones más ricas que incluyen atributos, relaciones o expresiones referenciales complejas (ej., "el hombre de abajo con la cabeza levantada")

Ejemplos prácticos

Aquí veremos cómo implementar la detección de objetos utilizando YOLOv10, una de las versiones más recientes y avanzadas del algoritmo "You Only Look Once" (YOLO), conocido por su rapidez y precisión. Comenzaremos asegurándonos de que el modelo pre-entrenado de YOLOv10 está disponible en nuestro entorno. Para ello, emplearemos un comando que verifica si el modelo ya está descargado; de no ser así, lo descargará automáticamente. Luego, utilizaremos las librerías `cv2` para el procesamiento de imágenes y `ultralytics` para cargar y utilizar el modelo YOLOv10.

```
# Importamos las librerías a utilizar
import cv2
from ultralytics import YOLO
from IPython.display import Image

# Load the pre-trained YOLOv10 model
```

```

model = YOLO('yolov1On') # This will automatically download the model weights

# Perform inference
source_img = cv2.imread('https://ultralytics.com/images/bus.jpg')
results = model(source_img)

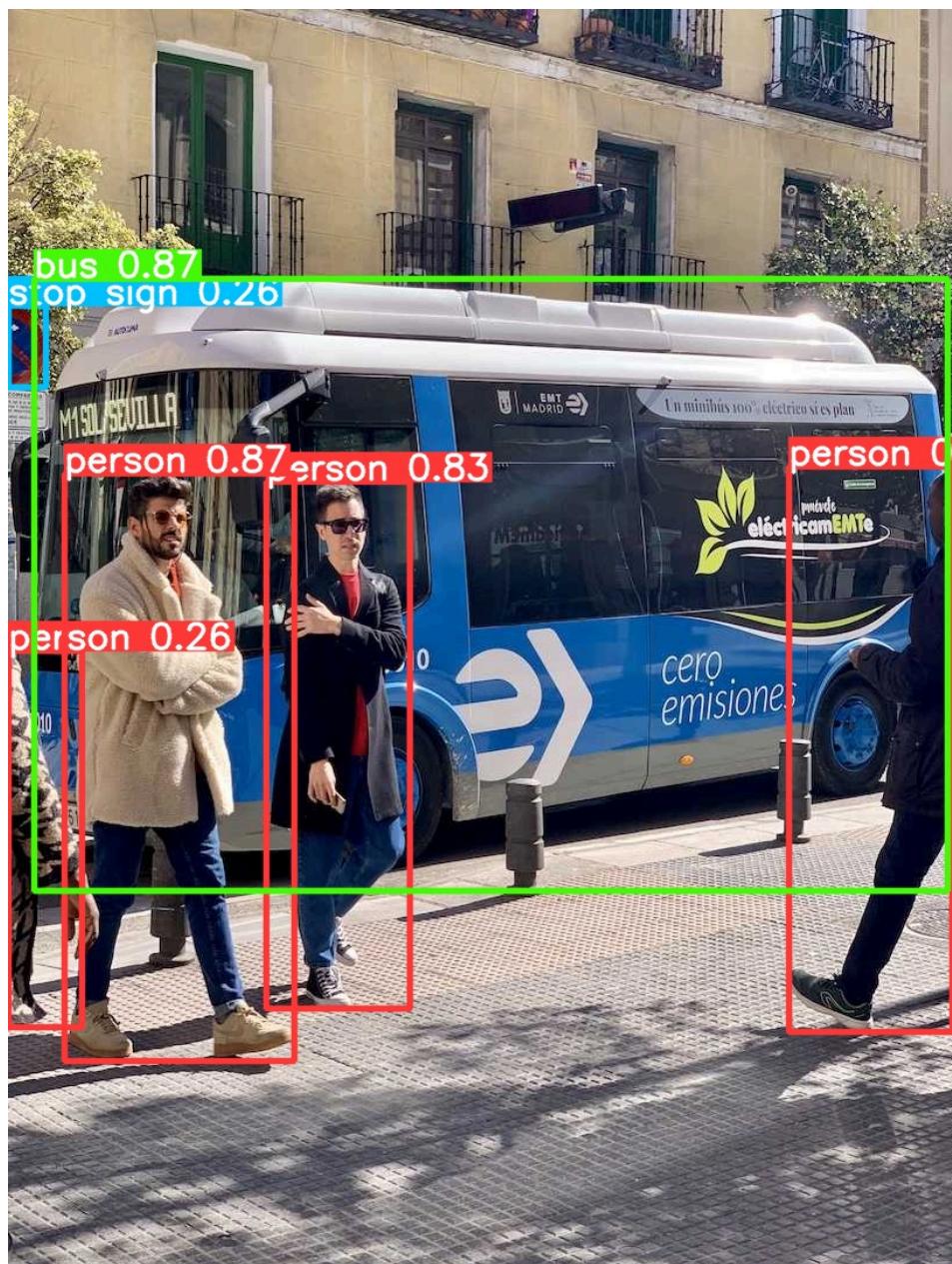
# Visualize the results on the frame
annotated_frame = results[0].plot()

# Guardamos el resultado en JPG
cv2.imwrite('bus_results.jpg', annotated_frame)

# Mostramos el resultado en Colab
Image('bus_results.jpg')

```

Como resultado obtendremos:



La librería [ultralytics](#), se encarga de graficar los bounding boxes hallados en la imagen. Sin embargo, si queremos acceder a los datos, podemos hacerlo del siguiente modo:

```
# Iteramos sobre los boung boxes obtenidos
for box in results[0].boxes:
    # Extrayendo los datos del tensor
    x1, y1, x2, y2, confidence, cls = box.data[0]

    # Obteniendo el nombre de la clase
    class_name = model.names[int(cls)]

    # Imprimir los resultados en un formato legible
    print(f"Clase: {class_name}, Box: ({x1:.2f}, {y1:.2f}, {x2:.2f}, {y2:.2f}), Confianza: {confidence:.2f}")
```

Y veremos el siguiente resultado, donde se muestran cada uno de los bounding box encontrados, con su clase y nivel de confidencia encontrado:

```
Clase: bus, Box: (22.87, 231.28, 805.00, 756.84), Confianza: 0.87
Clase: person, Box: (48.55, 398.55, 245.35, 902.70), Confianza: 0.87
Clase: person, Box: (669.47, 392.19, 809.72, 877.04), Confianza: 0.85
Clase: person, Box: (221.52, 405.80, 344.97, 857.54), Confianza: 0.83
Clase: person, Box: (0.00, 550.53, 63.01, 873.44), Confianza: 0.26
Clase: stop sign, Box: (0.06, 254.46, 32.56, 324.87), Confianza: 0.26
```

Ultralytics, para sus modelos preentrenados de YOLO, generalmente utiliza el dataset [COCO](#) (Common Objects in Context) para el entrenamiento. [COCO](#) es uno de los datasets más populares y ampliamente utilizados en el campo de la visión por computadora, especialmente para tareas como la detección de objetos, segmentación de instancias y reconocimiento de imágenes.

El dataset [COCO](#) contiene más de 200.000 imágenes etiquetadas, abarcando más de 80 categorías de objetos comunes en una amplia gama de contextos. Estas imágenes están anotadas con más de un millón de instancias de objetos, lo que proporciona una rica fuente de datos para entrenar modelos de detección de objetos robustos y precisos.

Confidencia

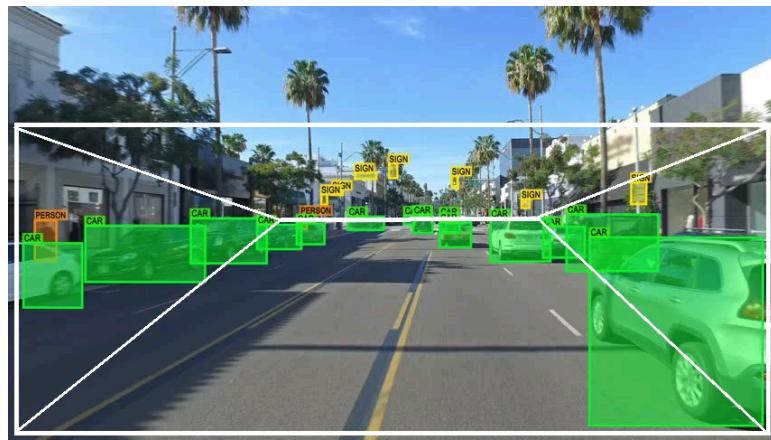
El nivel de confianza en los modelos de detección de objetos es una medida cuantitativa que indica la probabilidad que tiene el modelo de que una predicción sea correcta. Este nivel de confianza suele representarse como un porcentaje que indica cuán seguro está el modelo de que un objeto detectado en una imagen o video realmente corresponde a la categoría que se ha identificado.

Por ejemplo, si un modelo de detección de objetos identifica un perro en una imagen y asigna un nivel de confianza del 90% a esta detección, esto significa que el modelo está un 90% seguro de que el objeto identificado es, de hecho, un perro. Este valor puede mostrarse en la escala de 0 a 100%, o de 0 a 1.

Aplicaciones

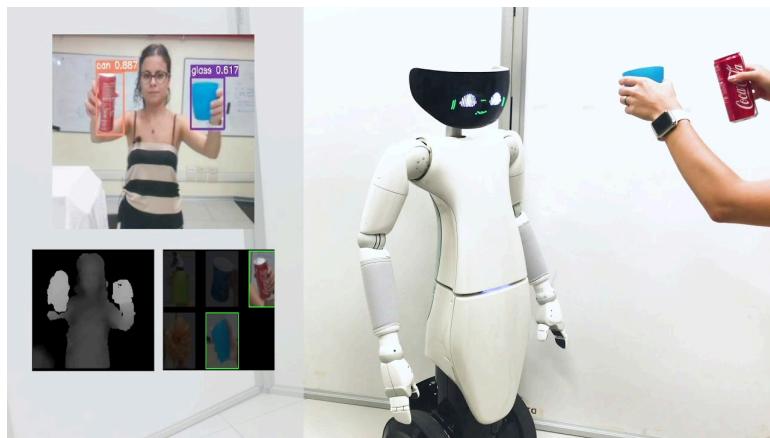
La detección de objetos tiene una amplia gama de aplicaciones importantes en diversos sectores. Algunas de las aplicaciones más destacadas incluyen:

- **Sopporte para Vehículos Autónomos:** Ayuda a los coches autónomos a navegar de manera segura en el tráfico.



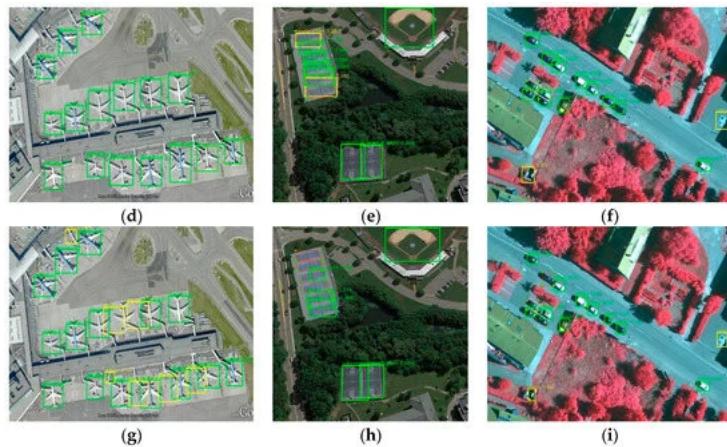
<https://www.tesmanian.com/blogs/tesmanian-blog/patent-enhanced-object-detection-for-autonomous-vehicles-based-on-field-view-Patente de Tesla>

- **Seguridad:** Utilizada en sistemas de seguridad y vigilancia para identificar comportamientos potencialmente peligrosos o ilegales.
- **Análisis de Comportamiento Humano:** Usado en investigaciones de mercado, seguridad y estudios de comportamiento social.
- **Imágenes Médicas:** Como en la detección de cáncer, donde ayuda a identificar tumores y otras anomalías en imágenes médicas.
- **Robótica:** Facilita la interacción y navegación de los robots en entornos diversos.



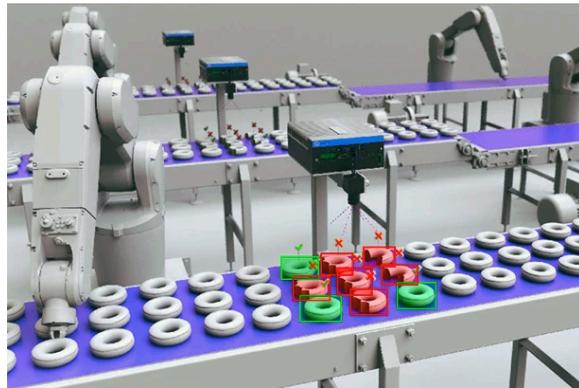
<https://www.youtube.com/watch?v=eT-2v6-xoSs>

- **Aplicaciones de Teledetección:** Utilizadas en la monitorización ambiental, la agricultura y la gestión de recursos naturales.



<https://www.mdpi.com/2072-4292/10/1/131>

- **Inspección de Calidad en Industrias:** Se utiliza para automatizar el control de calidad en líneas de producción. La detección de objetos puede identificar defectos de fabricación, inconsistencias en productos y asegurar que los estándares de calidad se mantengan de manera eficiente y confiable.



<https://robovision.ai/markets/manufacturing/food-quality-control/>

- **Monitoreo y Optimización de Tráfico Vehicular:** La tecnología se emplea para mejorar la gestión del tráfico, analizando patrones de flujo vehicular, detectando congestiones y accidentes, y contribuyendo a sistemas de semáforos inteligentes y otras soluciones de gestión de tráfico. Esto puede llevar a una mejor planificación urbana y una reducción en los tiempos de viaje.

5. Segmentación avanzada

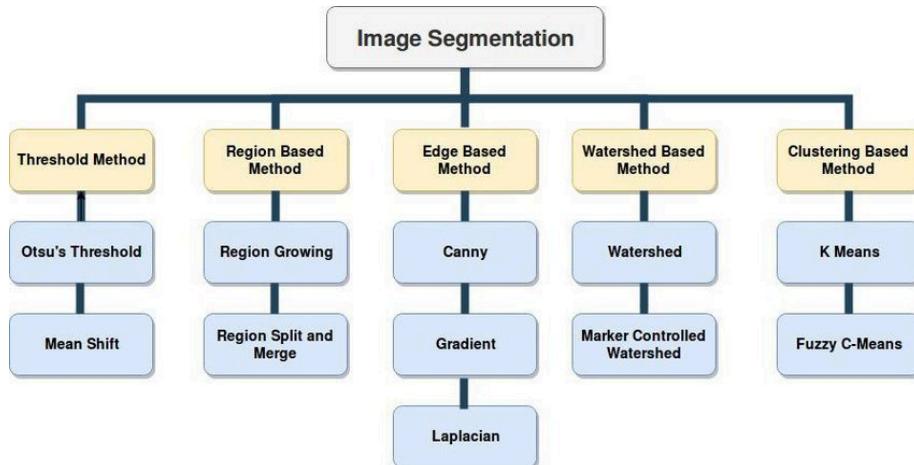
Introducción y conceptos teóricos

La segmentación de imágenes es una técnica crucial en el campo de la visión por computadora que implica dividir una imagen en partes o regiones, a menudo para identificar y analizar ciertos objetos o áreas de interés dentro de la imagen.

Existen dos grandes grupos de métodos de segmentación: los clásicos y los basados en deep learning.

Métodos Clásicos

Los métodos clásicos de segmentación se basan en técnicas de procesamiento de imágenes y visión por computadoras que no involucran aprendizaje automático o lo hacen de manera limitada. Estos métodos suelen depender de la detección de bordes, umbrales, crecimiento de regiones, y técnicas de agrupación para identificar y separar las regiones de interés en una imagen. Por ejemplo, el algoritmo de Canny para detección de bordes y el algoritmo de Watershed son técnicas clásicas de segmentación. Aunque estos métodos pueden ser muy eficaces en imágenes con alto contraste y bajo nivel de ruido, a menudo tienen dificultades para manejar la complejidad y variabilidad presentes en imágenes reales, especialmente cuando se requiere distinguir entre objetos similares o en condiciones de iluminación desafiantes.



Segmentación con métodos clásicos. Fuente: https://www.researchgate.net/figure/Image-Segmentation-Methods_fig2_340314328

Métodos Basados en Deep Learning

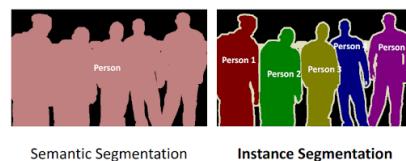
Con el auge del aprendizaje profundo, los métodos basados en deep learning han establecido el estado del arte en la tarea de segmentación de imágenes. Estos enfoques utilizan redes neuronales profundas, especialmente aquellas arquitecturas diseñadas para tareas de visión, como las redes convolucionales (CNNs), para aprender de grandes conjuntos de datos y realizar segmentaciones precisas y detalladas. Esta segmentación se puede clasificar en tres tipos principales: segmentación semántica, segmentación de instancias y segmentación panóptica. Cada uno de estos tipos tiene sus características y aplicaciones únicas.

- 1. Segmentación Semántica (Semantic Segmentation):** La segmentación semántica se centra en asignar un etiqueta de clase a cada píxel de la imagen, donde cada etiqueta corresponde a un tipo específico de objeto o región. Por ejemplo, en una imagen de una escena urbana, los píxeles pueden clasificarse como 'calle', 'edificio', 'vehículo', 'persona', etc.

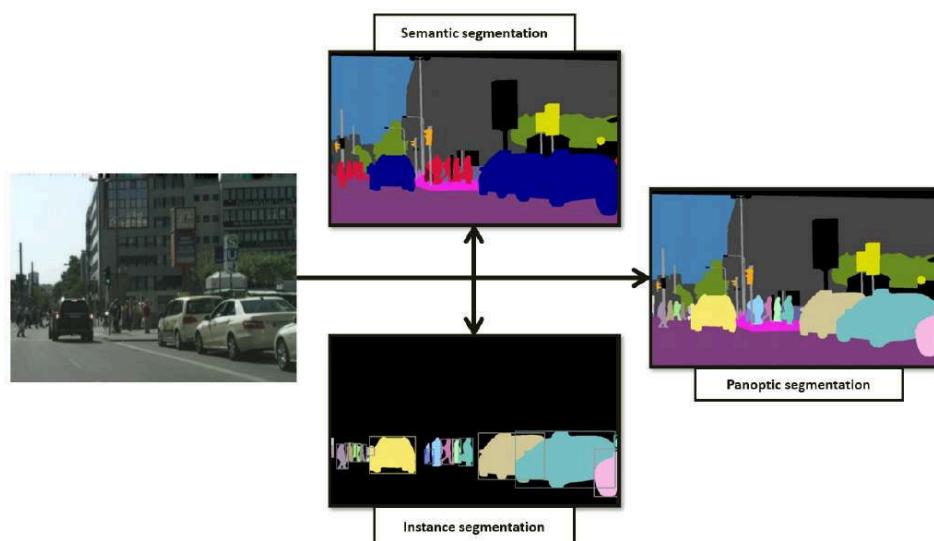
Es importante destacar que la segmentación semántica no distingue entre diferentes instancias del mismo tipo de objeto; por ejemplo, todas las personas en la imagen tendrían la misma etiqueta. En el documento "Fully Convolutional Networks for Semantic Segmentation" (Long et al., CVPR 2015), se explora el uso de redes neuronales convolucionales (CNNs) para la segmentación semántica, demostrando su eficacia en este tipo de tarea.



- 2. Segmentación de Instancias (Instance Segmentation):** A diferencia de la segmentación semántica, la segmentación de instancias no solo categoriza cada píxel, sino que también diferencia entre distintas instancias del mismo tipo de objeto. Utilizando el mismo ejemplo de la escena urbana, la segmentación de instancias identificaría y separaría cada vehículo o persona individualmente. Esto es útil en aplicaciones donde es importante no solo saber qué objetos están presentes en la imagen, sino también identificar y contar cada uno de ellos. Un ejemplo destacado de esta técnica es el modelo Mask R-CNN, descrito en el documento "Mask R-CNN" (He et al., ICCV 2017), que es capaz de realizar la segmentación de instancias a nivel de píxeles.



3. Segmentación Panóptica (Panoptic Segmentation): La segmentación panóptica combina aspectos de la segmentación semántica y de instancias para proporcionar una comprensión integral de la escena en una imagen. Este enfoque asigna a cada píxel una etiqueta de clase y, para las clases de 'cosas' (objetos contables como personas, vehículos, etc.), también distingue entre diferentes instancias. En otras palabras, realiza la segmentación de instancias para objetos contables y la segmentación semántica para regiones no contables o 'materia' (como cielo, carretera, césped). El documento "[Panoptic Segmentation](#)" (Kirillov et al., CVPR 2019) introduce y detalla esta tarea, proponiendo una métrica de calidad panóptica (PQ) para evaluar los modelos y fomentando la investigación en este campo.



Ejemplos de segmentación, que muestran la diferencia entre la segmentación semántica, la segmentación de instancias y la segmentación panóptica. (Fuente: [Panoptic Segmentation: A Review](#))

Metodologías y Arquitecturas

- **Modelos de Aprendizaje Profundo:** El éxito en estas áreas ha sido impulsado significativamente por modelos de aprendizaje profundo. Por ejemplo, las CNN (Redes Neuronales Convolucionales) han sido fundamentales en el aprendizaje de las representaciones complejas necesarias para una segmentación precisa. Arquitecturas como Mask R-CNN para segmentación de instancias y FCN (Redes Convolucionales Completamente Conectadas) para segmentación semántica han establecido referencias en sus respectivos campos.
- **Entrenamiento Conjunto y Arquitecturas Unificadas:** La investigación reciente se centra en modelos que pueden realizar tareas de segmentación semántica y de instancias dentro de una sola arquitectura unificada, buscando eficiencia y coherencia. Esto incluye esfuerzos para desarrollar modelos entrenables de extremo a extremo que puedan producir mapas de segmentación panóptica directamente.

Modelos Panoptic: <https://paperswithcode.com/task/panoptic-segmentation/>

Modelos Instance:

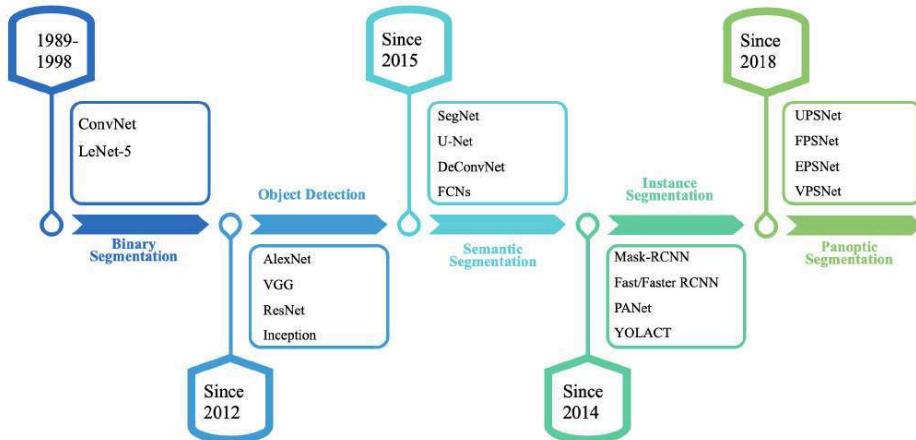
<https://paperswithcode.com/task/instance-segmentation/>

Modelos Semantic:

<https://paperswithcode.com/task/semantic-segmentation/>

Direcciones Futuras

- **Integración de Modalidades:** Incorporar modalidades de datos adicionales (por ejemplo, información de profundidad, datos temporales en secuencias de video) con entradas RGB tradicionales para mejorar la precisión de la segmentación.
- **Mejoras en Eficiencia:** Desarrollar modelos ligeros que mantengan una alta precisión mientras son implementables en dispositivos con recursos limitados.
- **Segmentación Granular:** Mejorar la capacidad de los modelos para delinear objetos a un nivel más granular, especialmente en escenas con múltiples occlusiones y variación en el tamaño de los objetos.



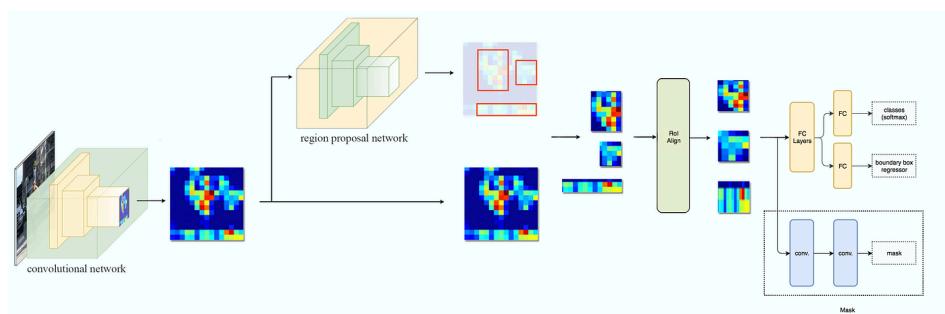
Evolución de los modelos de segmentación (Fuente: [Panoptic Segmentation: A Review](#))

Mask R-CNN

Mask R-CNN es una extensión del modelo Faster R-CNN, uno de los modelos más eficientes para la detección de objetos. Mientras que Faster R-CNN se enfoca en la detección de objetos proporcionando cuadros delimitadores (bounding boxes) y etiquetas de clase para cada objeto, Mask R-CNN va un paso más allá y agrega una capa de segmentación que proporciona una máscara de segmentación a nivel de píxel para cada instancia detectada. Esto significa que Mask R-CNN no solo identifica la ubicación de los objetos dentro de la imagen sino también su contorno.

Arquitectura de Mask R-CNN

La arquitectura de Mask R-CNN se divide en dos partes principales: una red de propuestas de regiones (Region Proposal Network, RPN) para detectar objetos y una red para predecir simultáneamente las clases, las cajas delimitadoras y las máscaras de los objetos detectados.



Fuentes: <https://jonathan-hui.medium.com/image-segmentation-with-mask-r-cnn-ebe6d793272>

1. **Backbone (Columna vertebral):** Primero, una imagen de entrada se pasa a través de una red de base (usualmente una ResNet modificada y añadida con una red Feature Pyramid Network, FPN) para extraer características a diferentes escalas. Pueden encontrarse versiones con diferentes backbones, por ejemplo ResNet101, ResNet50, o MobileNetV1.
2. **Region Proposal Network (RPN):** Estas características se utilizan para generar propuestas de regiones que podrían contener objetos. RPN es una red convolucional que escanea las características en todas las ubicaciones y predice, para cada ubicación, si hay un objeto y cuál es el cuadro delimitador de ese objeto.

3. **RoI Align:** A diferencia de Faster R-CNN que utiliza RoIPool, Mask R-CNN introduce RoIAlign para preservar la información espacial precisa. RoIAlign evita la cuantización (redondeo) durante la asignación de las propuestas de regiones a las características, permitiendo una extracción de características más precisa para la segmentación.

4. **Cabezas de clasificación, regresión de cuadros delimitadores y segmentación de máscaras:** Para cada propuesta generada por el RPN, Mask R-CNN utiliza tres cabezas en paralelo:

- Una cabeza de clasificación que determina la clase del objeto.
- Una cabeza de regresión de cuadros delimitadores que afina el cuadro delimitador para el objeto.
- Una cabeza de segmentación de máscaras que produce una máscara binaria a nivel de píxel para cada instancia.

SAM (Segment Anything)

SAM (Segment Anything Model) es un proyecto revolucionario [presentado por Meta](#), destinado a democratizar la segmentación de imágenes mediante la introducción de un modelo, dataset y tarea nuevos para la segmentación de imágenes. SAM es un modelo fundamental para la segmentación de imágenes que aprende una noción general de objetos, permitiendo generar máscaras para cualquier objeto en imágenes o videos, **incluyendo aquellos no vistos durante el entrenamiento**. Su diseño flexible y capacidades de generalización prometen amplias aplicaciones en múltiples dominios sin la necesidad de entrenamiento adicional, abriendo nuevas posibilidades para la investigación y aplicaciones prácticas en visión por computadora.

https://prod-files-secure.s3.us-west-2.amazonaws.com/db285804-d357-495f-b59b-667118e5e4ea/de97a4d7-0a99-4d95-858f-7e61f35247fa/402912772_617406953738599_4073655169170720504_n.mp4

La capacidad de segmentar objetos *que no fueron vistos durante el entrenamiento*, lo posiciona como un modelo fundamental en la segmentación de imágenes, prometiendo aplicaciones amplias en investigación y aplicaciones prácticas sin la necesidad de reentrenamiento específico. SAM es lo suficientemente general como para cubrir un amplio conjunto de casos de uso y se puede utilizar de forma inmediata en nuevos "dominios" de imágenes, ya sean fotografías submarinas o microscopía celular, sin requerir aprendizaje adicional (una capacidad a menudo denominada transferencia de disparo cero o zero-shot).

Ejemplos prácticos

Instance Segmentation

Una de las formas más sencillas y rápidas de realizar Instance Segmentation en Python, es usando la librería [pixellib](#) (<https://github.com/ayoolaolafenwa/Pixellib>). En pocas líneas de código podemos realizar segmentación sobre una imagen. Veremos un ejemplo utilizando el modelo [PointRend](#) para realizar la segmentación. A continuación se muestra una comparación entre [PointRend](#) y el tradicional modelo [Mask R-CNN](#):



Pares de resultados de ejemplo de Mask R-CNN con su cabezal de máscara estándar (imagen izquierda) frente a PointRend (imagen derecha), usando ResNet-50 con FPN (Feature Pyramid Network). Se puede observar cómo PointRend predice máscaras con detalles sustancialmente más finos alrededor de los límites de los objetos.

