

Guía Completa: Cómo Usar el Pipeline YOLO + SAM + MiDaS

Requisitos Previos

- Cuenta de Google (para usar Google Colab)
 - Conexión a internet estable
 - Imágenes para procesar (opcional, se incluye demo)
-

PASO 1: Configuración Inicial en Google Colab


1.1 Abrir Google Colab

1. Ve a <https://colab.research.google.com>
2. Inicia sesión con tu cuenta de Google
3. Crea un nuevo notebook: **Archivo > Nuevo notebook**

1.2 Activar GPU (Recomendado)

1. Ve a **Entorno de ejecución > Cambiar tipo de entorno de ejecución**
2. Selecciona **GPU** en "Acelerador de hardware"
3. Haz clic en **Guardar**

1.3 Copiar el Código

1. Copia todo el código del pipeline en la primera celda del notebook
 2. Ejecuta la celda (Ctrl+Enter o botón )
-

PASO 2: Instalación de Dependencias

2.1 Crear y Ejecutar Celda de Instalación

Crea una nueva celda y pega este código:


```
# Instalación de todas las dependencias necesarias
```

```
!pip install ultralytics
!pip install segment-anything
!pip install opencv-python
!pip install timm
!pip install matplotlib
!pip install torch torchvision
!pip install numpy
!pip install Pillow
!pip install pandas

# Descargar modelo SAM (archivo grande ~2.5GB)
!wget https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth

# Clonar repositorio SAM
!git clone https://github.com/facebookresearch/segment-anything.git

print("✅ Instalación completada!")
```

 **IMPORTANTE:** Esta instalación puede tomar 5-10 minutos. Espera a que termine antes de continuar.

PASO 3: Inicialización del Pipeline

3.1 Crear Celda de Inicialización

```
# Inicializar el pipeline
print("🔄 Inicializando modelos...")
pipeline = VisionPipeline()
print("✅ Pipeline listo para usar!")
```

 **Nota:** La primera vez puede tomar 2-3 minutos descargando modelos.

PASO 4: Opciones para Procesar Imágenes

OPCIÓN A: Demo con Imagen de Ejemplo (Más Fácil)



```
# Ejecutar demo automático
resultados = demo_con_imagen_ejemplo()
```

¿Qué hace esto?

- Descarga automáticamente una imagen de ejemplo
- Procesa la imagen completa
- Muestra todas las visualizaciones
- Guarda resultados en carpetas

OPCIÓN B: Subir Tu Propia Imagen

B.1 Subir Imagen a Colab

1. En el panel izquierdo, haz clic en el ícono de  **Archivos**
2. Arrastra tu imagen desde tu computadora
3. O haz clic en  **Subir** y selecciona la imagen

B.2 Procesar Tu Imagen

```
# Cambiar "mi_imagen.jpg" por el nombre de tu archivo
imagen_path = "/content/mi_imagen.jpg"
resultados = pipeline.procesar_imagen_completa(imagen_path)
```

OPCIÓN C: Usar URL de Imagen

```
import urllib.request
```

```
# Descargar imagen desde URL
url = "https://ejemplo.com/mi_imagen.jpg"
urllib.request.urlretrieve(url, "imagen_descargada.jpg")
```

```
# Procesar
resultados = pipeline.procesar_imagen_completa("imagen_descargada.jpg")
```

PASO 5: Explorar Funcionalidades Individuales

5.1 Solo Detección YOLO

```
# Probar solo detección de objetos
detecciones = pipeline.detectar_objetos_yolo("mi_imagen.jpg", confidence=0.3)
print(f"Objetos detectados: {detecciones['nombres_clases']}")
print(f"Confianzas: {detecciones['confianzas']}")
```

5.2 Solo Segmentación SAM

```
# Primero detectar objetos
detecciones = pipeline.detectar_objetos_yolo("mi_imagen.jpg")

# Luego segmentar
masks = pipeline.segmentar_con_sam(detecciones['imagen'], detecciones['boxes'])
print(f"Máscaras generadas: {len(masks)}")
```

