Universidad Nacional de San Antonio Abad del Cusco
Departamento Académico de Ing. Informática
VISION COMPUTACIONAL
Práctica Nº 10

RECONOCIMIENTO DE PATRONES
TEMPLATE MATCHING

Iván C. Medrano Valencia

1. **OBJETIVO.**

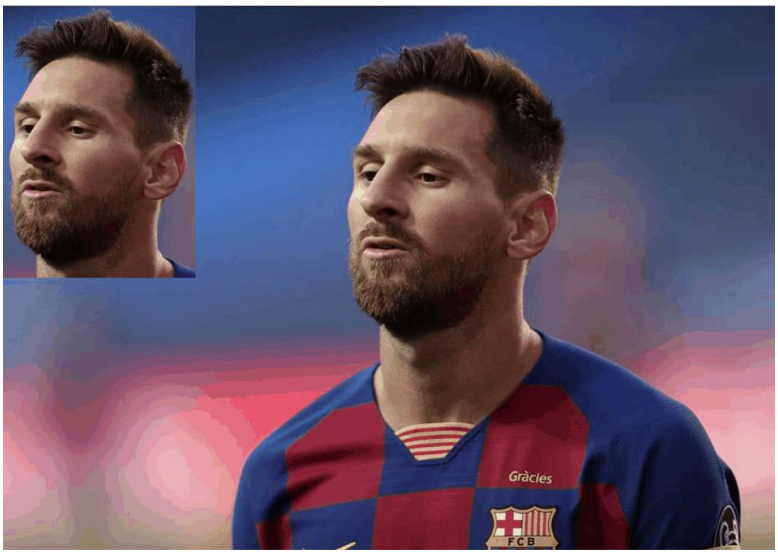
   - Conocer e implementar el algoritmo Template Matching para reconocimiento de patrones con OpenCV.

2. **INTRODUCCIÓN.**

   Template Matching es una técnica en el procesamiento de imágenes digitales que identifica las partes de una imagen que coinciden con una plantilla predefinida. Tiene varias aplicaciones y se utiliza en campos como reconocimiento facial y de voz, automatización y estimación de movimiento.

3. **TEMPLATE MATCHING.**

   En la siguiente animación GIF, podemos ver una foto de Lionel Messi. Esta es nuestra imagen de entrada. También tenemos la imagen de plantilla que es una parte recortada de la imagen de entrada. Ahora, simplemente vamos a escanear una imagen más grande con esta plantilla deslizándola por todas las posiciones posibles. Luego, compararemos una plantilla con regiones de imagen superpuestas en la imagen más grande, hasta que encontremos una coincidencia.



Para hacer esta comparación, OpenCV ha proporcionado varios métodos diferentes de coincidencia de plantillas. Aquí podemos ver las fórmulas que OpenCV ha calculado para cada método disponible. Tenga en cuenta que ***I*** denota una imagen, ***T*** la plantilla y ***R*** el resultado.

SqDiff – Squared difference

$$R(x,y) = \sum_{x',y'} \left(T\left(x',y'\right) - I\left(x+x',y+y'\right)\right)^2$$

SqDiffNormed – Normalized squared difference

$$R(x,y) = \frac{\sum_{x',y'} \left(T\left(x',y'\right) - I\left(x+x',y+y'\right)\right)^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

CCorr – Cross correlation

$$R(x,y) = \sum_{x',y'} \left(T\left(x',y'\right) \cdot I\left(x+x',y+y'\right)\right)$$

CCorrNormed – Normalized cross correlation

$$R(x,y) = \frac{\sum_{x',y'} \left(T\left(x',y'\right) - I\left(x+x',y+y'\right)\right)^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

CCoeff – Cosine coefficient

$$R(x,y) = \sum_{x',y'} \left(T'\left(x',y'\right) \cdot I'\left(x+x',y+y'\right)\right)$$

Donde:

$$T'\left(x',y'\right) = T\left(x',y'\right) - 1/(w \cdot h) \cdot \sum_{x'',y''} T\left(x'',y''\right)$$

$$I'\left(x+x',y+y'\right) = I\left(x+x',y+y'\right) - 1/(w \cdot h) \cdot \sum_{x'',y''} I\left(x+x'',y+y''\right)$$