Ahora si veamos el ejemplo práctico:

```
# Instalación de la librería PixelLib
# !pip install pixellib

# Importación de las bibliotecas necesarias
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation
from IPython.display import Image

# Creación de una instancia del modelo de segmentación
ins = instanceSegmentation()

# Carga del modelo preentrenado
ins.load_model("pointrend_resnet50.pkl")

# Segmentación de la imagen y guardado de la imagen segmentada con las cajas delimitadoras
ins.segmentImage("police_motorbike.jpg", show_bboxes=True, output_image_name="police_motorbike_segmented.jpg")

# Muestra la imagen original
display(Image(filename="police_motorbike.jpg"))

# Muestra la imagen segmentada
display(Image(filename="police_motorbike_segmented.jpg"))
```

Como resultado obtendremos las siguientes imágenes:



Imagen original

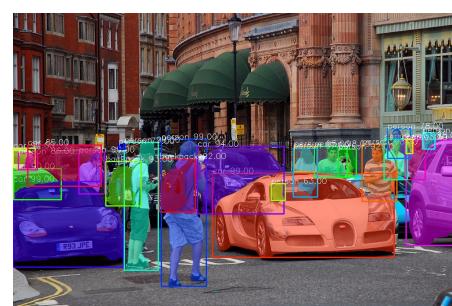


Imagen con máscaras y bounding boxes

Panoptic Segmentation

El siguiente ejemplo permite realizar segmentación panóptica. Mediante el uso de librerías como `PIL` (Python Imaging Library), `torch`, `numpy`, `matplotlib`, `seaborn`, y especialmente `transformers`, este código demuestra cómo cargar, procesar y visualizar una imagen desde una URL para luego identificar y segmentar cada objeto presente en ella. La segmentación se realiza con el modelo `DETR (Detection Transformer)`, específicamente con una variante preentrenada para panoptic segmentation conocida como `facebook/detr-resnet-50-panoptic`, demostrando la potencia de los modelos basados en transformers en tareas de visión por computadora.

El proceso inicia con el procesamiento del modelo DETR, el cual ha sido preentrenado para identificar y segmentar múltiples objetos en una sola pasada. A través de un forward pass, el modelo genera predicciones que luego son post-procesadas para obtener segmentaciones detalladas en formato COCO, un estándar en la comunidad de visión por computadora para la anotación, segmentación y detección de objetos.

```
import io
import requests
from PIL import Image, ImageDraw, ImageFont
import torch
import numpy as np
import matplotlib.pyplot as plt
import itertools
import seaborn as sns
from transformers import DetrFeatureExtractor, DetrForSegmentation
from transformers.models.detr.feature_extraction_detr import rgb_to_id

# Cargar la imagen
url = "https://raw.githubusercontent.com/matterport/Mask_RCNN/master/images/7933423348_c30bd9bd4e_z.jpg"
image = Image.open(requests.get(url, stream=True).raw)

# Cargar el modelo y el extractor de características
feature_extractor = DetrFeatureExtractor.from_pretrained("facebook/detr-resnet-50-panoptic")
model = DetrForSegmentation.from_pretrained("facebook/detr-resnet-50-panoptic")

# Preparar la imagen para el modelo
inputs = feature_extractor(images=image, return_tensors="pt")

# Paso forward
outputs = model(**inputs)

# Postprocesamiento
processed_sizes = torch.as_tensor(inputs["pixel_values"].shape[-2:]).unsqueeze(0)
result = feature_extractor.post_process_panoptic(outputs, processed_sizes, threshold=0.85)[0]

# Convertir la segmentación a numpy
panoptic_seg = Image.open(io.BytesIO(result["png_string"]))
panoptic_seg = np.array(panoptic_seg, dtype=np.uint8)
panoptic_seg_id = rgb_to_id(panoptic_seg)

# Preparar la paleta de colores
palette = itertools.cycle(sns.color_palette())

# Crear la imagen segmentada
segmented_image = Image.fromarray(np.zeros_like(panoptic_seg, dtype=np.uint8))
draw = ImageDraw.Draw(segmented_image)

# Añadimos un mapeo manual de los IDs de las categorías a los nombres, basado en las clases comunes de COCO
COCO_LABELS = {
    1: 'persona', 2: 'bicicleta', 3: 'coche', 4: 'motocicleta', 5: 'avión',
```

```

6: 'autobús', 7: 'tren', 8: 'camión', 9: 'barco', 10: 'semáforo',
11: 'hidrante', 13: 'señal de stop', 14: 'parquímetro', 15: 'banco', 16: 'pájaro',
17: 'gato', 18: 'perro', 19: 'caballo', 20: 'oveja', 21: 'vaca',
22: 'elefante', 23: 'oso', 24: 'cebra', 25: 'jirafa', 27: 'mochila',
28: 'paraguas', 31: 'bolso de mano', 32: 'corbata', 33: 'maleta', 34: 'frisbee',
36: 'tabla de snowboard', 37: 'pelota deportiva', 38: 'cometa', 39: 'bate de béisbol',
40: 'guante de béisbol', 41: 'patineta', 42: 'tabla de surf', 43: 'raqueta de tenis',
44: 'botella', 46: 'plato de vino', 47: 'taza', 48: 'tenedor', 49: 'cuchillo',
50: 'cuchara', 51: 'tazón', 52: 'banana', 53: 'manzana', 54: 'sándwich',
55: 'naranja', 56: 'brócoli', 57: 'zanahoria', 58: 'perrito caliente', 59: 'pizza',
60: 'donut', 61: 'pastel', 62: 'silla', 63: 'sofá', 64: 'maceta', 65: 'cama',
67: 'mesa de comedor', 70: 'inodoro', 72: 'TV', 73: 'computadora portátil', 74: 'ratón',
75: 'control remoto', 76: 'teclado', 77: 'teléfono celular', 78: 'microondas',
79: 'horno', 80: 'tostadora', 81: 'fregadero', 82: 'refrigerador', 84: 'libro',
85: 'reloj', 86: 'florero', 87: 'tijeras', 88: 'oso de peluche', 89: 'secador de pelo',
90: 'cepillo de dientes',
}

# Ajusta el bucle para dibujar segmentos y etiquetas correctamente
for segment_info in result["segments_info"]:
    class_id = segment_info["category_id"]
    class_name = COCO_LABELS.get(class_id, 'Desconocido') # 'Desconocido' si el ID no está en el diccionario
    id = segment_info["id"]

    # Generar la máscara para este segmento específico
    mask = panoptic_seg_id == id
    color = np.array(next(palette)) * 255 # Convertir el color a un array de numpy adecuado

    # Convertir la máscara a una imagen de PIL para usarla como máscara en 'paste'
    mask_image = Image.fromarray((mask * 255).astype(np.uint8))

    # Crear una imagen del color del segmento que tenga las dimensiones correctas
    color_image = Image.new("RGB", segmented_image.size, color=tuple(color.astype(int)))

    # Pegar usando la máscara para aplicar solo este segmento
    segmented_image.paste(color_image, (0,0), mask=mask_image)

    # Dibujar el nombre de la clase en la posición inicial del segmento
    draw = ImageDraw.Draw(segmented_image)
    where = np.where(mask)
    if where[0].size > 0 and where[1].size > 0:
        x, y = np.min(where[1]), np.min(where[0])
        draw.text((x, y), class_name, fill='white')

    # Mostrar la imagen original y la segmentada
    plt.figure(figsize=(30, 15))
    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.title('Imagen Original')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(segmented_image)
    plt.title('Imagen Segmentada con Etiquetas')
    plt.axis('off')

    plt.show()

```

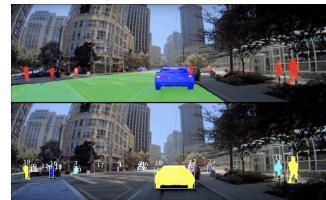
Para la visualización, el script emplea una combinación de técnicas. Utiliza una paleta de colores generada por `seaborn` para distinguir entre los diferentes objetos identificados y segmentados. Posteriormente, sobre la imagen segmentada, se dibujan los nombres de las clases correspondientes a cada objeto, utilizando un mapeo manual de identificadores a nombres de clases basado en el conjunto de datos COCO. Esta etapa destaca no solo la capacidad del modelo para segmentar precisamente los objetos, sino también para identificarlos correctamente según la categoría a la que pertenecen. Finalmente, se presenta la imagen original y su versión segmentada lado a lado para una comparación directa, mostrando el poderoso resultado de combinar DETR con técnicas de post-procesamiento y visualización en Python.



Aplicaciones

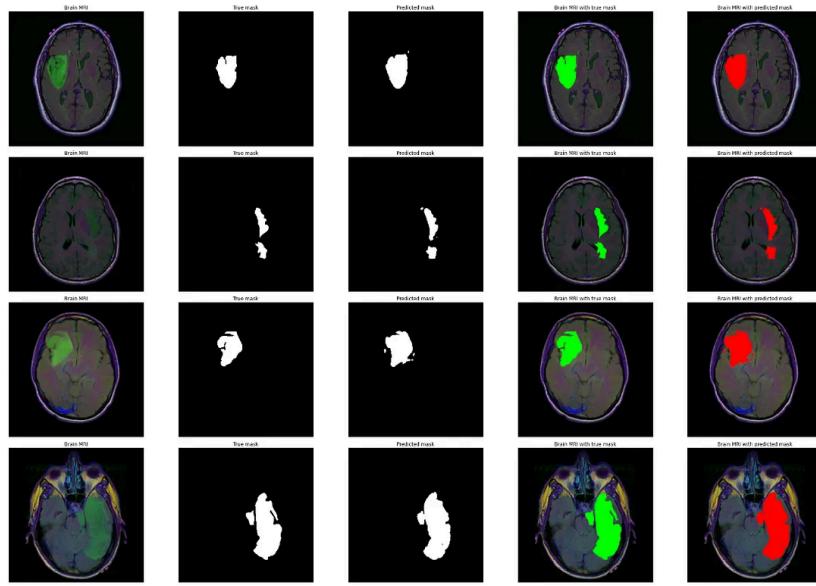
Las aplicaciones de la segmentación en el ámbito de la visión por computadora son vastas y abarcan una variedad de campos, demostrando su versatilidad y capacidad para impulsar innovaciones en múltiples dominios. A continuación, se detallan algunas aplicaciones clave:

- **Vehículos Autónomos:** La segmentación semántica y panóptica se utiliza para entender el entorno alrededor de los vehículos autónomos. Identificando y clasificando todos los elementos del entorno, como otros vehículos, peatones, señales de tráfico y la carretera, los sistemas autónomos pueden tomar decisiones informadas sobre cómo navegar de manera segura.



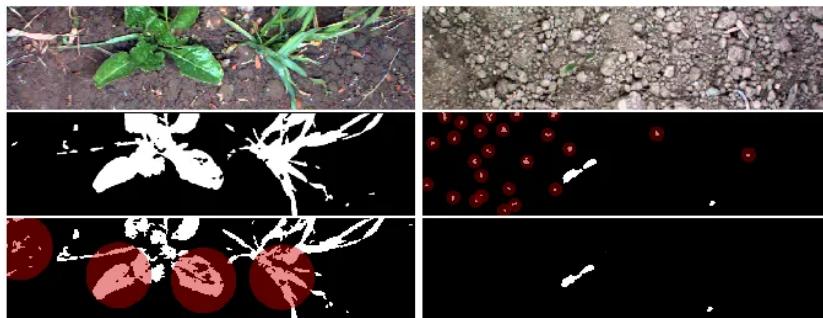
<https://blogs.nvidia.com/blog/drive-labs-panoptic-segmentation/>
(Video)

- **Diagnóstico Médico:** En el ámbito médico, la segmentación de imágenes juega un papel crucial en el diagnóstico y análisis de imágenes médicas. Se utiliza para segmentar regiones específicas de interés, como tumores en imágenes de resonancia magnética o rayos X, lo que permite a los médicos identificar y medir la progresión de enfermedades con precisión.



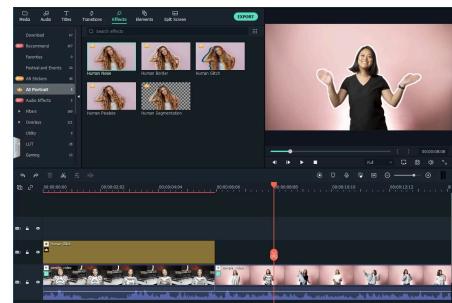
[Brain Tumor classification and detection from MRI images using CNN based on ResU-Net Architecture](#)

- **Agricultura de Precisión:** La segmentación semántica se emplea en la agricultura de precisión para analizar imágenes aéreas de cultivos, identificando áreas que necesitan atención, como aquellas afectadas por enfermedades, plagas o sequía. Esto permite una gestión más eficiente de los recursos y mejora la productividad.



[Real-Time Semantic Segmentation of Crop and Weed for Precision Agriculture Robots Leveraging Background Knowledge in CNNs](#)

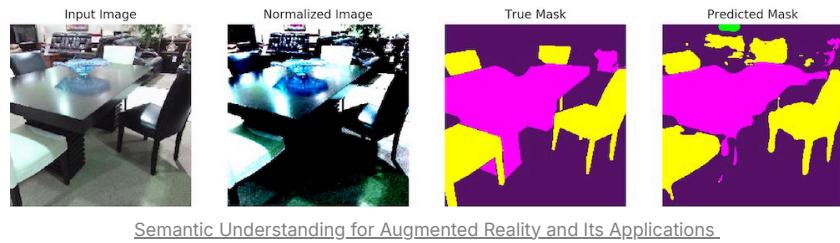
- **Control de Calidad en la Industria:** En la industria manufacturera, la segmentación de instancias se utiliza para inspeccionar productos en líneas de ensamblaje, identificando defectos o partes faltantes. Esto asegura que solo los productos que cumplen con los estándares de calidad sean distribuidos.
- **Edición y Análisis de Imágenes y Vídeos:** La segmentación es fundamental para aplicaciones de edición de imágenes y vídeos, permitiendo la manipulación precisa de objetos específicos, cambio de fondos o aplicación de efectos especiales. En el análisis de vídeos, facilita el seguimiento de objetos y la comprensión del comportamiento dentro de las escenas.



<https://filmora.wondershare.com/video-editing-tips/video-background-changer.html>

- **Robótica:** En robótica, la segmentación ayuda a los robots a comprender su entorno, permitiéndoles interactuar de manera efectiva con objetos y personas. Esto es crucial para tareas como la navegación autónoma, la manipulación de objetos y la colaboración con humanos.

- **Realidad Aumentada:** La segmentación permite integrar de manera realista elementos virtuales en entornos reales al entender la composición y estructura del espacio físico. Esto mejora la experiencia en aplicaciones de realidad aumentada, como juegos, herramientas educativas y aplicaciones de diseño.

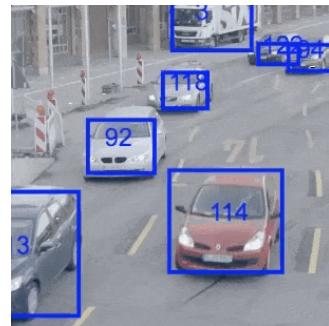


6. Object Tracking

Introducción y conceptos teóricos

El seguimiento de objetos, también conocido como "object tracking" en inglés, es un campo de estudio muy importante dentro de la visión por computadora que tiene como objetivo la identificación y seguimiento continuo de uno o varios objetos a lo largo del tiempo en secuencias de imágenes o videos. Esta disciplina combina algoritmos de detección de objetos, análisis de movimiento, y reconocimiento de patrones para mantener la continuidad de la identidad de los objetos en diferentes cuadros de un video, a pesar de los cambios en la orientación, escala, iluminación, y occlusiones.

El seguimiento de objetos se aplica en una amplia gama de áreas, desde la vigilancia de seguridad y el monitoreo de tráfico hasta la interacción humano-computadora, la edición de videos, la robótica autónoma, y la realidad aumentada. Cada una de estas aplicaciones presenta desafíos únicos, como la variabilidad en las condiciones de iluminación, la presencia de movimientos rápidos o irregulares, y la necesidad de procesamiento en tiempo real, lo que requiere el desarrollo de soluciones especializadas y eficientes.



El núcleo del seguimiento de objetos implica varios pasos fundamentales: la detección inicial del objeto o los objetos de interés, la predicción de su movimiento y cambio de estado, y la actualización de su posición en cada nuevo cuadro del video. A lo largo de los años, se han desarrollado diversas metodologías para abordar estos pasos, desde técnicas clásicas como el filtro de Kalman y el seguimiento basado en contornos, hasta enfoques más modernos que emplean aprendizaje profundo y redes neuronales convolucionales para mejorar la precisión y robustez del seguimiento.

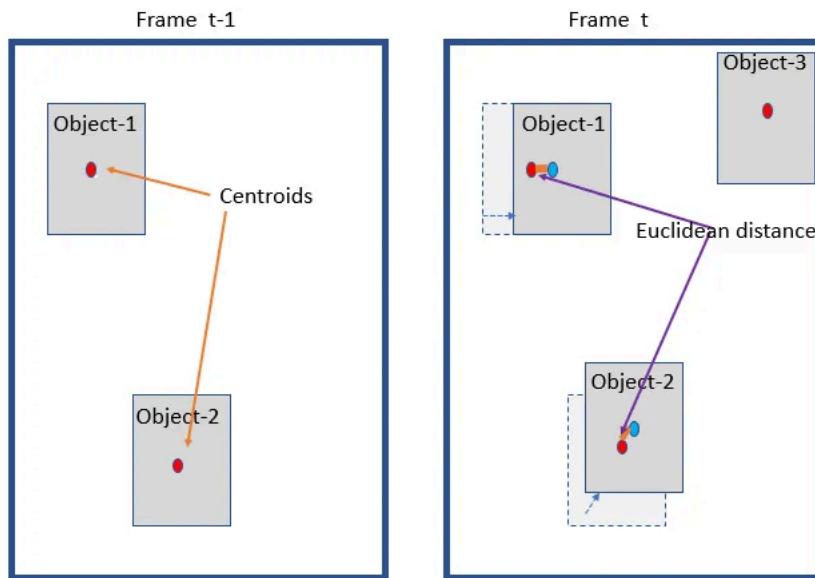
Los desafíos principales que enfrenta el tracking de múltiples objetos (MOT) son:

1. Detectar con precisión los objetos de interés en el cuadro con alta confianza. Los problemas con la detección precisa de objetos incluyen no detectar un objeto de interés, asignar una etiqueta de clase incorrecta a un objeto detectado o localizar incorrectamente un objeto identificado.
2. El cambio de identidad ocurre cuando dos objetos similares se superponen o se mezclan, causando el cambio de identidad; por lo tanto, es difícil mantener el seguimiento del ID del objeto.
3. Distorsión de fondo: Un fondo sobrecargado hace difícil detectar objetos pequeños durante la detección de objetos.
4. Oclusión: ocurre cuando algo que queremos ver está oculto u ocluido por otro objeto.
5. Múltiples espacios, deformación o rotación del objeto.
6. Iluminación de la imagen.
7. Rastros visuales o manchas capturadas en cámara debido al desenfoque de movimiento (motion blur).

 Generalmente, el tracking de objetos está asociado a un detector de objetos. La salida del detector de objetos (bounding boxes) se utiliza como entrada a algoritmos de tracking que intentan vincular las detecciones a lo largo del tiempo en el transcurso de un video con un único ID.
Algunos modelos fusionan la detección y el tracking end-to-end, pero suelen requerir de alto poder computacional para su funcionamiento en tiempo real.

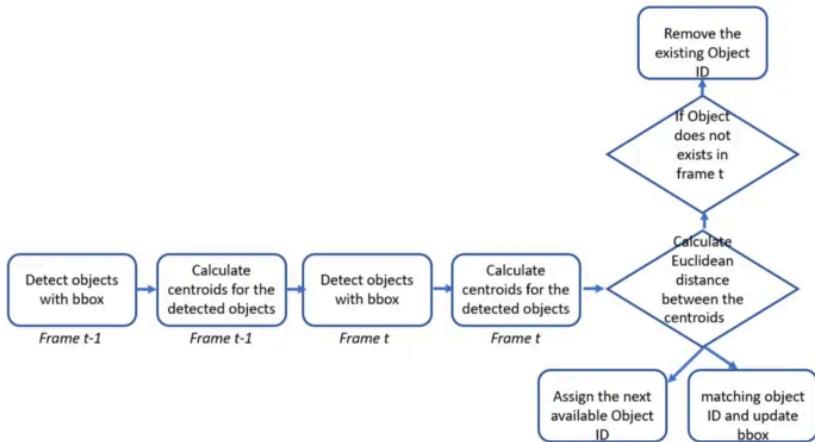
Tracking basado en centroides

Uno de los métodos más básicos para el seguimiento de los objetos, es el basado en las centroides de los objetos:



El seguimiento de objetos basado en centroides utiliza la distancia euclíadiana entre los centroides de los objetos detectados entre dos cuadros consecutivos en un video.

- **Paso 1:** Los objetos son detectados utilizando un cuadro delimitador para el cuadro en el tiempo t-1.
- **Paso 2:** Calcular los centroides para el objeto detectado para el cuadro en el tiempo t-1.
- **Paso 3:** Los objetos son detectados utilizando un cuadro delimitador para el cuadro en el tiempo t. Asignar un ID único a los objetos.
- **Paso 4:** Calcular los centroides del objeto detectado para el cuadro en el tiempo t.
- **Paso 5:** Calcular la distancia euclíadiana entre los centroides de todos los objetos detectados en los cuadros t-1 y t.
- **Paso 6:** Si la distancia entre el centroide en el tiempo t-1 y t es menor que el umbral, es el mismo objeto en movimiento. Por lo tanto, usar el ID de objeto existente y actualizar las coordenadas del cuadro delimitador del objeto al nuevo valor del cuadro delimitador.
- **Paso 7:** Si la distancia entre el centroide en el tiempo t-1 y t supera el umbral, agregar un nuevo ID de objeto.
- **Paso 8:** Cuando los objetos detectados en el cuadro anterior no pueden ser emparejados con ningún objeto existente, eliminar el ID del objeto del seguimiento.



Fuente: <https://medium.com/aiguy/a-centroid-based-object-tracking-implementation-455021c2c992>

Ventajas del seguimiento por centroides:

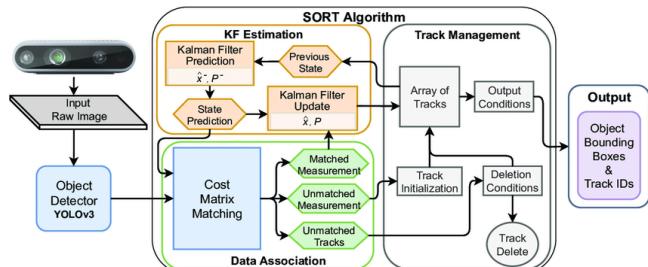
- Simplicidad:** El seguimiento por centroides es conceptualmente simple y fácil de implementar, lo que lo hace accesible para una amplia gama de aplicaciones.
- Eficiencia computacional:** Dado que solo se calculan y comparan las posiciones centrales de los objetos, el proceso puede ser bastante rápido, lo que lo hace adecuado para aplicaciones en tiempo real.
- Buen rendimiento en escenarios simples:** Para videos con objetos bien separados y sin occlusiones complejas, el seguimiento por centroides puede ser bastante efectivo y preciso.

Desventajas del seguimiento por centroides:

- Sensibilidad a las occlusiones:** El método puede fallar o dar resultados inexactos en situaciones donde los objetos se solapan o se ocultan entre sí, ya que los centroides pueden no representar correctamente la posición de los objetos ocluidos.
- Dependencia del proceso de detección:** La precisión del seguimiento por centroides está directamente ligada a la calidad de la detección de objetos. Los errores en la detección pueden propagarse a través del seguimiento, resultando en identificaciones incorrectas.
- Dificultades con la variación de forma y tamaño:** Los objetos que cambian drásticamente de forma o tamaño entre cuadros pueden ser difíciles de seguir correctamente, ya que el cambio en el área del objeto afecta la posición del centroide.
- Limitaciones en escenarios complejos:** Para videos con escenas congestionadas, interacciones frecuentes entre objetos o cambios rápidos en la dinámica de los objetos, el seguimiento basado en centroides puede no ser suficientemente robusto.

Algoritmo SORT

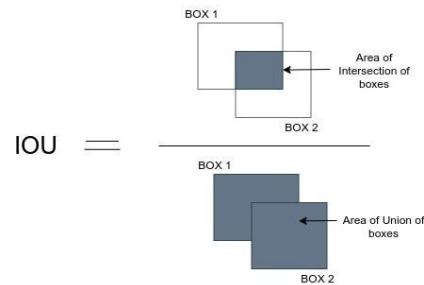
El algoritmo SORT (2016) es un algoritmo de seguimiento eficaz en tiempo real, que se transformó en un modelo de referencia por su eficacia, aunque luego fueron apareciendo muchos otros modelos superadores, por ejemplo DeepSORT o StrongSORT.



Fuente: [Sort and Deep-SORT Based Multi-Object Tracking for Mobile Robotics: Evaluation with New Data Association Metrics](https://arxiv.org/pdf/1612.00268.pdf)

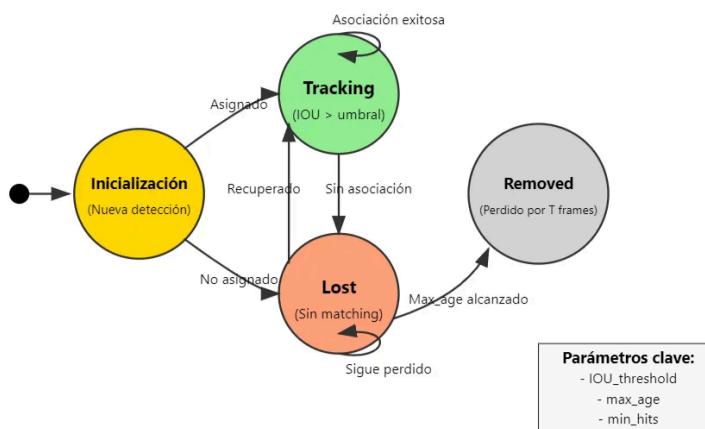
Combina los conceptos de filtros de Kalman y algoritmo húngaro para rastrear objetos.

- **Estimación:** las detecciones del cuadro actual se propagan al siguiente cuadro estimando la posición de los objetos utilizando un modelo de velocidad constante. Estos componentes de la velocidad se resuelven mediante el filtro de Kalman.
- **Detección:** el algoritmo SORT utiliza como fuente los resultados de un detector de objetos, para identificar objetos en el primer fotograma de la secuencia de vídeo. A cada objeto detectado se le asigna un número de identificación único.
- **Asociación de datos:** Una vez que se tienen predicciones para los objetos existentes y se detectan nuevos objetos en el cuadro actual, se utiliza el algoritmo húngaro para asociar las detecciones actuales con las predicciones. Esto se basa en la minimización de una métrica de costo, que normalmente es la IOU (Intersection Over Union), entre los cuadros delimitadores predichos y detectados.



Un valor de MOTA de 1 indica un seguimiento perfecto sin errores, mientras que un valor de 0 o negativo indica un rendimiento pobre, con muchos errores de seguimiento.

- **Actualización de estado:** una vez establecidas las asociaciones, el algoritmo SORT actualiza el estado de cada objeto rastreado incorporando la información recién detectada. Esto incluye actualizar la posición, la velocidad y otros atributos relevantes del objeto. Es por eso que mucho algoritmos de tracking implementan una máquina de estados o FSM (Finite State Machine).



Máquina de estados del algoritmo SORT. Se parte como base de una detección con un modelo que obtiene un bounding box, y luego se va siguiendo el estado según la actividad del objeto.

Datasets

En los últimos años, se han publicado varios conjuntos de datos para el seguimiento de múltiples objetos (MOT, por sus siglas en inglés), cada uno con características y propósitos específicos. Estos son algunos de los conjuntos de datos típicos utilizados en el campo del seguimiento de objetos:

- **MOTChallenge:** MOTChallenge es uno de los puntos de referencia más comunes para el seguimiento de múltiples objetos. Proporciona algunos de los conjuntos de datos más grandes para el seguimiento de peatones que están disponibles públicamente. Por cada conjunto de datos, se proporcionan las anotaciones de ground truth ("verdad fundamental") para el conjunto de entrenamiento y las detecciones tanto para los conjuntos de entrenamiento

como de prueba. Los conjuntos de datos de MOTChallenge facilitan la comparación de algoritmos de seguimiento al proporcionar detecciones públicas que todos los modelos pueden utilizar, eliminando la calidad de detección de la evaluación del rendimiento del seguimiento. Los datasets más conocidos son [MOT16](#) y [MOT17](#), que son versiones actualizadas del conjunto de datos [MOT15](#), con una mayor precisión de [ground truth](#) y protocolos estrictamente seguidos. [MOT20](#) es un desafío de detección de peatones que presenta secuencias de video desafiantes en entornos no restringidos.

- **KITTI:** El conjunto de datos KITTI permite el seguimiento tanto de personas como de vehículos y se recopiló conduciendo un automóvil por una ciudad. Publicado en 2012, consta de 21 videos de entrenamiento y 29 de prueba, con un total de aproximadamente 19,000 cuadros. Incluye detecciones obtenidas usando detectores DPM y RegionLets, así como información estéreo y láser.
- **Otros conjuntos de datos:** Además de los mencionados, hay otros conjuntos de datos menos utilizados actualmente, como [UA-DETRAC](#), que se centra en vehículos rastreados desde cámaras de tráfico, y los conjuntos de datos TUD y [PETS2009](#), que se centran en peatones. Muchos de sus videos ahora son parte de los conjuntos de datos de MOTChallenge.

