Proyecto final

Gael Vivanco Campos

Benemérita Universidad Autónoma de Puebla, Puebla Pue 72000, México
Puebla, Av. San Claudio S/N, 72570 Puebla, México
www.buap.mx

Abstract. Este artículo presenta el diseño e implementación de una aplicación de visión por computadora para el monitoreo automatizado de exámenes en línea. El sistema integra detección de objetos en tiempo real para identificar dispositivos prohibidos (celulares) y análisis biométrico facial para determinar la atención del estudiante mediante la estimación de la pose de la cabeza. Adicionalmente, se implementa un monitoreo de eventos del sistema operativo para detectar cambios de ventana o pérdida de foco en la aplicación. El desarrollo se realiza en Python utilizando una arquitectura basada en componentes, integrando las librerías OpenCV, Ultralytics YOLO y MediaPipe, siguiendo los principios de seguimiento de objetos descritos en la literatura especializada.

Keywords: Visión por Computadora, YOLO, MediaPipe, Proctoring, Detección de Atención.

1 Introducción

Al hacer pruebas online muchas veces no se sabe si la prueba es tomada de manera correcta, los participantes pueden no hacerlo correctamente, el monitoreo de cada uno de ellos lleva tiempo, dinero y esfuerzo, para solucionar esto el procesamiento de imágenes y la visión por computadora dan herramientas que funcionan de maravilla para estos casos. El sistema contempla los problemas de la atención, como voltear hacia los lados o cambiar de pestaña, también toma en cuenta el uso de celulares, al finalizar muestras métricas que muestran el comportamiento captado por el modelo.

2 Fundamentos

2.1 Geometría facial y estimación de pose

Para determinar hacia dónde mira el usuario se utiliza un modelo de malla facial (Face Mesh), este modelo este compuesto por 2 modelos neuronales uno que detecta el rostro BlazeFace y el Modelo de Malla Facial.
El primero es un modelo ligero para detectar rostros, el modelo localiza, recorta y la pasa al siguiente paso, el modelo de malla facial que proyecta 468 puntos (landmarks) sobre el rostro. Analizando los puntos clave: nariz, barbilla, frente y pómulos, se pueden

calcular vectores que indican la rotación (yaw) y la inclinación (pitch) de la cabeza sin necesidad de hardware de calibración costoso.

Para mantener la eficiencia en tiempo real, el detector solo se ejecuta en el primer fotograma o cuando el sistema pierde el rastro del rostro, para fotogramas posteriores el modelo predice la posición que sigue. Por ello en algunos modelos cuando te mueves mucho el modelo se ve más lento.

2.2 Detección de cambio de ventana

La detección de cambio de ventana se realiza mediante la monotorizacion de la cola de mensaje del sistema, los sistemas de ventanas como DWM en Windows solo otorgan el foco a una ventana la cual es la única que recibe eventos de teclado. La aplicación obtiene las señales de focusin y focusout para poder saber que la ventana del examen cambio.

2.3 Detección de objetos

Para este sistema, se utiliza el modelo YOLOv8 (You Only Look Once), una red neuronal convolucional que permite inferencia en tiempo real. A diferencia de técnicas clásicas de sustracción de fondo, YOLO permite clasificar semánticamente el objeto, permitiéndonos filtrar específicamente la clase "Teléfono Celular" (Clase 67 en COCO Dataset) e ignorar otras presencias como personas adicionales, cumpliendo con los requisitos de privacidad del examen.

El funcionamiento del modelo se desglosa en 3 componentes:

Blackbone: Es una Red Neuronal Convolucional que actúa como extractora de características. Reduce la resolución espacial de la imagen mientras aumenta la profundidad de los canales, extrayendo patrones complejos (bordes, texturas, formas de celulares).

Neck: Utiliza una Red Piramidal de Características (FPN). Su función es mezclar las características de diferentes escalas. Esto es crucial para detectar celulares tanto si están muy cerca de la cámara (grandes) como si están lejos (pequeños).

Head: La capa de salida predice simultáneamente tres tensores:

Clasificación: Probabilidad de que el objeto pertenezca a la clase k, k=67 para "Cell Phone".

Regresión de Caja: Coordenadas del centro (x, y), ancho y alto (w, h) del objeto.

Objectness: La probabilidad de que exista un objeto en esa región.

Para finalizar YOLO divide la imagen de entrada en una cuadrícula SxS. Si el centro de un celular cae en una celda específica, esa celda es responsable de detectarlo. Finalmente, se aplica el algoritmo de **Supresión de No-Máximos (NMS)** para eliminar detecciones duplicadas, dejando solo el recuadro con mayor probabilidad de confianza.

