DOCUMENTACIÓN: PRUEBA TÉCNICA PARA AI DEVELOPER JACOB. T.

Propuesta usando Visión artificial y NLP

Índice

Indice de Figuras

Introducción.

La visión por computadora es un campo dentro de la inteligencia artificial que permite a las máquinas analizar y comprender el mundo visual a partir de imágenes o videos. Esta tecnología es ampliamente utilizada en aplicaciones como la detección de objetos, el reconocimiento facial y la identificación de patrones anómalos en entornos dinámicos. En este proyecto, se emplea YOLOV**11 (You Only Look Once, V11)** este modelo de AI utiliza deep learning y redes neuronales convolucionales (CNN) para detectar objetos en avanzada para la detección de objetos en tiempo real. Adicionalmente, se combinan herramientas de visión por computadora como OpenCV para el procesamiento de imágenes y la implementación de un sistema de reconocimiento de anomalías en playeras (T-shirts).

El objetivo principal es desarrollar un sistema capaz de identificar y reconocer anomalías en tiempo real (de playeras), utilizando técnicas de visión artificial. Este sistema en etapas futuras busca detectar objetos o comportamientos fuera de lo común en la línea de producción de playeras Freeze Max, facilitando la detección de anomalías para el departamento de Control de Calidad (CC).

Este tipo de sistemas tiene aplicaciones en diversos campos, como:

- Seguridad y vigilancia para detectar errores de fabricación.
- Control de calidad en líneas de producción de Freeze Max.
- Monitoreo en los procesos de manufactura para identificar accidentes o eventos inusuales en líneas de producción.

El sistema desarrollado realiza las siguientes tareas:

1. Captura video en tiempo real desde una cámara.
2. Utiliza YOLOV11 para detectar objetos y clasificarlos.
3. Analiza patrones para identificar anomalías basándose en criterios predefinidos, (playeras con anomalías de fabricación: manchas, errores de costuras, orificios en tela).
4. Genera alertas con notificaciones a CC y almacenamiento de eventos para anomalías en un txt con:
    - Fecha formato ISO 8601 (YYYY-MM-DD)
    - Hora (UTC)
    - Línea de producción
    - Nombre del Producto
    - Tipo de anomalía (mancha, costura, orifico en tela)
    - Confianza %: Porcentaje de confianza en la lectura (0% – 100%), cuando se detecta una anomalía.

Para profundizar en las tecnologías utilizadas, se recomienda consultar la documentación oficial de YOLO, OpenCV y otros recursos relevantes en el campo de la visión por computadora:

- Ultralytics QuickStart
- YOLO Docs
- Repositorio Oficial YOLO11

Requisitos Técnicos

Para ejecutar este proyecto, es necesario contar con las siguientes herramientas y dependencias instaladas en el sistema:

1. Herramientas de Software

- **Python 3.12 o superior**: El lenguaje de programación utilizado para desarrollar el proyecto.
- **Gestor de paquetes pip**: Para instalar las bibliotecas necesarias.

2. Bibliotecas y Dependencias

Las siguientes bibliotecas deben estar instaladas en el entorno de Python:

- **Torch**: Para la creación y entrenamiento de modelos de aprendizaje profundo.
- **TorchVision**: Para trabajar con transformaciones de imágenes y modelos preentrenados en PyTorch.
- **Ultralytics**: Para la implementación de YOLO en sus diferentes versiones y tareas de detección de objetos.
- **OpenCV**: Para la captura, procesamiento y análisis de imágenes en tiempo real.
- **Matplotlib**: Para la visualización de datos y gráficos generados durante el análisis o entrenamiento.
- **NumPy**: Para realizar operaciones matemáticas avanzadas y manipulación de matrices.

Estas dependencias se pueden instalar fácilmente utilizando el archivo requirements.txt incluido en el repositorio. Para instalarlas, ejecuta el siguiente comando en la terminal:

```
pip install -r requirements.txt
```

3. Requisitos de Hardware

- **Cámara web**: Para capturar video en tiempo real.
- **Sistema operativo**: Windows 10.