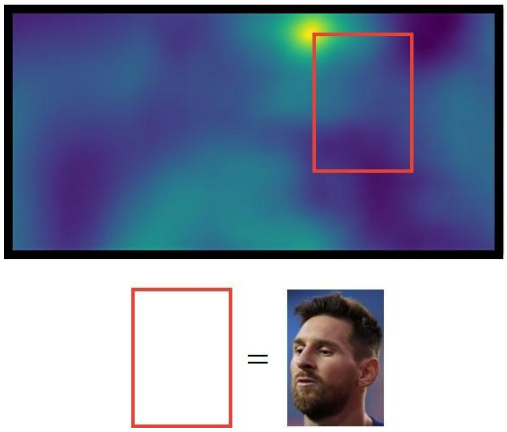
CCoeffNormed – Normalized cosine coefficient

$$R(x,y) = \frac{\sum_{x',y'} \left(T'\left(x',y'\right) \cdot I'\left(x+x',y+y'\right)\right)}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x'y'} I'(x+x',y+y')^2}}$$

Cuando deslizamos la imagen de la plantilla a través de la imagen de entrada, se calcula una métrica en cada ubicación de píxel. Esta métrica representa qué tan similar es la plantilla a esa área particular de la imagen de entrada. Para cada ubicación de ***T*** en la imagen ***I*** almacenamos la métrica en la matriz de resultados **R**. Cada ubicación **(x, y)** en **R** contiene la métrica de coincidencia. Tenga en cuenta que este proceso es muy similar a la convolución donde la salida de nuestra imagen se reducirá.

En la siguiente imagen podemos ver el mapa de los resultados de la comparación. Las ubicaciones más brillantes indican las coincidencias más altas. Como puede ver, la ubicación marcada por el círculo amarillo es probablemente la que tiene el valor más

alto, por lo que esa ubicación se considerará como la mejor candidata para nuestra plantilla. Este círculo amarillo representará la esquina superior izquierda del rectángulo, donde asumimos que se ubicará el candidato de coincidencia óptimo. El ancho y el alto del rectángulo son iguales a la imagen de la plantilla**.**



4. ACTIVIDAD I: APLICATION DE TEMPLATE MATCHING

Ejercicio 10.1

Primero, vamos a importar las bibliotecas necesarias y cargar la imagen de entrada y la imagen de la plantilla. También corregiremos el orden de los colores porque trazaremos estas imágenes con matplotlib.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

template = cv2.imread("images/template1.jpg")
template = cv2.cvtColor(template, cv2.COLOR_BGR2RGB)
img = cv2.imread("images/image1.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img)
plt.imshow(template)
```

Entonces, como puede ver, la imagen de entrada es Leo Messi y la imagen de la plantilla es solo la cara de Messi. Ahora, si miramos las dimensiones de una imagen de plantilla, podemos ver que son (269, 191, 3).

```
template.shape
```

Tenga en cuenta que si dibuja un rectángulo de tamaño (269, 191) alrededor de la cara de Messi en la imagen de entrada, esa área tendrá el tamaño y la forma exactos de una imagen de plantilla.

Ahora, usaremos la lista de cadenas que son los nombres de diferentes métodos de coincidencia de plantillas.

```
methods =["cv2.TM_CCOEFF" , "cv2.TM_CCOEFF_NORMED", "cv2.TM_CCORR" , "cv2.TM_CCORR_N
ORMED" , "cv2.TM_SQDIFF" , "cv2.TM_SQDIFF_NORMED"]
```

Sin embargo, necesitamos evaluar cada una de estas cadenas como si fuera una función OpenCV. Para hacer eso, usaremos la función eval (). De esa manera, podemos transformar directamente una cadena que coincida con cada una de estas funciones integradas de coincidencia de plantillas. Será más conveniente para nosotros escribir nuestro código de esa manera, en lugar de llamar a cada función manualmente. Ahora, sigamos adelante.

El siguiente paso es crear un bucle for que pase por cada uno de estos métodos. También crearemos una copia de la imagen de entrada. Luego, usando la función eval (), recorreremos todos los métodos de cadena y transformaremos esa cadena que vamos a usar en la función OpenCV real.
Ahora usaremos la función ***cv2.matchTemplate ()*** que consta de los siguientes parámetros.

- Input image: una imagen donde se está ejecutando la búsqueda. Tenga en cuenta que debe ser un número entero de 8 bits o un punto flotante de 32 bits.
- **Template**: imagen de plantilla buscada. Debe ser menor o igual a la imagen de entrada y debe tener el mismo tipo de datos.
- **Output**: el mapa resultante de la comparación. Es un punto flotante de 32 bits de un solo canal. Si las dimensiones de la imagen de entrada son **x × y** píxeles y las dimensiones de la plantilla son **m × n** píxeles, entonces el resultado es **(x – m + 1) × (y – n + 1).**
- **Method**: parámetro que especifica el método de comparación
- **Mask**: máscara de la plantilla buscada. Debe tener el mismo tipo de datos y tamaño que la plantilla. Opcional.