3 Metodología

3.1 Librerías

3.1.1 Tkinter

Con esta librería podemos hacer interfaces
Mandamos a llamar a root, es el objeto que tkinter usa para la base.
Los widgets son elementos que van encima como botones, etiquetas, lienzos.
Acomodadores, sin estos no se ven los objetos, existen pack(), grid(), place().
El bucle hace que lo que hagas se ejecute en loop.
Root -> frame->botón

3.1.2 OpenCV

Es la biblioteca estándar para la visión de computadora viene por defecto en el lenguaje Python.
De esta librería utilizaremos funciones como:
cv2.VideoCapture(0), cv2.rectangle, cv2.putText, cv2.cvtColor, etc.

3.1.3 Pillow

Esta librería también es para el procesamiento de imágenes, pero es más sencilla y es usada para traducir los array de OpenCV a un objeto que pueda mostrar la librería Tkinter.

3.1.4 Mediapipe

Es un framework desarrollado por Google que en este caso usaremos el modulo Face Mesh que crea una malla que detecta varios puntos del rostro, pero en este caso solo usaremos 5, nariz, barbilla, frente y pómulos, para poder saber si el rostro se movió y hace que lado.

3.1.5 Ultralitics YOLO

Esta librería es usada para poder detectar celulares, ya que se necesita el procesamiento en tiempo real y para una gran cantidad de dispositivos se usa el modelo más pequeño llamado nano.

3.2 Detección de rostro y atención

Para poder determinar el nivel de atención presentado en por ejemplo exámenes podemos hacer uso de MediaPipe Face Mesh, este método utiliza 2 métodos de .

```python
def analizar_cabeza(self, frame):
    alto, ancho, _ = frame.shape
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = self.face_mesh.process(rgb_frame)
    estado = "Desconocido"

    if results.multi_face_landmarks:
        for landmarks in results.multi_face_landmarks:
            def obtener_coords(idx):
                pt = landmarks.landmark[idx]
                return int(pt.x * ancho), int(pt.y * alto)

            # Puntos clave
            p_nariz = obtener_coords(1)
            p_izq_cara = obtener_coords(234)
            p_der_cara = obtener_coords(454)
            p_barbilla = obtener_coords(152)
            p_frente = obtener_coords(10)

            # --- LOGICA HORIZONTAL (Izquierda/Derecha) ---
            dist_nariz_izq = p_nariz[0] - p_izq_cara[0]
            dist_nariz_der = p_der_cara[0] - p_nariz[0]
            if dist_nariz_der == 0: dist_nariz_der = 0.001
            ratio_h = dist_nariz_izq / dist_nariz_der

            # --- LOGICA VERTICAL (Arriba/Abajo) ---
            dist_nariz_frente = p_nariz[1] - p_frente[1]
            dist_nariz_barbilla = p_barbilla[1] - p_nariz[1]
            if dist_nariz_barbilla == 0: dist_nariz_barbilla = 0.001
            ratio_v = dist_nariz_frente / dist_nariz_barbilla

            # Umbrales
            if ratio_h < 0.4:
                estado = "Izquierda"
            elif ratio_h > 2.5:
                estado = "Derecha"
            elif ratio_v < 0.6: # Nariz muy cerca de la frente
                estado = "Arriba"
            elif ratio_v > 1.8: # Nariz muy cerca de la barbilla
                estado = "Abajo"
            else:
                estado = "Centro"
```

Ilustración 1

3.3 Análisis de mirada

Para saber cuándo alguien está en la ventana del examen o no usamos los eventos del sistema operativo, podemos hacerlo con self.windows.bind('<FocusOut>', self.al_perder_foco).

```python
def al_perder_foco(self, event):
    if self.examen_activo:
        self.ventana_activa = False
        self.stats["ventanas_cambiadas"] += 1
        print("¡Alerta! Cambio de ventana detectado.")

def al_ganar_foco(self, event):
    if self.examen_activo:
        self.ventana_activa = True
```

Ilustración 2

3.4 Detección de celulares

La detección se hace con el modelo de YOLO, podemos indicarle que busque celulares poniendo la clase 67. Los resultados son pasados a un bucle que obtiene sus coordenadas y les dibujo un cuadro alrededor y texto que dice celular.

```python
class Detector:
    def __init__(self):
        print("Cargando modelo YOLO...")
        self.model = YOLO("yolov8n.pt")
        print("Modelo cargado.")

    def detectar_trampas(self, frame):
        # Solo buscamos celulares (clase 67)
        results = self.model(frame, stream=True, classes=[67], verbose=False)

        celular_detectado = False
        alerta = ""

        # deteccion de celular
        for r in results:
            boxes = r.boxes
            for box in boxes:
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                confianza = float(box.conf[0])
                if confianza > 0.4:
                    celular_detectado = True
                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
                    cv2.putText(frame, "CELULAR", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

        es_sospechoso = False
        if celular_detectado:
            alerta = "ALERTA: Uso de Celular"
            es_sospechoso = True

        return frame, es_sospechoso, alerta
```