Métricas y benchmarks

A continuación se mencionan algunas métricas utilizadas para evaluar los modelos de tracking:

MOTA: La precisión de seguimiento de múltiples objetos (Multi-Object Tracking Accuracy) es una métrica definida para evaluar la exactitud en el seguimiento de múltiples objetos. Se utiliza para contar la acumulación de errores en el seguimiento, incluyendo el número de objetos de seguimiento y si coinciden. La fórmula para calcular MOTA es:

$$\text{MOTA} = 1 - \frac{|\text{FN}| + |\text{FP}| + |\text{IDSW}|}{|\text{gtDet}|}$$

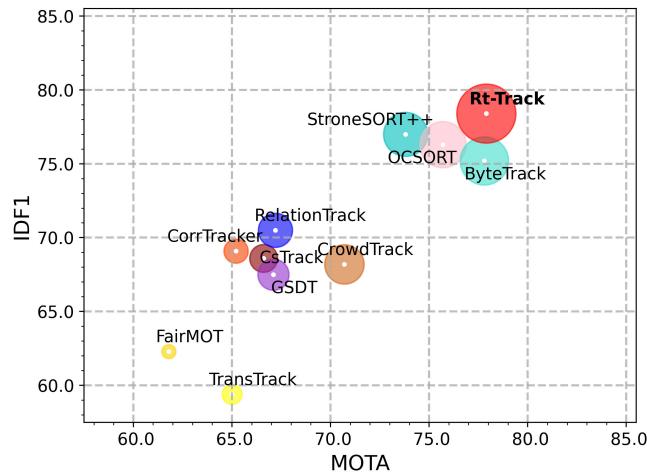
donde:

- **Falsos Negativos (FN):** Son objetos que existen en el ground truth (los objetos reales que deberían ser detectados y seguidos) pero que el algoritmo no detectó o no siguió correctamente. Es decir, son casos donde hay un objeto real en la escena, pero el sistema no generó ninguna predicción correspondiente.
- **Falsos Positivos (FP):** Son predicciones generadas por el algoritmo que no corresponden a ningún objeto real en el ground truth. El sistema "ve" algo que no existe realmente.
- **IDSW (Identity Switches):** Ocurren cuando el algoritmo cambia incorrectamente el identificador asignado a un objeto que está siguiendo.
- **gtDet** representa el número de objetos de seguimiento (ground truth).



Frecuentemente, se utiliza un modelo de detección de objetos asociado a un modelo de tracking. MOTA evalúa el rendimiento del sistema completo de detección-tracking como una unidad. No distingue explícitamente entre errores del detector y errores del tracker.

IDF1 (métricas de identificación): IDF1 enfatiza la precisión de la asociación en lugar de la detección y se utiliza como métrica secundaria en el punto de referencia MOTChallenge debido a su enfoque en medir la precisión de la asociación sobre la precisión de la detección.



Comparaciones de diferentes rastreadores en el conjunto de prueba de MOT20.

La fórmula de IDF1 es:

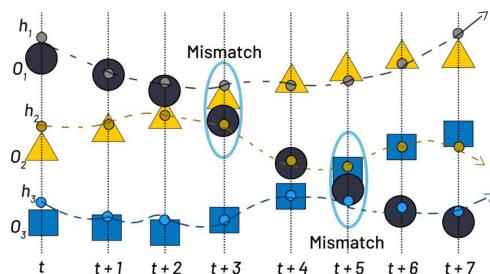
$$\text{IDF1} = \frac{2 \times \text{IDTP}}{\text{IDTP} + \text{IDFP} + \text{IDFN}}$$

Donde:

- **IDTP (True Positives de Identidad):** Número de coincidencias correctas de identidades entre las predicciones y las verdaderas.
- **IDFP (False Positives de Identidad):** Número de identidades incorrectamente asignadas.
- **IDFN (False Negatives de Identidad):** Número de identidades verdaderas que no fueron detectadas o asignadas incorrectamente.

Un valor alto de IDF1 indica que no solo se han detectado correctamente los objetos, sino que también se han mantenido consistentes sus identidades a lo largo de las secuencias de seguimiento, lo cual es esencial en aplicaciones donde la persistencia de identidad es crítica, como en el seguimiento de personas en videos de seguridad o en la gestión de interacciones entre vehículos en entornos de tráfico autónomo.

MOTP: La precisión de seguimiento de múltiples objetos (MOTP, por sus siglas en inglés) es otra métrica utilizada para evaluar si la posición del objeto está posicionada con precisión. Se calcula como la suma de las distancias entre la posición correspondiente del objeto en el cuadro t y la posición predicha, conocida también como el error de coincidencia, dividida por el número total de coincidencias entre el objeto y el objeto predicho en el cuadro t .



Discrepancias entre la posición del objeto "O" y la hipótesis "h"

Comparativa entre las métricas mencionadas:

Característica	MOTA	IDF1	MOTP
Significado	Multiple Object Tracking Accuracy	ID F1 Score	Multiple Object Tracking Precision
Qué mide	Precisión general del tracking	Consistencia de las identidades asignadas	Precisión de las posiciones estimadas
Fórmula básica	$1 - (FN + FP + ID) / GT$	$2 * IDTP / (2 * IDTP + IDFP + IDFN)$	$\Sigma(d_t) / (TP)$
Rango	-∞ a 1	0 a 1	0 a 1 (depende de la medida de distancia)
Interpretación	Mayor es mejor. 1 es perfecto.	Mayor es mejor. 1 es perfecto.	Menor es mejor. 0 es perfecto.
Sensibilidad a	Falsos negativos, falsos positivos, cambios de ID	Consistencia de IDs a lo largo del tiempo	Precisión de localización
Fortalezas	Medida global de rendimiento	Robusta a errores de corto plazo	Evaluúa la precisión de localización
Debilidades	No distingue entre tipos de errores	Puede ser menos intuitiva	No considera falsos positivos o negativos
Uso principal	Evaluación general del sistema	Evaluación de la consistencia de identidades	Evaluación de la precisión de localización

Donde:

- FN: Falsos Negativos
- FP: Falsos Positivos
- ID: Cambios de Identidad
- GT: Ground Truth
- IDTP: ID True Positive
- IDFP: ID False Positive
- IDFN: ID False Negative
- d_t: distancia entre la posición predicha y la real
- TP: True Positives

Referencias:

[Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics](#)

<https://paperswithcode.com/task/object-tracking>

Kalman:

<https://arshren.medium.com/an-easy-explanation-of-kalman-filter-ec2ccb759c46>

[Multiple Object Tracking in Recent Times: A Literature Review](#)

Ejemplos prácticos

Existen muchas implementaciones de modelos de tracking, y en muchos casos puede ser compleja la instalación de las librerías, ya que suelen requerir de GPU y a su vez de dependencias de frameworks como PyTorch, TensorFlow, CUDA, entre otros.

Sin embargo hay algunas librerías específicas, que tratan de facilitar el manejo de tracking, implementando diversos modelos. Las siguientes librerías son algunas de las que poseen un manejo de alto nivel para implementar MOT en Python:

<https://github.com/visionml/pytracking>

<https://github.com/open-mmlab/mmtracking>

<https://github.com/roboflow/supervision>

<https://github.com/wmuron/motpy>

<https://github.com/adipandas/multi-object-tracker>

<https://github.com/insight-platform/Similari>

En el ejemplo a continuación, se demuestra cómo integrar modelos de Object Detection con Multi-Object Tracking (MOT), para crear una solución completa que no solo detecta objetos en un video sino que también se sigue el movimiento a lo largo del tiempo. Se carga un modelo YOLO pre-entrenado para detectar objetos en cada cuadro de un video. Luego, se utiliza ByteTrack, implementado dentro de la librería **supervision** (<https://github.com/roboflow/supervision>), para realizar el seguimiento de estos objetos detectados. Además, se anotan los cuadros con bounding boxes y etiquetas que indican los identificadores de seguimiento, lo que facilita la visualización y comprensión del movimiento y la presencia de los objetos dentro del video:

```
# Importar las librerías necesarias
import supervision as sv # Librería para el seguimiento de objetos
from ultralytics import YOLO # Librería para la detección de objetos con YOLO
import numpy as np # Librería para operaciones matemáticas y manejo de arrays

# Definir rutas de los videos de entrada y salida, y el nombre del modelo
SOURCE_VIDEO_PATH = "highway_600.mp4" # Ruta al video de entrada
TARGET_VIDEO_PATH = "highway_600_tracking.mp4" # Ruta donde se guardará el video con seguimiento de objetos
MODEL_NAME = "yolov8x.pt" # Nombre del modelo preentrenado de YOLO a utilizar

# Inicializar el modelo de detección y el rastreador de objetos
model = YOLO(MODEL_NAME) # Cargar el modelo de YOLO
tracker = sv.ByteTrack() # Inicializar ByteTrack para el seguimiento de objetos

# Inicializar los anotadores para las cajas delimitadoras y las etiquetas
bounding_box_annotator = sv.BoundingBoxAnnotator() # Para dibujar cajas delimitadoras
label_annotator = sv.LabelAnnotator() # Para dibujar etiquetas (IDs de seguimiento)

# Definir la función de callback que procesa cada frame
def callback(frame: np.ndarray, index: int) → np.ndarray:
    results = model(frame)[0] # Detectar objetos en el frame actual con YOLO
    detections = sv.Detections.from_ultralytics(results) # Convertir resultados a formato de supervision
    detections = tracker.update_with_detections(detections) # Actualizar el estado del rastreador con las detecciones

    labels = [f"#{tracker_id}" for tracker_id in detections.tracker_id] # Crear etiquetas con los IDs de seguimiento

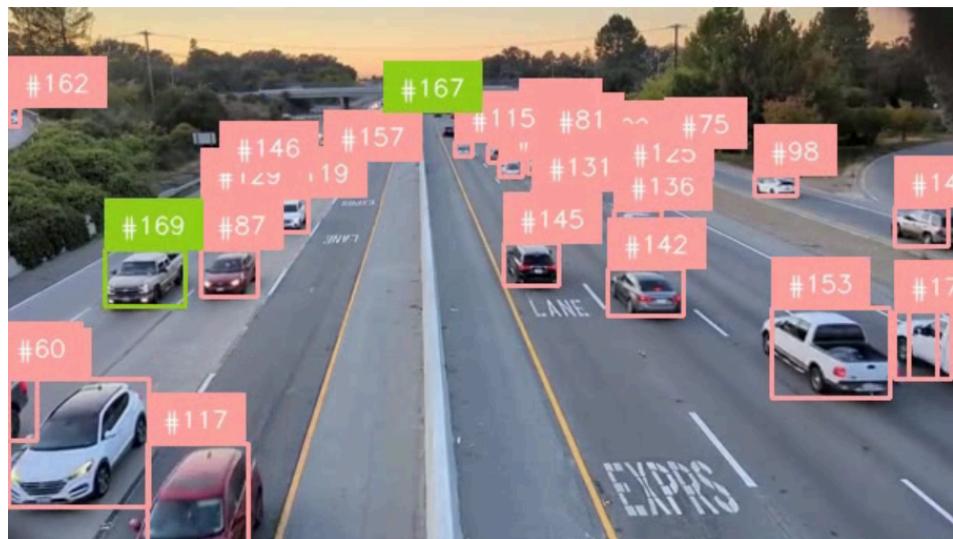
    # Anotar el frame con bounding boxes y etiquetas
    annotated_frame = bounding_box_annotator.annotate(
        scene=frame.copy(), detections=detections)
    annotated_frame = label_annotator.annotate(
        scene=annotated_frame)
```

```

scene=annotated_frame, detections=detections, labels=labels)
return annotated_frame # Devolver el frame anotado

# Procesar el video: leer, aplicar callback a cada frame y guardar el resultado
sv.process_video(
    source_path=SOURCE_VIDEO_PATH, # Ruta del video original
    target_path=TARGET_VIDEO_PATH, # Ruta del video resultante
    callback=callback # Función de callback para procesar cada frame
)

```



Los números sobre los bounding boxes, indican los IDs asignados, producto de aplicar el algoritmo de tracking ByteTrack

Aplicaciones

El Seguimiento de Múltiples Objetos (MOT) en visión por computadora tiene una amplia gama de aplicaciones en diversos sectores comerciales e industriales. Algunas de las aplicaciones más destacadas incluyen:

- **Vigilancia y Seguridad:** El MOT se utiliza ampliamente en sistemas de vigilancia para monitorear y rastrear el movimiento de personas, vehículos u objetos en áreas públicas o restringidas, ayudando en la detección de actividades sospechosas o no autorizadas.
- **Transporte y Logística:** En el sector del transporte, el MOT facilita el seguimiento de vehículos en autopistas, cruces urbanos y estacionamientos para mejorar la gestión del tráfico y la seguridad vial. En logística, puede ayudar en el seguimiento de paquetes en almacenes o durante el proceso de transporte.
- **Retail y Análisis de Comportamiento del Consumidor:** El MOT permite a los minoristas rastrear el movimiento de los clientes dentro de las tiendas, analizar patrones de comportamiento y optimizar la disposición de los productos y el diseño de las tiendas para mejorar la experiencia de compra.
- **Automoción y Vehículos Autónomos:** Se utiliza en el desarrollo de vehículos autónomos y sistemas de asistencia al conductor para detectar y rastrear otros vehículos, peatones y obstáculos en el entorno, contribuyendo a la toma de decisiones seguras en tiempo real.
- **Gestión de Eventos y Multitudes:** El MOT ayuda en la planificación y gestión de eventos a gran escala al monitorear y analizar el flujo y la densidad de las multitudes, lo que facilita la toma de decisiones para la evacuación de emergencia y el control de multitudes.
- **Análisis Deportivo:** En el deporte, el MOT se utiliza para rastrear el movimiento de los jugadores y la pelota durante los juegos, proporcionando datos valiosos para el análisis del rendimiento, la estrategia del equipo y la mejora del entrenamiento.
- **Robótica y Automatización Industrial:** El MOT es fundamental en aplicaciones de robótica para la detección y seguimiento de objetos, lo que permite a los robots realizar tareas complejas como la recogida y colocación, inspección y ensamblaje en entornos industriales.

- **Salud y Rehabilitación:** En el sector de la salud, el MOT se utiliza para rastrear los movimientos de los pacientes durante la rehabilitación física, proporcionando datos precisos sobre el progreso del tratamiento y ayudando en la personalización de los planes de terapia.
- **Entretenimiento y Medios:** El MOT permite efectos visuales avanzados en películas y videojuegos al rastrear el movimiento de actores o elementos interactivos, mejorando la creación de contenidos inmersivos y experiencias de usuario.

7. Face detection & Recognition

Introducción y conceptos teóricos

La detección de rostros (Face Detection) y el reconocimiento de rostros (Face Recognition) son dos conceptos relacionados, pero distintos dentro del campo de la visión por computadora, ambos enfocados en el análisis de imágenes para identificar y gestionar características humanas. A continuación, se explican estos conceptos en detalle:

Detección de Rostros (Face Detection)

La detección de rostros se refiere al proceso de identificar la presencia y la ubicación de rostros humanos en imágenes digitales o secuencias de video. Este proceso no se enfoca en la identidad de la persona en la imagen, sino simplemente en determinar si hay caras presentes y, en caso afirmativo, dónde están localizadas. La detección de rostros es el primer paso esencial en cualquier sistema que busque realizar tareas más complejas relacionadas con rostros, como el reconocimiento facial o el análisis de expresiones.

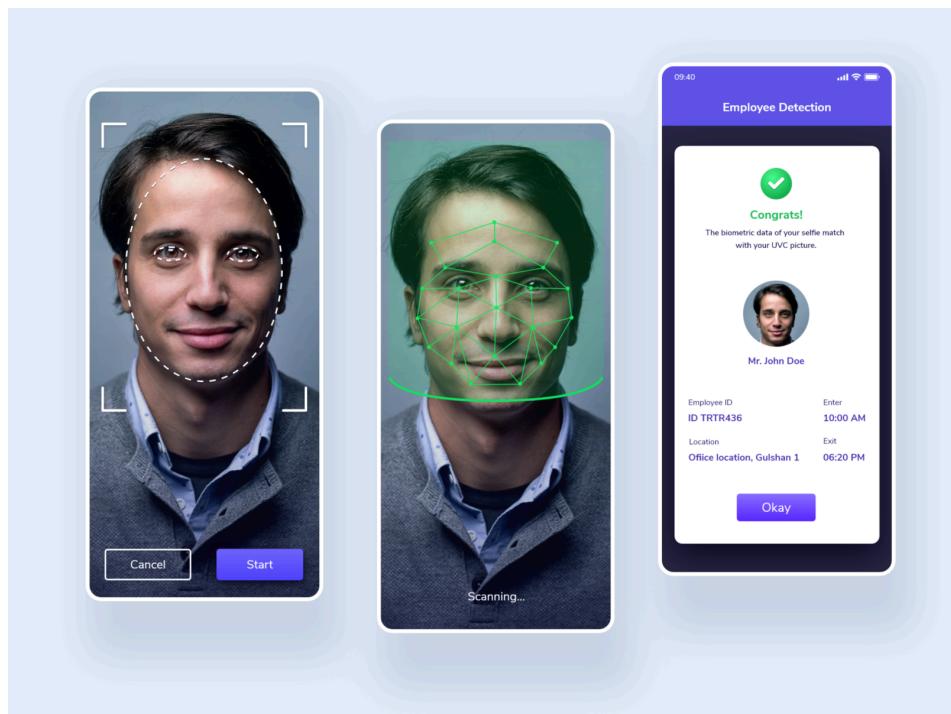


La detección de rostros utiliza algoritmos de visión por computadora y aprendizaje automático para analizar las características visuales de las imágenes. Estos algoritmos pueden buscar patrones específicos que se asemejen a componentes faciales (como los ojos, la nariz, la boca) y la forma general de un rostro. Uno de los desafíos en la detección de rostros es lograr precisión y eficacia bajo diversas condiciones, como diferentes iluminaciones, poses, expresiones faciales y occlusiones (barbijos, anteojos, gorras, etc.).

Reconocimiento de Rostros (Face Recognition)

El reconocimiento de rostros es un paso más allá de la detección. Una vez que se ha detectado un rostro, el reconocimiento busca identificar o verificar la identidad de la persona en la imagen o el video. Este proceso implica comparar las características faciales detectadas con una base de datos de rostros conocidos para encontrar una coincidencia.

El reconocimiento de rostros utiliza técnicas de aprendizaje profundo, especialmente redes neuronales convolucionales, para aprender y memorizar características faciales distintivas. Estas características se extraen de las imágenes y se convierten en vectores de características, que luego se pueden comparar con vectores de características en una base de datos para determinar la identidad.



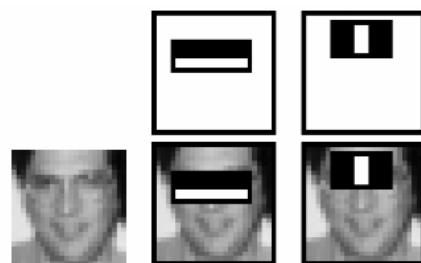
El reconocimiento facial se utiliza en una amplia gama de aplicaciones, desde sistemas de seguridad y vigilancia hasta autenticación biométrica en dispositivos personales y sistemas de gestión de identidades. También plantea desafíos importantes en términos de privacidad y ética, dado su potencial para el seguimiento y la identificación de individuos sin su consentimiento.

Algoritmos y métodos de detección

Métodos de detección tradicionales

Antes de la popularización de las redes neuronales convolucionales para la detección de rostros, existían varios métodos basados en técnicas de visión por computadora y aprendizaje automático tradicional. Estos métodos se centraban en la extracción de características y el uso de clasificadores para identificar la presencia de rostros en imágenes. Aunque los métodos basados en CNN han dominado el campo recientemente debido a su alta precisión y capacidad para manejar variaciones complejas en las imágenes, los métodos no basados en CNN jugaron un papel fundamental en el desarrollo inicial de la detección de rostros. Algunos de estos métodos incluyen:

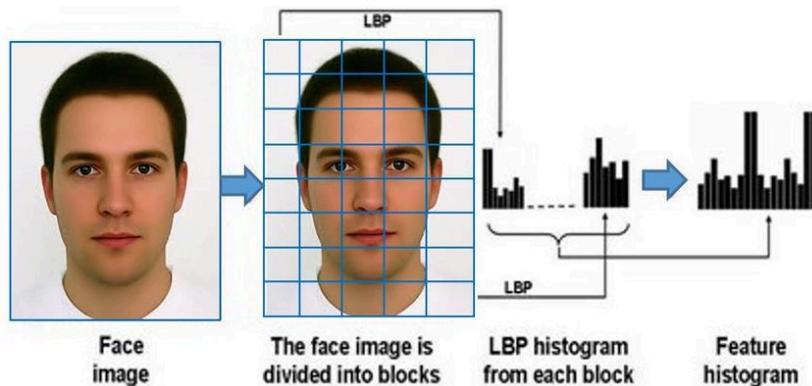
- 1. Viola-Jones Object Detection Framework:** También conocido como Haar Cascades, este es uno de los métodos más conocidos y ampliamente utilizado para la detección rápida de objetos, incluidos los rostros. Utiliza características Haar, una imagen integral para el cálculo rápido de características, y un clasificador en cascada basado en AdaBoost para seleccionar y utilizar las características más relevantes. Este método es eficaz para la detección de rostros en tiempo real y fue innovador en su capacidad para operar en condiciones de iluminación variadas y con rostros en diferentes orientaciones.



Paper original:
<https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>

- 1. Local Binary Patterns (LBP):** El método LBP se utiliza para la extracción de características texturales de las imágenes. En el contexto de la detección de rostros, LBP puede aplicarse para describir las texturas faciales, y luego se utilizan técnicas de clasificación, como máquinas de vectores de soporte (SVM), para identificar si una

región de la imagen contiene un rostro. LBP es eficaz para manejar variaciones en la iluminación y es relativamente rápido en comparación con los métodos basados en aprendizaje profundo.



1. Histogram of Oriented Gradients (HOG): El descriptor HOG se utiliza para capturar la estructura de bordes y gradientes de una imagen, lo que puede ser indicativo de la presencia de rostros. Similar al método LBP, después de extraer características HOG de una imagen o región de interés, se utilizan clasificadores como SVM para determinar la presencia de rostros. Este método ha sido popular para la detección de peatones y se adaptó para la detección de rostros con resultados competitivos.

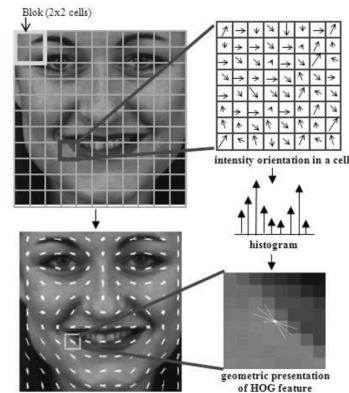
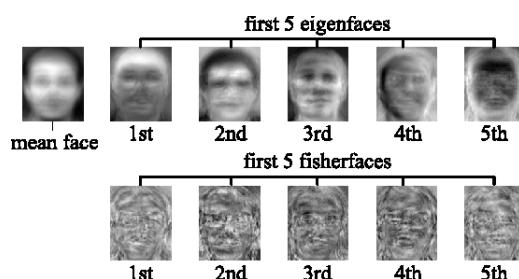


Fig. 3. Histogram of oriented gradient extraction from face.

1. Eigenfaces y Fisherfaces: Aunque más asociados con el reconocimiento de rostros, estos métodos también pueden aplicarse para la detección mediante la modelización del espacio de características faciales. Eigenfaces utiliza análisis de componentes principales (PCA) para reducir la dimensión de los datos de imagen facial y capturar las principales variaciones entre rostros. Fisherfaces extiende este enfoque utilizando análisis discriminante lineal (LDA) para maximizar la separación entre clases (rostros y no rostros).



Resumen comparativo de métodos tradicionales

Método	Velocidad	Precisión	Robustez a Iluminación	Robustez a Poses	Recursos Computacionales	Casos de Uso Ideales
Viola-Jones (Haar Cascades)	★★★★★	★★★	★★	★★	★ (muy bajos)	Aplicaciones en tiempo real, dispositivos móviles, sistemas embebidos

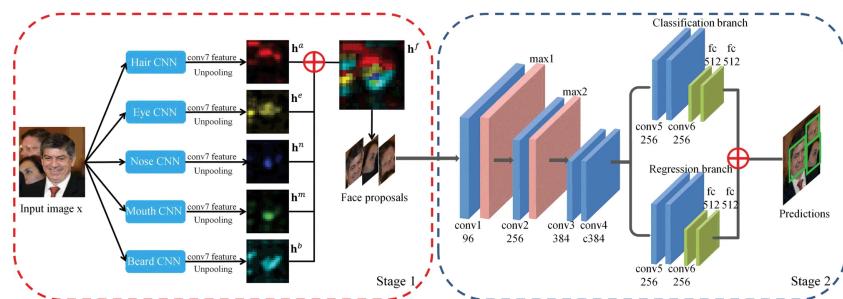
Método	Velocidad	Precisión	Robustez a Iluminación	Robustez a Poses	Recursos Computacionales	Casos de Uso Ideales
Local Binary Patterns (LBP)	★★★★★	★★★	★★★★	★★	★★ (bajos)	Sistemas con iluminación variable, balance velocidad-precisión
Histogram of Oriented Gradients (HOG)	★★★	★★★★★	★★★★★	★★★	★★★ (moderados)	Aplicaciones que priorizan precisión sobre velocidad, vigilancia
Eigenfaces	★★	★★★	★	★★	★★★ (moderados)	Sistemas controlados, bases de datos grandes de rostros
Fisherfaces	★★	★★★	★★	★★	★★★ (moderados)	Mejor discriminación entre rostros y no-rostros

Fuente: Claude 3.7 Sonnet y validado por Gemini 2.5 Pro

Métodos de detección basados en CNN

Estos métodos varían en su enfoque, desde el uso de características faciales específicas hasta el empleo de redes neuronales complejas para aprender a detectar rostros de manera más general. A continuación, se describen algunos de los métodos más destacados:

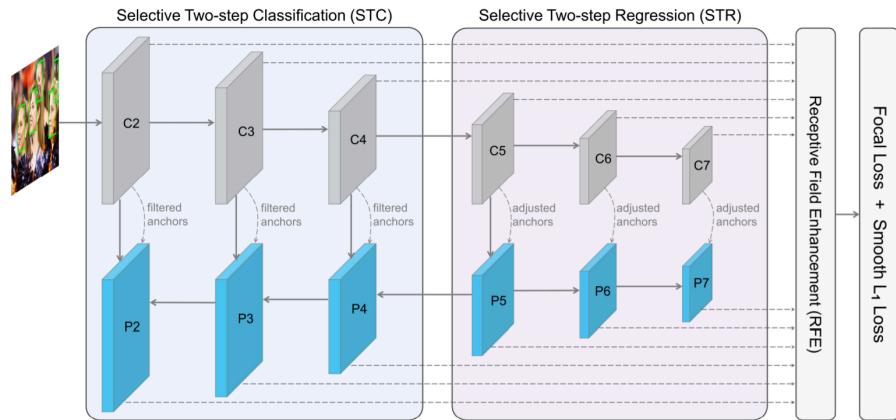
1. **Faceness-Net**: La supervisión basada en atributos faciales puede mejorar efectivamente la capacidad de una red de detección de rostros para manejar occlusiones severas. Propone una red de dos etapas donde la primera etapa aplica varias ramas para generar mapas de respuesta de diferentes partes faciales y la segunda etapa refina la ventana candidata usando una CNN de múltiples tareas.



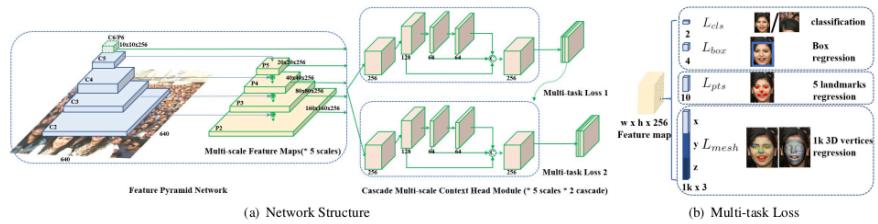
1. **SRN (Selective Refinement Network)**: Se destaca por su alto rendimiento, especialmente en la detección de rostros pequeños, y su capacidad de funcionar en tiempo real. Introduce operaciones novedosas de clasificación y regresión en dos pasos de forma selectiva en un detector de rostros basado en anclas:

- a. Módulo de clasificación en dos pasos selectivo (STC): Su objetivo es filtrar la mayoría de los "anclas negativas simples" (regiones de la imagen que no contienen rostros) de las capas de detección de bajo nivel. Esto reduce la cantidad de elementos que el clasificador posterior debe analizar, mejorando la eficiencia.
- b. Módulo de regresión en dos pasos selectivo (STR): Su objetivo es ajustar de manera general la ubicación y el tamaño de las anclas de las capas de detección de alto nivel. Esto proporciona una mejor inicialización para el regresor posterior, que refina la ubicación del rostro con mayor precisión.

