UNIVERSIDAD EUROPEA DEL ATLÁNTICO

GRADO EN
INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE ASIGNATURA

INTELIGENCIA ARTIFICIAL

Trabajo realizado por

NICOLÁS CALVACHE JIMÉNEZ

Índice

Problemática presentada

Para el trabajo de fin de asignatura se nos pone como problemática el desarrollar un programa de inteligencia artificial para detectar a partir de una imagen los siguientes:

Funcionalidades principales

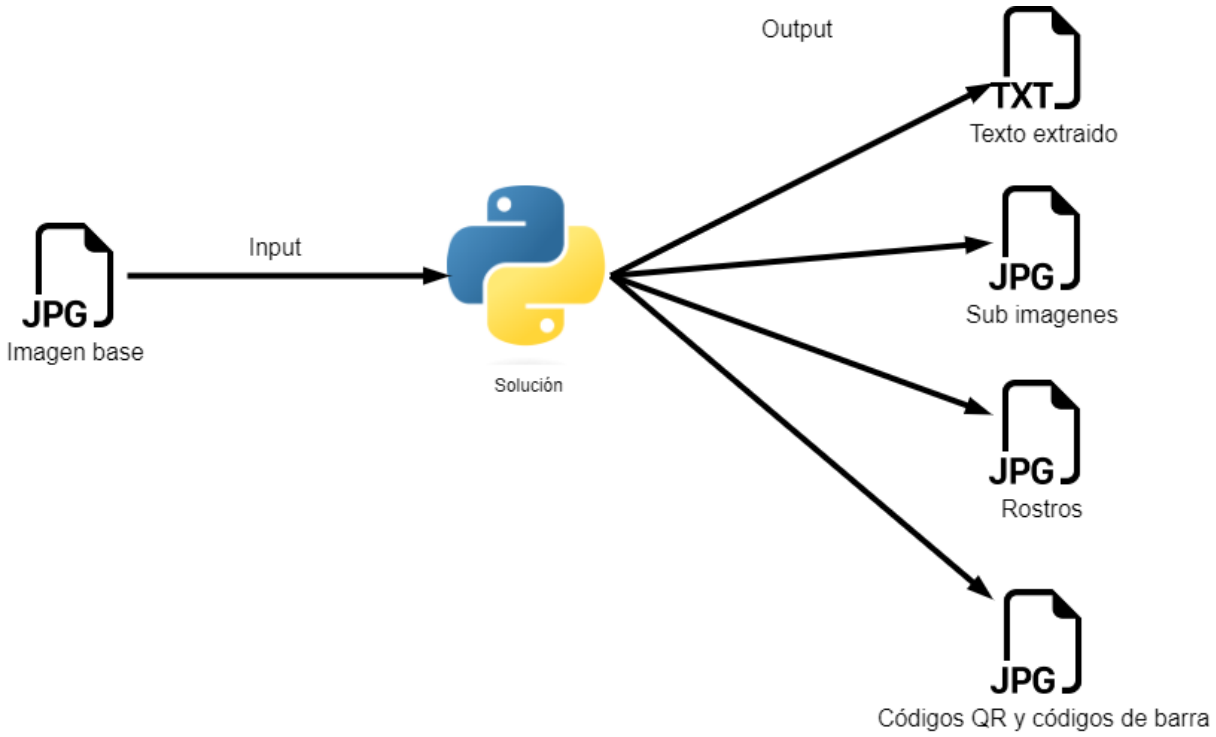
- Texto digital
- Texto a mano

Funcionalidades secundarias

- Imágenes
- Tablas
- Códigos de barra/QR
- Características faciales

Solución propuesta

Como solución, primeramente se propone desarrollar la herramienta en el lenguaje de programación Python. Esto debido a su extensa librería de paquetes open source dedicados a la inteligencia artificial y la simplicidad que nos ofrece este lenguaje a la hora de desarrollar soluciones.
Como entrada al sistema, sé requiere una imagen base en formato PNG/JPG, que al pasar por la solución exporta 4 tipos de características extraídas de la imagen base, sub imágenes, sub imágenes que contienen rostros, códigos QR/Barra y el texto extraído.
En primer lugar, se toma la imagen base desde el directorio de datos, modificando la ruta en el parámetro designado. Luego, se obtienen distintas muestras de la misma imagen con filtros (Escala de grises, desenfoque gaussiano y tresh gausiano) con el fin de ser utilizada por los distintos módulos de extracción como lo son el de texto, codigos de barra y qr y sub imagenes clasificándolas por si contienen rostros o no.
Finalmente, los outputs detectados se escriben en disco en el directorio de output.