```
for m in methods:
  img_copy = img.copy()
  method = eval(m)
  res = cv2.matchTemplate(img_copy,template,method)
```

Ahora, necesitamos encontrar los valores máximo y mínimo del mapa resultante, así como las ubicaciones de los valores mínimo y máximo. Para eso, usaremos la función cv2.minMaxLoc () y como parámetro, simplemente pasaremos nuestra imagen que es la salida de nuestro método de coincidencia de plantillas. Esta función toma el mapa, encuentra las ubicaciones mínima y máxima y las devuelve como una tupla. Luego, podemos descomprimir esa tupla, para encontrar el valor mínimo, el valor máximo, así como las ubicaciones mínima y máxima.

```
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
```

Una cosa importante a tener en cuenta es que dos métodos que usan diferencias cuadradas, SqDiff y SqDiffNormed, serán ligeramente diferentes porque la ubicación con el valor mínimo se considerará la coincidencia. Por otro lado, para otros métodos, la coincidencia se ubicará donde la función encuentre el valor máximo.

Para solucionar este problema, necesitamos crear un ciclo que verifique si un método dado considera la ubicación con el valor mínimo como una coincidencia, o la ubicación con el valor máximo. Entonces diremos que, si usamos un método SqDiff o

SqDiffNormed, la esquina superior izquierda del rectángulo es igual a la ubicación mínima. Para todos los demás métodos dirá que la esquina superior izquierda del rectángulo es igual a la ubicación máxima.

```
if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
    top_left = min_loc
  else:
    top_left = max_loc
print(top_left)
```

Entonces, ahora sabemos la ubicación de la esquina superior izquierda de nuestro rectángulo. A continuación, necesitamos encontrar la esquina inferior derecha del rectángulo. Ya sabemos que el ancho y el alto del rectángulo son iguales a la imagen de la plantilla. Por lo tanto, solo necesitamos obtener la forma del rectángulo. Luego, vamos a definir la esquina inferior derecha del rectángulo que es igual a la esquina superior izquierda indexada en cero más el ancho, y luego la esquina superior izquierda indexada en uno más la altura.

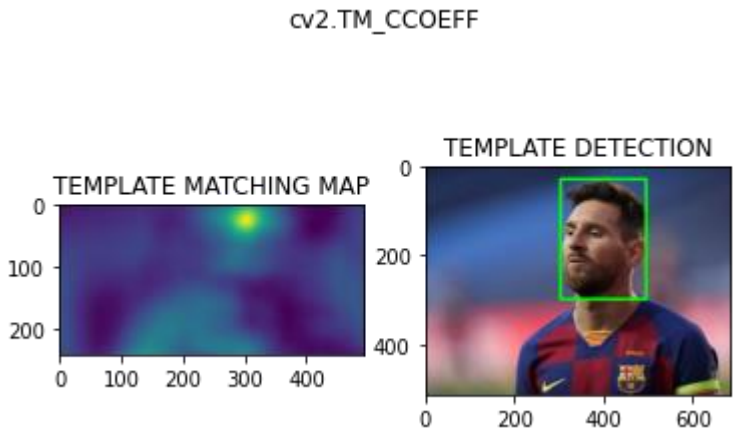
```
height, width, channels = template.shape
bottom_right = (top_left[0]+width, top_left[1]+height)
```

Ahora, solo necesitamos dibujar nuestro rectángulo con la función cv2.rectangle ().

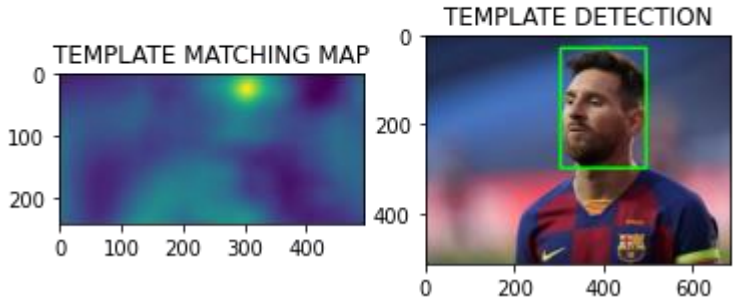
```
cv2.rectangle(img_copy, top_left, bottom_right, (0,255,0),6)
```

El paso final es graficar todos los resultados que obtuvimos.

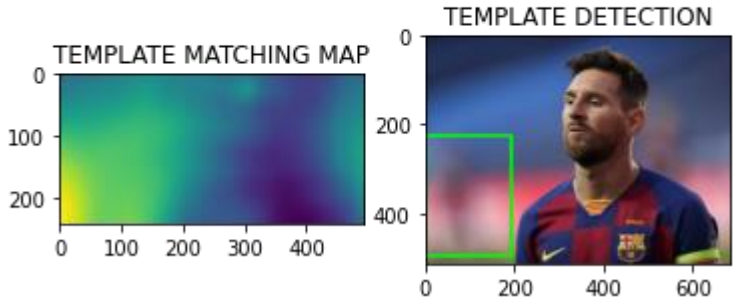
```
 plt.subplot(121)
plt.imshow(res)
plt.title("TEMPLATE MATCHING MAP")
plt.subplot(122)
plt.imshow(img_copy)
plt.title("TEMPLATE DETECTION")
plt.suptitle(m)