4. Clonación del Repositorio

Para obtener el código del proyecto, clona el repositorio desde GitHub:

```
#Clonar el repositorio
git clone https://github.com/Jacob-Tinoco/Python-Developer-AI.git
```

```
# Cambiar al directorio del proyecto
cd Python-Developer-AI
```

5. Ejecución del Proyecto

Una vez instaladas las dependencias, puedes ejecutar cualquiera de los scripts proporcionados:

- detect_personv3_JT.py

La siguiente lista son los pseudocódigos propuestos para desarrollar a fondo el modelo:

- train.py
- detect.py

Fundamentos Teóricos

1. YOLO

YOLO es un modelo de detección de objetos basado en redes neuronales convolucionales que permite identificar y localizar objetos en imágenes o videos en tiempo real. A diferencia de otros enfoques, YOLO procesa la imagen completa en una sola pasada, dividiéndola en cuadrículas y prediciendo simultáneamente las clases y las cajas delimitadoras (bounding boxes) para cada objeto detectado. Esta arquitectura es rápida y eficiente, siendo ideal para aplicaciones en tiempo real como vigilancia, conducción autónoma y análisis de video.

2. COCO (Common Objects in Context)

COCO es un conjunto de datos ampliamente utilizado en tareas de visión por computadora, diseñado específicamente para entrenar y evaluar modelos de detección, segmentación y reconocimiento de objetos. Contiene más de 300,000 imágenes con anotaciones detalladas que incluyen categorías como personas, animales, vehículos y objetos cotidianos. La diversidad y complejidad de las imágenes en COCO lo convierten en un estándar de referencia para medir el rendimiento de modelos como YOLO en tareas de detección de objetos.

3. Detección de Objetos

La detección de objetos es una técnica de visión por computadora que combina dos tareas principales: la localización y la clasificación de objetos en imágenes o videos. La localización implica identificar la posición exacta de un objeto mediante cajas delimitadoras, mientras que la clasificación asigna una etiqueta o categoría al objeto detectado. Modelos como YOLO utilizan estas técnicas para identificar múltiples objetos en una sola imagen, incluso en escenarios complejos con múltiples categorías y superposiciones, ver Figura 1.

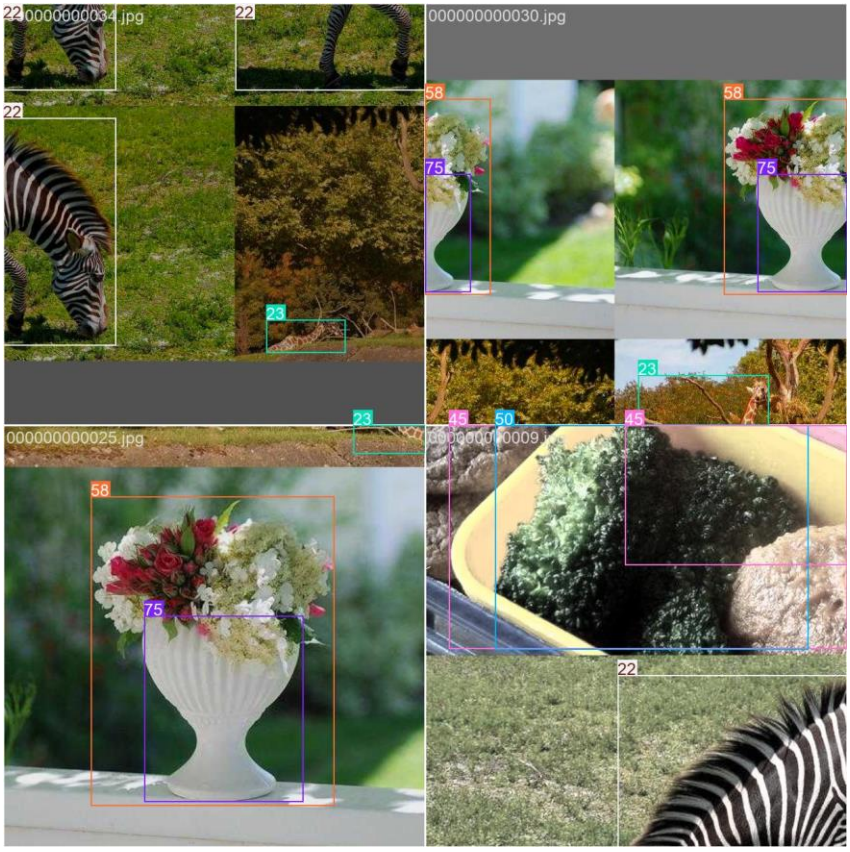

Figura 1 Ejemplos de detección de objetos con YOLO, imagen creada a partir de las pruebas con documentación oficial

