ATRIA INNOVATION

Quality Innovation Award 2021

Índice

1. Introducción: SIARA

Como se ha comentado anteriormente, el sistema SIARA mejora la eficiencia de los procesos de las plantas de selección de residuos de envases de plástico, latas y briks a través de un sistema de Visión Artificial basado en Deep Learning (redes neuronales) para la detección y retirada de residuos que pueden entorpecer el proceso de selección y recuperación.

Durante la realización de este proyecto se realizó su propia página http://siaraproject.es/ para difundir el proyecto. En esta, se actualizan los avances del proyecto y los próximos pasos.

Figura 1: Página web SIARA

2. Las fases del desarrollo

A continuación, se muestran las **5 fases distintas** en las que se ha dividido el proyecto.

1. **Definición de objetivos y condiciones:** Estudio entre todos los participantes de las condiciones, dónde se va a instalar el sistema de adquisición y sus dimensiones; qué medidas de evaluación se van a utilizar para el Proyecto, qué herramientas de software…
2. **Diseño e instalación del sistema de visión artificial.** Diseño de la estructura que integrará todos los sistemas y selección los sensores a incluir y su posición, el ordenador de control, el sistema de iluminación y cómo se van a conectar todos los sistemas.
3. **Creación de la base de datos.** Captura de imágenes en el entorno real y etiquetado de los objetos.
4. **Desarrollo y entrenamiento de la visión artificial** Selección de la arquitectura de Inteligencia Artificial a implementar y los parámetros iniciales. Entrenamiento de dicha arquitectura en el ámbito particular del Proyecto con el dataset creado en la fase anterior y evaluar.
5. **Evaluación real y mejora continua:** Implementación y puesta en marcha del algoritmo desarrollado en la fase 4 en el sistema de adquisición instalado en la planta. Evaluar su funcionamiento y monitorizar errores para realimentar el entrenamiento.

Figura 2: Fases del proyecto

Una de las partes más importantes del proyecto es la realización del código. A continuación, se detallan los pasos más importantes a la hora de realizar este proyecto.

3. Desarrollo del código de Deep Learning

Para el desarrollo del sistema de visión los algoritmos se han desarrollado en el lenguaje Python para cada una de las distintas fases del proceso. Según la función y objetivo que tienen estos programas, se pueden distinguir 4 fases o módulos cada uno de los cuales está formado por varios programas:

- **Toma de datos**: adquisición de imágenes con residuos.
- **Tratamiento de datos**: operaciones sobre los datos obtenidos para facilitar y mejorar los resultados del entrenamiento.
- **Entrenamiento**: creación y aprendizaje de modelos de Redes Neuronales (de varios tipos) con los datos anteriormente obtenidos.
- **Evaluación**: métricas con los resultados obtenidos del desempeño de los modelos entrenados.

Cada uno de estos módulos está formado por varios programas con una función específica.

Para el proceso de entrenamiento, se utilizan, como se ha mencionado, Redes Neuronales. Estas Redes, son modelos de Inteligencia Artificial que emulan el comportamiento del cerebro humano. Están compuestas por diversos atributos que se modifican, simulando el proceso de aprendizaje, en función de los datos recibidos y los resultados que quieren conseguir.

4. Toma de Imágenes

Para el desarrollo de una red neuronal, son necesarios una gran cantidad de datos, cuantos más datos se introduzcan, más fiabilidad tendrá nuestra red neuronal. En este caso, los datos son las imágenes de los residuos. Para el desarrollo del sistema, se han etiquetado y utilizado un total de 23.500 imágenes etiquetadas.

Para la obtención de las fotografías de los residuos se desarrolló un código en Python, encargado de gestionar la toma de imágenes. También se desarrolló una librería de control para controlar de forma sencilla la cámara utilizada para este proceso.

La librería de control de la cámara, se trata de un programa de Python que contiene las funciones básicas necesarias para el control de la cámara seleccionada:

- Funciones de inicialización de la cámara
- Función de carga de parámetros
- Función para la toma de una imagen

- Función de cierre de la cámara

Por otro lado, el programa de gestión se encarga de utilizar la librería, anteriormente descrita, para tomar imágenes y guardarlas en el disco duro del ordenador. Este programa toma imágenes cada 3 segundos, para evitar de este modo guardar demasiadas fotografías de momentos muy similares. De esta forma, además, se evita sobrecargar la cámara.

Para los casos en los que la cinta transportadora de residuos está detenida, se implementó una función que compara 2 imágenes. Si el programa observa que estas 2 imágenes son muy parecidas entre sí, no guardara una nueva imagen, así se evita llenar el disco del ordenador con imágenes que no aportan información nueva. En la Figura 3, se puede observar el programa en ejecución.