plt.show()
print("\n")
print("\n")


Los resultados son:
```



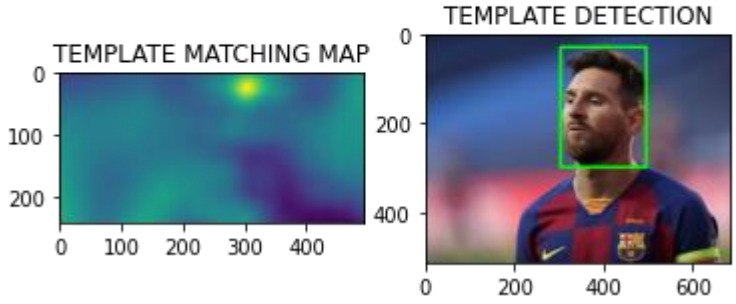
cv2.TM_CCOEFF_NORMED



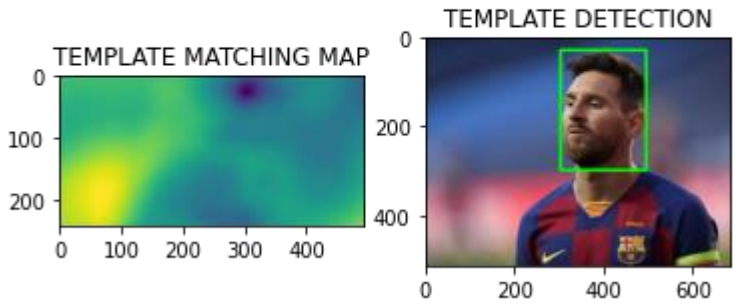
cv2.TM_CCORR



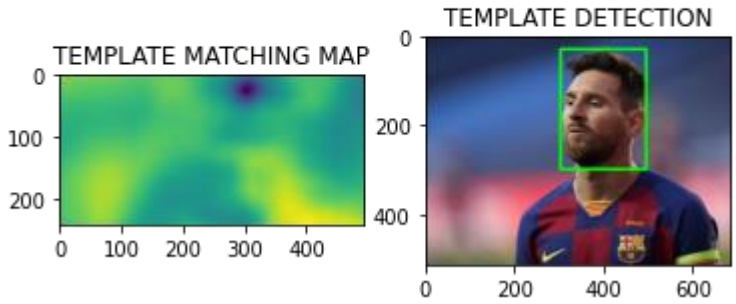
cv2.TM_CCORR_NORMED



cv2.TM_SQDIFF



cv2.TM_SQDIFF_NORMED



Ejercicio 10.2

```
import cv2
import numpy as np

img_rgb = cv2.imread('images/image2.jpg')
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

template = cv2.imread('images/template2.jpg',0)
w, h = template.shape[::-1]

res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
#--% de coindidencia
threshold = 0.8
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,255,255), 2)