4. Etiquetas para YOLO

En el contexto de YOLO, las etiquetas representan la información necesaria para entrenar el modelo en la detección de objetos, ver Figura 2. Estas etiquetas se almacenan en archivos de texto asociados a cada imagen y contienen datos específicos sobre los objetos presentes, como la clase, la posición y el tamaño de las cajas delimitadoras. Cada línea de un archivo de etiquetas describe un objeto con el siguiente formato:

`<clase> <x_centro> <y_centro> <ancho> <alto>`

Donde las coordenadas y dimensiones están normalizadas en relación con el tamaño de la imagen. Este formato simplificado permite que YOLO procese las etiquetas de manera eficiente durante el entrenamiento, asegurando una detección precisa.

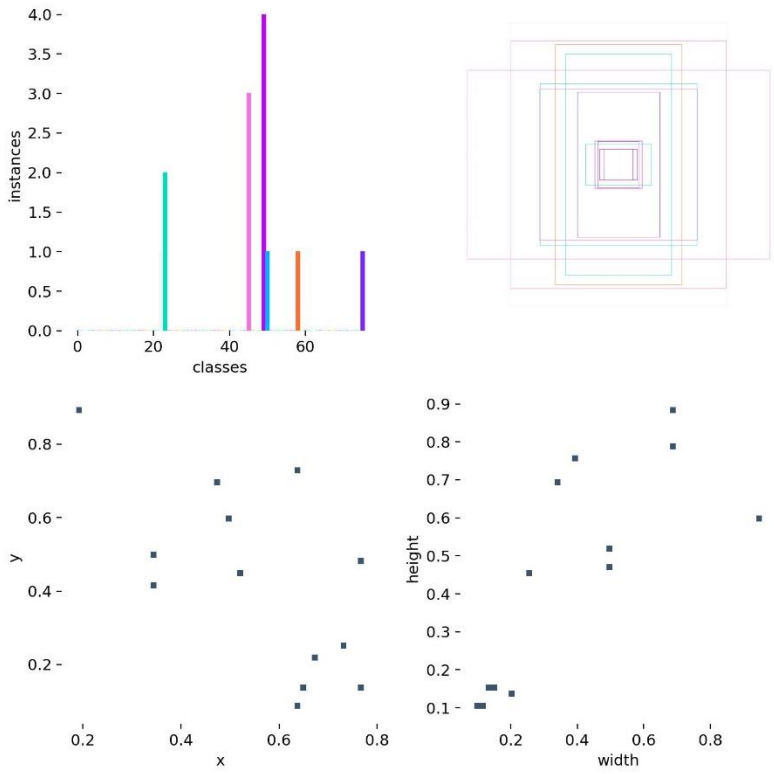

Figura 2Ejemplo en el uso de etiquetas durante el procesamiento de imágenes

5. Confianza de Detección

La confianza de detección es un valor entre 0 y 1 (0% y 100%) que indica la precisión con la que el modelo ha identificado un dedo como extendido.

6. Torch

Torch es una biblioteca de aprendizaje profundo que proporciona herramientas para construir y entrenar modelos de redes neuronales. Es flexible y eficiente, lo que la hace ideal para tareas que requieren cálculos complejos y procesamiento en paralelo. En este proyecto, se utiliza para implementar modelos de detección de objetos y realizar inferencias rápidas.

7. TorchVision

TorchVision es un módulo complementario de Torch diseñado específicamente para trabajar con datos de imágenes. Proporciona herramientas para cargar y preprocesar conjuntos de datos, así como modelos preentrenados para tareas de visión por computadora. En este proyecto, se emplea para manejar transformaciones de imágenes y facilitar el entrenamiento de modelos en un futuro.

8. Ultralytics

Ultralytics es una biblioteca que ofrece implementaciones avanzadas del modelo YOLO (You Only Look Once). Facilita tareas como la detección de objetos, segmentación y clasificación

con configuraciones predefinidas y personalizables. En este proyecto, se utiliza para cargar el modelo YOLO y realizar detecciones de manera eficiente.

9. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto que proporciona herramientas para el procesamiento de imágenes y videos. En este proyecto, se utiliza para:

- Capturar video en tiempo real desde la cámara web.

- Mostrar los resultados de la detección de objetos en una ventana.

10. Matplotlib

Matplotlib es una biblioteca de Python utilizada para la visualización de datos. Permite crear gráficos, histogramas y figuras personalizadas para la visualización de información. En este proyecto, se emplea para representar visualmente los resultados de las detecciones y analizar el rendimiento del modelo durante su ejecución.

11. NumPy

NumPy es una biblioteca de Python utilizada para realizar operaciones numéricas y de matrices. En este proyecto, se emplea para calcular distancias entre puntos clave (LandMarks) y normalizar valores de confianza durante las detecciones.

Metodología.

Sobre la metodología, se muestra en la Figura 3, el cual sigue el proceso de:

- **Inicio del Sistema**: El programa comienza cargando el modelo YOLO y abriendo el flujo de video desde la cámara especificada.
- **Procesamiento del Video**: Cada cuadro del video es leído y procesado mediante el modelo YOLO para identificar posibles anomalías.
- **Análisis de Resultados**: Los resultados de YOLO son analizados para determinar el tipo de anomalía detectada (orificio, mancha o costura incorrecta).
- **Registro y Notificación**: Si se detecta una anomalía, se registra en un archivo de texto y se envía una notificación simulada.
- **Ciclo de Procesamiento**: El sistema continúa procesando los cuadros del video hasta que no haya más datos disponibles o se detenga manualmente.
- **Finalización**: Una vez que se procesan todos los cuadros, el sistema se cierra.