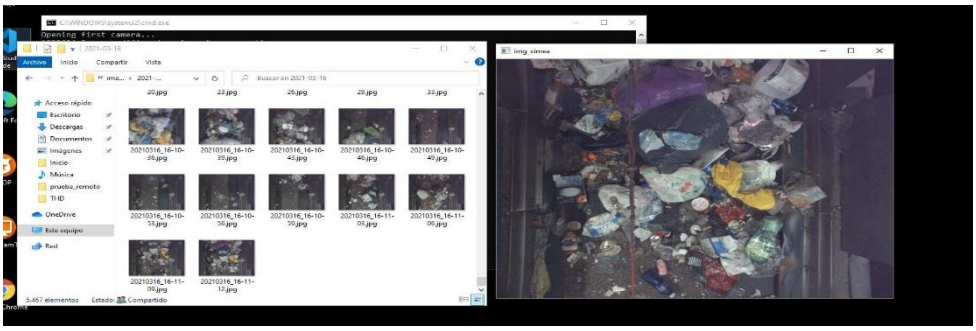

Figura 3: Toma de imágenes en ejecución

5. Tratamiento de datos

Por otro lado, se ha desarrollado otro programa en Python para modificar las imágenes obtenidas y optimizar el entrenamiento. Debido a los cambios que se realizan, también puede ser necesario adaptar las etiquetas (posición de los objetos de interés en las imágenes) al nuevo formato, debido a cambios en el tamaño de la imagen. En la Figura 4 se encuentra la función principal del programa, en el que se itera sobre las antiguas etiquetas creando las nuevas.

Figura 4: Izquierda: imagen original sin modificar. Derechas: Imagen tras ser procesada por el programa de eliminación de márgenes

Para poder realizar el posterior entrenamiento de la red, es necesario etiquetar los datos obtenidos. Para este tipo de sistemas, la función de etiquetado consiste en señalar en cada imagen la posición de los objetos que se desean encontrar, además de indicar la clasificación de dichos objetos. Estos datos son introducidos en las Redes Neuronales, que adaptan sus diversos atributos para poder detectar los objetos que se han señalado.

Para realizar el etiquetado se ha hecho uso de CVAT, un programa de etiquetado de imágenes que permite señalar de forma intuitiva la localización y tipos de objetos de diversas imágenes, tal y como puede observarse en la Figura 5, además de generar los archivos que serán introducidos en las Redes Neuronales.

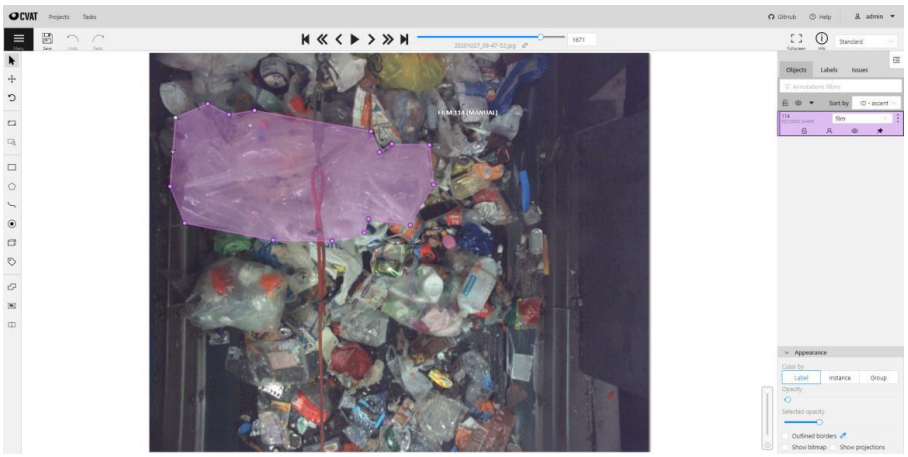

Figura 5: Etiquetado de imágenes mediante CVAT

6. Entrenamiento del algoritmo

Para el entrenamiento de las Redes Neuronales se han generado comandos de ejecución cuya principal función es referenciar a los datos generados y a los ficheros de configuración necesarios para realizar dicho entrenamiento. Se han utilizado principalmente dos tipos de redes, Tensorflow y YOLO. Debido a esto, sus ficheros de configuración y comandos son diferentes.

```
D:/Proyectos/THD_Ecoembes/codigo/THD/THD_venv/Scripts/python
D:/Tensorflow/models/research/object_detection/model_main_tf2.py
--pipeline_config_path=D:/Proyectos/THD_Ecoembes/codigo/THD/Tensorflow/experiments/training/
ssd_resnet50_v1_fpn_1024x1024_coco17_tpu-8/THD.config
--model_dir=D:/Proyectos/THD_Ecoembes/codigo/THD/Tensorflow/experiments/evaluation
--alsologtostderr
```

```
darknet.exe detector train D:/Proyectos/THD_Ecoembes/codigo/THD/YOLO/data/THD.data
D:/Proyectos/THD_Ecoembes/codigo/THD/YOLO/data/yolov4-leaky-416.cfg
D:/Proyectos/THD_Ecoembes/codigo/THD/YOLO/data/yolov4-leaky-416.weights -map -gpus 0
```