cv2.imshow('Detected',img_rgb)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Ejercicio 10.3

```
# importar los paquetes necesarios
import numpy as np
import glob
```

```python
import os
import cv2
#Ruta a la plantilla
templatePath = 'template.jpeg'
#templatePath = 'task3_bonus/Bonus_1/t_1.jpg'
#templatePath = 'task3_bonus/Bonus_2/t_2.jpeg'
#templatePath = 'task3_bonus/Bonus_3/t_3.jpeg'
#ruta a la carpeta de imágenes
imageFolderPath = 'images'
#imageFolderPath = 'task3_bonus/Bonus_1'
#imageFolderPath = 'task3_bonus/Bonus_2'
#imageFolderPath = 'task3_bonus/Bonus_3'

def resize(image, width = None, height = None, inter = cv2.INTER_AREA):
    # inicializar las dimensiones de la imagen que se cambiará de tamaño y
    # toma el tamaño de la imagen
    dim = None
    (h, w) = image.shape[:2]

    # si tanto el ancho como el alto son Ninguno, entonces devuelve la
    # imagen original
    if width is None and height is None:
        return image

    # compruebe si el ancho es None
    if width is None:
        # calcular la relación de la altura y construir las
        # dimensiones
        r = height / float(h)
        dim = (int(w * r), height)

    # de lo contrario, la altura es None
    else:
        # calcular la relación del ancho y construir las
        # dimensiones
        r = width / float(w)
        dim = (width, int(h * r))

    # cambiar el tamaño de la imagen
    resized = cv2.resize(image, dim, interpolation = inter)

    # devolver la imagen redimensionada
    return resized

#imagen de plantilla de lectura


template = cv2.imread(templatePath,0)

#aplicando transformación laplaciana a la plantilla
template = cv2.Laplacian(template,cv2.CV_64F)
template = np.float32(template)
#th y tw son la altura y el ancho de la plantilla
(tH, tW) = template.shape[:2]
i=0
# recorre las imágenes para encontrar la plantilla
for imagePath in glob.glob(imageFolderPath + "/*.jpg"):
    i+=1
    #lee la imagen
    image = cv2.imread(imagePath)

    #convierte la imagen a escala de grises
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #aplica desenfoque gaussiano con un tamaño de kernel de 3 en la imagen
    blur = cv2.GaussianBlur(gray,(3,3),0)
    gray = cv2.Laplacian(blur,cv2.CV_64F)
    gray = np.float32(gray)
    found = None
    pos=0
    # bucle sobre las escalas de la imagen
    for scale in np.linspace(0.5, 2, 30):
        # cambiar el tamaño de la imagen de acuerdo con la escala
        # y realizar un seguimiento de la proporción del cambio de tamaño

        resized = resize(gray, width = int(gray.shape[1] * scale))
```

```
            r = gray.shape[1] / float(resized.shape[1])

            # si la imagen redimensionada es más pequeña que la plantilla, salir del bucle
            if resized.shape[0] < tH or resized.shape[1] < tW:
                break

            # coincidencia para encontrar la plantilla en la imagen edged = cv2.Canny
            # (redimensionado, 50, 200)
            result = cv2.matchTemplate(resized, template, cv2.TM_CCOEFF)
            (_, maxVal, _, maxLoc) = cv2.minMaxLoc(result)

            # si hemos encontrado un nuevo valor máximo de correlación,
            # entonces actualice encontrado
            if found is None or maxVal > found[0]:
                found = (maxVal, maxLoc, r)
                (startX, startY) = (int(maxLoc[0] * r), int(maxLoc[1] * r))
                (endX, endY) = (int((maxLoc[0] + tW) * r), int((maxLoc[1] + tH) * r))


    # descomprime la variable de contabilidad y calcula las coordenadas (x, y)
    # del cuadro delimitador en función de la proporción redimensionada
    if found is not None:#and pos>5:
        # dibujar un cuadro delimitador alrededor del resultado detectado y mostrar la imagen
        (maxVal, maxLoc, r) = found
        if(maxVal>350000):
            print(imagePath,'maxval=',maxVal,'Cursor detected at location:',(int(maxLoc[0] *
r), int(maxLoc[1] * r)))
            (startX, startY) = (int(maxLoc[0] * r), int(maxLoc[1] * r))
            (endX, endY) = (int((maxLoc[0] + tW) * r), int((maxLoc[1] + tH) * r))
            cv2.rectangle(image, (startX, startY), (endX, endY), (0, 0, 255), 2)
        else:
            print(imagePath,'Cursor not detected')


    cv2.imshow(imagePath, image)
    cv2.waitKey(0)
```

5. TAREA.

Analizar el algoritmo Template Matching cuando la imagen cambia de tamaño o cuando es rotada o oscurecida. ¿Cuál de los métodos se desempeña mejor?

6. REFERENCIAS BIBLIOGRÁFICAS

- Dawson-Howe, K. (2014). A Practical Introduction to Computer Vision With OpenCV, Wiley & Sons Ltd.
- Hafsa Asad, W. R., Nikhil Singh (2020). The Computer Vision Workshop. Birmingham U.K., Pack Publishing.
- Szeliski, R. (2011). Computer Vision Algorithms and Applications. London, Springer.