Universidad Iberoamericana de Puebla

Ingeniería en Sistemas Computacionales

Tecnologías Emergentes en Computación

Basura Inteligente: Identificación y Clasificación de Residuos con IA

Diaz Stevanato Leon Lameda
Hernández López José Mauro
Pacheco Valdez Diego

Otoño 2024

Índice

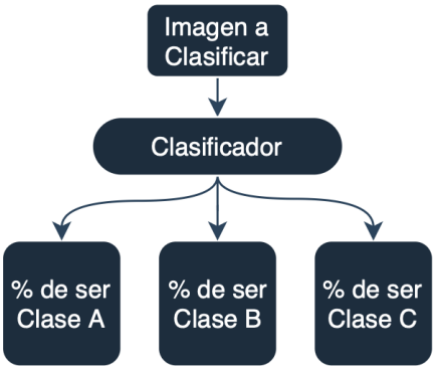
1. Selección y Análisis del Modelo de Machine Learning

1.1 Análisis de la Problemática

Nuestra misión con este proyecto es aquel de crear una herramienta que facilite la correcta separación de la basura en nuestro estado. Con esto en mente tenemos claro que se trata de un problema de clasificación.

En el aprendizaje automático, la **clasificación** es el proceso de asignar una etiqueta discreta a una muestra de datos. Los principales tipos de clasificación son:
c

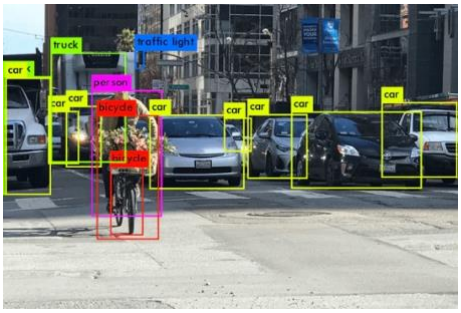
1. **Clasificación Binaria**: Asigna una muestra a una de dos categorías.
2. **Clasificación Multiclase**: Asigna una muestra a una de varias categorías.
3. **Clasificación Multietiqueta**: Asigna múltiples etiquetas a una muestra.
4. **Clasificación Jerárquica**: Las categorías están organizadas en una estructura jerárquica.



Sin embargo, esto solo sería parte de la problemática, pues no se puede clasificar la basura si la computadora es incapaz de procesarla.

Dentro de la inteligencia artificial existe la rama de la visión por computadora y, dentro de esta misma, la detección de objetos. La detección de objetos permite a las computadoras usar redes neuronales para calcular la posibilidad de que lo que están viendo sea un objeto en específico. En específico se trabajó con YOLOv8.

YOLOv8 es una arquitectura de red neuronal profunda para detección de objetos en tiempo real, mejorando la precisión y eficiencia de versiones anteriores. Puede identificar múltiples objetos en una sola imagen o video, siendo rápida y adecuada para aplicaciones en tiempo real.