Figura 6: Muestra de comandos de ejecución empleados (Arriba: Tensorflow, Abajo: YOLO)

En la Figura 6 se pueden observar ejemplos de los comandos empleados para ejecutar el entrenamiento. En ambos casos al principio se indica el programa a ejecutar (siendo distintos para Tensorflow y YOLO) para posteriormente indicar los ficheros de configuración que contienen el resto de información relevante, en el caso de Tensorflow tambien se indica al final la ruta en la que deseamos que guarde los parámetros entrenados.

7. Evaluación del funcionamiento del algoritmo

Una vez entrenado el modelo de detección de objetos, se evalúa su funcionamiento extrayendo diversas métricas (mAP, LRP, tiempo medio…). Al igual que con el entrenamiento también existen diferencias en el código según el detector de objetos usado, ya que los modelos entrenados se cargan con funciones diferentes y las etiquetas de los datos de test también son tratadas de con otros formatos. A continuación, se muestran las métricas obtenidas con dos de las Redes Neuronales utilizadas, MobileNet y YOLOv4.

- SSD MobileNet V1 FPN 640x640
  - AP:
    - AP@Barquilla: 64 %
    - AP@Cartón: 53 %
    - AP@Film: 60 %
    - mAP: 59 %
  - LRP:
    - LRP@Barquilla: 43 %
    - LRP @Cartón: 49 %
    - LRP @Film: 56 %
    - mLRP: 50 %
  - Tiempo por imagen: 0,11 segundos
  - Matriz de Confusión:

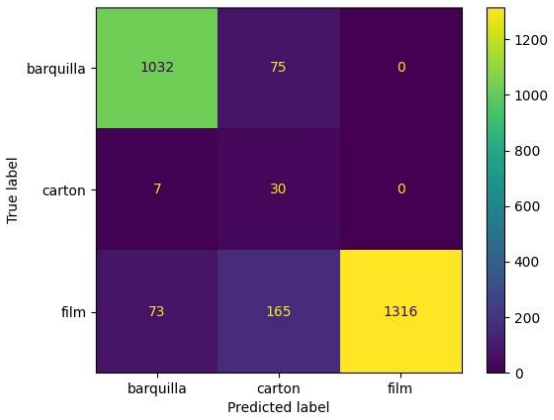

Figura 7: Matriz de confusión Mobilenet

- YOLOv4-Leaky-416
  - AP:
    - AP@Barquilla: 90 %
    - AP@Cartón: 86 %
    - AP@Film: 80 %
    - mAP: 88 %
  - LRP:
    - LRP@Barquilla: 35 %
    - LRP @Cartón: 41 %
    - LRP @Film: 36 %
    - mLRP: 37 %
  - Tiempo por imagen: 0,04 segundos
  - Matriz de Confusión:

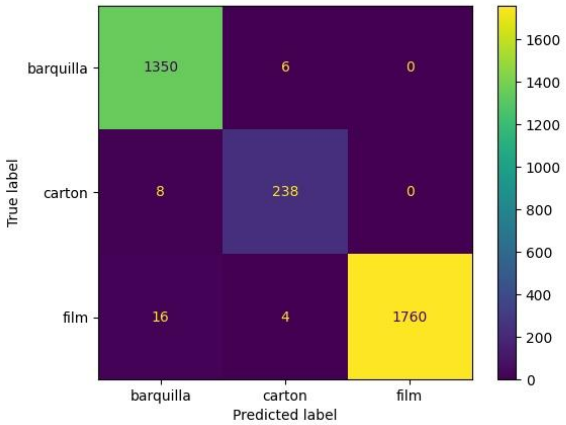

Figura 8: Matriz de confusión YOLOv4

La AP representa la precisión ponderada para cada una de las categorías seleccionadas, indicando el porcentaje de acierto en clasificación, la mAP es la media de todas las AP. Por otro lado, LRP es una métrica que cuantifica el error de la capacidad de localización de los diferentes tipos de elemento, cuanto menor es su valor mejor son los resultados.

Las matrices de confusión indican el número de elementos detectados, como han sido clasificados y a que grupo pertenece realmente.