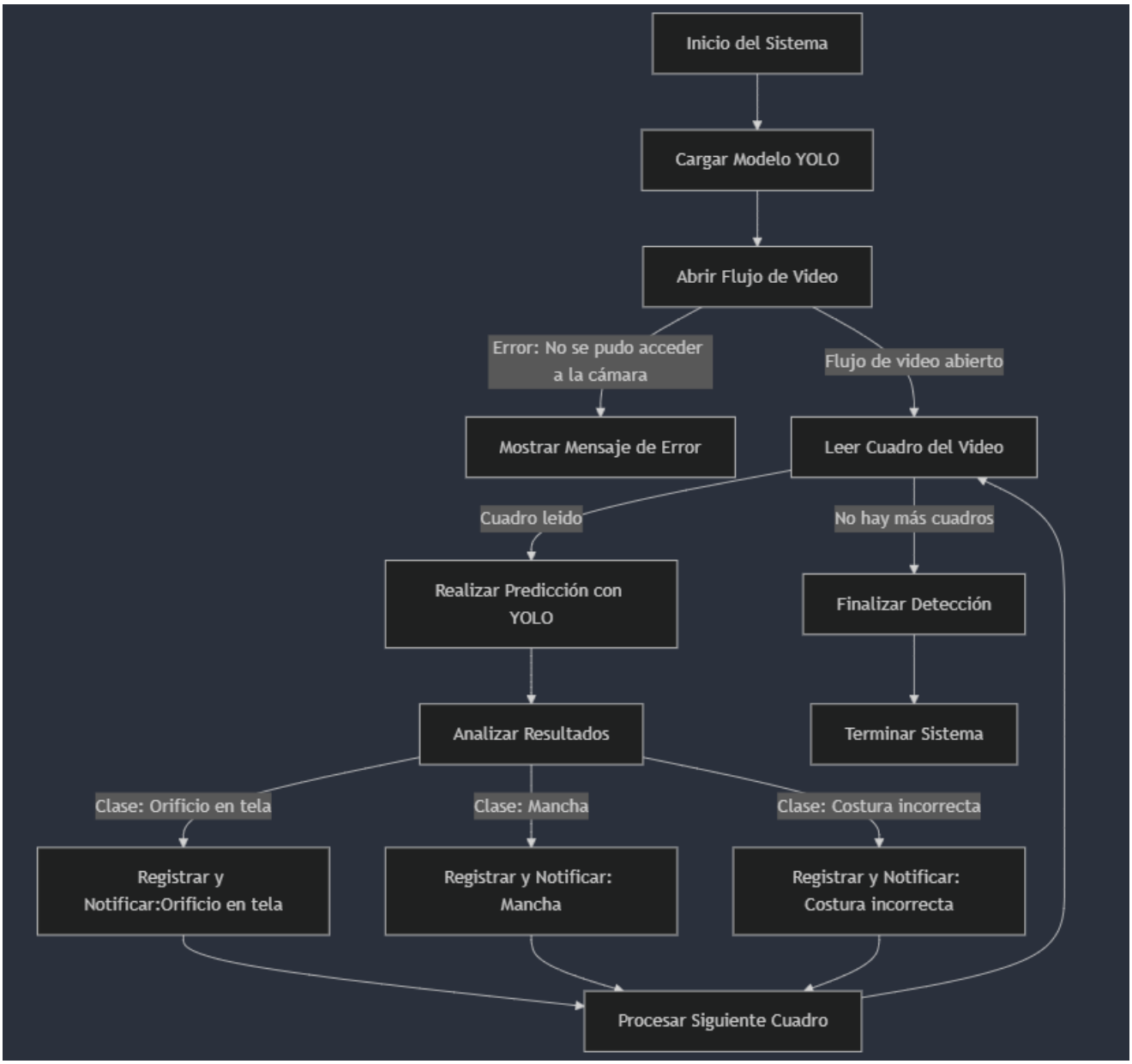

Figura 3 Metodología

Código de Ejemplo (Pseudocódigo)

Este pseudocódigo describe un bucle principal que captura frames de la cámara, detecta manos y dedos extendidos utilizando MediaPipe, y muestra los resultados en tiempo real con OpenCV. La lógica se basa en la comparación de landmarks y el cálculo de confianza para determinar el estado de los dedos. El proceso es eficiente y está diseñado para ejecutarse en tiempo real.

```
"""
© 2025. Todos los derechos reservados. Este script está
protegido por derechos de autor (Jacob Tinoco).
Uso no autorizado está prohibido.
"""

# Importar las bibliotecas necesarias
from ultralytics import YOLO
import cv2
import datetime
import os

# Función para registrar eventos en un archivo .txt
def registrar_evento(anomalia, confianza):
    """
    Registra eventos de anomalías en un archivo .txt.
    """
    # Obtener la fecha actual
    fecha_actual = datetime.datetime.now()
    # Crear el nombre del archivo con la fecha
    nombre_archivo =
f"anomalias_{fecha_actual.strftime('%d%m%y')}_producto.txt"

    # Verificar si el archivo no existe y crearlo si es
necesario
    if not os.path.exists(nombre_archivo):
        with open(nombre_archivo, 'w') as archivo:
            archivo.write("Registro de anomalías en la línea
de producción\n")

    # Registrar el evento en el archivo
    with open(nombre_archivo, 'a') as archivo:
        archivo.write(
            f"| Fecha (ISO 8601) | Hora (UTC)  | Línea de
```

```
Producción | Nombre del Producto | Tipo de anomalía:
{anomalia} | Confianza: {confianza}%\n"
        )

# Función para enviar notificaciones
def enviar_notificacion(anomalia):
    """
    Simula el envío de una notificación (correo a Control de
Calidad o mensaje de WhatsApp al encargado de area).
    """
    print(f"Notificación enviada: {anomalia}")

# Función principal para detectar anomalías
def detectar_anomalias(stream_video):
    """
    Detecta anomalías en un flujo de video utilizando un
modelo YOLO.
    """
    # Cargar el modelo YOLO preentrenado
    modelo = YOLO("dirección/PATH/modelo/preentrenado")  #
Reemplazar con la ruta real del modelo

    try:
        # Abrir el flujo de video
        video = cv2.VideoCapture(stream_video)
        if not video.isOpened():
            print("Error: No se pudo acceder a la cámara.")
            return

        # Procesar cada cuadro del video
        while video.isOpened():
            read, frame = video.read()
            if read:
                # Realizar predicción con el modelo YOLO
                resultados = modelo.predict(source=frame,
conf=0.5)

                # Analizar los resultados de la predicción
                for resultado in resultados.boxes.data:
                    x1, y1, x2, y2, confianza, clase_id =
resultado
```

```
                    anomalía = ""
                    if clase_id == 1:
                        anomalía = "Orificio en tela"
                    elif clase_id == 2:
                        anomalía = "Mancha"
                    elif clase_id == 3:
                        anomalía = "Costura incorrecta"

                    # Si se detecta una anomalía, registrar y
notificar
                    if anomalía != "":
                        registrar_evento(anomalía, confianza
* 100)
                        enviar_notificacion(anomalía)

    except Exception as e:
        print(f"Error durante la detección de anomalías:
{e}")

# Iniciar el sistema
print("Sistema iniciado.")
detectar_anomalias("ruta/a/stream/camara")  # Reemplazar con
la ruta real del stream de la cámara
```

