UA

Universidad

de Alicante

ESCUELA POLITÉCNICA SUPERIOR

Servicios de Computación en la Nube

Curso académico 2024 / 2025

Semestre 2

Grado en Ingeniería en Inteligencia Artificial

Grupo 1

Profesora: Ricardo Moreno Rodríguez

Alumno: Santiago Álvarez Geanta

Universitat d'Alacant
Universidad de Alicante

Mayo 2025

ÍNDICE

Introducción

El objetivo de esta práctica es diseñar y evaluar una solución completa que aproveche webcams públicas, modelos de inteligencia artificial (YOLOv8) y diferentes estrategias de procesamiento en la nube para detectar presencia en zonas urbanas y tomar decisiones sobre la iluminación. La práctica se estructura en cinco partes, desde el desarrollo local hasta la externalización total del procesamiento.

1. Trabajo Local

1.1. Identificación de Webcam Pública

Para realización de la práctica he seleccionado la siguiente webcam pública para realizar el seguimiento de una vía con flujo peatonal y vehicular casi constante:

[Webcam Enlace]

1.2. Obtención de frames

Con la cámara seleccionada, desarrollé un pequeño script en **Python** que capturaba un frame cada cierto tiempo. Aunque parecía una tarea sencilla, no estuvo exenta de desafíos: asegurar la conexión estable con el directo de **YouTube**. Teniendo que utilizar herramientas como **pytube**, **yt-dlp y ffmpeg** para obtener el enlace directo a la transmisión utilizando:

```
yt-dlp -g https://www.youtube.com/watch?v=Rg2QZ5zYukQ
```

Obteniendo como resultado el siguiente enlace para la transmisión:

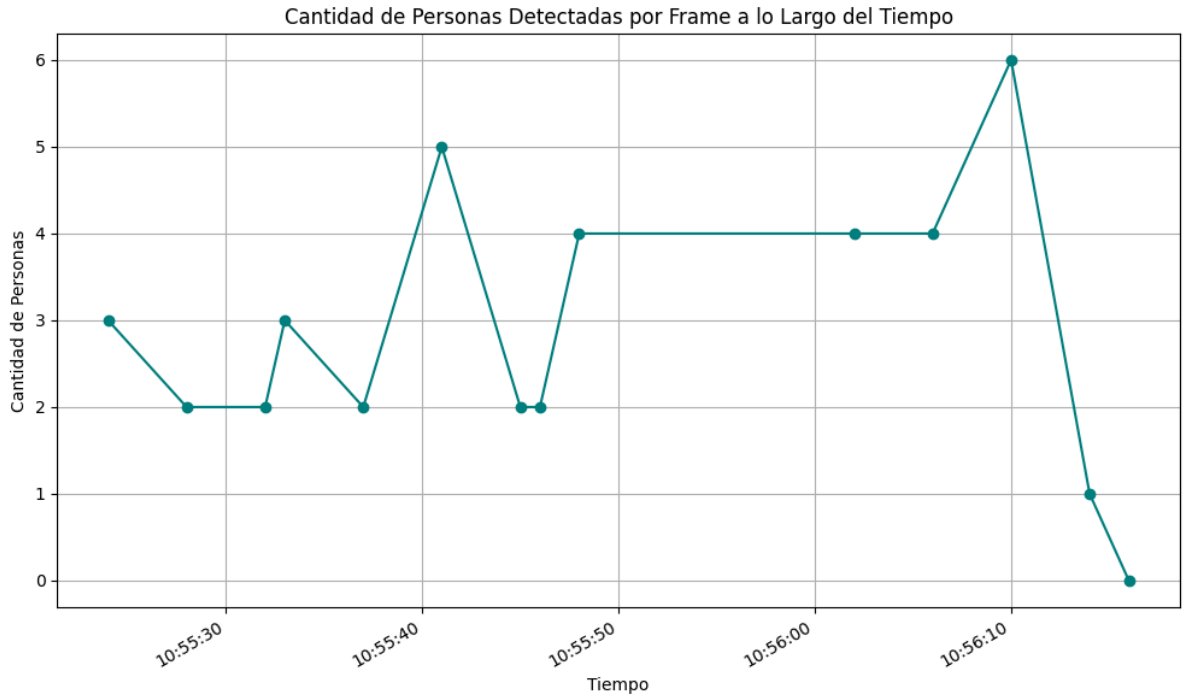
```
# URL obtenida mediante 'yt-dlp -g'
stream_url = "https://manifest.googlevideo.com/api/manifest/hls_playlist/expire/1747666307/" \
"ei/I_EqaJiyCu7YxN8P17nemAs/ip/92.190.147.110/id/Rg2QZ5zYukQ.1/itag/96/source/yt_live_broadcast/" \
"requiressl/yes/ratebypass/yes/live/1/sgoap/gir%3Dyes%3Bitag%3D140/sgovp/gir%3Dyes%3Bitag%3D137/" \
"rqh/1/hls_chunk_host/rr2---sn-w51luxa-5ajl.googlevideo.com/xpc/EgVo2aDSNQ%3D%3D/playlist_duration/" \
"30/manifest_duration/30/bui/AecWEAYEvh5KPHlHA8ss3lKYQaibJL_p8w9VMNMY0nRFplzcDjzQe4Av-qVI2_Rwjl79M_ptc0GoGw4E/" \
"spc/wk1kZrf3y4GCYDT6CPz3idg-s9CKiiqrn_Cs40wyBXEmTwAMY92K7K_jfdiR4ALNCGuiATE/vprv/1/playlist_type/DVR/initcwndbps/" \
"3411250/met/1747644708,/mh/aQ/mm/44/mn/sn-w51luxa-5ajl/ms/lva/mv/m/mvi/2/pl/23/rms/lva,lva/dover/11/pacing/0/keepalive/" \
"yes/fexp/51355912,51466698/mt/1747644268/sparams/expire,ei,ip,id,itag,source,requiressl,ratebypass,live,sgoap,sgovp,rqh,xpc," \
"playlist_duration,manifest_duration,bui,spc,vprv,playlist_type/sig/AJfQdSswRQIgauhbq8hewUnb2CsqHTV-tHuPNNOt5SE7QLTn-" \
"bElnSECIQCUpb-yzNXTzr_Px3ZiGrw_FADo4ZlyV34F3f3Co9pUdg%3D%3D/lsparams/hls_chunk_host,initcwndbps,met,mh,mm,mn,ms,mv,mvi," \
"pl,rms/lsig/ACuhMU0wRQIgGFTMvJ-uZiIobFZdLa46BeZUsxYkb3D_OKU4oJ8Cu2kCIQDjEQpRH3oMj6_49QXEtiplTukL68vtsKm3EZFBjSX0Eg%3D%3D" \
"/playlist/index.m3u8"
```

Finalmente, logré capturar imágenes listas para el análisis a partir del siguiente código:

```python
cap = cv2.VideoCapture(stream_url)

if not cap.isOpened():
    print("No se pudo abrir el stream.")
    exit()

while True:
    ret, frame = cap.read()
    if not ret:
        print("No se pudo leer un frame.")
        continue
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"frames/frame_{timestamp}.jpg"
    #cv2.imwrite(filename, frame)
    print(f"Guardado: {filename}")
    #time.sleep(random.randint())
```

1.3. Aplicación del modelo YOLOv8

Integré el modelo **YOLOv8** en mi entorno local para que, a partir de cada frame, pudiera identificar personas automáticamente. Los resultados se registran en un archivo CSV junto a su fecha y hora. Para visualizar mejor la actividad en la escena, generé una gráfica que mostraba cómo fluctúa la presencia de personas con el paso del tiempo.