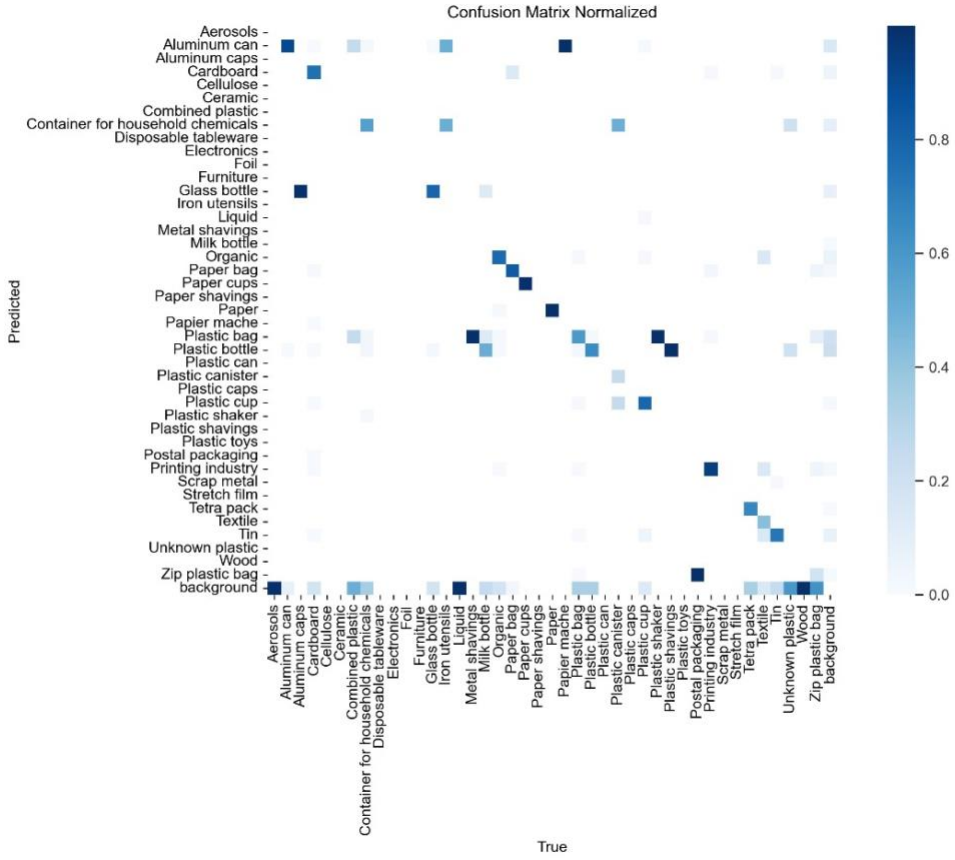


Fuente: *Rich feature hierarchies for accurate object detection and semantic segmentation*

1.2 Análisis General de Datos

Para lograr nuestro cometido es necesario el uso de un dataset correctamente etiquetado, para esto se utilizó el dataset YOLO Waste Detection Dataset v1 creado por ProjectVerba. Se eligio el siguiente dataset debido a su tamaño (13,104 imágenes), sus cuarenta y dos clases (desde botellas hasta textiles) y su licencia de código libre. Este demuestra ser uno de los mejores data sets para nuestro diseño.

2. Matriz de Confusión Normalizada



Lo que vemos aquí es la matriz de relación normalizada creada con la información de nuestra base de datos. Al observar la matriz de confusión, existe algo en específico que llaman nuestra atención.

Las clases como Aerosols, Aluminum can, y Aluminum caps muestran clara confusión entre sí, esto claramente se debe a lo similar visual entre estas, lo mismo parece sucederé entre las bolsas y vasos de plástico y el papel y cartón.

Esto, aunque puede parecer un gran problema no tiene repercusiones extremadamente problemáticas para nuestro proyecto gracias a que estos errores son en objetos de un mismo material.

3. Primer Modelo y Resultados

3.1. Configuración del Entrenamiento

La función `setup_training_config()`: se encarga de configurar los parámetros de entrenamiento del modelo, basándose en el hardware disponible (GPU o CPU). Los parámetros incluyen:

`epochs`

Número de épocas de entrenamiento.

`batch`

Tamaño del lote de entrenamiento.

`imgsz`

Tamaño de las imágenes de entrada.

`device`

Selección entre CPU o GPU dependiendo de la disponibilidad.

`optimizer`

El optimizador utilizado es Adam, conocido por su estabilidad.

`lr0` y `lrf`

Definen el aprendizaje inicial y final, lo que ayuda en la convergencia del modelo.