Flujo


JPG
Preparación de entorno
Eliminar output anterior
Procesamiento de Imagen
Extracción de QR y Códigos de Barra
Extracción de Texto
Extracción de Imágenes
Fin de flujo

Desarrollo de solución

Requisitos

Para ejecutar la herramienta desarrollada se requiere tener los siguientes:

- Windows 10 o mayor.
- Python 3.10.8 o cualquier versión 3.10.
- Google Tesseract OCR, link a instalador de Windows.
- Visual C++ Redistributable Packages for Visual Studio 2013, requerido en el caso de Windows para la libreria pyzbar.

Seguido de las siguientes librerías que se pueden obtener por medio del comando `pip install`

- pytesseract (`pip install pytesseract`)
- cv2 (`pip install opencv-python`)
- pyzbar (`pip install pyzbar`)
- PIL (`pip install Pillow`)

Directorio de la solución

- data (Casos de prueba para la solución)
  - Kernels (Kernels utilizados para detectar caras y expresiones)
- res (Directorio output)
  - faces (Directorio output de imagenes que contienen rostros)
  - images (Directorio output de imagenes detectadas)
  - qr (Directorio output de codigos de barra y QR)
  - lectura_imagen.txt (Fichero conteniendo el texto extraído de la imagen)
- main.ipynb (Fichero de la solución en formato Jupyter notebook)
- main.py (Fichero de la solución en formato Python)

Preparación de entorno

En esta sección se importan todas las librerías y además se declaran las funciones generales que más adelante serán utilizadas.

```
# Importar librerías necesarias
import pytesseract
import cv2
import pyzbar.pyzbar as pyzbar
import os
```

```
# Funcion que encapsula la librería pytesseract que extrae el texto
digital o manual a un string
def image_to_string(image, lang="eng") -> str:
    return pytesseract.image_to_string(image, lang=lang)
```

```
# Funcion para escribir archivos
def save_to_file(filename: str, content: str) -> None:
    with open(filename, "w") as f:
        f.write(content)
```

```
# Funcion para almacenar cualquier tipo de imagen
def save_image(filename: str, image) -> None:
    cv2.imwrite(filename, image)
```

```
# Funcion para eliminar todo contenido de una carpeta en particular
def delete_folder_content(folder_path):
    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path, filename)
        try:
            if os.path.isfile(file_path) or os.path.islink(file_path):
                os.unlink(file_path)
        except Exception as e:
            print('Failed to delete %s. Reason: %s' % (file_path, e))
```

Eliminar output anterior

Para este apartado eliminamos el output anterior para realizar una nueva ejecución.

```
# Por cada folder de output ejecutar lo siguiente
for folder_path in ["res", "res/faces", "res/images", "res/qr/"]:
    # En base a la ruta eliminar el contenido
    delete_folder_content(folder_path)
```

Procesamiento de Imagen

Ahora pasamos a detectar la imagen, extraerla y pasarla por distintos filtros que nos ayudaran a hacer una mejor detección de los elementos requeridos.

```
# Constante que define la imagen a tratar
FILENAME = "data/....jpg"
```

```
# Lectura de la imagen base
image = cv2.imread(FILENAME)

# Escritura en disco del output de la imagen base
save_image("res/0_image_inicial.jpg", image)
```

```
# Procesar la imagen base a escala de grises
# (Esto permite eliminar información innecesaria)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Escritura en disco del output de la imagen base a escala de grises
save_image("res/1_imagen_escala_gris.jpg", gray)
```

```
# Procesar la imagen a escala de grises con un desenfoque gausiano
# (Esto permite suavizar la imagen y eliminar información innecesaria)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Escritura en disco del output de la imagen a escala de grises con
desenfoque
save_image("res/2_imagen_desenfocada.jpg", blurred)
```

```
# Aplicamos un flitro para dejar solamente contornos
thresh = cv2.adaptiveThreshold(
    blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11,
2)

# Escritura en disco del output de la imagen con el filtro nuevo
save_image("res/4_imagen_tresh.jpg", thresh)
```

Extracción de QR

Esta sección se enfoca exclusivamente en la detección de codigos qr y de barra, haciendo uso de la libreria pyzbar

```
# Detectar los codigos QR con la libreria pyzbar
codes = pyzbar.decode(blurred)

# Ejecutar lo siguiente por cada codigo encontrado
for code in codes:
    # Obtenemos sus dimensiones
    x, y, w, h = code.rect

    # Con las dimensiones obtenemos el codigo qr y lo guardamos en disco
en el output
    save_image("res/qr/{}.jpg".format(codes.index(code)),
               blurred[y:y+h, x:x+w])

    # De la imagen desenfocada eliminamos el codigo qr quitar
información innecesaria
    blurred[y:y+h, x:x+w] = 255
```

```
# Escritura en disco del output de la imagen a escala de grises con
desenfoque sin codigos qr
save_image("res/3_imagen_sin_qrs.jpg", blurred)
```

Extracción de Texto

```
# Detectar cualquier texto presente en la imagen a escala de grises y
desenfocada
save_to_file("res/lectura_imagen.txt", image_to_string(blurred))
```