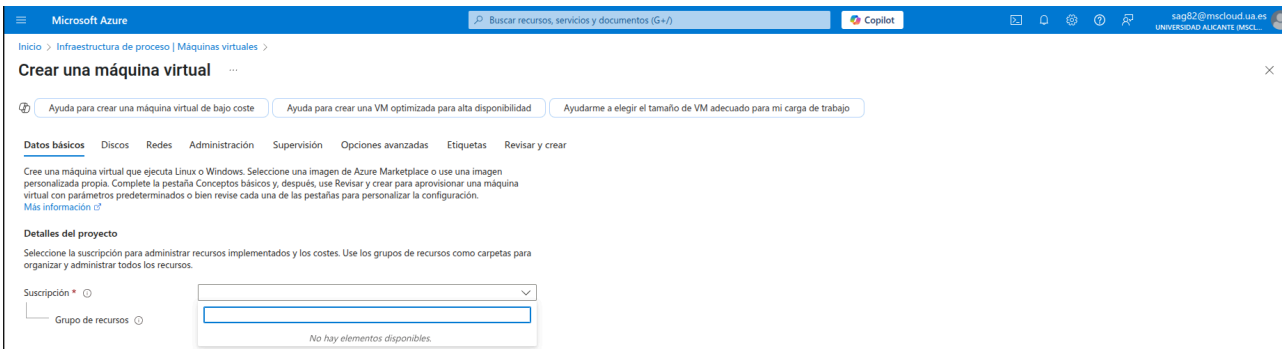


2. Subiendo a la nube: Procesamiento en Máquina Virtual

Durante el desarrollo de esta parte de la práctica, cuyo objetivo principal era crear una máquina virtual (VM) en la nube de Microsoft Azure para externalizar el procesamiento de imágenes, no fue posible completar las tareas propuestas debido a limitaciones técnicas relacionadas con el acceso a los recursos de Azure.

Específicamente, al intentar c**rear la máquina virtua**l requerida, el sistema solicitaba la selección de una suscripción activa, sin la cual no es posible aprovisionar servicios en la plataforma. Sin embargo, a pesar de estar utilizando una cuenta institucional del tipo **@mscloud.ua.es**, no se encontraba disponible ninguna suscripción asignada para la creación de recursos. Esta situación me impidió continuar con el despliegue de la infraestructura necesaria.

El mismo problema



3. Serverless

3.1 Modelo Serverless en la nube

Desarrollé una aplicación serverless utilizando **Azure Functions** con Python para procesar frames enviados a través de peticiones HTTP. Este enfoque permite delegar el procesamiento en la nube **sin necesidad de gestionar infraestructura dedicada**, beneficiándose de la escalabilidad automática

Creé un nuevo proyecto Azure Functions con soporte para Python:.

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P1optativa$ func init MiFuncionServerless --python
Found Python version 3.12.3 (python3).
The new Python programming model is generally available. Learn more at https://aka.ms/pythonprogrammingmodel
Writing requirements.txt
Writing function_app.py
Writing .gitignore
Writing host.json
Writing local.settings.json
Writing /home/sant_vz6/Escritorio/UA-2/2 CUATRI/CAR/P1optativa/MiFuncionServerless/.vscode/extensions.json
```

También añadí una función llamada **procesar_frame** que responde a peticiones HTTP:

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P1optativa/MiFuncionServerless$ func new --name procesar_frame --template "HTTP trigger"
Appending to /home/sant_vz6/Escritorio/UA-2/2 CUATRI/CAR/P1optativa/MiFuncionServerless/function_app.py
The function "procesar_frame" was created successfully from the "HTTP trigger" template.
```

La función recibe datos JSON con el frame, procesa (simulado) y responde con un mensaje JSON indicando el resultado. Medí el tiempo de respuesta para monitorizar la latencia de la función.