`Data Augmentation`

```
def setup_training_config():
    """Configurar parámetros de entrenamiento basados en hardware disponible"""
    config = {
        'epochs': 100,  # Aumentado para mejor convergencia
        'batch': 16,
        'imgsz': 640,
        'patience': 20,  # Early stopping
        # Usar GPU si está disponible
        'device': 'cuda' if torch.cuda.is_available() else 'cpu',
        'workers': min(8, max(1, torch.cuda.device_count() * 4)),
        'optimizer': 'Adam',  # Optimizer más estable
        'lr0': 0.001,  # Learning rate inicial
        'lrf': 0.01,   # Learning rate final
        'momentum': 0.937,
        'weight_decay': 0.0005,
        'warmup_epochs': 3,
        'warmup_momentum': 0.8,
        'warmup_bias_lr': 0.1,
        'box': 7.5,    # Box loss gain
        'cls': 0.5,    # Cls loss gain
        'dfl': 1.5,    # DFL loss gain
        # Data Augmentation
        'hsv_h': 0.015,  # Aumento de datos HSV-Hue
        'hsv_s': 0.7,    # Aumento de datos HSV-Sat
        'hsv_v': 0.4,    # Aumento de datos HSV-Val
        'degrees': 0.0,   # Rotación imagen
        'translate': 0.1,  # Traslación
        'scale': 0.5,    # Escala
        'shear': 0.0,    # Shear
        'perspective': 0.0,  # Perspectiva
        'flipud': 0.0,    # Flip up-down
        'fliplr': 0.5,    # Flip left-right
        'mosaic': 1.0,    # Mosaico
        'mixup': 0.0,     # Mixup
```

Diversas técnicas de aumento de datos, como la rotación, traslación, escalado y volteo, para mejorar la generalización del modelo.

```
    'copy_paste': 0.0  # Copy-paste
}
```

3.2. Entrenamiento del Modelo

```
def train_model(data_path, config):
    """Entrenar el modelo con los parámetros optimizados"""
    try:
        # Inicializar modelo
        model = YOLO("yolov8n.pt")

        # Entrenamiento
        results = model.train(
            data=data_path,  # Ruta correcta al archivo data.yaml
            epochs=config['epochs'],
            batch=config['batch'],
            imgsz=config['imgsz'],
            project="yolo_waste_detection",
            name="yolov8_training9",
            device=config['device']
        )

        # Validación
        val_results = model.val()
        logger.info(f"Métricas de validación: {val_results}")

        # Exportar modelo
        model.export(format="onnx", dynamic=True)
        logger.info("Modelo exportado en formato ONNX")

        return model, results

    except Exception as e:
        logger.error(f"Error durante el entrenamiento: {str(e)}")
        raise
```

`train_model()`

Se inicializa el modelo YOLOv8 utilizando el archivo preentrenado yolov8n.pt.

El entrenamiento se realiza usando el dataset cargado y los parámetros previamente configurados.

`model.train()`

Esta función entrena el modelo con los datos del dataset, configurando el número de épocas, el tamaño del batch, el tamaño de las imágenes y el dispositivo (CPU o GPU) en el que se realizará el entrenamiento.