Resultados

En términos generales, la propuesta de aplicación tiene que tener tiempo de al menos 3 meses para entrenamiento de un modelo personalizado. El modelo YOLOV11, combinado con las herramientas de visión artificial y procesamiento de imágenes, permite detectar anomalías en playeras con un nivel de precisión adecuado para aplicaciones en tiempo real.

El sistema puede en un futuro ser capaz de identificar manchas, errores de costura y orificios en tela, dependiendo de la calidad de las imágenes capturadas y las condiciones de iluminación. Además, el tiempo de procesamiento por cuadro permite una lectura estable dado que YOLO puede procesar a una velocidad de 30 fps siendo una lectura aceptable en un requisito de streaming, lo que asegura un rendimiento eficiente en entornos industriales.

Los datos generados durante la detección de anomalías pueden ser almacenados correctamente en archivos de texto bajo el formato ISO 8601, garantizando una trazabilidad precisa de los eventos detectados en las líneas de producción. Este registro incluye información clave como la fecha, hora, línea de producción, tipo de anomalía y porcentaje de confianza, resultando en un recurso útil para el análisis futuro de rendimiento o de mejoras continuas al proyecto.

Análisis de Precisión, Limitaciones y Desafíos

Precisión

El modelo puede llegar a ser útil en la detección de anomalías en playeras, alcanzando precisiones superiores al 90% con un entrenamiento adecuado supervisado y semisupervisado con escenarios controlados. Sin embargo, la precisión puede llegar a disminuir ligeramente en condiciones de iluminación deficiente o cuando las anomalías eran de menor tamaño y difícil visibilidad.

Limitaciones

Entre las principales limitaciones identificadas, se encuentra la dependencia del modelo en imágenes de alta calidad para mantener su precisión. En entornos con variaciones significativas en iluminación o movimiento rápido de las prendas, el sistema puede presentar dificultades para identificar anomalías pequeñas. Asimismo, el entrenamiento del modelo estuvo limitado por la cantidad y diversidad de datos disponibles, lo que limita a un pseudocódigo.

Desafíos

El principal desafío radica en la necesidad de optimizar el modelo para entornos industriales reales, donde las condiciones son menos controladas. Adicionalmente, la integración del sistema con otros procesos de manufactura requerirá ajustes en la infraestructura tecnológica y capacitación del personal para interpretar los resultados generados por el sistema.

Análisis de los Resultados

El análisis de los resultados obtenidos evidencia que el sistema desarrollado tiene un alto potencial para ser implementado en líneas de producción industriales. La capacidad de identificar anomalías en tiempo real permite reducir significativamente los errores en el control de calidad, lo que se traduce en una mejora en la eficiencia y reducción de costos asociados a productos defectuosos.

Sin embargo, los resultados también resaltan la importancia de continuar perfeccionando el modelo mediante la inclusión de más datos de entrenamiento y la adaptación del sistema a diferentes condiciones de operación. Esto permitirá incrementar su robustez y garantizar un desempeño consistente en entornos dinámicos indurtriales.

Conclusión

El sistema propuesto es eficaz para la detección de anomalías en playeras utilizando técnicas avanzadas de visión artificial y aprendizaje profundo. La combinación del modelo YOLO con herramientas como OpenCV y Torch permite desarrollar una solución capaz de operar en tiempo real, cumpliendo con los requisitos establecidos para su aplicación en entornos industriales.

A pesar de las limitaciones identificadas, los beneficios que ofrece este sistema son evidentes, especialmente en términos de automatización y precisión en el control de calidad. En futuras iteraciones, se recomienda obtener un conjunto de datos de entrenamiento, mejorar la integración con otras tecnologías y optimizar el modelo para escenarios más complejos. Esto asegurará una implementación exitosa y escalable en diferentes líneas de producción.