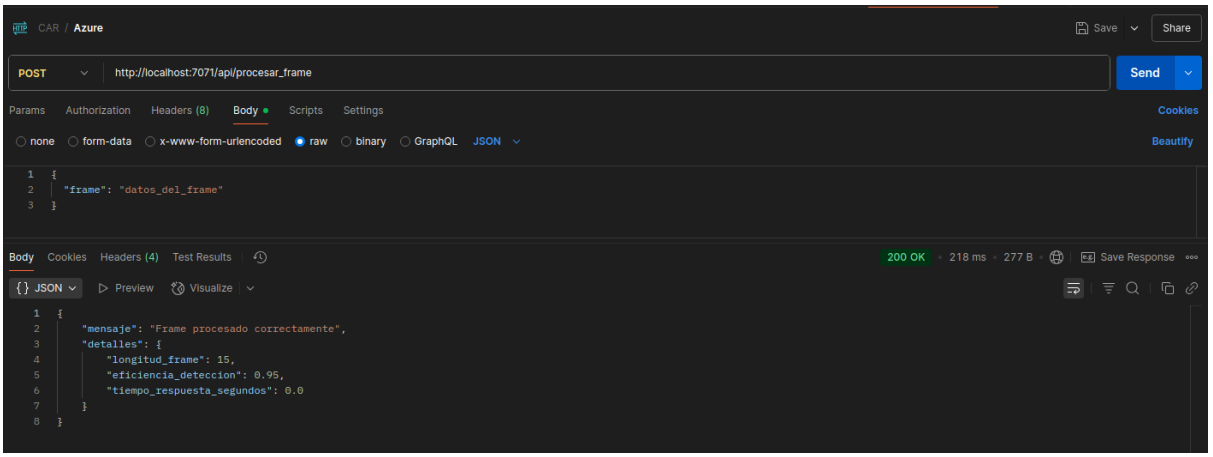
```python
app = func.FunctionApp()

@app.route(route="procesar_frame")
def procesar_frame(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Procesando frame recibido.')

    start_time = time.time()

    try:
        req_body = req.get_json()
        frame_data = req_body.get('frame')
        if not frame_data:
            return func.HttpResponse("No se ha recibido ningún frame en la petición.", status_code=400)

        # NOTE
        # Aquí iría el procesamiento real del frame, por ejemplo detección.
        # Para simular, calculamos una "eficiencia" ficticia.
        longitud_frame = len(frame_data)
        eficiencia = 0.95

        elapsed_time = time.time() - start_time

        # Resultado con métricas
        resultado = {
            "mensaje": "Frame procesado correctamente",
            "detalles": {
                "longitud_frame": longitud_frame,
                "eficiencia_deteccion": eficiencia,
                "tiempo_respuesta_segundos": round(elapsed_time, 4)
            }
        }

        logging.info(f"Procesado frame: eficiencia={eficiencia}, tiempo={elapsed_time:.4f}s")

        return func.HttpResponse(json.dumps(resultado), status_code=200, mimetype="application/json")

    except Exception as e:
        logging.error(f"Error procesando frame: {e}")
        return func.HttpResponse("Error procesando la petición.", status_code=500)
```

3.2. Rendimiento

Medición del tiempo de respuesta
Se registró el tiempo desde el inicio de la función hasta finalizar el procesamiento simulado. El tiempo de respuesta promedio observado fue útil para validar la eficiencia y escalabilidad de la función.

Eficiencia en la detección
En este ejemplo se simula una eficiencia del 95% (eficiencia_deteccion: 0.95). En un caso real, la función incluiría lógica de detección y calcularía métricas reales basadas en el análisis del frame.

Resultados obtenidos:
Mediante POSTMAN pude realizar una petición POST a nuestro endpoint implementado mediante AZURE.



4. IA como servicio: simplificando con Azure Computer Vision

Al igual que en la segunda parte, la limitación de Microsoft Azure no me deja continuar con el desarrollo de esta cuarta parte.

5. Escalando la ciudad digital

Debido a la ausencia de una suscripción Cloud para desplegar y probar la aplicación en la nube, junto con las dificultades técnicas para obtener acceso a varias cámaras consecutivas unas a otras, no fue posible completar completamente las tareas 5.2 y 5.3 del proyecto.

No obstante, avancé en la implementación del sistema de gestión local para múltiples cámaras, que en un entorno con acceso a cámaras adecuadas y una infraestructura Cloud disponible, permitiría cumplir con los objetivos planteados.

6. Conclusiones

Esta práctica fue mucho más que una simple tarea técnica: fue una exploración real de cómo construir sistemas inteligentes distribuidos usando recursos locales y en la nube. Aprendí a balancear costes, rendimiento, escalabilidad y simplicidad, y descubrí el valor real de externalizar tareas complejas a plataformas especializadas.