Ilustración 3

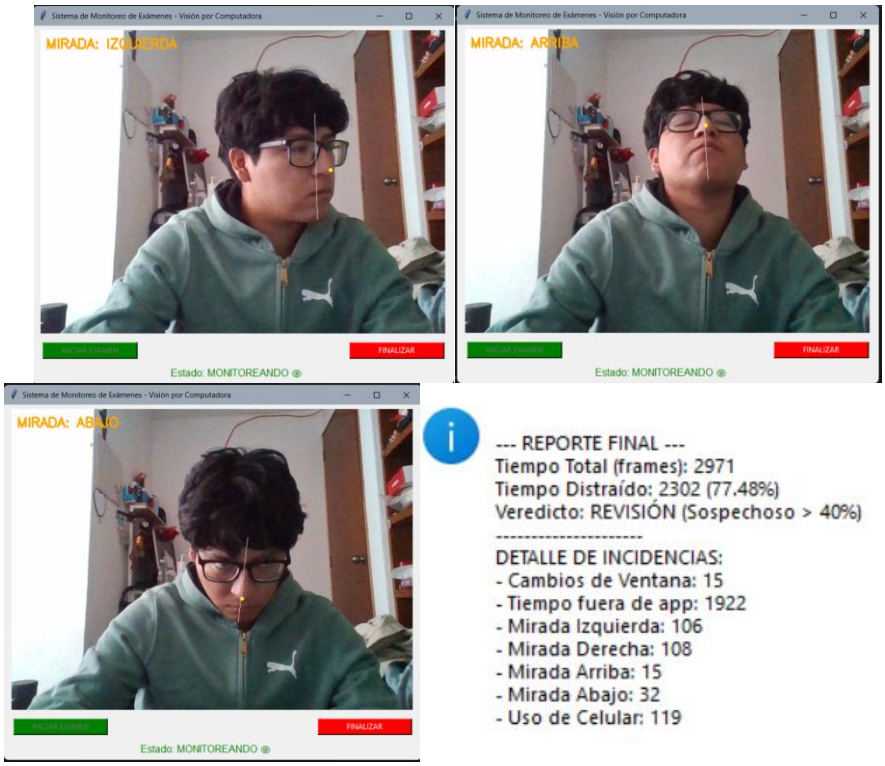
4 Resultados y discusión

4.1 Análisis de funcionamiento

La interfaz muestra la imagen tomada por la cámara, tiene 2 botones, el de iniciar examen y el de terminar examen, la detección no empieza hasta que se le da a empezar, una vez que empieza se puede ver una línea que pasa por mi rostro y un circulo que enfoca mi nariz. Cuando ponemos un celular o hay alguna otra persona se detecta colocándole encima una etiqueta como celular o intruso, además cuando cambiamos de pagina se puede ver el mensaje de que regresemos al examen.

4.2 Comparación de imágenes

El las siguientes imágenes se puede ver cómo se ven las alertas, está la de regresar al examen, alerta por celular, mirada hace varios puntos y finalmente el reporte de la cantidad de frames según cada comportamiento.



5 Conclusión

Los modelos clásicos se quedan cortos al analizar una imagen o son computacionalmente costosos, el análisis de una imagen resulta ser una tarea compleja que hace que el uso de redes neuronales sea un camino óptimo para el procesamiento de estas, los modelos de YOLO funcionan de maravilla en al detectar objetos, pero no son tan útiles al tratar de obtener hacia donde es que la atención de las personas esta puesta. El modelo de Google es poderosísimo no solo para saber hacia donde se dirige la atención si no también para las expresiones faciales, algunos filtros como los usados en TikTok o Instagram,

References

1. Szeliski, R. (2022). Computer vision: algorithms and applications (2nd ed.). Springer Nature.
2. Bradski, G., & Kaehler, A. (2023). Learning OpenCV 5: computer vision with the OpenCV library (5th ed.). O'Reilly Media..
3. Soy Dalto. (2023, 22 enero). *Curso de PYTHON desde CERO (Completo)* [Vídeo]. YouTube. https://www.youtube.com/watch?v=nKPbfIU442g.