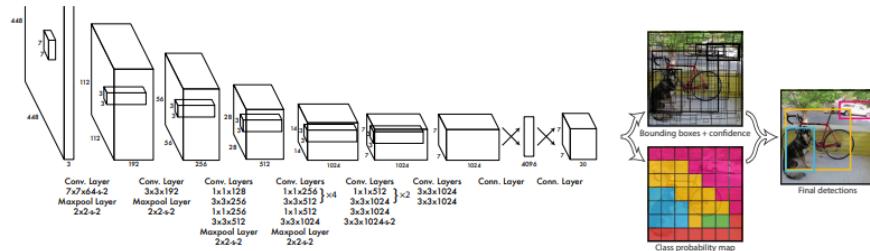
Además posee un bloque de Mejora del Campo Receptivo (RFE) para capturar mejor los rostros en poses extremas.



2. **RetinaFace**: Utiliza FPN para extraer características de imagen en varios niveles. A diferencia de los métodos de detección de dos pasos como SRN, RetinaFace presenta un detector de rostros de una sola etapa y realiza la detección de rostros de manera multi-tarea, prediciendo al mismo tiempo un puntaje de rostro, un cuadro de rostro, cinco puntos de referencia facial y vértices faciales 3D densos proyectados en el plano de la imagen.

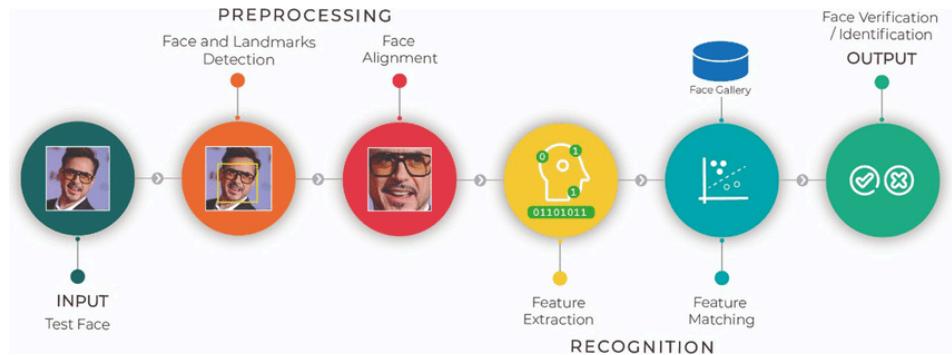


3. **YOLO (You Only Look Once)**: Aunque originalmente diseñado para la detección de objetos en general, las versiones de YOLO se han adaptado para la detección de rostros, ofreciendo una detección en tiempo real mediante la predicción simultánea de múltiples clases y ubicaciones de objetos en una sola pasada a través de la red. Existen variantes de YOLO, diseñados específicamente para detección de rostros, como [Yolo5Face](#).



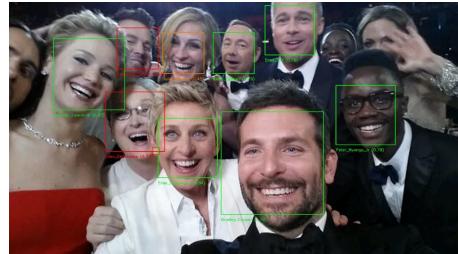
Reconocimiento de rostros

El reconocimiento de rostros es un proceso computacional que identifica o verifica a una persona a partir de una imagen digital o un marco de video de una fuente de video. El pipeline típico de procesamiento es el siguiente:



Fuente: [Benchmarking lightweight face architectures on specific face recognition scenarios](#)

- Detección de Rostros:** El primer paso es detectar la presencia de rostros en una imagen. Esto implica identificar las áreas de la imagen que contienen rostros y separarlas del resto. Técnicas como la detección de bordes y regiones, y algoritmos como Viola-Jones (Haar Cascades) se utilizan comúnmente para este propósito.

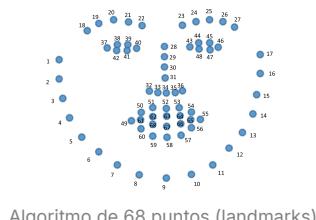


- Preprocesamiento:** Una vez detectado el rostro, la imagen se preprocesa para mejorar la calidad y reducir las variaciones debidas a la iluminación, orientación, y expresión facial. Esto puede incluir ajustes de iluminación, recorte, y escalado.

Para alinear los rostros, se aplica la detección de puntos de referencia faciales para detectar puntos de referencia clave en los rostros. La intuición es localizar puntos de referencia como ojos, nariz, cejas y labios que se encuentran en todo rostro humano. Hay diversos algoritmos de detección de puntos de referencia ampliamente utilizados, como por ejemplo, los algoritmos de puntos de referencia de 68 puntos (librería [Dlib](#)) y de [puntos de referencia de 5 puntos](#).



Alineación del rostro



Algoritmo de 68 puntos (landmarks)

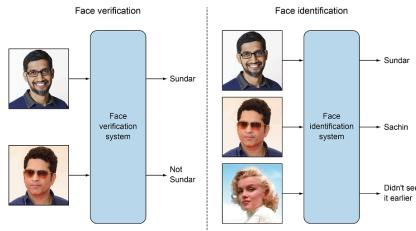


Algoritmo de 5 puntos

- Extracción de Características:** El siguiente paso es extraer características distintivas del rostro que puedan ser utilizadas para distinguirlo de otros. Esto puede incluir rasgos como la forma de los ojos, nariz y boca, así como texturas y patrones de la piel. Métodos populares incluyen el uso de puntos de referencia faciales, [análisis de componentes principales \(PCA\)](#) para obtener [Eigenfaces](#), o el uso de [filtros Gabor](#).

- Comparación y Reconocimiento:** Las características extraídas se comparan con un conjunto de datos conocido (base de datos de rostros) para encontrar coincidencias. Esto se puede hacer utilizando diversas técnicas de clasificación y comparación, como máquinas de vector soporte (SVM), redes neuronales convolucionales, o comparación directa de características. El objetivo es determinar cuán similar es el rostro detectado a cualquier rostro dentro de la base de datos, sin asignar una identidad específica aún.

- Verificación o Identificación:** En la etapa final, el sistema verifica si el rostro coincide con una identidad específica (verificación) o identifica a la persona comparando el rostro con una base de datos de rostros conocidos (identificación).

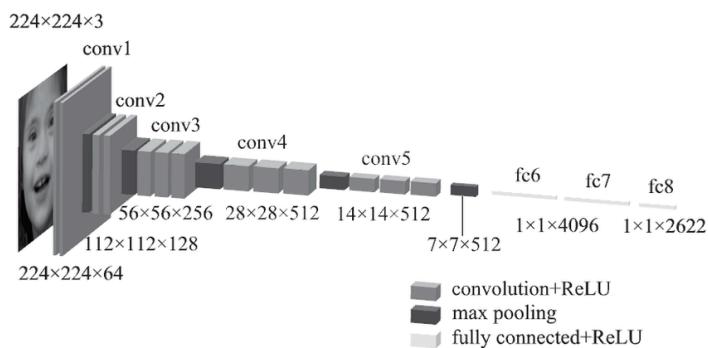


Métodos de reconocimiento de rostros basados en CNN

Los modelos de reconocimiento de rostros basados en Redes Neuronales Convolucionales (CNN) han transformado significativamente el campo del reconocimiento facial, ofreciendo una precisión, eficiencia y robustez sin precedentes en comparación con los enfoques tradicionales basados en características manuales y técnicas de aprendizaje de máquina más simples. La adaptabilidad y profundidad de las CNN permiten que estos modelos aprendan una representación rica y jerárquica de las imágenes faciales, capturando desde características visuales básicas hasta aspectos complejos de la identidad facial en distintos niveles.

Varios modelos basados en CNN han marcado pauta en el desarrollo de tecnologías de reconocimiento facial, estableciéndose como referencias clave en el área. A continuación, se presentan algunos de estos modelos emblemáticos:

- **VGG-Face** (2015): Este modelo es una adaptación del VGG-Net, diseñado específicamente para el reconocimiento facial. Con su arquitectura profunda y el uso de pequeños filtros convolucionales, VGG-Face consigue una representación detallada de las características faciales, lo que le permite alcanzar un rendimiento sobresaliente en diversas bases de datos de reconocimiento facial. Es popular también el dataset utilizado para su entrenamiento (https://www.robots.ox.ac.uk/~vgg/data/vgg_face/)

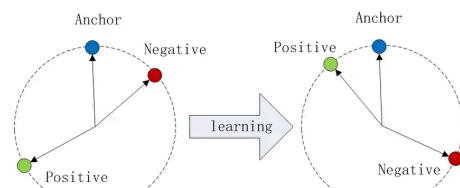
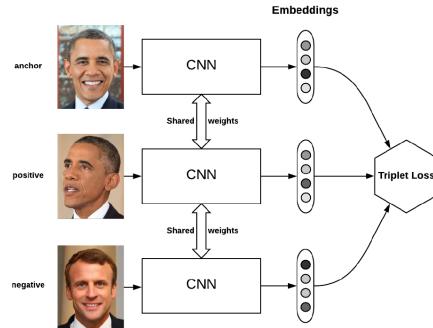


El modelo VGG-Face fue entrenado originalmente como un clasificador con exactamente 2,622 clases. Durante el entrenamiento, la red aprendió a distinguir entre 2,622 identidades diferentes (celebridades y figuras públicas), donde cada una representaba una clase distinta. La capa final softmax de 2,622 neuronas se entrenó para producir la probabilidad de que una imagen de entrada perteneciera a cada una de estas identidades específicas.

Este enfoque de clasificación supervisada permitió que la red aprendiera representaciones faciales robustas y discriminativas. Sin embargo, lo que hace útil a VGG-Face para aplicaciones generales de reconocimiento facial es que, aunque fue entrenado para estas 2,622 personas específicas, las capas anteriores de la red (especialmente la capa FC7 de 4,096 dimensiones) aprendieron características faciales universales que funcionan bien para representar cualquier rostro, incluso de personas que no estaban en el conjunto de entrenamiento original.

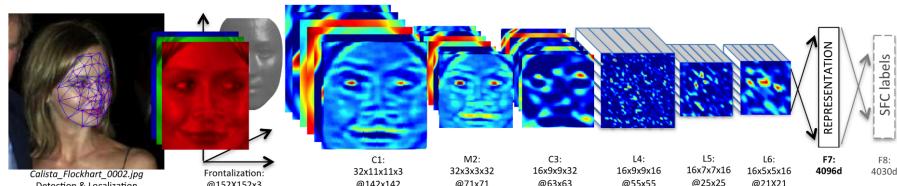
Por eso en la práctica, muchas implementaciones eliminan la capa de clasificación final y utilizan el modelo como un extractor de características faciales de propósito general.

- **FaceNet** (2015): Desarrollado por investigadores de Google, FaceNet introdujo el concepto de "triplet loss" para mejorar la calidad de las representaciones faciales aprendidas. Este modelo no solo identifica o verifica personas; también genera un embedding (vector de características) de cada rostro, lo que facilita la comparación de similitudes entre rostros distintos. El sistema es entrenado utilizando una CNN, y una función de pérdida, denominada por los autores pérdida de la tripleta (del inglés triplet loss), que es el eje principal de la propuesta de FaceNet. Esta función ajusta sus parámetros en función de tripletas con dos imágenes de la misma persona (una imagen ancla y un ejemplo positivo), y una de un sujeto diferente (un ejemplo negativo), como puede observarse en la figura. Una vez entrenado el sistema, se tiene una función de similitud que puede ser utilizada para verificar e identificar rostros.



Triplet loss estima el error (distancia) entre el vector de características de la imagen de anclaje y la imagen positiva, el error entre el vector de características de la imagen positiva y la imagen negativa.

- **DeepFace** (2014): Desarrollado por Facebook, DeepFace implementa una arquitectura CNN profunda que realiza una alineación 3D de los rostros antes de la clasificación, mejorando significativamente la precisión al lidiar con variaciones en la pose y la orientación del rostro. Este modelo logró una precisión cercana al desempeño humano en ciertos conjuntos de datos.



Esquema de la arquitectura de DeepFace. Imagen extraída de Taigman et al. (Taigman y cols., 2014).

La representación del rostro se obtiene mediante el uso de una red neuronal de nueve capas, cuya arquitectura puede observarse en la figura. Esta red incluye las etapas de frontalización, extracción de características y aplicación de filtros. Finalmente, se obtiene un vector de características que puede ser utilizado como representación del rostro y la distribución sobre las clases, asociada a la probabilidad de que el rostro pertenezca a cada una de dichas clases.

- **ArcFace** (2018): Considerado uno de los modelos más eficientes para la tarea de reconocimiento facial, ArcFace introduce una métrica de distancia angular en el espacio de características, lo que permite obtener embeddings más discriminativos para la identificación facial. Su capacidad para generar representaciones precisas y distinguibles de cada rostro lo convierte en un modelo de referencia en el estado del arte.

Ejemplos prácticos

Detección de rostros

En este ejemplo, utilizaremos OpenCV, para detectar rostros, ojos y sonrisas en una imagen. La detección de características faciales es un componente fundamental en muchas aplicaciones, desde la seguridad hasta las interfaces de usuario interactivas. Gracias a los clasificadores pre-entrenados de Haar Cascades que proporciona OpenCV, podemos lograr estas detecciones de manera eficiente y con relativamente poco código.

Primero, cargaremos los clasificadores pre-entrenados para rostros, ojos y sonrisas desde los recursos de OpenCV. Estos clasificadores han sido entrenados en miles de imágenes y son capaces de detectar las características mencionadas en imágenes nuevas. Despues, leeremos una imagen de prueba y la convertiremos a escala de grises. La conversión a escala de grises es un paso común en el procesamiento de imágenes para detección de características porque reduce la complejidad computacional al trabajar con un solo canal de color en lugar de tres.

Utilizando el clasificador de rostros, detectaremos rostros en la imagen. Para cada rostro detectado, dibujaremos un rectángulo alrededor de él y luego usaremos las regiones de interés (ROI) dentro de estos rectángulos para buscar ojos y sonrisas. Por cada ojo y sonrisa detectados, dibujaremos rectángulos alrededor de ellos en colores distintos para diferenciar las características faciales.

Finalmente, mostraremos la imagen resultante con los rectángulos dibujados alrededor de los rostros, ojos y sonrisas detectados. Este ejemplo ilustra cómo se pueden combinar varios clasificadores de Haar Cascades para realizar detecciones de características faciales complejas en imágenes utilizando OpenCV.

```
import cv2

# Cargar los clasificadores preentrenados
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
smile_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_smile.xml')

# Cargar la imagen y convertirla a escala de grises
image = cv2.imread('face_test.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

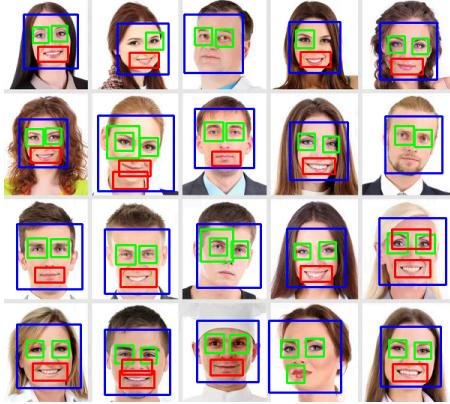
# Detectar rostros en la imagen
faces = face_cascade.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:
    cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = image[y:y+h, x:x+w]

    # Detectar ojos dentro del área del rostro
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

    # Detectar sonrisa dentro del área del rostro
    mouth = smile_cascade.detectMultiScale(roi_gray)
    for (mx,my,mw,mh) in mouth:
        cv2.rectangle(roi_color,(mx,my),(mx+mw,my+mh),(0,0,255),2)

# Mostrar la imagen resultante
cv2_imshow(image)
```



Reconocimiento de rostros

La identificación y verificación facial se han convertido en tareas fundamentales en una amplia gama de aplicaciones, desde sistemas de seguridad hasta soluciones de autenticación y más allá. En este contexto, las tecnologías de reconocimiento facial han evolucionado rápidamente, ofreciendo un rendimiento impresionante que puede ser aprovechado a través de diversas herramientas y librerías. Una de esas librerías es DeepFace, un framework de Python para el reconocimiento facial profundamente robusto, que simplifica el proceso de detectar, analizar y comparar rostros en imágenes.

DeepFace soporta varios modelos de aprendizaje profundo, cada uno de los cuales ha sido entrenado en vastos conjuntos de datos para aprender a distinguir entre características faciales únicas. Estos modelos incluyen [VGG-Face](#), [Facenet](#), [Facenet512](#), [OpenFace](#), [DeepFace](#), [DeepID](#), [ArcFace](#), [Dlib](#) y [SFace](#), permitiendo a los usuarios elegir el más adecuado para necesidades específicas.

El siguiente ejercicio demuestra cómo utilizar DeepFace para comparar dos imágenes faciales y determinar si pertenecen a la misma persona. Se selecciona el modelo ArcFace, conocido por su precisión y eficiencia en la codificación de características faciales en un espacio compacto que facilita la comparación. Este proceso implica la verificación de dos imágenes: una de Barack Obama y otra de Joe Biden, para ilustrar cómo la librería [deepface](#) puede discernir entre dos individuos diferentes.

```
from deepface import DeepFace

# Modelos soportados
models = [
    "VGG-Face",
    "Facenet",
    "Facenet512",
    "OpenFace",
    "DeepFace",
    "DeepID",
    "ArcFace",
    "Dlib",
    "SFace",
]

# Métodos disponibles para el cálculo de distancia
metrics = ["cosine", "euclidean", "euclidean_l2"]

# Ruta de las imágenes que deseas comparar
image1 = "face_db/obama/obama.jpg"
image2 = "face_db/biden/biden.jpg"

# Verificar si las imágenes pertenecen a la misma persona
result = DeepFace.verify(img1_path = image1, img2_path = image2, model_name=models[6], distance_metric = metric)
```

```
print("¿Son la misma persona?: ", result["verified"])
```

La capacidad de identificar con precisión a las personas en las imágenes, a través del reconocimiento facial, abre un abanico de posibilidades en campos como la seguridad, la vigilancia, el marketing, y más. En este contexto, la función `find()` de DeepFace emerge como una solución potente y flexible, diseñada para facilitar la identificación facial en conjuntos de datos amplios o bases de datos de rostros predefinidas. El siguiente ejercicio ilustra cómo utilizar la función `find()` de DeepFace para buscar rostros en una imagen específica dentro de una base de datos predefinida. Esta función no solo detecta rostros, sino que también los compara con los existentes en la base de datos para encontrar coincidencias posibles, brindando una herramienta poderosa para la identificación de individuos. Además, el ejercicio integra PIL (Python Imaging Library) para visualizar los resultados, dibujando rectángulos alrededor de los rostros identificados y etiquetando cada uno con el nombre correspondiente, directamente sobre la imagen original:

```
from deepface import DeepFace
from PIL import Image, ImageDraw, ImageFont
from IPython.display import display

# Definir la base de datos de rostros y la imagen de entrada
db_path = "face_db"
img_path = "g8_italy_onpage.jpg"

# Cargar los modelos disponibles y elegir uno específico
models = ["VGG-Face", "Facenet", "OpenFace", "DeepFace", "DeepID", "ArcFace", "Dlib"]
model_name = models[5]

# Métodos disponibles para el cálculo de distancia
metrics = ["cosine", "euclidean", "euclidean_l2"]

# Realizar la búsqueda del rostro en la base de datos
# threshold nos permite regular la sensibilidad. Su valor depende del modelo.
# Podemos dejar threshold sin definición, para usar el umbral por defecto
results_list = DeepFace.find(img_path=img_path, db_path=db_path, model_name=model_name,
                             distance_metric = metrics[2], threshold=None)

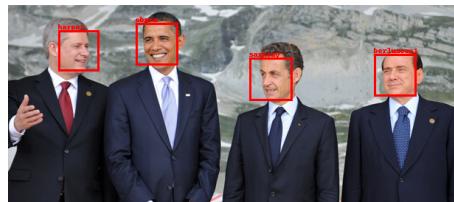
# Cargar la imagen original
imagen = Image.open(img_path)
dibujar = ImageDraw.Draw(imagen)
fuente = ImageFont.load_default()

# Verificar si se encontraron resultados
if results_list:
    # Iterar sobre cada DataFrame en la lista de resultados
    for results in results_list:
        if not results.empty:
            # Iterar sobre cada resultado encontrado
            for index, row in results.iterrows():
                # Extraer el nombre desde la ruta de la identidad encontrada, asumiendo estructura de directorios específica
                nombre = row['identity'].split('/')[-2]

                dibujar.rectangle(
                    [row['source_x'], row['source_y'],
                     row['source_x'] + row['source_w'], row['source_y'] + row['source_h']],
                     outline="red", width=3)
                # Dibujar la etiqueta con el nombre justo arriba del bounding box
                dibujar.text((row['source_x'], row['source_y'] - 10), nombre, fill="red", font=fuente)
else:
    print("No se encontraron coincidencias.")
```

```
# Mostrar la imagen modificada  
display(imagen)
```

El resultado obtenido será:



Aplicaciones

Los modelos de detección de rostros y los modelos de reconocimiento de rostros son tecnologías avanzadas que juegan roles cruciales en numerosas aplicaciones en el mundo digital y físico de hoy. Aunque están relacionados, sirven a propósitos distintos y, cuando se combinan, pueden crear sistemas complejos e inteligentes capaces de identificar y reconocer individuos con alta precisión. A continuación, mencionamos las aplicaciones de ambos tipos de modelos:

Modelos de Detección de Rostros

Los modelos de detección de rostros están diseñados para identificar y localizar rostros humanos dentro de imágenes o secuencias de video. No identifican a quién pertenece cada rostro; su tarea principal es señalar la presencia y posición de los rostros en el material visual. Algunas aplicaciones de estos modelos incluyen:

- **Seguridad y Vigilancia:** Mejorar los sistemas de seguridad y vigilancia al detectar la presencia de personas en áreas restringidas o monitorizar multitudes en tiempo real.
- **Mejora de Experiencia de Usuario:** Integrar en aplicaciones y dispositivos para atención al cliente, como kioscos interactivos y asistentes virtuales, permitiendo una interacción más natural y basada en la presencia del usuario.

Modelos de Reconocimiento de Rostros

Los modelos de reconocimiento de rostros van un paso más allá de la simple detección, identificando a quién pertenecen los rostros detectados al compararlos con una base de datos de rostros conocidos. Estos modelos encuentran aplicaciones en:

- **Autenticación Personal Segura:**
 - Desbloqueo de dispositivos personales (smartphones, laptops).
 - Acceso seguro a aplicaciones (bancarias, etc.) y cuentas en línea.
- **Seguridad y Aplicación de la Ley:**
 - Identificación de personas de interés (sospechosos, desaparecidos) por parte de fuerzas policiales y agencias de seguridad, utilizando bases de datos específicas.
 - Control fronterizo y verificación de identidad en aeropuertos para agilizar procesos y seguridad.
- **Control de Acceso Físico:**
 - Verificación de identidad para permitir o denegar el ingreso a instalaciones seguras (oficinas, edificios gubernamentales, áreas restringidas).

8. Facial Expression Recognition

Introducción y conceptos teóricos

En las últimas décadas, la interacción humano-computadora (HCI, por sus siglas en inglés) ha traspasado los límites tradicionales, buscando no solo comprender las instrucciones explícitas de los usuarios, sino también interpretar sus estados emocionales subyacentes. El Reconocimiento de Emociones Faciales (FER) emerge como un campo de

estudio multidisciplinario que se sitúa en la intersección de la psicología, la neurociencia, la inteligencia artificial (IA) y la visión por computadora. Esta tecnología se dedica a analizar expresiones faciales, tanto en imágenes estáticas como en videos, para inferir emociones humanas, lo que abre un amplio espectro de aplicaciones desde la seguridad hasta la salud mental, la educación, y más allá.

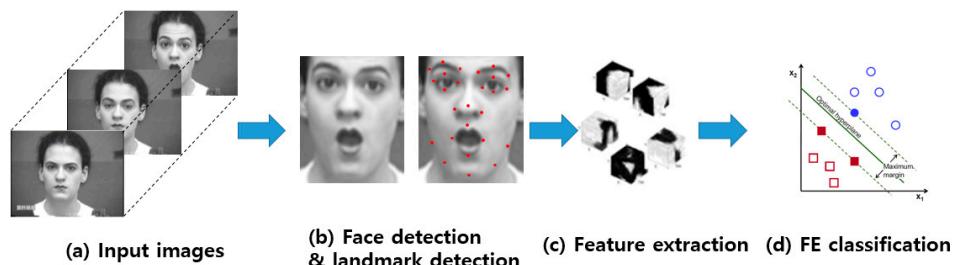
La importancia de FER radica en su capacidad para proporcionar una interfaz más natural e intuitiva entre humanos y máquinas, facilitando así una comunicación más rica y efectiva. La habilidad para reconocer y responder a las emociones humanas puede mejorar significativamente la experiencia del usuario en numerosos sistemas y servicios, personalizando las interacciones y anticipando necesidades o preocupaciones no expresadas explícitamente.

Sin embargo, la implementación de FER no está exenta de desafíos. La variabilidad individual en la expresión de emociones, las diferencias culturales, las condiciones ambientales durante la captura de imágenes, y las limitaciones inherentes a los algoritmos de aprendizaje automático son solo algunos de los obstáculos que esta tecnología debe superar para alcanzar una precisión robusta. Además, las implicaciones éticas y de privacidad relacionadas con la recopilación y análisis de datos biométricos sensibles requieren una consideración cuidadosa y marcos regulatorios adecuados.

Métodos de reconocimiento de expresiones

Las emociones faciales son factores importantes en la comunicación humana que nos ayudan a entender las intenciones de los demás. En general, las personas infieren los estados emocionales de otras personas, como la alegría, la tristeza y la ira, mediante expresiones faciales y tono vocal. Según [diferentes estudios](#), los componentes verbales transmiten un tercio de la comunicación humana, y los componentes no verbales transmiten dos tercios. Entre varios componentes no verbales, al llevar un significado emocional, las expresiones faciales son uno de los principales canales de información en la comunicación interpersonal. Por lo tanto, es natural que la investigación de la emoción facial haya estado atrayendo mucha atención durante las últimas décadas con aplicaciones no solo en las ciencias perceptuales y cognitivas, sino también en la informática afectiva y las animaciones por computadora.

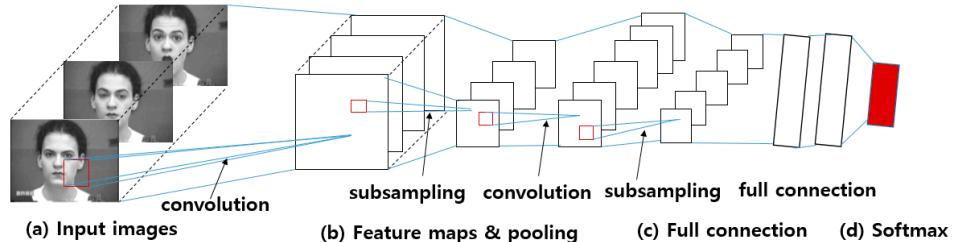
El FER se puede clasificar en dos grupos, según si las características son hechas a mano o generadas a través de la salida de una red neuronal profunda. En los enfoques convencionales, el FER está compuesto por tres pasos principales, como se muestra en la Figura: (1) detección de la cara y componentes faciales, (2) extracción de características y (3) clasificación de expresión. Primero, se detecta una imagen facial a partir de una imagen de entrada, y los componentes faciales (por ejemplo, ojos y nariz) o puntos de referencia son detectados desde la región facial. Segundo, se extraen diversas características espaciales y temporales de los componentes faciales. Tercero, los clasificadores de expresiones faciales pre-entrenados, como una máquina de vectores de soporte (SVM), AdaBoost y Random Forest, producen los resultados de reconocimiento utilizando las características extraídas:



Procedimiento utilizado en enfoques convencionales de FER: A partir de imágenes de entrada (a), se detecta la región de la cara y los puntos de referencia faciales (b), se extraen características espaciales y temporales de los componentes de la cara y los puntos de referencia (c), y la expresión facial se determina basándose en una de las categorías faciales utilizando modelos pre-entrenados.

En contraste con los enfoques tradicionales que utilizan características hechas a mano, el aprendizaje profundo ha surgido como un enfoque general para el aprendizaje automático, obteniendo resultados del estado del arte en muchos estudios de visión por computadora con la disponibilidad de grandes datos.

Los enfoques de FER basados en aprendizaje profundo reducen considerablemente la dependencia de modelos basados en la física facial y otras técnicas de preprocesamiento al permitir que el aprendizaje "de extremo a extremo" (end-to-end) ocurra directamente desde las imágenes de entrada. En los enfoques basados en CNN, la imagen de entrada es convolucionada a través de una colección de filtros para producir un mapa de características. Cada mapa de características se combina luego con redes completamente conectadas, y la expresión facial es reconocida como perteneciente a una clase particular basada en la salida del algoritmo softmax. La figura muestra el procedimiento utilizado por los enfoques de FER basados en CNN:

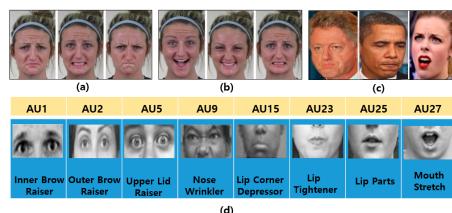


El FER también puede dividirse en dos grupos según si utiliza **imágenes fijas o imágenes de video**. Primero, el FER estático (basado en cuadros) depende únicamente de características faciales estáticas obtenidas al extraer características hechas a mano de cuadros de expresión máxima seleccionados de secuencias de imágenes. Segundo, el FER dinámico (basado en video) utiliza características espacio-temporales para capturar la dinámica de expresión en secuencias de expresión facial. Aunque se sabe que el FER dinámico tiene una tasa de reconocimiento más alta que el FER estático porque proporciona información temporal adicional, sufre de algunos inconvenientes. Por ejemplo, las características dinámicas extraídas tienen diferentes duraciones de transición dependiendo de las características de la expresión facial de las caras particulares. Además, la normalización temporal para obtener secuencias de expresión con un número fijo de cuadros puede resultar en una pérdida de información de la escala temporal.