5.3 Solo Estimación de Profundidad

```
import cv2

# Cargar imagen
imagen = cv2.imread("mi_imagen.jpg")
imagen_rgb = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)

# Estimar profundidad
depth_map = pipeline.estimated_profundidad_midas(imagen_rgb)

# Visualizar
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(imagen_rgb)
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(depth_map, cmap='plasma')
plt.title('Mapa de Profundidad')
plt.axis('off')
plt.show()
```

PASO 6: Efectos Creativos

6.1 Efecto Bokeh (Desenfoque de Fondo)

```
# Procesar imagen completa primero
resultados = pipeline.procesar_imagen_completa("mi_imagen.jpg")

# Aplicar efecto bokeh más intenso
```

```

imagen_bokeh = pipeline.aplicar_efecto_bokeh(
    resultados['detecciones']['imagen'],
    resultados['masks'],
    resultados['depth_map'],
    blur_strength=25 # Más desenfoque
)

# Mostrar resultado
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.imshow(resultados['detecciones']['imagen'])
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(imagen_bokeh)
plt.title('Efecto Bokeh')
plt.axis('off')
plt.show()

```

6.2 Análisis de Objetos por Distancia

```

# Obtener análisis detallado
df = resultados['analisis']

# Mostrar objetos ordenados por cercanía
print("🏆 OBJETOS MÁS CERCANOS:")
print(df[['clase', 'distancia_relativa', 'profundidad_media']].head())

# Filtrar solo objetos cercanos
objetos_cercanos = df[df['distancia_relativa'] > 0.7]
print(f"\n 📍 Objetos en primer plano: {len(objetos_cercanos)}")

```

PASO 7: Trabajar con Múltiples Imágenes

7.1 Crear Carpeta de Imágenes

```

import os
os.makedirs('mis_imagenes', exist_ok=True)
print("📁 Carpeta 'mis_imagenes' creada")
print("👉 Sube tus imágenes a esta carpeta usando el panel de archivos")

```

7.2 Procesar Todas las Imágenes

```
# Procesar todas las imágenes de la carpeta
procesar_multiples_imagenes('mis_imagenes')
```



PASO 8: Ver y Descargar Resultados

8.1 Explorar Archivos Generados

```
# Ver estructura de archivos creados
!ls -la outputs/
!ls -la outputs/depth_maps/
!ls -la outputs/recortes/
```

8.2 Descargar Resultados

1. Ve al panel de 📁 **Archivos** en Colab
2. Navega a la carpeta **outputs**
3. Haz clic derecho en cualquier archivo
4. Selecciona **Descargar**

8.3 Ver CSV con Estadísticas

```
import pandas as pd
```

```
# Leer archivo CSV generado
df = pd.read_csv('outputs/imagen_analisis.csv')
print("📊 ESTADÍSTICAS COMPLETAS:")
print(df.to_string())
```

```
# Crear gráfico de distancias
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.bar(df['clase'], df['distancia_relativa'])
plt.title('Distancia Relativa por Objeto')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

PASO 9: Personalización Avanzada

9.1 Ajustar Parámetros de Detección

```
# Detectar más objetos (menor confianza)
detecciones = pipeline.detectar_objetos_yolo("mi_imagen.jpg", confidence=0.2)

# Detectar menos objetos (mayor confianza)
detecciones = pipeline.detectar_objetos_yolo("mi_imagen.jpg", confidence=0.8)
```

9.2 Crear Visualizaciones Personalizadas

```
def mi_visualizacion_personalizada(imagen, masks, depth_map, nombres):
    """Crear tu propia visualización"""
    plt.figure(figsize=(15, 5))

    # Panel 1: Imagen original
    plt.subplot(1, 3, 1)
    plt.imshow(imagen)
    plt.title('Mi Imagen')
    plt.axis('off')