Extracción de Imágenes

```
# Obtenemos todos los contornos presentes en la imagen
contours, _ = cv2.findContours(
    cv2.Canny(thresh, 50, 200), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
# Cargamos de disco los kernels para detectar caras y ojos
face_cascade = cv2.CascadeClassifier(
    'data/kernels/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(
    'data/kernels/haarcascade_eye.xml')
```

```
# Valores minimos para detectar imagenes
w_min = 200
h_min = 200

# Por cada contorno ejecutar lo siguiente
for i, cnt in enumerate(contours):
    # Obtener las dimensiones del contorno
    x, y, w, h = cv2.boundingRect(cnt)

    # Determinamos las coordenadas del ROI (Region of interest)
    left = x
    top = y
    right = x + w
    bottom = y + h

    # Obtenemos el ROI (Region of interest) a partir de la imagen base
    roi = blurred[top:bottom, left:right]
    roi_original = image[top:bottom, left:right]

    # Flag que nos ayudará al finalizar a determinar si la imagen
contiene una cara o no
```

```
    face = False

    # Detectamos todas las posibles caras
    faces = face_cascade.detectMultiScale(roi)

    # Detectar el incremento necesario
    increase = 1 if w > w_min and h > h_min else 3

    # incrementar la imagen en caso de ser muy pequeña
    roi = cv2.resize(
        roi, None, fx=increase, fy=increase,
interpolation=cv2.INTER_LINEAR)
    roi_original = cv2.resize(
        roi_original, None, fx=increase, fy=increase,
interpolation=cv2.INTER_LINEAR)
    # Esta es una validación que nos ayuda a descartar contornos
demasiado pequeños que no son una imagen
    if (w >= w_min and h >= h_min) or len(faces) > 0:
        # Por cada cara ejecutar lo siguiente
        for (x, y, w, h) in faces:
            # Incrementamos las coordenadas para actuar
            x *= increase
            y *= increase
            w *= increase
            h *= increase

            # Pintar un rectangulo en sus coordenadas
            cv2.rectangle(roi_original, (x, y), (x + w, y + h), (255, 0,
0), 2)

            # Detectamos todas los posibles ojos
            eyes = eye_cascade.detectMultiScale(roi[y:y+h, x:x+w])
            # Por cada par de ojos ejecutar lo siguiente
            for (ex, ey, ew, eh) in eyes:
                cv2.rectangle(roi_original[y:y+h, x:x+w], (ex, ey),
                              (ex+ew, ey+eh), (0, 255, 0), 2)

            # Determinamos que si hay caras presentes
            face = True

        # Guardamos en disco los ROI determinando si son imagenes o si
contienen caras
        save_image("res/faces/roi_{}.jpg".format(i)
                   if face else "res/images/roi_{}.jpg".format(i),
roi_original)
```

Uso de la herramienta

Ver video.

Limitaciones de uso

- Las sub imagenes deben ser de un tamaño mayor o igual a 200x200 píxeles.
- Las sub imagenes deben de tener un color de fondo distinto al color de la imagen base.
- El texto no debe estar pixelado, es decir que debe tener una resolución decente
- No todas las tipografías son compatibles.
- La imagen debe ser PNG o JPG.
- Las caras deben estar mirando hacia al frente.
- El texto a mano debe ser legible.

Conclusiones

En conclusión, la detección de sub imágenes, rostros, texto y códigos QR/Barras utilizando Python es una tarea compleja pero alcanzable. Empleando librerías especializadas como OpenCV y PyTesseract, se pueden implementar soluciones eficientes para estas tareas de detección. Asimismo, usando una combinación de técnicas de procesamiento de imágenes y aprendizaje automático, se pueden mejorar aún más la precisión y la velocidad de estas soluciones.
Es importante destacar que, aunque se ha logrado un buen nivel de precisión en estas tareas de detección, todavía hay espacio para mejoras y optimizaciones adicionales. Además, es crucial seguir investigando y actualizando las librerías y técnicas empleadas para asegurar que estas soluciones se mantengan relevantes y eficaces en el futuro.

Posibles mejoras

Se logró implementar las 2 funcionalidades principales y 3 de las 4 secundarias, únicamente ha faltado el detectar las tablas, por lo tanto, se toma como una posible mejora a futuro el detectar tablas y exportarlas en un formato determinado.
Además, dentro de cada funcionalidad se encuentran distintas mejoras como:

- Detección imagenes
    - A veces detecta la misma imagen debido a contornos cercanos generando un duplicado de este
- Detección rostros
    - Debido a la duplicidad de imagenes por contornos cercanos, se exportan duplicados de imagenes que contienen rostros
    - Agregar kernel de rostros en distintas posiciones
- Detección de texto a mano
    - Siempre se puede agregar más entrenamiento a la herramienta utilizada para mejorar la detección de distintos tipos letra.

Por último, siempre se tiene cabida para realizar detecciones más eficaces y de mayor potencia.