Después de completar el entrenamiento, se valida el modelo utilizando

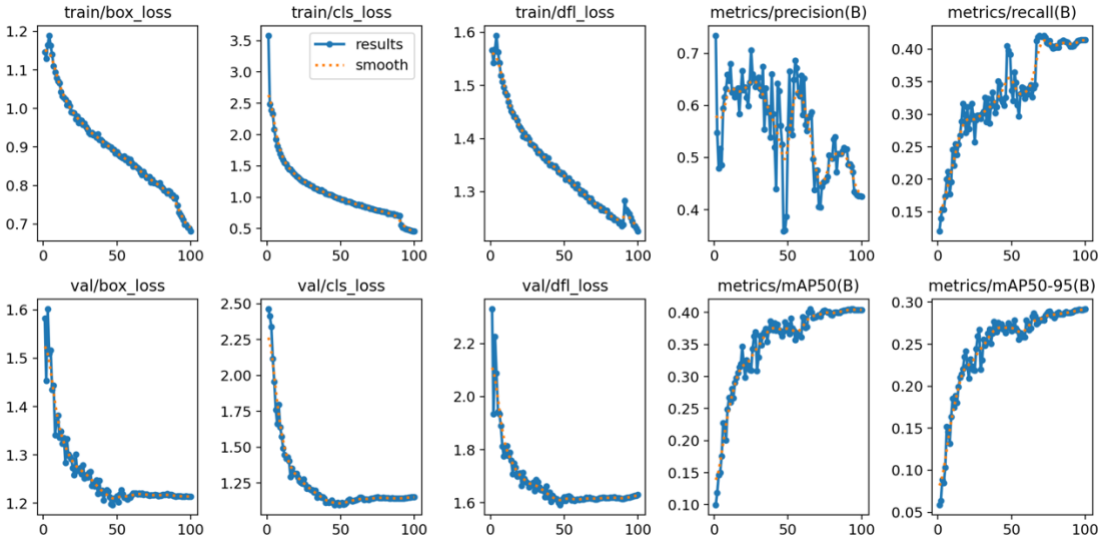
`model.val()`

Esta función evalúa el modelo con un conjunto de validación para obtener métricas como precisión, recall y F1-score.

Finalmente, el modelo entrenado se exporta en el formato ONNX, un formato estándar de interoperabilidad entre frameworks de aprendizaje automático. La exportación a ONNX permite usar el modelo en diferentes plataformas y dispositivos.

El modelo entrenado es, por último, almacenado para poder ser usado más adelante en yolo_waste_detection/yolov8_training95/weights/ donde se nos dan 3 opciones: la primera etapa, la última etapa y la mejor etapa.

3.3 Resultados



Se presentan los gráficos generados durante el entrenamiento del modelo YOLOv8. Cada uno de los gráficos muestra el comportamiento de diferentes métricas y pérdidas a lo largo de las épocas.

El modelo muestra una tendencia positiva en el entrenamiento y la validación, con una reducción consistente en las pérdidas y una mejora en las métricas de rendimiento (precisión, recall y mAP). Esto indica que el modelo YOLOv8 está aprendiendo de manera efectiva a clasificar y detectar los objetos del dataset de residuos

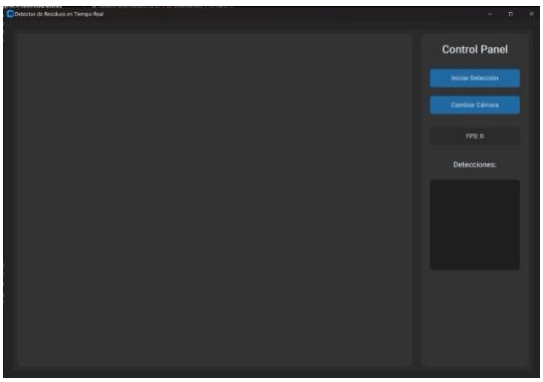
4. Creación de la Interfaz

La creación de la interfaz del proyecto de detección de residuos en tiempo real se llevó a cabo utilizando la biblioteca **customtkinter** para la estructura gráfica y **OpenCV** para la captura y procesamiento de video.

4.1 Diseño de la Estructura General

La interfaz gráfica está compuesta por una ventana principal la cual se divide en:

- Un panel para mostrar el video en tiempo real de la cámara.
- Un panel de control con botones y estadísticas relacionadas con el proceso de detección.



El objetivo principal es proporcionar al usuario una experiencia intuitiva y funcional para realizar detecciones de residuos de forma sencilla.

4.2 Configuración Inicial de la Ventana Principal

La ventana principal se inicializa como un objeto de la clase CTk.

Se define el título de la ventana: *"Detector de Residuos en Tiempo Real"*.

Se establece el tamaño predeterminado: 1200x800.