Terminología

FACS: El Facial Action Coding System (FACS) es un sistema basado en los cambios de los músculos faciales y puede caracterizar las acciones faciales para expresar emociones humanas individuales, tal como fue definido por Ekman y Friesen en 1978. El FACS codifica los movimientos de músculos faciales específicos llamados unidades de acción (AUs), que reflejan cambios momentáneos distintos en la apariencia facial.

AUs: Los Facial Action Units (AUs) son las unidades fundamentales de acción de los músculos faciales individuales o grupos de músculos que se observan típicamente al producir las expresiones faciales asociadas con una emoción particular. Estas unidades son la base del Facial Action Coding System (FACS).



Ejemplos de muestra de varias emociones faciales y AUs: (a) emociones básicas (triste, temeroso y enfadado), (b) emociones compuestas (sorprendido felicemente, asqueado felizmente, y temeroso tristemente), (c) expresiones espontáneas, y (d) AUs (parte superior e inferior de la cara).

La siguiente tabla muestra las unidades de acción facial (AUs) prototípicas observadas en cada categoría de emoción básica y compuesta.

Categoría	AUs	Categoría	AUs
Feliz	12, 25	Tristemente asqueado	4, 10
Triste	4, 15	Miedosamente enojado	4, 20, 25
Temeroso	1, 4, 20, 25	Sorprendido con miedo	1, 2, 5, 20, 25
Enojado	4, 7, 24	Asqueado con miedo	1, 4, 10, 20, 25
Sorprendido	1, 2, 25, 26	Enojado sorprendido	4, 25, 26
Asqueado	9, 10, 17	Asqueado sorprendido	1, 2, 5, 10
Tristemente feliz	4, 6, 12, 25	Felizmente temeroso	1, 2, 12, 25, 26
Sorprendido felizmente	1, 2, 12, 25	Enojado asqueado	4, 10, 17
Asqueado felizmente	10, 12, 25	Sobrecogido	1, 2, 5, 25
Temeroso tristemente	1, 4, 15, 25	Consternado	4, 9, 10
Enojado tristemente	4, 7, 15	Odio	4, 7, 10
Sorprendido tristemente	1, 4, 25, 26	-	-

Recursos adicionales:

<https://paperswithcode.com/task/facial-expression-recognition>

<https://github.com/kdhht2334/awesome-SOTA-FER>

Ejemplos prácticos

En el siguiente ejemplo proporcionado, se utiliza la librería `FER` (<https://github.com/JustinShenk/fer>) para detectar emociones en una imagen. Esta librería aprovecha modelos de aprendizaje automático pre-entrenados para identificar emociones humanas a partir de expresiones faciales, simplificando enormemente el proceso de reconocimiento. El código demuestra cómo cargar una imagen, inicializar el detector de emociones, y procesar la imagen para detectar emociones y dibujar bounding boxes alrededor de los rostros detectados. Además, se ordenan las emociones detectadas por su puntuación y se muestran las dos principales directamente en la imagen, ofreciendo una interfaz intuitiva para visualizar tanto la localización de los rostros como las emociones interpretadas por la IA.

```
from fer import FER
import cv2
from google.colab.patches import cv2_imshow

# Carga la imagen
img = cv2.imread("emotions.jpg")

# Inicializa el detector
detector = FER()

# Detecta emociones en la imagen
resultados = detector.detect_emotions(img)

# Dibuja los bounding boxes y las dos emociones principales detectadas en la imagen
for resultado in resultados:
    # Obtiene las coordenadas del bounding box
    (x, y, w, h) = resultado["box"]

    # Dibuja un rectángulo alrededor de la cara
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

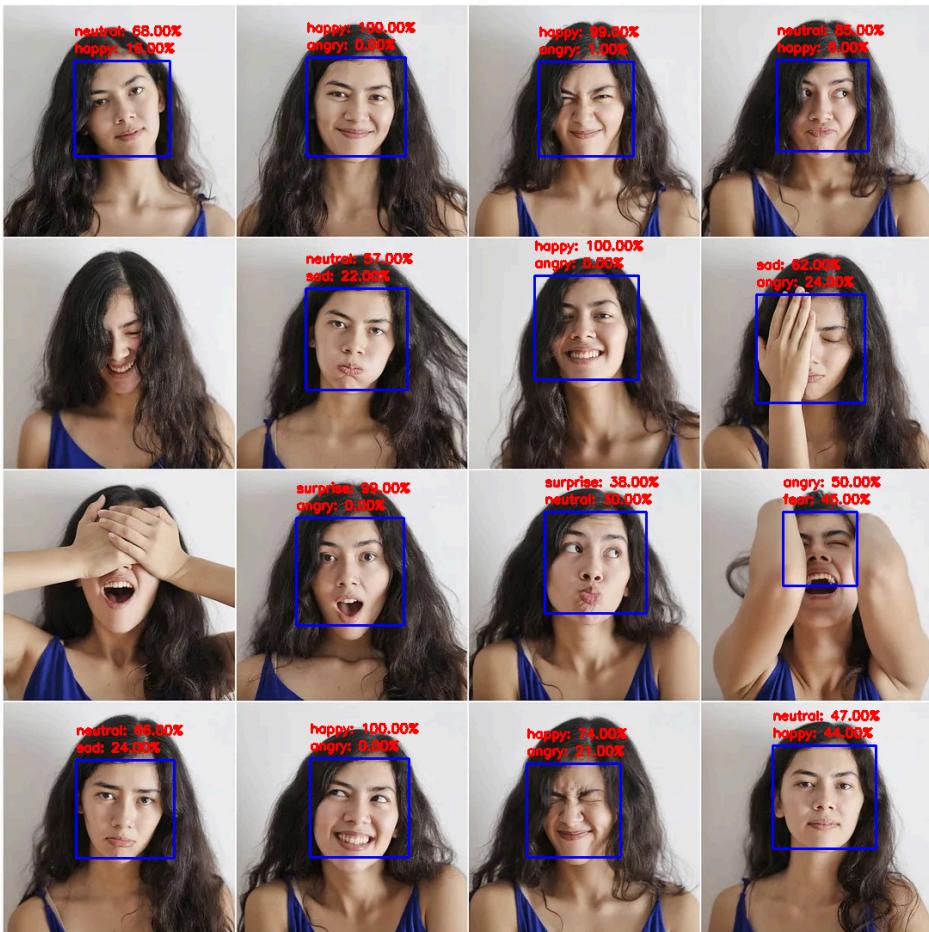
    # Obtiene las emociones y las ordena por score de manera descendente
    emociones = resultado["emotions"]
    emociones_ordenadas = sorted(emociones.items(), key=lambda x: x[1], reverse=True)

    # Prepara el texto con las dos emociones principales y sus scores
    texto1 = "{}: {:.2f}%".format(emociones_ordenadas[0][0], emociones_ordenadas[0][1] * 100)
    texto2 = "{}: {:.2f}%".format(emociones_ordenadas[1][0], emociones_ordenadas[1][1] * 100)

    # Dibuja el texto de las emociones sobre el bounding box
    cv2.putText(img, texto1, (x, y - 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
    cv2.putText(img, texto2, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

# Muestra la imagen modificada
cv2_imshow(img)
```

El resultado de la ejecución, arroja la siguiente imagen:



Este ejemplo práctico demuestra cómo utilizar `py-feat` (<https://github.com/cosanlab/py-feat>) para identificar no solo rostros dentro de una imagen, sino también para analizar puntos de referencia específicos, unidades de acción (microexpresiones), emociones, y la orientación del rostro, proporcionando así un análisis facial exhaustivo. El proceso inicia con la importación de la clase `Detector` de `py-feat` y otras herramientas esenciales para el manejo de imágenes, como `cv2` para la lectura y manipulación de la imagen, y `cv2_imshow` de Google Colab para su visualización dentro de notebooks. Se configura un objeto `Detector` especificando modelos avanzados para la detección de rostros, puntos de referencia, unidades de acción, emociones, y orientación del rostro. Posteriormente, se carga una imagen para su análisis y se utiliza el detector para extraer y procesar las características faciales detectadas. Este enfoque detallado no solo subraya la capacidad de `py-feat` para integrar diversos modelos en una sola instancia de detección, sino también su flexibilidad y potencia para aplicaciones de procesamiento de imágenes faciales en tiempo real o estudios de comportamiento humano a través de expresiones faciales:

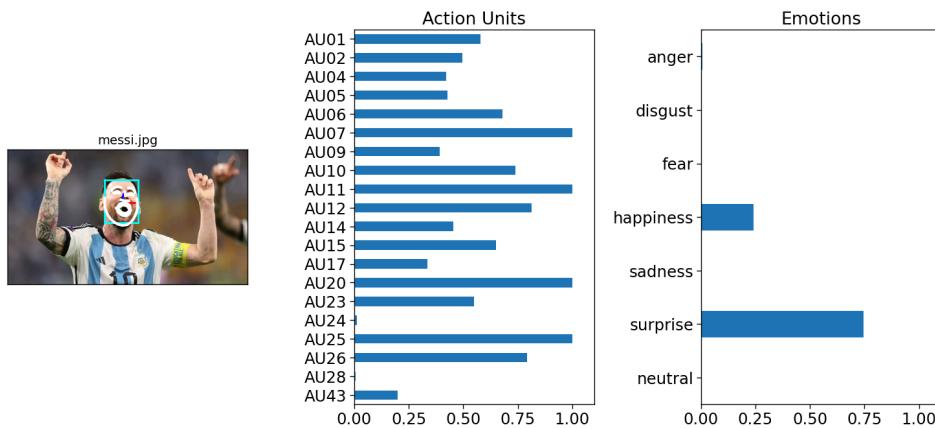
```
# Importa la clase Detector de la librería py-feat, utilizada para detectar características faciales en imágenes.
from feat import Detector

# Importa la función imshow del módulo plotting de py-feat, que permite mostrar imágenes.
from feat.plotting import imshow
from google.colab.patches import cv2_imshow
import cv2

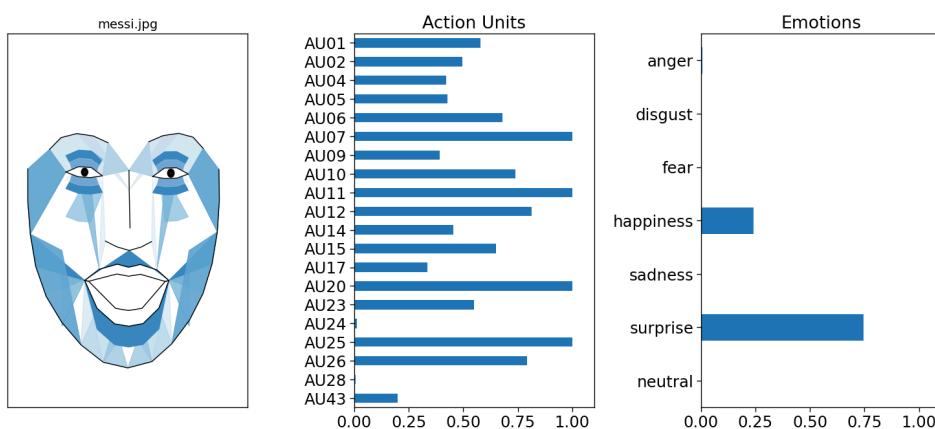
# Crea un objeto Detector con configuraciones específicas para los modelos de detección de rostro, puntos de referencia, unidades de acción (AU), emociones y pose del rostro.
detector = Detector(
    face_model="retinaface", # Modelo para la detección de rostros. RetinaFace es conocido por su precisión.
    landmark_model="mobilefacenet", # Modelo para detectar puntos de referencia en el rostro, como ojos y boca.
    au_model='xgb', # Modelo basado en XGBoost para identificar las unidades de acción, que son microexpresiones faciales.
    emotion_model="resmasknet", # Modelo para clasificar emociones basado en las expresiones faciales detectadas.
    facepose_model="img2pose", # Modelo para estimar la orientación del rostro en la imagen.
```

```
)
```

```
# Define la ruta de la imagen a analizar.  
image_path = "messi.jpg"  
  
# Utiliza el detector para analizar la imagen especificada y almacenar los resultados de la detección en single_face_prediction.  
# Esto incluye la detección de rostro, puntos de referencia, unidades de acción, emociones y la orientación del rostro.  
single_face_prediction = detector.detect_image(image_path)  
  
# Podemos usar el método .plot_detections() para generar una figura resumen de los rostros detectados,  
# unidades de acción y emociones. Siempre devuelve una lista de figuras de matplotlib.  
figs = single_face_prediction.plot_detections(poses=True)
```



```
# Por defecto, .plot_detections() superpondrá líneas faciales sobre la imagen de entrada.  
# Sin embargo, también es posible visualizar un rostro utilizando el modelo de puntos de  
# referencia de unidades de acción (AU) estandarizado de Py-Feat, el cual toma las AUs detectadas  
# y las proyecta sobre un rostro plantilla. Podemos controlar esto cambiando la configuración  
# a faces='aus' en lugar del valor predeterminado faces='landmarks'.  
figs = single_face_prediction.plot_detections(faces='aus', muscles=True)
```



Aplicaciones

El Reconocimiento de Expresiones Faciales (FER, por sus siglas en inglés) tiene una amplia gama de aplicaciones en diversos campos. Algunas de las principales aplicaciones son:

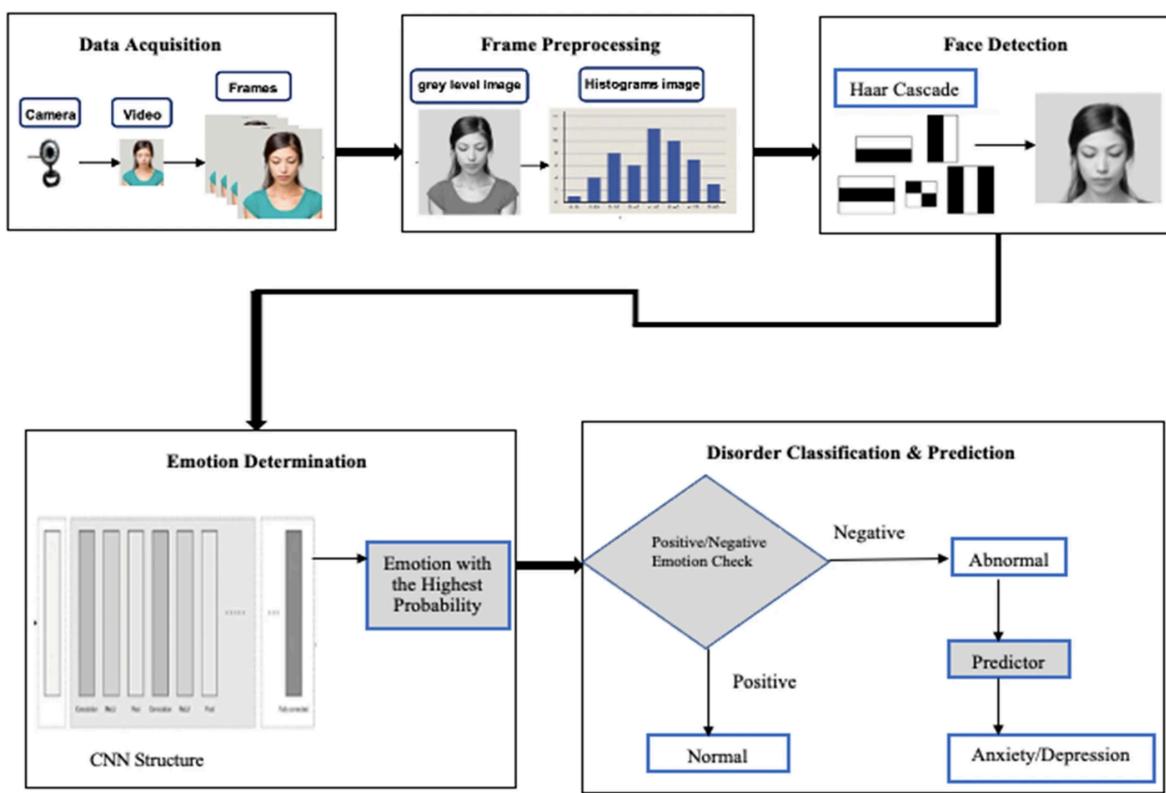
- 1. Interacción Humano-Computadora (HCI):** FER se utiliza para mejorar la comunicación entre los humanos y las computadoras o dispositivos inteligentes. Al detectar las expresiones faciales del usuario, los sistemas pueden adaptarse y responder de manera más natural y empática.

2. **Análisis de Mercado y Publicidad:** Las empresas pueden utilizar FER para evaluar las reacciones emocionales de los consumidores ante productos o campañas publicitarias, lo que les permite ajustar sus estrategias de marketing.
3. **Seguridad y Vigilancia:** FER se utiliza en sistemas de seguridad y vigilancia para detectar comportamientos sospechosos o amenazas potenciales basándose en las expresiones faciales.
4. **Conducción Asistida:** Los sistemas de asistencia al conductor pueden utilizar FER para monitorear el estado emocional del conductor y alertarlo si detectan signos de fatiga, distracción o enojo.



[Driver's Facial Expression Recognition in Real-Time for Safe Driving](#)

5. **Atención Médica:** FER se puede utilizar en el campo de la salud mental y la psicología para evaluar el estado emocional de los pacientes y ayudar en el diagnóstico y tratamiento de trastornos como la depresión o la ansiedad.



6. **Videojuegos y Realidad Virtual:** FER se implementa en videojuegos y aplicaciones de realidad virtual para mejorar la experiencia del usuario, adaptando el contenido y la interacción en función de las expresiones faciales del jugador.



[Recognizing Facial Expressions of Occluded Faces using Convolutional Neural Networks](#)

7. Análisis de Retroalimentación: FER se utiliza para analizar las expresiones faciales de los estudiantes o participantes en entornos educativos o de capacitación, para evaluar su nivel de comprensión y ajustar el material o la metodología de enseñanza.

9. Human Pose Estimation

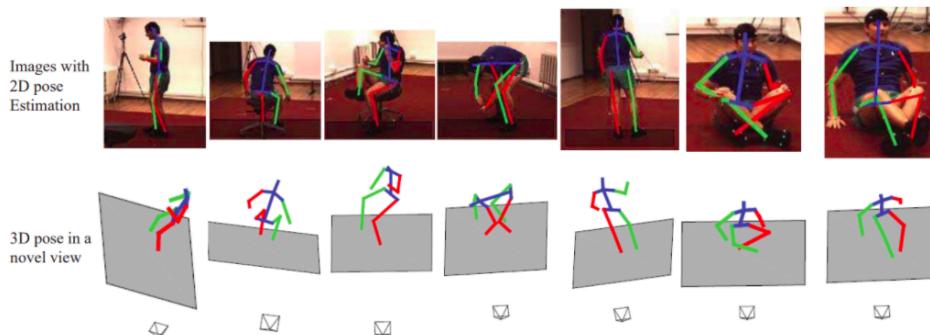
Introducción y conceptos teóricos

La estimación de pose humana (HPE) es una tarea fundamental en el ámbito de la visión por computadora, la cual describe la pose humana al localizar los puntos clave de las articulaciones en una imagen o video. A lo largo de los años, ha recibido una atención considerable en la investigación debido a su papel crucial en numerosas aplicaciones, como el análisis de comportamiento, la realidad virtual y las interacciones humano-computadora.

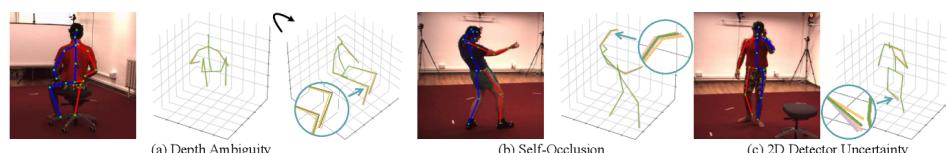
<https://www.youtube.com/watch?v=pW6nZXeWIG&t=4s>

[Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields](#)

La estimación de pose puede dividirse en estimaciones de pose bidimensionales (2D) y tridimensionales (3D), donde los métodos 2D se centran en localizar las coordenadas de los ejes X e Y de las articulaciones, mientras que los métodos 3D agregan un eje adicional para estimar la posición de las articulaciones en el espacio 3D, proporcionando información de profundidad. Este avance hacia la representación 3D ofrece un valor de investigación más alto y permite aplicaciones más amplias debido a la riqueza de información de profundidad proporcionada .



El éxito reciente en la estimación de pose 3D (3D HPE) ha sido impulsado por la tecnología de aprendizaje profundo y la disponibilidad de grandes conjuntos de datos. Sin embargo, el problema inherente de ambigüedad de profundidad aún limita seriamente la precisión de la 3D HPE. La ambigüedad de profundidad surge debido a la proyección de poses 3D en 2D, donde múltiples poses 3D pueden corresponder a la misma pose 2D. Este problema se produce debido a la proyección de poses 3D en 2D se conoce comúnmente como "**ambigüedad de punto de fuga**" (**foreshortening ambiguity**) o "**ambigüedad de perspectiva**" (**perspective ambiguity**). Por ejemplo, un brazo extendido puede parecer más corto en la imagen dependiendo de su orientación con respecto a la cámara.

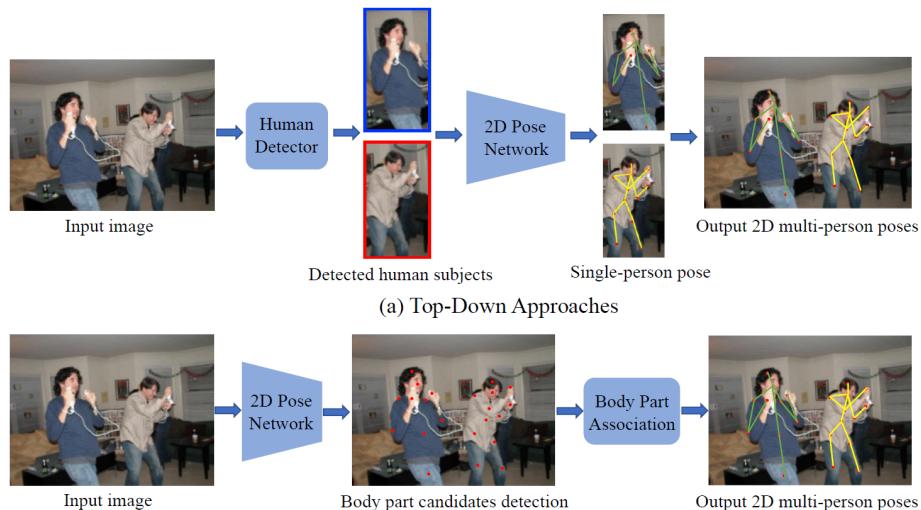


En el ámbito de la estimación de pose 2D, se han realizado esfuerzos considerables para desarrollar modelos basados en el aprendizaje profundo que pueden identificar y predecir la ubicación de los puntos clave del cuerpo humano en

imágenes o videos. Estos modelos se han vuelto cada vez más precisos gracias a arquitecturas de red neuronal profundas innovadoras, grandes conjuntos de datos para entrenamiento y validación, y métricas de evaluación avanzadas que facilitan la comparación justa entre diferentes enfoques. Además, se han identificado y explorado direcciones futuras para la investigación en HPE, como el desarrollo de métodos que pueden manejar eficazmente occlusiones, datos limitados para personas con discapacidades y la necesidad de modelos que puedan entrenarse en múltiples tareas para abordar la variabilidad en las poses y las apariencias del cuerpo humano.

Enfoques de los métodos

Los enfoques Top-Down y Bottom-Up son dos metodologías fundamentales utilizadas en la estimación de pose humana (HPE), especialmente en contextos donde se deben estimar las poses de múltiples personas en una misma imagen o video.



Cada uno tiene sus propias características y aplicaciones dependiendo del escenario y los objetivos específicos.

- **Top-Down Approach:** El enfoque Top-Down para la estimación de pose de múltiples personas se realiza en dos pasos principales:
 - Detección de Personas:** Primero, se identifican todas las personas en la imagen utilizando un detector de personas. Este detector genera cajas delimitadoras (bounding boxes) alrededor de cada persona en la imagen. Algunos ejemplos de detectores de personas incluyen Faster R-CNN, YOLO (You Only Look Once), y SSD (Single Shot MultiBox Detector).
 - Estimación de Pose por Persona:** Después de identificar las cajas delimitadoras, se aplica un modelo de estimación de pose para cada caja de manera individual, estimando la pose dentro de cada caja delimitadora como si fuera un problema de estimación de pose de una sola persona. Esto significa que el modelo de estimación de pose no necesita ser diseñado para manejar explícitamente múltiples personas; en cambio, el enfoque se centra en obtener una estimación de alta calidad para una persona a la vez.

Ventajas:

- Alta Precisión: Al centrarse en una persona a la vez, este enfoque puede utilizar modelos de estimación de pose detallados y complejos, lo que puede llevar a estimaciones de alta precisión.
- Flexibilidad: Es posible usar cualquier modelo avanzado de estimación de pose para la segunda etapa, lo que permite incorporar fácilmente mejoras y avances en la modelación de pose individual.
- **Bottom-Up Approach:** En contraste, el enfoque Bottom-Up para la estimación de pose de múltiples personas funciona identificando todos los componentes de la pose (como articulaciones o partes del cuerpo) en la imagen y luego agrupándolos en poses individuales.
 - Detección de Partes del Cuerpo:** Primero, se detectan todas las partes del cuerpo o puntos clave (como articulaciones) en toda la imagen, sin saber inicialmente a qué persona pertenece cada parte.
 - Agrupación de Partes en Poses:** Despues de detectar las partes del cuerpo, el siguiente paso es agrupar estas partes en conjuntos individuales que corresponden a poses completas de diferentes personas. Este paso

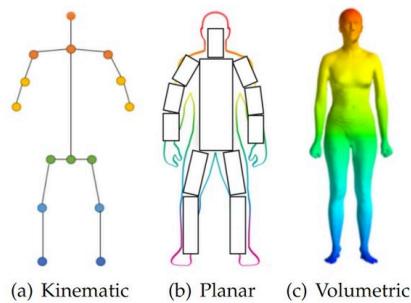
puede requerir algoritmos de agrupación o asociación que puedan manejar la ambigüedad y determinar cuáles partes pertenecen a la misma persona.

Ventajas:

- Eficiencia: Este enfoque puede ser más eficiente desde el punto de vista computacional para escenas con muchas personas porque las partes del cuerpo se detectan en una sola pasada por la imagen.
- Robustez ante Oclusiones: Puede manejar mejor las occlusiones parciales o la interacción entre personas, ya que la detección se realiza a nivel de partes del cuerpo.

Modelado de Cuerpo Humano

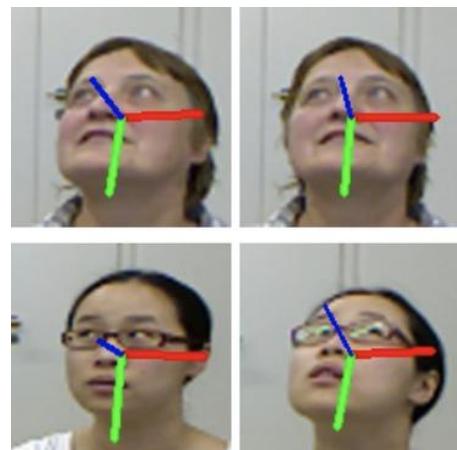
En la estimación de pose humana, la ubicación de las partes del cuerpo humano se utiliza para construir una representación del cuerpo humano (como una pose del esqueleto del cuerpo) a partir de datos de entrada visuales. Por lo tanto, el modelado del cuerpo humano es un aspecto importante de la estimación de pose humana. Se utiliza para representar características y puntos clave extraídos de los datos de entrada visuales. Típicamente, se utiliza un enfoque basado en modelos para describir e inferir poses del cuerpo humano y renderizar poses en 2D o 3D. La mayoría de los métodos utilizan un modelo cinemático rígido de N-articulaciones donde un cuerpo humano se representa como una entidad con articulaciones y extremidades, conteniendo información de la estructura cinemática del cuerpo y la forma del cuerpo. Generalmente se utilizan tres tipos de modelos para el modelado del cuerpo humano:



- **Modelo Cinemático (kinematic)**, también llamado modelo basado en esqueleto, se utiliza para la estimación de pose en 2D y 3D. Este modelo de cuerpo humano flexible e intuitivo incluye un conjunto de posiciones de articulaciones y orientaciones de extremidades para representar la estructura del cuerpo humano. Por lo tanto, los modelos de estimación de pose de esqueleto se utilizan para capturar las relaciones entre diferentes partes del cuerpo. Sin embargo, los modelos cinemáticos tienen limitaciones en la representación de información de textura o forma.
- **Modelo Plano**, o modelo basado en contornos, se utiliza para la estimación de pose en 2D. Los modelos planos se utilizan para representar la apariencia y forma de un cuerpo humano. Por lo general, las partes del cuerpo se representan mediante múltiples rectángulos que aproximan los contornos del cuerpo humano. Un ejemplo popular es el Modelo de Forma Activa (ASM, por sus siglas en inglés), que se utiliza para capturar el grafo completo del cuerpo humano y las deformaciones de silueta, usando análisis de componentes principales.
- **Modelo Volumétrico**, que se utiliza para la estimación de pose en 3D. Existen múltiples modelos de cuerpo humano en 3D populares que se utilizan para el análisis de pose basado en aprendizaje profundo para recuperar la malla humana en 3D. Por ejemplo, GHUM & GHUML, son pipelines de aprendizaje profundo completamente entrenables de principio a fin, entrenados en un conjunto de datos de alta resolución de escaneos de cuerpo completo de más de 60,000 configuraciones humanas para modelar formas y poses del cuerpo humano en 3D articuladas. Puede ser utilizado para inferencia.