Como ejemplo, se puede observar en los resultados de MobileNet que esta red, tiene una mayor facilidad para clasificar Barquillas y Films que Cartones, al tener este un menor AP. Si se comparan los resultados con los correspondientes LRP, se comprueba que los elementos con mayor LRP (Cartón) se corresponden con los de menor AP. Al indicar LRP una mejor detección cuanto menor es su valor, se puede concluir que MobileNet necesita que el proceso de detección se realice lo más aproximado posible para poder conseguir una correcta clasificación. Finalmente, como se puede ver en la Matriz de Confusión, el número de cartones encontrados

por MobileNet es muy bajo, además de confundir un alto porcentaje con Barquillas (como se podía intuir al ver la AP), confirmando que este modelo de red Neuronal tiene grandes dificultados con ese elemento en concreto, mientras que, los resultados de YOLOv4, muestran un desempeño mucho más consistente con todos los tipos de residuos.

8. Ejecución del algoritmo

Para comprobar en tiempo real cuanto se tarda en detectar objetos en cada imagen y observar que objetos se detectan sobre la imagen, se ha creado un programa cuya función es ejecutar el modelo definido sobre una serie de imágenes y mostrar los resultados. Al igual que en el proceso de evaluación, el método de carga y los parámetros a definir son diferentes según el detector utilizado.

Durante la ejecución, se cargan las imágenes de la ruta definida y se envían al modelo declarado. Una vez obtenidos los objetos detectados por el modelo, si es que los hay, estos se señalan sobre la propia imagen (Figura 9). Para facilitar la labor del usuario, se ha hecho que solo se aplique el detector sobre una nueva imagen al pulsar la tecla *intro*, además de poder detener la ejecución al pulsar *esc*.

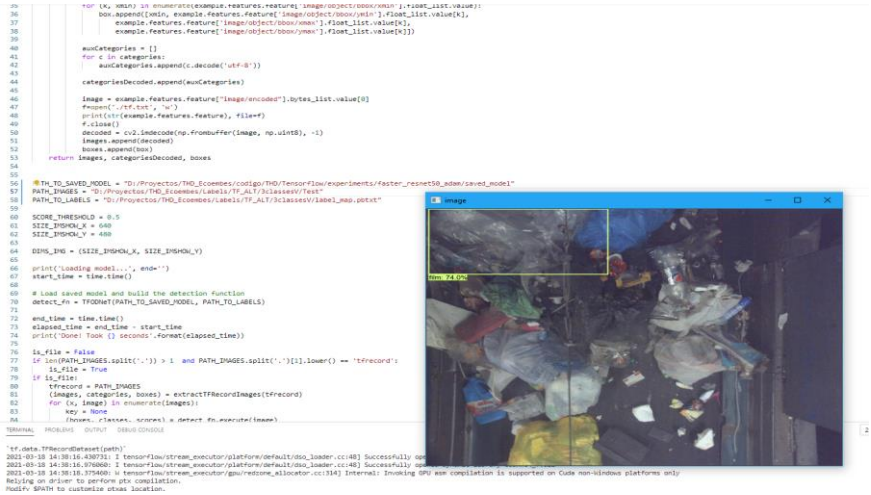

Figura 9: Ejecución en tiempo real de una red entrenada

9. Resultados

Al realizar el entrenamiento, se observó que había un número bajo de muestras de algunos de los residuos, haciendo inviable su entrenamiento, por este motivo se limitó la detección a 3 tipos de residuos, barquilla, Cartón y Film, siendo estos los que tenían un mayor número de

muestras. El entrenamiento con estos elementos obtuvo resultados más consistentes, en particular con las Redes Neuronales YOLOv4 y MobileNet. Esto se debe a que son redes de menor tamaño y que trabajan con imágenes pequeñas, lo que les permite aprender más rápido, lo cual, para este sistema, es beneficioso debido al bajo número de muestras.

En el caso de MobileNet (Figura 10) puede observarse una mayor dificultad en la detección de cartones, esto se debe al bajo número de muestras de este elemento, siendo el que menos cantidad de imágenes tenia de los 3 seleccionados. Por otro lado, YOLOv4 (Figura 11) es capaz de conseguir buenos resultados con los 3 elementos y con un gran porcentaje de acierto en clasificación. Esta diferencia se debe a que estos modelos utilizan distintos algoritmos de detección siendo el de YOLO más robusto para este tipo de aplicación.

A continuación, se muestran ejemplos de las detecciones realizadas con MobileNet y YOLOv4:

- **Mobilenet**:

Figura 10: Detecciones de MobileNet

- **YOLOv4**:

Figura 11: Detecciones de YOLOv4