Se utiliza el modo de apariencia oscura con ctk.set_appearance_mode("dark").

```
self.window = ctk.CTk()
self.window.title("Detector de Residuos en Tiempo Real")
self.window.geometry("1200x800")
ctk.set_appearance_mode("dark")
```

El modelo de detección se carga desde un archivo previamente entrenado

```
self.model = YOLO('yolo_waste_detection/yolov8_training95/weights/best.pt')
```

4.3 Componentes de la Interfaz

4.3.1 Panel de Video

El panel izquierdo de la interfaz está destinado a mostrar el video en tiempo real capturado por la cámara, Mostrando en el frame una etiqueta donde se renderiza el video procesado.

4.3.2 Panel de Control

El panel derecho está diseñado para ofrecer controles y estadísticas al usuario.

1. Botones de interacción:

Iniciar/Detener Detección: Controla el inicio o la pausa de la detección en tiempo real.

Cambiar Cámara: Alterna entre múltiples cámaras conectadas.

```
self.start_button = ctk.CTkButton(
            self.control_frame,
            text="Iniciar Detección",
            command=self.toggle_detection,
            font=("Roboto", 14),
            height=40
        )
        self.start_button.pack(pady=10, padx=20, fill="x")

        self.camera_button = ctk.CTkButton(
            self.control_frame,
```

```
            text="Cambiar Cámara",
            command=self.toggle_camera,
            font=("Roboto", 14),
            height=40
        )
self.fps_label = ctk.CTkLabel(
            self.stats_frame,
            text="FPS: 0",
        )
self.detections_text = ctk.CTkTextbox(
            self.control_frame,
            height=200,
        )
```

2. Estadísticas:

FPS: Muestra los cuadros por segundo procesados en tiempo real.

Detecciones: Lista los objetos detectados junto con su categoría y confianza.

4.4 Integración de la Cámara

La cámara se configura para capturar video con OpenCV:

- Se inicializa con una resolución de 640x480 píxeles.
- La función toggle_camera permite cambiar entre diferentes dispositivos de captura

```
def setup_camera(self):
self.cap = cv2.VideoCapture(0)
self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

4.5 Detección en Tiempo Real

El núcleo de la funcionalidad de detección se basa en:

1. Capturar los cuadros de la cámara.
2. Procesarlos con el modelo YOLO para identificar residuos.
3. Dibujar las detecciones y actualizarlas en la interfaz.

```
self.detection_thread =
threading.Thread(target=self.detect_video)
self.detection_thread.daemon = True
self.detection_thread.start()
```

La detección ocurre en un hilo separado para evitar bloquear la interfaz gráfica

Las detecciones se clasifican en tres categorías principales: reciclables, orgánicos y no reciclables. La lista de detecciones se actualiza dinámicamente en el panel de control.

4.6 Ejecución de la Interfaz

El método run lanza la ventana principal y permite al usuario interactuar con la aplicación

```
if __name__ == "__main__":
    app = TrashDetector()
    app.run()
```

4.7 Mantenimiento y Limpieza

Al cerrar la aplicación, los recursos de la cámara se liberan automáticamente para evitar problemas de acceso

```
if __name__ == "__main__":
    app = TrashDetector()
    app.run()
```

5. Conclusión.

A pesar de los desafíos técnicos enfrentados, como la optimización del rendimiento en tiempo real y la correcta clasificación de residuos menos comunes, el proyecto logró cumplir sus objetivos principales de manera satisfactoria. Este trabajo demuestra el impacto significativo que la inteligencia artificial y el diseño de software pueden tener en la resolución de problemas ambientales, proporcionando una base sólida para futuras mejoras y expansiones del sistema. Además, reconocemos el potencial para ampliar este proyecto considerablemente, como la creación de datasets propios con una mayor cantidad de categorías y divisiones, entre otras posibilidades.

Referencias

Girshick, R., Donahue, J., Darrell, T., Malik, J., & Berkeley, U. (s. f.). *Rich feature hierarchies for accurate object detection and semantic segmentation: R-CNN: Regions with CNN features*. https://arxiv.org. https://arxiv.org/pdf/1311.2524