    # Panel 2: Solo máscaras
    plt.subplot(1, 3, 2)
    combined_mask = np.zeros(imagen.shape[:2])
    for i, mask in enumerate(masks):
        combined_mask[mask] = i + 1
    plt.imshow(combined_mask, cmap='tab10')
    plt.title('Segmentación')
    plt.axis('off')

    # Panel 3: Profundidad invertida (más claro = más cerca)
    plt.subplot(1, 3, 3)
    plt.imshow(1 - depth_map, cmap='hot')
    plt.title('Cercanía (Claro = Cerca)')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

# Usar tu visualización
resultados = pipeline.procesar_imagen_completa("mi_imagen.jpg")
mi_visualizacion_personalizada(
    resultados['detecciones']['imagen'],
```

```
resultados['masks'],
resultados['depth_map'],
resultados['detecciones']['nombres_clases']
)
```

PASO 10: Solución de Problemas Comunes

Error: "No se detectaron objetos"

Solución:

```
# Reducir umbral de confianza
detecciones = pipeline.detectar_objetos_yolo("mi_imagen.jpg", confidence=0.1)
```

Error: "Out of memory"

Soluciones:

1. Redimensionar imagen antes de procesar:

```
from PIL import Image
img = Image.open("mi_imagen.jpg")
img = img.resize((800, 600)) # Imagen más pequeña
img.save("imagen_pequena.jpg")
```

2. Usar modelo YOLO más pequeño:

```
pipeline.yolo_model = YOLO('yolov8n.pt') # Nano (más rápido)
```

Error: "Archivo no encontrado"

Verificar ruta:

```
import os
print("Archivos disponibles:")
print(os.listdir('/content/'))
```

PASO 11: Casos de Uso Específicos

11.1 Análisis de Seguridad

```
# Detectar personas y vehículos
def analizar_seguridad(imagen_path):
    detecciones = pipeline.detectar_objetos_yolo(imagen_path, confidence=0.4)

    personas = [i for i, clase in enumerate(detecciones['nombres_clases']) if 'person' in clase]
    vehiculos = [i for i, clase in enumerate(detecciones['nombres_clases']) if any(v in clase for v in
['car', 'truck', 'bus'])]

    print(f"👤 Personas detectadas: {len(personas)}")
    print(f"🚗 Vehículos detectados: {len(vehiculos)}")

    return personas, vehiculos

# Usar
personas, vehiculos = analizar_seguridad("mi_imagen.jpg")
```

11.2 Análisis de Retail

```
# Contar productos en estantería
def contar_productos(imagen_path):
    resultados = pipeline.procesar_imagen_completa(imagen_path)
    df = resultados['analisis']

    # Productos por distancia
    primer_plano = df[df['distancia_relativa'] > 0.6]
    fondo = df[df['distancia_relativa'] <= 0.6]

    print(f"📦 Productos en primer plano: {len(primer_plano)}")
    print(f"📦 Productos en fondo: {len(fondo)}")

    return primer_plano, fondo
```




PASO 12: Recursos Adicionales

Documentación de Modelos:

- YOLO: <https://docs.ultralytics.com>

- **SAM:** <https://github.com/facebookresearch/segment-anything>
- **MiDaS:** <https://github.com/intel-isl/MiDaS>

Tipos de Imágenes Ideales:

-  **Buenas:** Fotos claras, buena iluminación, objetos bien definidos
-  **Perfectas:** Escenas urbanas, interiores, retratos con fondo
-  **Difíciles:** Imágenes muy oscuras, objetos muy pequeños, escenas complejas

Límites del Sistema:

- **Memoria:** Imágenes muy grandes (>4K) pueden causar errores
 - **Precisión:** Objetos muy pequeños pueden no detectarse
 - **Velocidad:** Procesamiento puede tomar 30-60 segundos por imagen
-