Estimación de la Pose de la Cabeza

La estimación de la pose de la cabeza de una persona es un problema popular en visión por computadora. La estimación de la pose de la cabeza tiene múltiples aplicaciones, como ayudar en la estimación de la mirada, modelar la atención, ajustar modelos 3D a video y realizar alineación facial. Tradicionalmente, la pose de la cabeza se calcula con el uso de puntos clave del rostro objetivo y resolviendo el problema de correspondencia de pose 2D a 3D con un modelo de cabeza humana promedio. La capacidad de recuperar la pose 3D de la cabeza es un subproducto del análisis de expresión facial basado en puntos clave, que se basa en la extracción de puntos clave faciales 2D con métodos de aprendizaje profundo. Estos métodos son robustos a occlusiones y cambios extremos de pose.



<https://paperswithcode.com/task/head-pose-estimation>

Métodos de Detección de Pose basados en Aprendizaje Profundo

Debido a que la estimación de pose es una técnica de visión por computadora fácilmente aplicable, podemos implementar un estimador de pose personalizado utilizando arquitecturas existentes. Las arquitecturas existentes para comenzar a desarrollar un estimador de pose personalizado incluyen:

- **OpenPose** es uno de los enfoques bottom-up más populares para la estimación de pose en tiempo real y de múltiples personas. OpenPose es un marco de código abierto que es adecuado para lograr alta precisión en la detección de puntos clave corporales, de pies, manos y faciales. Una ventaja de OpenPose es que es una API que ofrece a los usuarios la flexibilidad de seleccionar imágenes fuente de campos de cámaras, webcams y otros, más importante aún para aplicaciones en sistemas embebidos (por ejemplo, integración con cámaras y sistemas CCTV). Admite diferentes arquitecturas de hardware, como GPUs CUDA, GPUs OpenCL o dispositivos solo con CPU. La versión ligera es lo suficientemente eficiente para aplicaciones de inferencia Edge con procesamiento en dispositivo en tiempo real con dispositivos edge.
- **High-Resolution Net (HRNet)** es una red neuronal para la estimación de pose humana. Es una arquitectura utilizada en problemas de procesamiento de imágenes para encontrar lo que conocemos como puntos clave (articulaciones) con respecto al objeto o persona específico en una imagen. Una ventaja de esta arquitectura sobre otras es que la mayoría de los métodos existentes igualan representaciones de alta resolución de posturas desde representaciones de baja resolución con respecto al uso de redes de alta-baja resolución. En lugar de este sesgo, la red neuronal mantiene representaciones de alta resolución al estimar posturas. Por ejemplo, esta arquitectura HRNet es útil para la detección de la postura humana en deportes televisados.
- **DeepCut** es un enfoque Bottom-Up popular para la estimación de pose humana de múltiples personas. En lugar de detectar primero el número de personas en una imagen, DeepCut comienza identificando todos los puntos clave del cuerpo (como articulaciones) en toda la imagen. Luego, utiliza un modelo matemático para agrupar estos puntos en poses individuales que corresponden a diferentes personas, sin necesidad de detectar previamente a cada persona mediante cajas delimitadoras.
- **Estimación de Pose Regional de Múltiples Personas (AlphaPose)** es un método top-down popular de estimación de pose. Es útil para detectar poses en presencia de cajas delimitadoras humanas inexactas. La arquitectura AlphaPose es aplicable para detectar poses tanto de una sola persona como de múltiples personas en campos de imágenes o videos.
- **DeepPose**: Este es un estimador de pose humana que aprovecha el uso de redes neuronales profundas. La red neuronal profunda (DNN) de DeepPose captura todas las articulaciones, incluye una capa de agrupamiento, una capa de convolución y una capa completamente conectada para formar parte de estas capas.
- **PoseNet**: PoseNet es una arquitectura de estimador de pose construida sobre tensorflow.js para ejecutarse en dispositivos ligeros como navegadores o dispositivos móviles. Por lo tanto, PoseNet se puede utilizar para estimar una pose única o múltiples poses.
- **DensePose**: Esta es una técnica de estimación de pose que mapea todos los píxeles humanos de una imagen RGB a la superficie 3D del cuerpo humano. DensePose también se puede utilizar para estimación de pose única y múltiple.

- **Estimación de Pose TensorFlow:** TensorFlow Lite proporciona estimación de pose con un modelo de ML ligero optimizado para dispositivos edge de baja potencia.
- **OpenPifPaf:** es una librería (<https://github.com/openpifpaf/openpifpaf>) y framework de visión por computadora de código abierto para la comprensión de pose, que implica identificar y localizar partes del cuerpo humano en imágenes o videos. Se construye sobre el marco de aprendizaje profundo PyTorch y utiliza un enfoque de aprendizaje multitarea para lograr una estimación de pose precisa y eficiente. OpenPifPaf ha ganado popularidad por su facilidad de uso, robustez y capacidad para manejar escenarios desafiantes de seguimiento de movimiento, como occlusiones y fondos desordenados.
- **Estimación de Pose y clasificación de puntos clave YOLO:** Los modelos de pose de YOLO usan el sufijo -pose (por ejemplo, yolov8n-pose.pt). Estos modelos optimizados en tiempo real están entrenados en el conjunto de datos de puntos clave COCO y son adecuados para una variedad de tareas de estimación de pose.

Más recursos:

Deep Learning-Based Human Pose Estimation: A Survey: <https://arxiv.org/abs/2012.13392>

Ejemplos prácticos

Este ejemplo demuestra cómo utilizar el modelo pre-entrenado YOLOv11n para la estimación de poses en imágenes, utilizando específicamente una fotografía de Lionel Messi. Después de cargar y predecir la pose con el modelo YOLO, el código procede a visualizar la imagen anotada directamente. Además, se extraen los puntos clave (keypoints) de las poses detectadas, y luego, mediante OpenCV, se grafican estos puntos sobre la imagen original, destacando así las áreas de interés identificadas por el modelo, como las articulaciones y partes del cuerpo. Este proceso subraya el potencial de la tecnología en aplicaciones de análisis de movimiento y deportes.

```
from ultralytics import YOLO
import matplotlib.pyplot as plt
import cv2
from google.colab.patches import cv2_imshow

# Leer la imagen local usando OpenCV
img = cv2.imread('messi_miami.jpg')

# Cargar el modelo preentrenado YOLOv8n para la estimación de pose
model = YOLO("yolo11n-pose.pt")
# Realizar la predicción en la imagen cargada
results = model(img, save=True)

# Extraer y mostrar la imagen anotada con las detecciones y estimaciones de pose
annotated_frame = results[0].plot()
cv2_imshow(annotated_frame)

# Extraer los keypoints de los resultados
result_keypoints = results[0].keypoints.xyn.cpu().numpy()

# Obtener las dimensiones de la imagen
w, h = img.shape[1], img.shape[0]

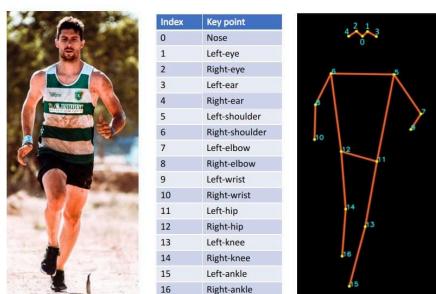
# Dibujar los keypoints en la imagen original
for keypoints in result_keypoints: # Iterar sobre cada conjunto de keypoints
    for j, keypoint in enumerate(keypoints): # Iterar sobre cada keypoint
        x, y = keypoint # Extraer las coordenadas x, y del keypoint
        # Dibujar un círculo en la posición del keypoint
        cv2.circle(img, (int(x * w), int(y * h)), 3, (0, 255, 0), -1)
        # Añadir el número del keypoint al lado del círculo
        cv2.putText(img, str(j), (int(x * w), int(y * h)), cv2.FONT_HERSHEY_SIMPLEX, 0.3, (255, 0, 0), 1)
```

```
# Mostrar la imagen original con los keypoints dibujados
cv2_imshow(img)
```

El resultado serán dos imágenes. En la primera se muestran las anotaciones que realiza la librería [ultralytics](#), en cambio en la segunda, iteramos sobre los resultados y dibujamos los *keypoints* detectados.

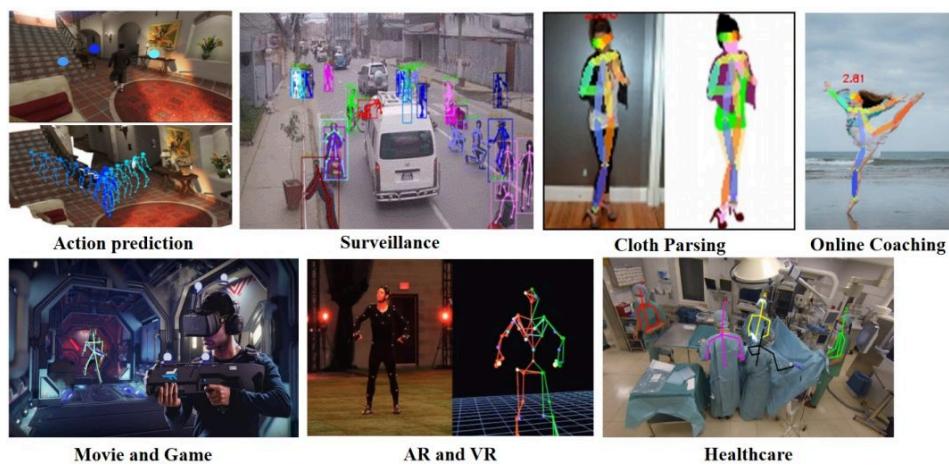


Los índices de cada *keypoint*, corresponden a las siguientes etiquetas:



Aplicaciones

La estimación de pose ha encontrado aplicaciones en una amplia gama de campos, gracias a su capacidad para entender y predecir la posición y orientación de las personas en el espacio.

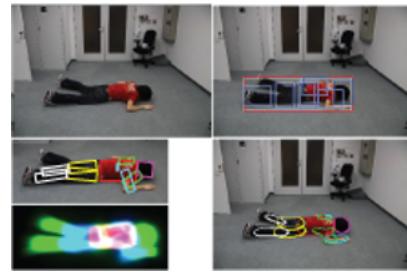


A continuación, se detallan algunas de las aplicaciones más populares de la estimación de pose:

- 1. Estimación de Actividad Humana:** La estimación de la actividad humana es fundamental en áreas como la vigilancia de seguridad, el análisis deportivo y la interacción humano-computadora. Al identificar las poses y movimientos específicos de las personas, los sistemas pueden reconocer actividades como correr, saltar, caer o realizar gestos específicos. Esta capacidad permite, por ejemplo, analizar el rendimiento deportivo para mejorar la técnica o detectar comportamientos sospechosos en entornos vigilados.
- 2. Transferencia de Movimiento y Realidad Aumentada:** La transferencia de movimiento implica capturar los movimientos de una persona y aplicarlos a un avatar digital o personaje de videojuego. Esto se utiliza no solo en la

industria del entretenimiento, para crear animaciones más realistas, sino también en realidad aumentada (AR) para proyecciones interactivas. Por ejemplo, en aplicaciones de prueba de ropa virtual, los movimientos del usuario se capturan y se reflejan en un avatar que lleva la ropa seleccionada, permitiendo al usuario ver cómo le quedaría esa prenda en diferentes poses y movimientos.

3. **Captura de Movimiento para Entrenamiento de Robots:** La captura de movimiento es esencial para enseñar a los robots cómo realizar tareas humanas, desde la manipulación de objetos hasta la navegación en entornos complejos. Al capturar y analizar los movimientos humanos, los robots pueden aprender a imitar estas acciones de manera efectiva. Esto se aplica en la automatización industrial, donde los robots aprenden a realizar tareas de ensamblaje, y en la asistencia personal, donde los robots ayudan en tareas domésticas.
4. **Seguimiento de Movimiento para Consolas:** En el mundo de los videojuegos, la estimación de pose permite una interacción más inmersiva y física con los juegos. Consolas como la Wii de Nintendo, PlayStation Move de Sony y Kinect de Xbox han utilizado tecnologías de seguimiento de movimiento para permitir que los jugadores utilicen sus propios movimientos corporales como controles de juego, desde deportes virtuales hasta juegos de baile y aventuras interactivas.
5. **Detección de Caídas Humanas:** La detección de caídas es crucial en el cuidado de personas mayores o individuos con condiciones que aumentan el riesgo de caídas. Utilizando cámaras y software de estimación de pose, los sistemas pueden detectar de manera confiable cuando una persona ha caído y alertar a cuidadores o servicios de emergencia. Esto mejora significativamente la capacidad de proporcionar una respuesta rápida, potencialmente salvando vidas y reduciendo las complicaciones asociadas con las caídas.



<https://www.graphics.rwth-aachen.de/publication/0042/>

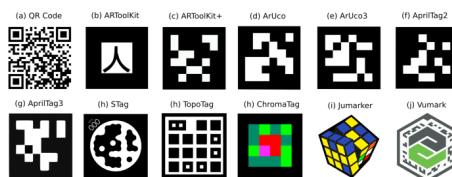
10. Marker detection

Introducción y conceptos teóricos

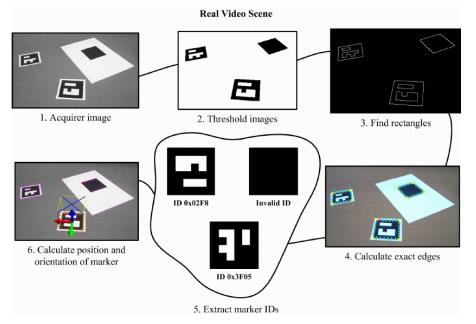
La detección de marcadores o fiduciales en el campo de la visión por computadora ha surgido como una solución robusta para problemas de estimación de pose en aplicaciones como realidad aumentada, interacción con dispositivos de mano y navegación robótica. Estos sistemas suelen consistir en marcadores, un algoritmo de detección y un sistema de codificación. Tradicionalmente, los algoritmos de detección han sido diseñados a mano, basándose en técnicas de procesamiento de imágenes de bajo nivel como la detección de bordes, detección de manchas y binarización de imágenes. Sin embargo, estas técnicas limitan la apariencia de los marcadores y su robustez de detección, llevando a la necesidad de un sistema de codificación completamente diseñado para superar estas limitaciones.

Los marcadores fiduciales ofrecen características confiables en cantidad y unicidad suficientes, siendo necesarios al menos cuatro puntos clave no colineales para calcular una pose única a partir de un solo marcador. Además, se requiere de un conjunto de diferentes marcadores para distinguir múltiples objetos.

Entre los sistemas de marcadores fiduciales más conocidos se encuentran ARToolkitPlus, AprilTag, ArUco, TopoTag y RuneTag. Algunos sistemas han empleado marcadores de tablero de ajedrez en blanco y negro, donde un límite cuadrilátero proporciona cuatro puntos de esquina para la estimación de la pose, y las celdas en blanco y negro dispuestas en una cuadrícula regular representan los símbolos digitales "0" o "1". Los sistemas de marcadores más recientes introducen puntos clave redundantes definidos con estructuras dentro del marcador, como TopoTag, que limita la disposición de bits en una disposición de tablero de ajedrez y proporciona regiones fijas, y RuneTag, que dispone puntos sólidos en círculos concéntricos.



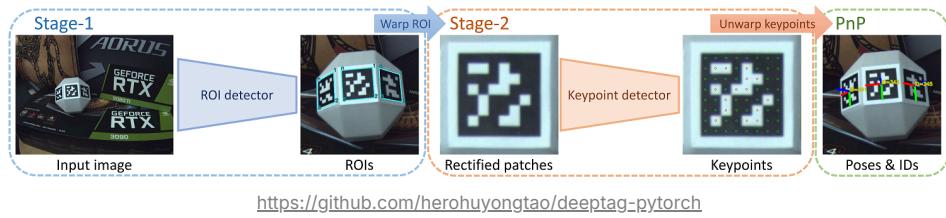
La disposición de los elementos en un marcador **debe ser asimétrica para evitar la ambigüedad de la pose**. Además, para reducir la posibilidad de confusión con elementos ambientales, se fomenta un alto número de transiciones de bits.



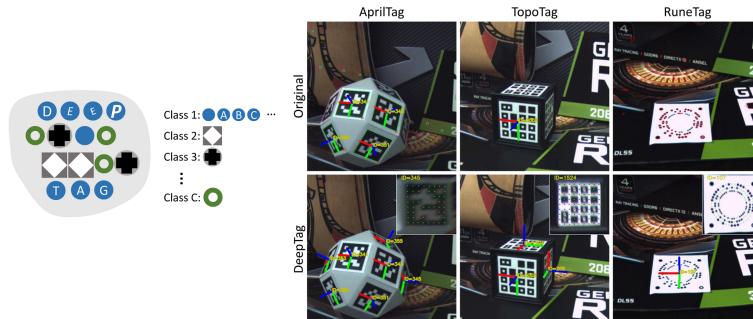
Pipeline típico para la detección de marcadores ([Fuente](#))

Deep Learning

La introducción de DeepTag, un marco basado en aprendizaje profundo para el diseño y detección de marcadores fiduciales, propone una mejora significativa en la flexibilidad y robustez para diversas aplicaciones.



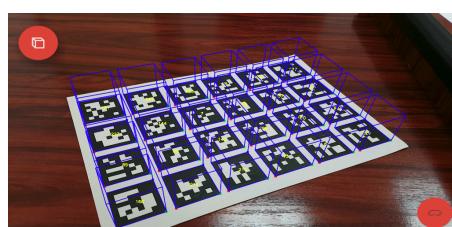
DeepTag soporta la detección de una amplia variedad de familias de marcadores existentes y facilita el diseño de nuevas familias de marcadores con patrones locales personalizados. Además, propone un procedimiento efectivo para sintetizar datos de entrenamiento sobre la marcha sin anotaciones manuales, permitiendo que DeepTag se adapte fácilmente a familias de marcadores existentes y recién diseñadas.



Ejemplos prácticos

AruCo

Una forma sencilla de probar el reconocimiento de marcadores Aruco, es utilizar la siguiente página:
<https://mmla.gse.harvard.edu/tools/js-aruco-marker-detector/>



AprilTags

AprilTags, una familia particular de marcadores fiduciales, ha ganado popularidad debido a su robustez, facilidad de detección y la precisión con la que permite la estimación de la pose de un objeto o cámara en el espacio 3D.

AprilTags son esencialmente patrones cuadrados que pueden ser fácilmente impresos y pegados en objetos o entornos para servir como puntos de referencia claros y detectables. Cada AprilTag está diseñado para ser único, lo que permite su identificación inequívoca dentro de un conjunto de tags, y posee propiedades que facilitan su detección incluso bajo condiciones subóptimas, como ángulos de visión extremos, iluminación variable y presencia de occlusiones parciales.

La detección eficiente de estos marcadores en imágenes digitales implica identificar su presencia y orientación en el espacio de la imagen, decodificar su identidad única y, si es necesario, estimar la posición y orientación (pose) del tag respecto a la cámara. Esta información puede ser utilizada para una variedad de aplicaciones, desde el posicionamiento preciso de objetos virtuales en un entorno de realidad aumentada hasta la navegación y mapeo en robots autónomos.

Para facilitar la detección de AprilTags en imágenes utilizando Python, podemos aprovechar librerías especializadas que implementan algoritmos optimizados para este propósito. Una de estas librerías es `apriltag`, específicamente diseñada para detectar AprilTags y ampliamente utilizada en la comunidad de visión por computadora por su eficacia y simplicidad de uso.

A continuación, se presenta un ejemplo práctico que demuestra cómo utilizar la biblioteca `apriltag` en un entorno de Google Colab para detectar AprilTags en una imagen estática:

```
# !pip install apriltag
# Importa las librerías necesarias
import cv2
import apriltag
import matplotlib.pyplot as plt

images = ['apriltag.jpg', 'apriltag_3.jpg']
for image_path in images:
    # Carga la imagen y convierte a escala de grises
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Crea una instancia del detector AprilTag
    detector = apriltag.Detector()

    # Realiza la detección de AprilTags en la imagen
    results = detector.detect(gray)

    # Dibuja los resultados de la detección
    for r in results:
        # Extrae las coordenadas de las esquinas del AprilTag detectado
        (ptA, ptB, ptC, ptD) = r.corners
        ptA = (int(ptA[0]), int(ptA[1]))
        ptB = (int(ptB[0]), int(ptB[1]))
        ptC = (int(ptC[0]), int(ptC[1]))
        ptD = (int(ptD[0]), int(ptD[1]))

        # Dibuja el contorno del AprilTag detectado en la imagen
        cv2.line(image, ptA, ptB, (0, 255, 0), 2)
        cv2.line(image, ptB, ptC, (0, 255, 0), 2)
        cv2.line(image, ptC, ptD, (0, 255, 0), 2)
        cv2.line(image, ptD, ptA, (0, 255, 0), 2)

        # Dibuja el ID del AprilTag en el centro del AprilTag detectado
        (cX, cY) = (int(r.center[0]), int(r.center[1]))
        cv2.putText(image, str(r.tag_id), (cX, cY), cv2.FONT_HERSHEY_SIMPLEX,
```

```
0.5, (0, 0, 255), 2)
```

```
# Muestra la imagen con los AprilTags detectados usando matplotlib
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.show()
```



Aplicaciones

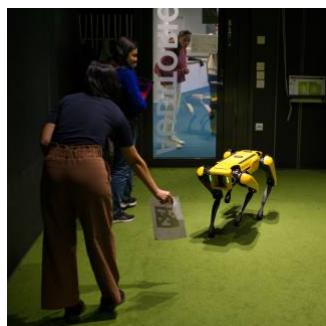
La detección de marcadores, especialmente en el contexto de la visión por computadora, ha encontrado aplicaciones en una amplia gama de campos debido a su capacidad para proporcionar referencias precisas y fiables en el mundo físico para sistemas digitales. Los marcadores fiduciales facilitan la interacción entre el mundo real y el digital, permitiendo una multitud de aplicaciones innovadoras y prácticas:

1. Realidad Aumentada (AR)

La detección de marcadores es fundamental en aplicaciones de realidad aumentada, donde objetos virtuales se superponen en el mundo real. Los marcadores actúan como puntos de anclaje para colocar con precisión estos objetos virtuales, permitiendo aplicaciones en educación, diseño, entretenimiento y comercio, como la visualización de productos en 3D en entornos reales.



2. Robótica



En robótica, los marcadores se utilizan para la navegación y localización de robots en un entorno, permitiéndoles identificar su posición y orientación con precisión. Esto es crucial para tareas de mapeo, localización y navegación autónoma en entornos estructurados o semiestructurados.

4. Control de Movimiento y Captura de Movimiento

La detección de marcadores se emplea en la captura de movimiento para registrar los movimientos de actores y objetos, utilizándose en la creación de efectos visuales para películas y videojuegos, así como en análisis biomecánicos y deportivos.



Figure 1 - Markers in the user's body

5. Manufactura y Automatización Industrial

En la industria, los marcadores facilitan la automatización de procesos al permitir que las máquinas reconozcan y manipulen objetos con precisión. Esto se aplica en el ensamblaje automatizado, la inspección de calidad y la logística.



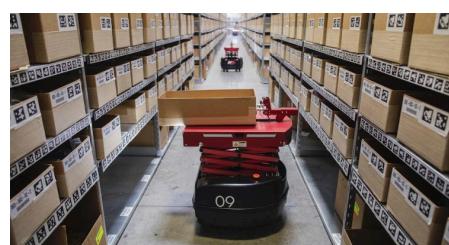
6. Educación y Capacitación



Los marcadores se utilizan para crear materiales educativos interactivos y herramientas de capacitación, permitiendo a los estudiantes interactuar con modelos virtuales 3D que mejoran la comprensión de conceptos complejos.

7. Logística automatizada

La detección de marcadores fiduciales en logística automatizada permite el rastreo preciso de inventario. Los marcadores en productos o contenedores facilitan su identificación y localización en almacenes. Robots y vehículos autónomos utilizan marcadores para navegar eficientemente en entornos de almacenamiento.



11. Color Quantization

Introducción y conceptos teóricos

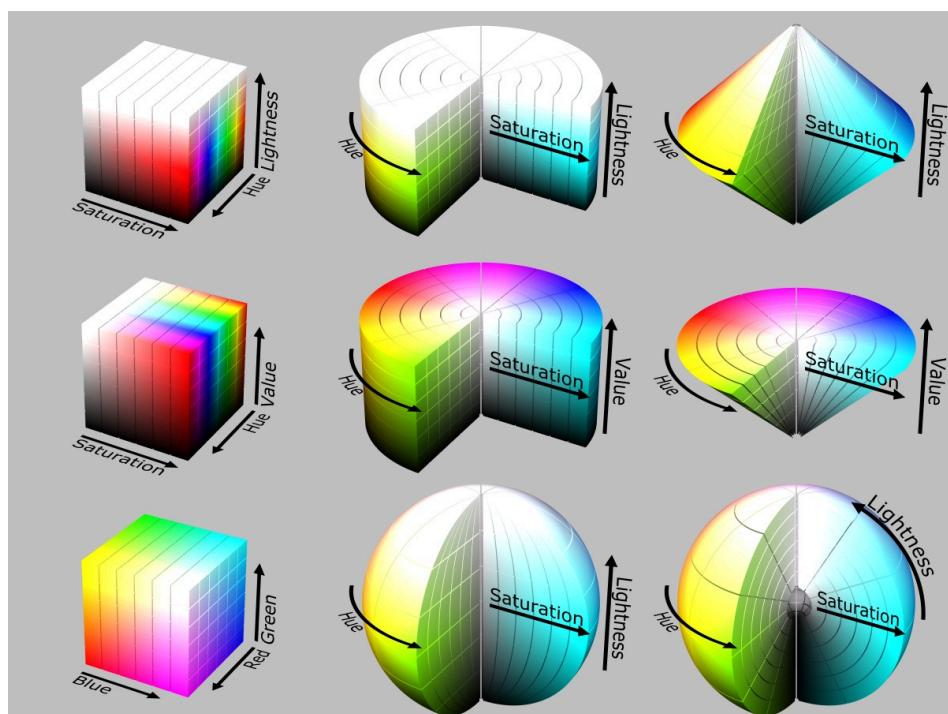
La cuantización de color es un proceso mediante el cual se reduce el número de colores distintos en una imagen con el fin de hacer que el almacenamiento y procesamiento de la imagen sean más eficientes. Este proceso es esencial en diversas aplicaciones, como la búsqueda de imágenes por similitud, la edición de colores, la generación de paletas de colores, la segmentación y el análisis de esquemas de colores de imágenes.

Antes de sumergirnos en la cuantización, debemos profundizar en el manejo de modelos de color.

Modelos de color

Un modelo de color es un modelo matemático abstracto que describe la forma en la que los colores pueden representarse como tuplas de números, normalmente como tres o cuatro valores o componentes de color (p.e. RGB y CMYK son modelos de color).

Los modelos de color usan formas geométricas como cilindros, conos y cubos porque estos facilitan la representación, comprensión y manipulación de colores en un espacio tridimensional. Estas formas ayudan a modelar las relaciones entre los diferentes componentes del color (como tono, saturación y brillo) de una manera que es tanto intuitivamente comprensible como matemáticamente práctica.



A continuación, se detallan las razones específicas por las que cada una de estas formas se usa en modelos de color específicos:

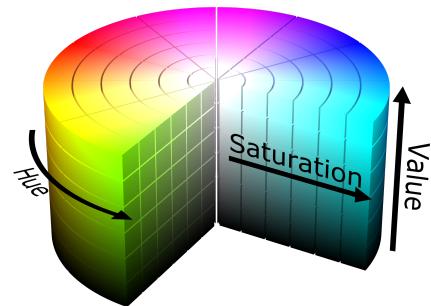
- Cilindros:** El modelo de color HSL (Hue, Saturation, Lightness) y HSV (Hue, Saturation, Value) usan la forma cilíndrica porque permite separar el componente de color (hue) de los componentes de blancura/negruzca (saturation) y de claridad/oscuridad (lightness o value). El ángulo alrededor del eje central representa el tono, la distancia al eje representa la saturación, y la altura a lo largo del eje representa la luminosidad o el valor. Esta separación facilita la manipulación de colores en términos de su perceptibilidad para el ojo humano.
- Conos:** El modelo de color Luv* y Lab* (CIELUV y CIELAB) utilizan formas geométricas que se asemejan a conos o doble conos. Estos modelos están diseñados para ser uniformemente perceptuales, lo que significa que la misma cantidad de cambio numérico en estos espacios de color corresponde aproximadamente al mismo grado de cambio en la percepción del color. La forma de cono ayuda a representar la variabilidad de la percepción del color en relación con la luminosidad.
- Cubos:** El modelo de color RGB (Red, Green, Blue) utiliza la forma de un cubo porque cada uno de los tres colores primarios se puede representar como una dimensión en un espacio tridimensional, con cada vértice del cubo representando uno de los colores primarios o una mezcla de ellos. Esto facilita la representación digital de colores.

en dispositivos electrónicos como monitores y cámaras digitales, donde cada color se puede ajustar independientemente para crear una amplia gama de colores.

Cada una de estas formas se elige por su capacidad para representar visualmente las propiedades complejas del color de una manera que es accesible y útil para aplicaciones específicas, desde la edición de imágenes digitales hasta la interpretación científica de colores en diversas condiciones de iluminación.

Modelo de color HSV

Cuando es necesario trabajar con imágenes con operaciones de color, es frecuente utilizar HSV sobre otros modelos como RGB. Se debe a cómo HSV separa la información de color (tono) de la luminancia (valor), y eso facilita el tratamiento por ejemplo cuando necesitamos seleccionar o extraer un color o rango de color. Por ejemplo, podemos trabajar en HSV para extraer el color verde de fondo, especialmente para aplicaciones como el chroma keying.

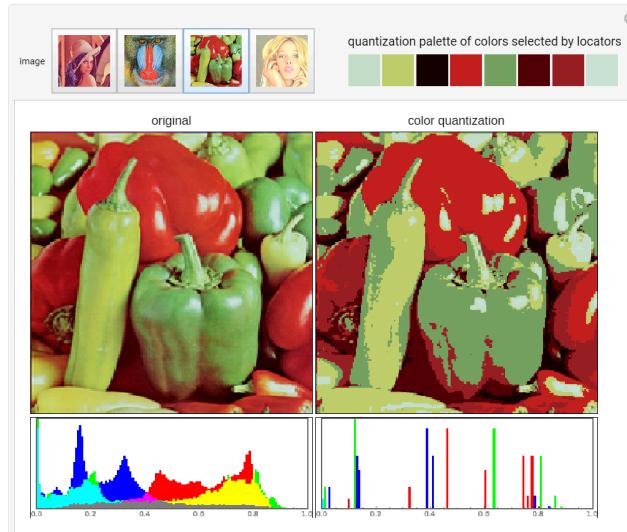


El espacio de color HSV representa los colores en términos de su tono (Hue), saturación (Saturation) y valor (Value, a veces denominado brillo). Esta separación hace que sea particularmente útil para tareas de procesamiento de imágenes como la detección y extracción de colores específicos, debido a que:

1. **Tonos bien diferenciados:** El componente de tono (H) representa el color puro, independientemente de su intensidad o saturación. Esto permite identificar y extraer colores basándose únicamente en su característica de tono, lo cual es ideal para identificar el verde en un fondo de pantalla verde.
2. **Flexibilidad en variaciones de iluminación:** La separación del componente de brillo (valor) del color permite a HSV manejar mejor las variaciones de iluminación. Los objetos o fondos verdes bajo diferentes condiciones de luz pueden ser detectados más consistentemente que con el modelo RGB, donde los cambios en la iluminación afectan directamente a los tres canales de color.
3. **Ajuste de saturación:** El componente de saturación ayuda a distinguir entre diferentes intensidades del mismo color, permitiendo filtrar los tonos verdes desaturados o muy brillantes que pueden no ser relevantes para la extracción deseada.

Reducción de colores

La reducción de paleta de colores, es un proceso que mapea los colores de una imagen digital a una paleta más pequeña de colores representativos. Esto se hace mediante la agrupación de colores similares en un solo valor de color.



<https://demonstrations.wolfram.com/ColorQuantizationOfPhotographicImages/PaletteFromColorsInThe/>

Cada pixel de la imagen original se sustituye por el color más cercano de la paleta reducida. Este proceso es útil por varias razones:

1. **Reducción del espacio de almacenamiento:** Algunas imágenes digitales, como las de formato de mapa de bits (BMP) sin compresión, pueden ocupar una gran cantidad de espacio en disco debido a la necesidad de almacenar la información de color para cada píxel. La cuantización de colores reduce el número de colores distintos, lo que permite un almacenamiento más eficiente.
2. **Visualización en dispositivos con capacidades limitadas:** Algunos dispositivos antiguos o de baja gama, como pantallas de baja profundidad de color o impresoras monocromáticas, no pueden representar la gama completa de colores de una imagen. La cuantización de colores permite adaptar la imagen a las capacidades del dispositivo.



3. **Compresión de imágenes:** La cuantización de colores es a menudo un paso previo a la compresión de imágenes, ya que reduce la cantidad de información de color que debe codificarse, lo que mejora la eficiencia de la compresión.
4. **Simplificación de imágenes para análisis:** La cuantización de colores simplifica significativamente el contenido visual de una imagen, lo cual es especialmente útil en aplicaciones de análisis de imágenes, ya que la reducción de colores simplifica la segmentación por colores.

Existen varios algoritmos para realizar la cuantización de colores, como el muestreo de colores, la agrupación de colores (clustering) y la búsqueda de paleta óptima. Estos algoritmos difieren en su complejidad computacional, calidad de la imagen resultante y capacidad para preservar los detalles importantes de la imagen original.

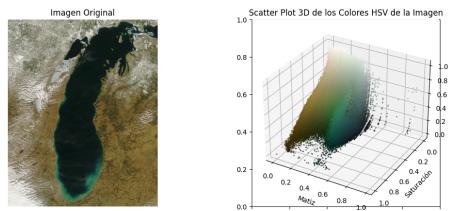


Es importante tener en cuenta que la cuantización de colores implica una pérdida de información, ya que se reduce el número de colores distintos presentes en la imagen. Por lo tanto, existe un compromiso entre la calidad de la imagen y el ahorro de espacio o los requisitos de visualización.

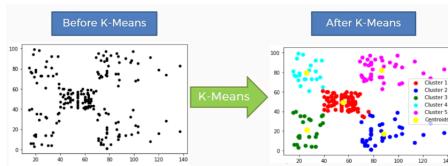
Clustering

En el procesamiento de imágenes, los métodos de clustering juegan un papel crucial en la segmentación, compresión, y análisis de imágenes. Estos métodos agrupan píxeles o características similares para simplificar el contenido visual de las imágenes, facilitando su interpretación y análisis.

Por ejemplo, en una fotografía, encontraremos un abanico grande de colores. Si los mostramos en una gráfica 3D con el modelo de colores HSV, veríamos algo como esto:



El objetivo que buscamos con la agrupación, es juntar por similitud o distancia, aquellos colores que se parecen, en k grupos. Los algoritmos de clustering nos permiten encontrar el punto central de esos grupos (centrodes), a través de diferentes técnicas.



Concepto del efecto de clustering. Luego de aplicar el método de K-Means agrupamos por cercanía, al rededor de un centroide para cada conjunto.

Aquí mencionamos algunos de los métodos de clustering más conocidos en procesamiento de imágenes:

1. **K-Means Clustering:** Es uno de los algoritmos de clustering más populares y ampliamente utilizado para la cuantización de color y la segmentación de imágenes. K-Means busca dividir los píxeles de una imagen en K grupos basados en la similitud de sus valores de color o intensidad, minimizando la varianza dentro de los grupos.
2. **Clustering Jerárquico:** Este método construye una jerarquía de clusters ya sea mediante un enfoque aglomerativo, que comienza con clusters individuales para cada píxel y los fusiona progresivamente, o mediante un enfoque divisorio, que comienza con un solo cluster que engloba todos los píxeles y lo divide sucesivamente. Es útil para la segmentación de imágenes basada en diferentes niveles de granularidad.
3. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):** Es un algoritmo basado en la densidad que puede identificar clusters de formas arbitrarias al agrupar píxeles densamente conectados e identificar puntos de ruido o de fondo. Es especialmente útil en la segmentación de imágenes cuando los objetos presentan variabilidad en su forma y tamaño.
4. **Mean Shift Clustering:** Este método localiza los centros de clusters buscando los modos o picos de densidad en el espacio de características. Es particularmente efectivo para la segmentación de imágenes cuando no se conoce de antemano el número de clusters, ya que no requiere especificar el número de clusters como un parámetro.

Ejemplos prácticos

La cuantización de colores mediante el algoritmo de K-means es una técnica poderosa y versátil en el procesamiento de imágenes, que permite simplificar la complejidad visual de una imagen reduciendo su número de colores a una paleta más pequeña. Este proceso no solo facilita la compresión y el almacenamiento eficiente de imágenes sino que también puede resaltar características visuales importantes y mejorar el rendimiento de algoritmos posteriores de análisis de imágenes.

El siguiente ejemplo demuestra cómo aplicar la cuantización de color a una imagen del Lago Michigan, utilizando el algoritmo K-means de la biblioteca `scikit-learn`. Partiendo de una imagen en color, el objetivo es reducir su diversidad de colores a solo cuatro colores representativos, seleccionados a través del agrupamiento de K-means basado en la similitud de color en el espacio RGB. Este proceso no solo simplifica la apariencia visual de la imagen sino que también destila su paleta de colores a los elementos más significativos, lo que puede ser especialmente útil para tareas como la visualización de datos, análisis de contenido, o simplemente para reducir el espacio de almacenamiento necesario para la imagen.



Inicialmente, la imagen se carga y se convierte al formato RGB para un manejo consistente del color. Luego, se redimensiona para optimizar el tiempo de procesamiento, seguido por la reestructuración de los datos de la imagen para que se ajusten al formato requerido por el algoritmo K-means. Después de ejecutar K-means, los colores de cada píxel en la imagen se reemplazan por el color del centroide del cluster más cercano, resultando en una versión cuantizada de la imagen original. Finalmente, el ejemplo culmina con la visualización de la imagen cuantizada junto con su paleta de colores reducida, proporcionando una representación visual clara del resultado de la cuantización.

```
# !wget 'https://raw.githubusercontent.com/jpmanson/tuia-unr/main/images/modis-lake-michigan.jpg' -O modis-lake-michigan.jpg

import cv2
import numpy as np
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt

# Cargar la imagen usando OpenCV
imagen = cv2.imread('modis-lake-michigan.jpg')
imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB) # Convertir de BGR a RGB

# Redimensionar la imagen para un procesamiento más rápido
imagen = cv2.resize(imagen, (512, 512), interpolation=cv2.INTER_AREA)

# Preparar los datos para K-means
pixels = imagen.reshape((-1, 3))
pixels = np.float32(pixels)

# Definir y ejecutar K-means
k = 4 # Número de clusters
kmeans = KMeans(n_clusters=k, random_state=0).fit(pixels)

# Reconstruir la imagen con los colores de los centroides
centroides = np.uint8(kmeans.cluster_centers_)

# Mostrar la imagen resultante y la paleta de colores
fig, ax = plt.subplots(1, 2, figsize=(12, 6), gridspec_kw={'width_ratios': [3, 1]})

# Imagen resultante
imagen_kmeans = centroides[kmeans.labels_.flatten()]
imagen_kmeans = imagen_kmeans.reshape(imagen.shape)
```

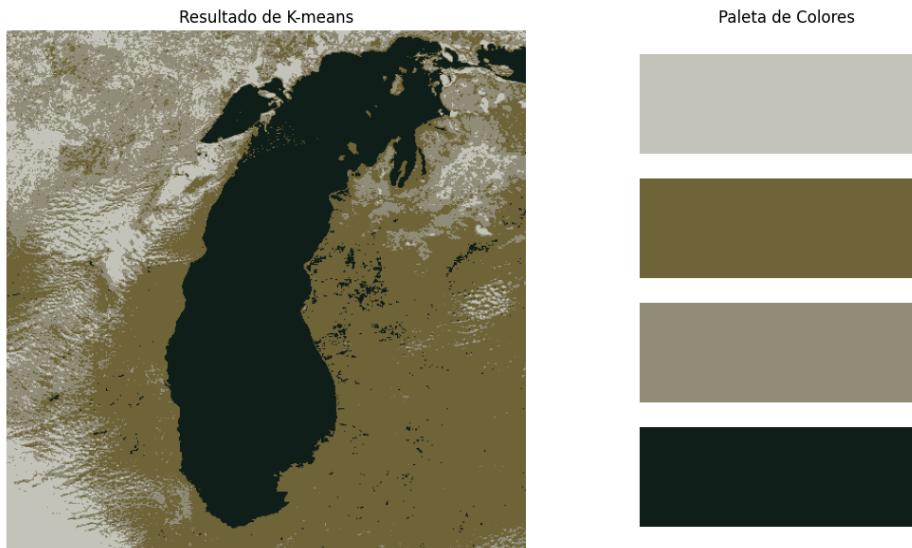
```

ax[0].imshow(imagen_kmeans)
ax[0].set_title('Resultado de K-means')
ax[0].axis('off')

# Paleta de colores
for i, color in enumerate(centroides):
    ax[1].barh(i, 1, color=color/255, edgecolor='none')
ax[1].set_title('Paleta de Colores')
ax[1].axis('off')
plt.tight_layout()
plt.show()

```

Y el resultado será el siguiente:



Repositorios con más ejemplos y métodos:

<https://github.com/Chadys/QuantizeImageMethods>

<https://github.com/annoviko/pyclustering>

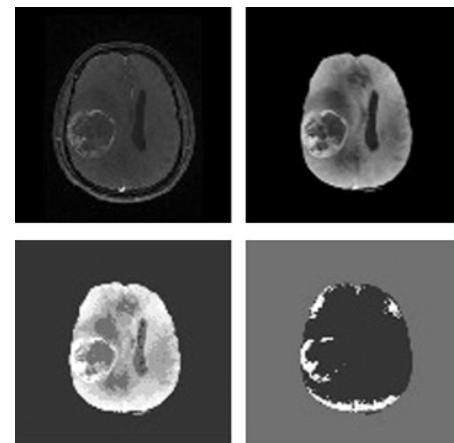
Aplicaciones

La cuantización de color en imágenes es una técnica crucial en visión por computadora con aplicaciones prácticas significativas en diversas industrias. Al reducir el número de colores en una imagen, esta técnica permite simplificar el procesamiento de imágenes y mejorar el rendimiento de los algoritmos de visión por computadora, facilitando desde la segmentación hasta el análisis avanzado de imágenes. A continuación, se exploran algunas aplicaciones comerciales:

1. Clasificación de objetos e inspección: La reducción de colores es útil para clasificar objetos por su composición de colores. Puede combinarse con un detector de objetos tradicional, de modo que primero detectamos el objeto y luego lo clasificamos de acuerdo a su composición de colores. En la manufactura, la cuantización de color ayuda en el control de calidad visual automatizado. La cuantización de color reduce la complejidad de las imágenes capturadas, facilitando la detección rápida y precisa de defectos.

<https://www.youtube.com/shorts/lr1MqOV3qls>

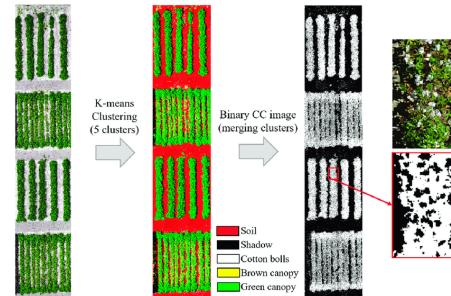
Análisis de Imágenes Médicas: La cuantización de color es fundamental en el análisis de imágenes médicas para simplificar la visualización y el análisis de datos complejos. Hospitales y centros de investigación utilizan esta técnica para mejorar la visualización de imágenes de resonancia magnética (MRI) y tomografía computarizada (CT). La cuantización de color ayuda en la segmentación de tejidos en imágenes médicas, facilitando diagnósticos más rápidos y precisos.



[Evaluation of k-Means and fuzzy C-means segmentation on MR images of brain](#)

Agricultura de Precisión

En la agricultura de precisión, la cuantización de color facilita el análisis de imágenes satelitales y de drones para monitorear la salud de los cultivos y la gestión del agua. Estas técnicas se utilizan para analizar la variabilidad de colores en los campos, ayudando a los agricultores a optimizar el uso de recursos y mejorar las técnicas de cultivo. Las técnicas de cuantización se utilizan en el cálculo del NDVI, que el Índice de vegetación de diferencia normalizada, el cual se utiliza para estimar la cantidad, calidad y desarrollo de la vegetación con datos aportados por medio de sensores remotos, como satélites o drones.



[A Comparative Study of RGB and Multispectral Sensor-Based Cotton Canopy Cover Modelling Using Multi-Temporal UAS Data](#)

12. Image captioning

Introducción y conceptos teóricos

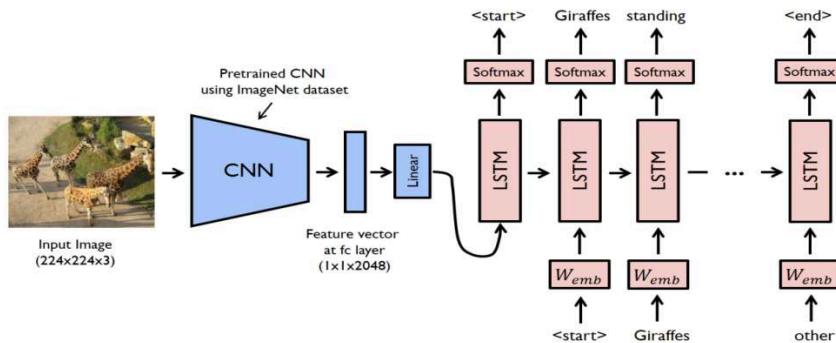
Image captioning (subtítulos de imágenes) son la tarea de describir el contenido de una imagen en palabras. Esta tarea se encuentra en la intersección de la visión por computadora y el procesamiento del lenguaje natural.

A young boy is playing basketball. 	Two dogs play in the grass. 	A dog swims in the water. 	A little girl in a pink shirt is swinging.
A group of people walking down a street. 	A group of women dressed in formal attire. 	Two children play in the water. 	A dog jumps over a hurdle.

Componentes Clave del *Image Captioning*

- Extracción de Características de la Imagen:** Usualmente, se utiliza una red neuronal convolucional (CNN) para analizar la imagen y extraer características visuales relevantes. La CNN actúa como un "encoder" que transforma la imagen en un conjunto compacto de características numéricas que describen los elementos visuales importantes de la imagen, como objetos, texturas y colores.
- Generación de Texto:** A partir de las características extraídas por la CNN, se utiliza una red neuronal recurrente (RNN) o una variante como LSTM (Long Short-Term Memory) o GRU (Gated Recurrent Units) para generar texto descriptivo. Esta parte del modelo actúa como un "decoder", que convierte el conjunto de características visuales en una secuencia de palabras que forman una oración coherente y relevante para la imagen.

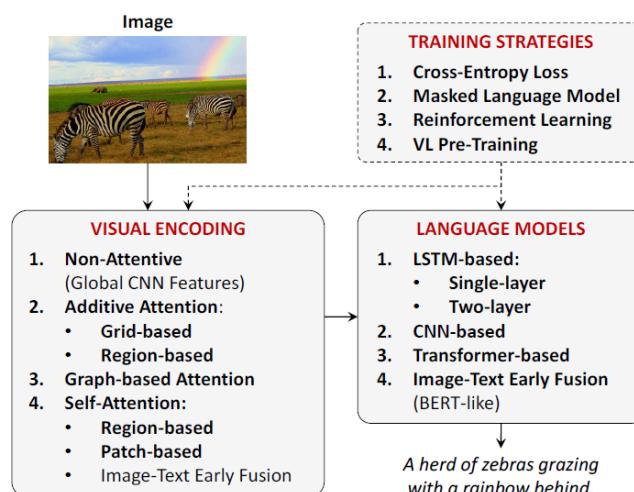
La mayoría de los sistemas modernos de subtítulos de imágenes utilizan un marco codificador-decodificador, donde una imagen de entrada se codifica en una representación intermedia de la información de la imagen y luego se decodifica en una secuencia de texto descriptivo:



[A Survey on Various Deep Learning Models for Automatic Image Captioning](#)

El proceso aquí se puede descomponer en los siguientes pasos:

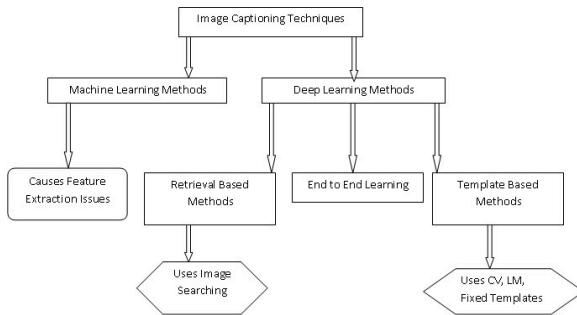
- Pre-procesamiento:** La imagen es pre-procesada para estandarizar su tamaño y formato, adecuándola para la entrada en la CNN.
- Codificación:** La CNN procesa la imagen y crea un vector de características que resume el contenido visual de la imagen.
- Decodificación:** El vector de características es la entrada para la RNN, que genera una secuencia de palabras una por una, formando una oración que describe la imagen. Durante este proceso, cada palabra generada es influenciada por las palabras anteriores y las características visuales de la imagen, asegurando que la descripción sea relevante y contextualmente adecuada.



Resumen de la tarea de descripción de imágenes y taxonomía de los enfoques más relevantes (Paper: "[From Show to Tell: A Survey on Deep Learning-based Image Captioning](#)")

Métodos y clasificaciones

Existen diferentes líneas de trabajo y modelos para hacer el subtitulado:



Enfoques basados de aprendizaje automático

En el enfoque tradicional de aprendizaje automático, los datos de entrada se utilizan para extraer características. Este enfoque puede emplearse para la descripción de imágenes, pero no es altamente eficiente. La razón es que extraer características artesanales como los Patrones Binarios Locales (LBP), Histograma de Gradientes Orientados (HOG) y la Transformada de Características Invariantes a la Escala (SIFT) de un conjunto de datos grande no es fácil ni factible. Las imágenes del mundo real son más complejas y los conjuntos de datos pueden ser muy diversificados. Por lo tanto, el enfoque de aprendizaje automático para la descripción de imágenes no es muy eficiente.

Enfoque basado en aprendizaje profundo

En los últimos años, se han publicado diversos trabajos sobre aprendizaje profundo aplicados a la descripción de imágenes, proponiendo varios enfoques dentro de este marco.

Veremos diversos marcos basados en el aprendizaje profundo para la descripción de imágenes:

1. Enfoque basado en recuperación para la descripción de imágenes:

La descripción de imágenes mediante el enfoque de recuperación utiliza redes neuronales convolucionales profundas (CNN) y autoencoders para encontrar características en las imágenes y detectar diversas palabras relacionadas con estas características. Este método busca imágenes con descripciones similares en el conjunto de datos y luego ajusta la descripción final según las descripciones encontradas. Algunos modelos que usan esta metodología son

DeViSE, NeuralTalk, TWINs, VSE++

2. Enfoque basado en plantillas para la descripción de imágenes:

El enfoque de recuperación es útil para la descripción de imágenes, pero tiene algunas limitaciones, ya que genera descripciones utilizando únicamente palabras encontradas en imágenes similares. Este enfoque puede omitir algunos objetos importantes que no están presentes en las palabras previamente encontradas. Por lo tanto, los investigadores utilizan el método de plantillas para superar los problemas encontrados en el enfoque de recuperación. En este método, se identifican algunas palabras clave que representan objetos y sus atributos. Luego, estas palabras generadas se fijan en la plantilla para finalmente generar una descripción. Es un enfoque efectivo, pero tiene la limitación de generar descripciones de longitud fija solamente. Ejemplo:

Baby_Talk

3. Enfoque de aprendizaje de extremo a extremo para la descripción de imágenes:

El enfoque de aprendizaje de extremo a extremo es el mejor hasta la fecha y es utilizado por casi todos los investigadores para la descripción de imágenes. El aprendizaje de los parámetros se puede realizar directamente a través del entrenamiento. Se utiliza una Red Neuronal Convolutiva Profunda (DCNN) como codificador para leer y convertir imágenes de entrada en vectores de características, que tienen la misma dimensión que la dimensión de entrada de una Red Neuronal Recurrente (RNN) que se utilizará para traducir el vector en una oración.

Ejemplos prácticos

Para trabajar con modelos de procesamiento de imágenes y generación de texto en Python, una de las herramientas que podemos usar es el paquete salesforce-lavis (<https://github.com/salesforce/lavis>). Esta librería facilita la carga y aplicación de modelos de inteligencia artificial avanzados para tareas como la generación automática de descripciones de imágenes (*image captioning*) y responder preguntas sobre imágenes (*visual question answering*). A continuación, se muestra cómo configurar y utilizar este paquete para generar descripciones automáticas de una imagen y

responder una pregunta específica sobre ella.

```
import torch
from PIL import Image
from lavis.models import load_model_and_preprocess

# Configuración del dispositivo para usar GPU si está disponible, de lo contrario CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Cargamos el modelo junto con sus preprocesadores para facilitar la inferencia
model, vis_processors, _ = load_model_and_preprocess(
    name="blip_caption", model_type="large_coco", is_eval=True, device=device
)

# Preprocesamiento de la imagen
raw_image = Image.open("/content/france.jpg").convert("RGB")
image = vis_processors["eval"]((raw_image).unsqueeze(0).to(device))

# Generación de descripciones
# Debido a la naturaleza no determinista del muestreo de núcleo, es posible obtener descripciones diferentes cada vez
result = model.generate({"image": image}, use_nucleus_sampling=True, num_captions=3)
print(result)
```

Además de generar descripciones, también podemos utilizar el modelo para responder preguntas sobre la imagen. Primero, cargamos el modelo adecuado y preparamos tanto la imagen como la pregunta:

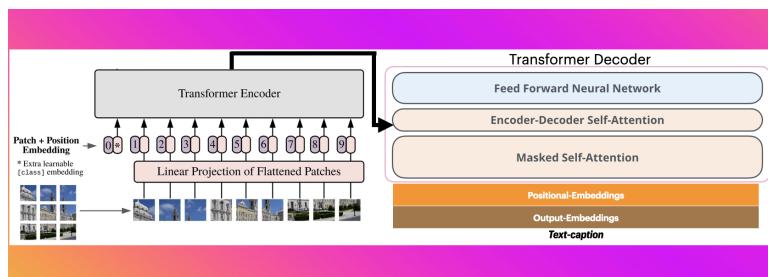
```
from lavis.models import load_model_and_preprocess

# Cargamos el modelo para el Visual Question Answering (VQA)
model, vis_processors, txt_processors = load_model_and_preprocess(
    name="blip_vqa", model_type="vqav2", is_eval=True, device=device
)

# Formulamos una pregunta sobre la imagen
question = "Which city is this photo taken?"
image = vis_processors["eval"]((raw_image).unsqueeze(0).to(device))
question = txt_processors["eval"]((question))

# Predecimos la respuesta a la pregunta formulada
result = model.predict_answers(samples={"image": image, "text_input": question}, inference_method="generate")
print(result)
```

Otro ejemplo es usar el modelo VIT GPT-2 basado en transformers, de tipo Vision Encoder Decoder Model.



```
from transformers import GPT2TokenizerFast, ViTImageProcessor, VisionEncoderDecoderModel
from torch.utils.data import Dataset
from torchtext.data import get_tokenizer
```

```

import requests
import torch
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import os
from tqdm import tqdm

import warnings
warnings.filterwarnings('ignore') # Ignora las advertencias para evitar distracciones en la salida

# Carga del modelo preentrenado para descripción de imágenes
model_raw = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
image_processor = ViTImageProcessor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
tokenizer = GPT2TokenizerFast.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

# Definición de la función para mostrar y generar descripciones de imágenes
def show_n_generate(url, greedy=True, model=model_raw):
    image = Image.open(requests.get(url, stream=True).raw) # Descarga y carga la imagen desde una URL
    pixel_values = image_processor(image, return_tensors="pt").pixel_values # Procesa la imagen para el modelo
    plt.imshow(np.asarray(image)) # Muestra la imagen
    plt.show()

    if greedy:
        # Genera una descripción usando el modo 'greedy' (el más probable)
        generated_ids = model.generate(pixel_values, max_new_tokens=30)
    else:
        # Genera una descripción permitiendo cierta aleatoriedad en la elección de palabras
        generated_ids = model.generate(
            pixel_values,
            do_sample=True,
            max_new_tokens=30,
            top_k=5)
    generated_text = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0] # Decodifica los tokens generados
    print(generated_text) # Imprime la descripción generada

# URL de la imagen a describir
url = "http://images.cocodataset.org/val2017/000000039769.jpg"

# Mostramos los resultados
show_n_generate(url, greedy = False)

```

Y el resultado será:



"two cats that are sleeping in a green blanket"

Aplicaciones

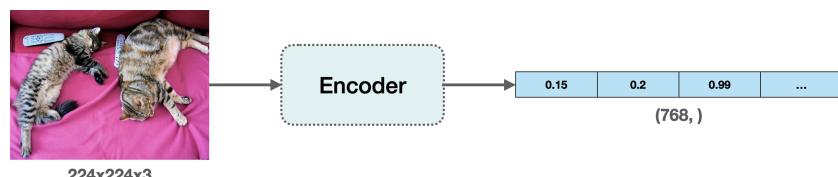
El *image captioning*, o subtítulo de imágenes, es una tecnología avanzada que encuentra aplicaciones en varios campos, aprovechando la intersección entre visión por computadora y procesamiento de lenguaje natural. Aquí están algunas de las aplicaciones más destacadas:

1. **Asistencia para personas con discapacidad visual:** Una de las aplicaciones más significativas del *image captioning* es ayudar a las personas con discapacidad visual a entender el contenido visual que les rodea. Al describir verbalmente lo que aparece en las imágenes, estos sistemas pueden ayudar a mejorar la accesibilidad y la navegación en entornos tanto digitales como físicos. Plataformas de redes sociales utilizan *image captioning* para generar descripciones automáticas de imágenes que se publican en sus sitios, permitiendo que los usuarios con discapacidad visual puedan participar más plenamente en estas plataformas.
2. **Automatización en bibliotecas de imágenes y archivos:** En medios de comunicación y bibliotecas digitales donde se manejan grandes volúmenes de imágenes, el *image captioning* facilita la organización, búsqueda y recuperación de imágenes basándose en su contenido descriptivo. Esto es especialmente útil para archivistas que necesitan catalogar grandes cantidades de material visual.
3. **Análisis de contenido multimedia en investigación y medios:** Investigadores y profesionales de medios utilizan el *image captioning* para analizar rápidamente grandes cantidades de material visual, identificando patrones y extrayendo información útil que puede ser utilizada para diversos fines analíticos.
4. **Automatización en seguridad y vigilancia:** En sistemas de vigilancia, la capacidad de generar descripciones textuales automáticas de escenas capturadas por cámaras puede facilitar la monitorización y respuesta a eventos en tiempo real, mejorando la seguridad y la eficiencia operativa.
5. **Interfaz de usuario para búsqueda visual avanzada:** En aplicaciones de comercio electrónico y plataformas de búsqueda, el *image captioning* puede mejorar las interfaces de usuario permitiendo búsquedas basadas en descripciones de imagen naturales, facilitando a los usuarios encontrar productos o información basada en descripciones visuales simples.

13. Image embeddings

Introducción y conceptos teóricos

Los *image embeddings* son representaciones numéricas de imágenes, que condensan información visual relevante en vectores de dimensiones más reducidas.



Estos embeddings se generan usualmente mediante modelos de aprendizaje profundo y tienen una amplia gama de aplicaciones, incluyendo la búsqueda de imágenes, la recomendación de productos, y la mejora de la interacción entre humanos y máquinas.

La creación de un embedding de una imagen implica transformar los píxeles brutos en un espacio vectorial donde las distancias entre los puntos reflejan alguna forma de similitud semántica entre las imágenes correspondientes. Este proceso permite que tareas como la clasificación y la recuperación de imágenes sean más eficientes y efectivas, incluso cuando se enfrentan a un gran volumen de datos visuales.

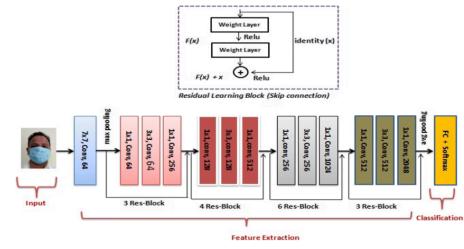


Este marco conceptual es importante por sus aplicaciones prácticas en industrias creativas, de seguridad y de comercio electrónico, donde la capacidad de filtrar, organizar y recomendar contenido visual de manera eficaz puede ser un gran activo.

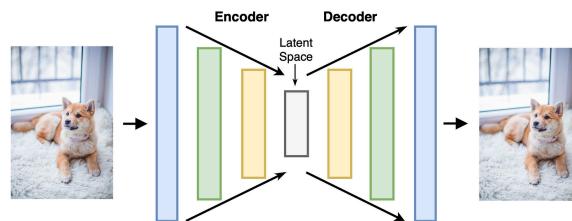
Métodos para generar embeddings

Existen diversos métodos para generar embeddings de imágenes. Aquí mencionamos algunos de los más comunes:

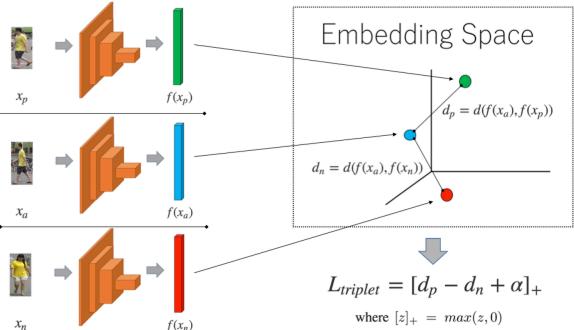
- 1. Redes Neuronales Convolucionales (CNN):** Las CNN son especialmente eficaces para extraer características visuales relevantes de las imágenes. Las capas intermedias de una CNN preentrenada en un gran conjunto de datos, como ImageNet, pueden ser utilizadas para generar embeddings de imágenes. Modelos como Resnet realizan un buen trabajo para conseguir los embeddings, quitando por ejemplo las últimas capas de clasificación.



- 2. Autoencoders:** Los autoencoders son redes neuronales que se entran para reconstruir sus entradas. La capa oculta intermedia, más pequeña que la entrada, se utiliza como embedding, comprimiendo la información de la imagen en un espacio de menor dimensión. Llamamos a ésta "espacio latente", y podemos usar esa representación como un embedding de la imagen de entrada.

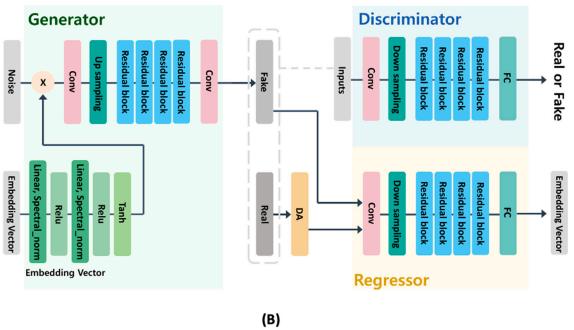
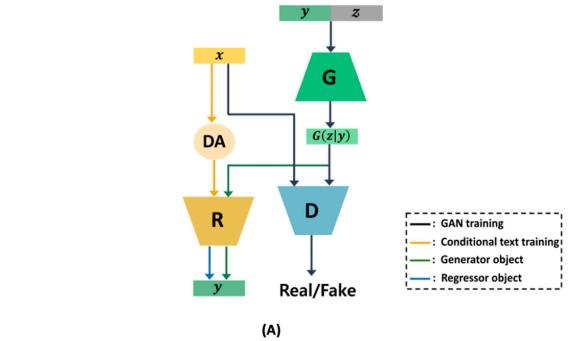


3. Redes Siamesas y Triplet Networks: Estas redes se entrena para generar embeddings que reflejan la similitud semántica entre imágenes. Las Redes Siamesas toman dos imágenes como entrada y se entrena para que los embeddings estén cerca si las imágenes son similares y lejos si son diferentes. Las Triplet Networks amplían este concepto tomando tres imágenes como entrada: una imagen de anclaje, una imagen positiva (similar a la anclaje) y una imagen negativa (diferente de la anclaje). Se entrena para que el embedding del anclaje esté más cerca del positivo que del negativo.

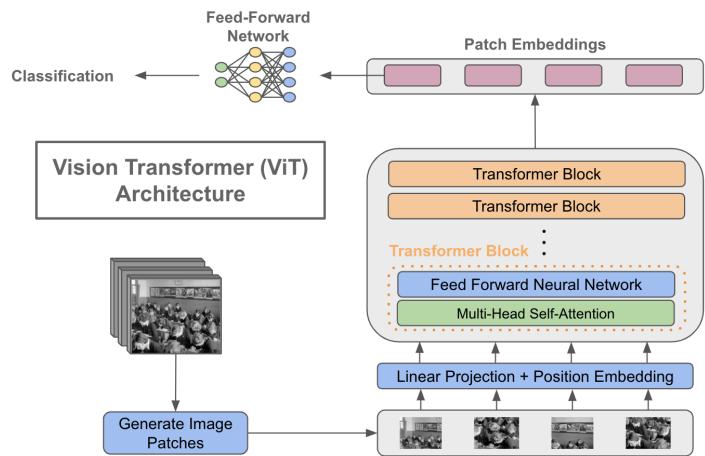


<https://levelup.gitconnected.com/metric-learning-using-siamese-and-triplet-convolutional-neural-networks-ed5b01d83be3>

4. Redes Generativas Adversarias (GAN): Las GAN tienen dos componentes, un generador y un discriminador. Aunque se utilizan más comúnmente para generar nuevas imágenes, el discriminador de una GAN entrenada puede usarse para generar embeddings de imágenes.



1. Visual Transformers: Aunque son más conocidos por su uso en el procesamiento del lenguaje natural, los Transformers también se utilizan en visión por computadora. Los Transformers pueden generar embeddings de imágenes que capturan relaciones de largo alcance entre diferentes partes de la imagen.



Fuente: [Using Transformers for Computer Vision](#)

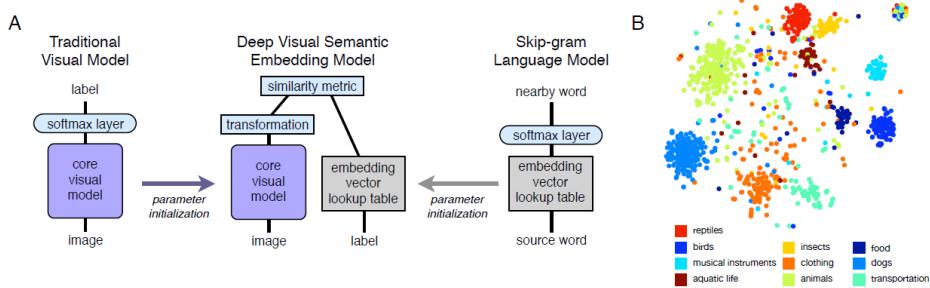
Embeddings de imágenes y texto

La integración de embeddings de imágenes con texto representa un avance significativo en la intersección de la visión por computadora y el procesamiento del lenguaje natural (NLP). Esta convergencia de modalidades no solo mejora la interpretación semántica de los contenidos visuales, sino que también amplía las capacidades de las aplicaciones de inteligencia artificial para comprender y operar en entornos más complejos y humanos. Esto permite nuevas aplicaciones como:

- **Mejora del Aprendizaje Zero-Shot:** Una de las aplicaciones más importantes de la integración de embeddings de imágenes con texto es en el aprendizaje zero-shot. En este contexto, los modelos no solo reconocen los objetos para los cuales fueron explícitamente entrenados, sino también objetos nunca vistos durante el entrenamiento, utilizando el conocimiento semántico aprendido de los textos. Esto es crucial para aplicaciones en entornos dinámicos donde no es práctico recopilar datos de entrenamiento para cada posible objeto que podría necesitar ser reconocido.
- **Búsqueda y Recuperación de Imágenes Mejorada:** La capacidad de buscar imágenes basadas en descripciones textuales detalladas mejora significativamente con la integración de embeddings de texto e imagen. Los sistemas que utilizan embeddings visuales y textuales combinados pueden entender mejor las consultas en lenguaje natural, permitiendo una recuperación más precisa y relevante de imágenes que corresponden a descripciones textuales específicas.
- **Generación Automática de Descripciones de Imágenes (Captioning):** El captioning automático de imágenes, que implica generar descripciones textuales para imágenes, se beneficia enormemente de la integración de embeddings visuales y textuales. Al entender tanto el contenido visual como el contexto semántico, los modelos pueden producir descripciones más naturales y precisas. Esto tiene aplicaciones importantes en accesibilidad digital, como ayudar a las personas con discapacidades visuales a entender el contenido visual a través de descripciones textuales.

DeViSe

El modelo DeViSE (Deep Visual-Semantic Embedding) es un enfoque avanzado en el campo del aprendizaje profundo y la visión por computadora, diseñado para crear un puente entre el espacio visual de las imágenes y el espacio semántico del texto. Desarrollado por Andrea Frome y sus colegas en Google en 2013, DeViSE se introdujo como una solución innovadora para el problema de la clasificación de imágenes en contextos donde no están disponibles etiquetas de entrenamiento para todas las categorías posibles. Esto se aborda mediante la incorporación de información semántica derivada de texto no etiquetado, lo que permite al modelo realizar clasificaciones "zero-shot", es decir, reconocer categorías de objetos que nunca ha visto durante el entrenamiento.



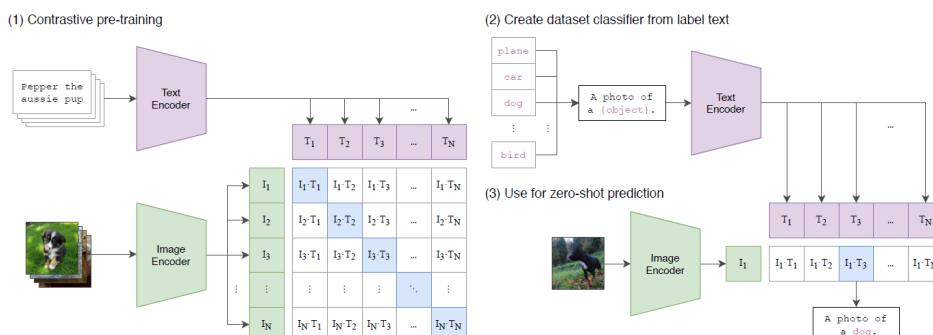
(a) Izquierda: una red de categorización de objetos visuales con una capa de salida softmax; Derecha: un modelo de lenguaje skip-gram; Centro: El modelo DeViSE, que se inicializa con parámetros preentrenados en las capas inferiores de los otros dos modelos. (b) Visualización t-SNE de un subconjunto de los embeddings de etiquetas de 1K del ILSVRC 2012 aprendidos utilizando skip-gram.

El modelo DeViSE es especialmente útil para la clasificación zero-shot, donde el modelo necesita clasificar imágenes en categorías que no estaban presentes en el conjunto de datos de entrenamiento. Esto se logra al aprender a mapear imágenes a un espacio semántico en el que se pueden comparar directamente con descripciones textuales. DeViSE puede ser utilizado para mejorar los sistemas de búsqueda de imágenes que permiten a los usuarios buscar imágenes utilizando consultas de texto. El modelo también se presta para tareas de generación automática de descripciones de imágenes, donde el objetivo es crear descripciones textuales precisas y ricas semánticamente para imágenes.

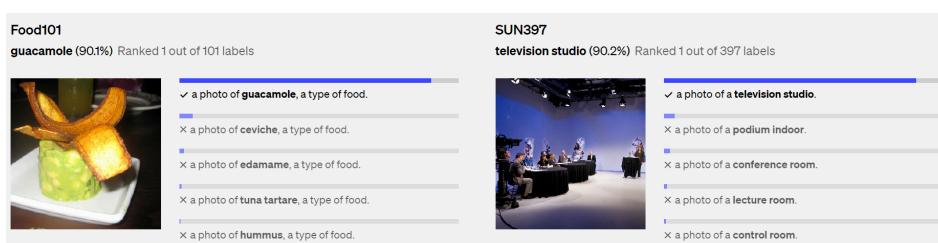
CLIP

El modelo CLIP (Contrastive Language–Image Pretraining), publicado por OpenAI en 2021, es un modelo de embedding multimodal que ha marcado una diferencia significativa en comparación con los modelos anteriores. Fue diseñado para entender y clasificar imágenes con un conjunto masivo de datos de más de 400 millones de pares de imágenes y textos recopilados de Internet y utiliza una combinación de Transformers y DCNN (ResNet) en su arquitectura.

Su capacidad distintiva es realizar tareas de clasificación de imágenes sin necesidad de entrenamiento adicional específico para las categorías objetivo ("zero-shot learning"). Antes de CLIP, los modelos de embedding visual-semánticos, como DeViSE y otros basados en redes neuronales profundas, también buscaban mapear imágenes a un espacio semántico utilizando texto. Sin embargo, estos modelos generalmente requerían reajustes o adaptaciones específicas para cada nueva categoría de clasificación y no escalaban bien con el número de categorías debido a la necesidad de reentrenar con cada expansión del conjunto de datos. CLIP, al operar efectivamente en un modo de cero disparos y ser capaz de entender directamente las descripciones textuales sin entrenamiento adicional.



Mientras que los modelos de imágenes estándar entran conjuntamente un extractor de características de imágenes y un clasificador lineal para predecir alguna etiqueta, CLIP entrena conjuntamente un codificador de imágenes y un codificador de texto para predecir las combinaciones correctas de un lote de ejemplos de entrenamiento (imagen, texto). En el momento de la prueba, el codificador de texto aprendido sintetiza un clasificador lineal de cero disparos mediante la incrustación de los nombres o descripciones de las clases del conjunto de datos objetivo.



Limitaciones de CLIP: Aunque CLIP generalmente funciona bien reconociendo objetos comunes, tiene dificultades en tareas más abstractas o sistemáticas, como contar el número de objetos en una imagen, y en tareas más complejas, como predecir qué tan cerca está el coche más próximo en una foto. CLIP también tiene dificultades, comparado con modelos específicos para tareas, en clasificaciones muy detalladas, como distinguir entre modelos de coches, variantes de aviones o especies de flores. CLIP también sigue teniendo una pobre generalización en imágenes que no están cubiertas en su conjunto de datos de pre-entrenamiento. Por ejemplo, aunque CLIP es un sistema OCR competente, cuando se evalúa en dígitos manuscritos del conjunto de datos MNIST, CLIP en modo de cero disparos solo alcanza un 88% de precisión, muy por debajo del 99.75% de los humanos en ese conjunto de datos. Además se ha observado que los clasificadores en modo de cero disparos de CLIP pueden ser sensibles a la formulación o fraseo y a veces requieren de ensayo y error en la "ingeniería de prompts" para funcionar bien.

Ejemplos prácticos

Visualización de embeddings

En el siguiente ejemplo, utilizaremos la biblioteca FiftyOne, una herramienta poderosa para la visualización, exploración y análisis de datasets de machine learning, en conjunto con modelos de embedding de imágenes pre-entrenados y técnicas de reducción de dimensionalidad como PCA, t-SNE y UMAP. Este enfoque nos permite no solo entender mejor las características subyacentes de los datos de imagen, sino también comparar la efectividad de diferentes modelos de embedding y métodos de visualización. El proceso detallado es:

- 1. Instalación de Dependencias:** Comenzamos instalando las librerías necesarias, incluyendo FiftyOne (<https://github.com/voxel51/fiftyone>), scikit-learn y umap-learn (<https://github.com/lmcinnes/umap>), que facilitarán la carga de modelos, la manipulación de datasets y la aplicación de algoritmos de reducción de dimensionalidad.
- 2. Carga de Modelos y Dataset:** Utilizamos modelos de embeddings pre-entrenados como CLIP (<https://github.com/openai/CLIP>) y ResNet101, y cargamos el dataset CIFAR-10 desde el zoo de datasets de FiftyOne. Este conjunto de datos es ampliamente utilizado para tareas de clasificación de imágenes y proporciona una base sólida para nuestros experimentos de embeddings.
- 3. Cómputo de Embeddings:** Calculamos los embeddings para las imágenes del dataset utilizando los modelos seleccionados. Estos embeddings capturan las características esenciales de las imágenes en vectores de baja dimensión.
- 4. Aplicación de Técnicas de Reducción de Dimensionalidad:** Aplicamos métodos como PCA, t-SNE y UMAP a los embeddings generados para visualizar cómo las imágenes están agrupadas en el espacio de baja dimensión. Cada técnica tiene sus propias fortalezas en preservar diferentes aspectos de la estructura de los datos (local o global), y su comparación puede ofrecer insights valiosos sobre la naturaleza de los datos y la efectividad de los embeddings.
- 5. Visualización:** Finalmente, lanzamos la aplicación FiftyOne para visualizar los resultados de nuestras reducciones dimensionales. Esta interfaz permite una exploración interactiva de cómo las imágenes están organizadas y agrupadas según los diferentes embeddings y métodos de reducción.

```
# !pip install -U fiftyone scikit-learn umap-learn

# Importamos las bibliotecas necesarias
import fiftyone as fo
import fiftyone.brain as fob
import fiftyone.zoo as foz

# Cargamos modelos preentrenados de embeddings de imágenes desde FiftyOne Zoo
clip = foz.load_zoo_model("clip-vit-base32-torch")
resnet101 = foz.load_zoo_model("resnet101-imagenet-torch")

# Cargamos el dataset CIFAR-10, especificando que solo queremos el split de 'test'
dataset = foz.load_zoo_dataset("cifar10", split="test")

## Computamos y almacenamos los embeddings usando el modelo ResNet101
dataset.compute_embeddings(
    resnet101,
    embeddings_field="resnet101_embeddings"
)
```

```

)

## Computamos y almacenamos los embeddings usando el modelo CLIP
dataset.compute_embeddings(
  clip,
  embeddings_field="clip_embeddings"
)

## Aplicamos PCA a los embeddings de ResNet101 y almacenamos el resultado
fob.compute_visualization(
  dataset,
  embeddings="resnet101_embeddings",
  method="pca",
  brain_key="resnet101_pca"
)

## Aplicamos PCA a los embeddings de CLIP y almacenamos el resultado
fob.compute_visualization(
  dataset,
  embeddings="clip_embeddings",
  method="pca",
  brain_key="clip_embeddings_pca"
)

## Aplicamos t-SNE a los embeddings de ResNet101 y almacenamos el resultado
fob.compute_visualization(
  dataset,
  embeddings="resnet101_embeddings",
  method="tsne",
  brain_key="resnet101_tsne"
)

## Aplicamos t-SNE a los embeddings de CLIP y almacenamos el resultado
fob.compute_visualization(
  dataset,
  embeddings="clip_embeddings",
  method="tsne",
  brain_key="clip_embeddings_tsne"
)

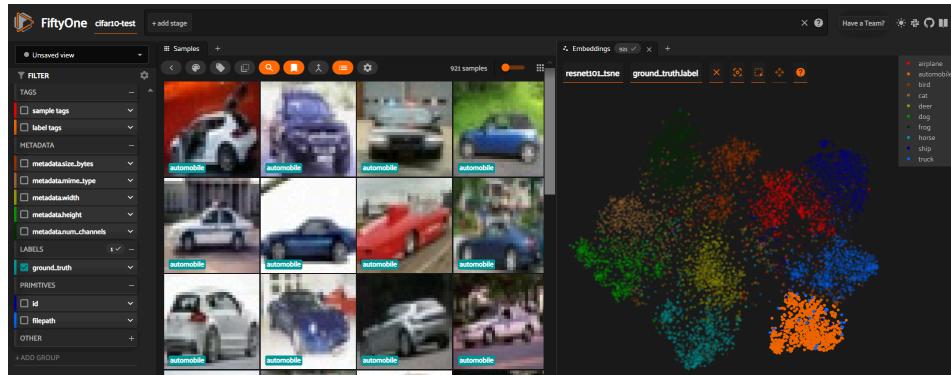
## Aplicamos UMAP a los embeddings de ResNet101 y almacenamos el resultado
fob.compute_visualization(
  dataset,
  embeddings="resnet101_embeddings",
  method="umap",
  brain_key="resnet101_umap"
)

## Aplicamos UMAP a los embeddings de CLIP y almacenamos el resultado
fob.compute_visualization(
  dataset,
  embeddings="clip_embeddings",
  method="umap",
  brain_key="clip_embeddings_umap"
)

```

```
# Iniciamos la aplicación FiftyOne para explorar visualmente los datos y las visualizaciones creadas
session = fo.launch_app(dataset)
```

Podremos luego navegar el dataset y los embeddings obtenidos, a través de los diferentes métodos de reducción dimensional que hemos utilizado:



Aquí vemos un ejemplo con otro dataset (rostros) utilizando CLIP y UMAP:

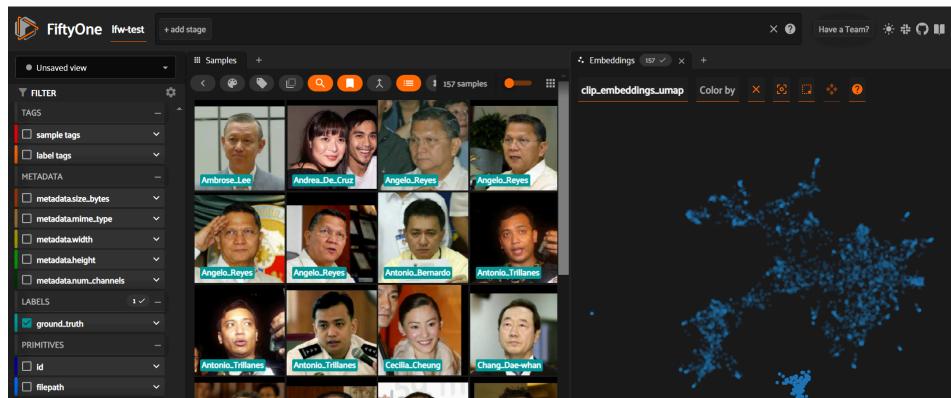
```
# Dataset "Labeled Faces in the Wild" (Caras Etiquetadas en Entornos Naturales)
dataset = foz.load_zoo_dataset("lfw", split="test")

## Computa y almacena los embeddings de CLIP
dataset.compute_embeddings(
    clip,
    embeddings_field="clip_embeddings"
)

## UMAP con embeddings de CLIP
fob.compute_visualization(
    dataset,
    embeddings="clip_embeddings",
    method="umap",
    brain_key="clip_embeddings_umap"
)

# Lanza la aplicación FiftyOne
session = fo.launch_app(dataset)
```

Veremos luego algo como esto, utilizando la herramienta de selección en el lado derecho:

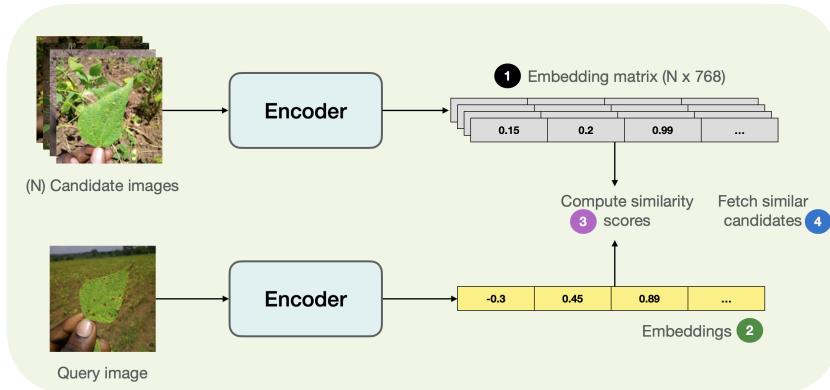


Búsqueda por similaridad

Otro ejemplo interesante, es el propuesto por Hugging Face en el siguiente cuaderno:

https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/image_similarity.ipynb

El modelo que utilizaremos allí es `vit-base-beans`. En dicho ejemplo, podemos buscar las imágenes más cercanas o similares, a partir de una imagen de consulta (query):



CLIP

Para ver como utilizar el modelo CLIP, podemos usar el cuaderno oficial publicado en <https://github.com/openai/CLIP> :
https://colab.research.google.com/github/openai/clip/blob/master/notebooks/Interacting_with_CLIP.ipynb

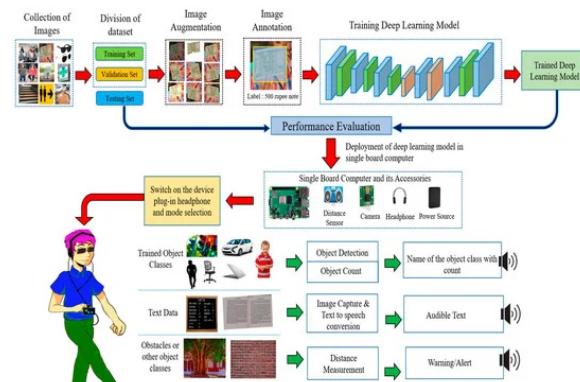
Más información en: <https://anirbansen2709.medium.com/beginners-guide-to-one-of-the-best-vision-model-clip-contrastive-learning-image-pre-training-46edf55e11c1>

Aplicaciones

Aquí se mencionan algunas aplicaciones de los embeddings de imágenes:

- **Búsqueda Visual de Imágenes (Image Retrieval):** Los embeddings permiten la búsqueda de imágenes por similitud visual en grandes bases de datos. Al convertir imágenes en vectores numéricos, se pueden utilizar métricas de distancia (como la distancia euclídea o la similitud coseno) para encontrar imágenes "similares" de manera eficiente. Esto es útil en plataformas de comercio electrónico para recomendaciones de productos basadas en la apariencia y en sistemas de gestión de contenido digital para organizar y recuperar activos visuales.
- **Reconocimiento Facial:** En la seguridad y en aplicaciones móviles, los embeddings de imágenes se utilizan para el reconocimiento facial. Los sistemas de seguridad pueden identificar o verificar la identidad de individuos en cámaras de seguridad, mientras que los smartphones pueden utilizar reconocimiento facial para autenticación. Los embeddings ayudan a transformar las imágenes faciales en un espacio donde las caras del mismo individuo se agrupan juntas, facilitando la comparación y reconocimiento rápidos.
- **Clasificación Automática de Imágenes:** Los embeddings de imágenes son fundamentales para sistemas de inteligencia artificial que clasifican imágenes en categorías predefinidas. Esto es esencial para la organización automática de grandes conjuntos de datos de imágenes.

- **Mejora de la Accesibilidad en la Web:** Los embeddings pueden ser utilizados para generar descripciones automáticas de imágenes para usuarios con discapacidades visuales, mejorando la accesibilidad en la web. Al comprender el contenido visual de una imagen, los modelos de IA pueden describir imágenes en texto, que luego puede ser leído por software de asistencia, permitiendo a los usuarios comprender el contenido visual sin verlo directamente. **Orcam** es una compañía que desarrolla dispositivos de este tipo (<https://www.orcam.com/en-us/blog/smart-glasses-blind>)



Fuente: [Efficient Multi-Object Detection and Smart Navigation Using Artificial Intelligence for Visually Impaired People](#)

- **Asistentes Virtuales Mejorados:** Los asistentes o sistemas de chat, pueden beneficiarse enormemente de capacidades multimodales al integrar texto con imágenes, para ofrecer respuestas más completas y contextuales a partir de imágenes que aporte el usuario como parte del diálogo. En el ámbito del servicio al cliente, los sistemas de diálogo multimodales pueden manejar consultas complejas más eficazmente. Pueden permitir a los usuarios subir imágenes de un producto dañado y recibir instrucciones específicas paso a paso en formato de video o diagramas interactivos para ayudar en la resolución de problemas.