TRABAJO FIN DE MÁSTER

Máster Big Data y Business Analytics

Curso 2022-2023

Object Detection for Waste Management

UNIVERSIDAD COMPLUTENSE DE MADRID

Alumnos: Esteban Joaquín Jiménez Párraga, Sofía Castro Domínguez, Eduardo Maza Ortega, Julio Martínez Sánchez, César Sánchez Martín

Septiembre, 2023

Índice general

1 Introducción **8**

1.1 Motivación 8

1.2 Estructura de la memoria 8

2 Conceptos previos **10**

2.1 Problemas de detección de objetos 10

3 Implementación **15**

3.1 Modelo de clasificación 15

3.2 Modelo de detección 16

3.2.1 Instalación 16

3.2.2 Preparación de los datos 17

3.2.3 Configuración del modelo 17

4 Evaluación Experimental **18**

4.1 Datos de prueba 18

4.2 Entorno de desarrollo 18

4.3 Evaluación del modelo de clasificación 19

4.3.1 Entrenamiento y testeo 19

4.3.2 Resultados obtenidos 19

4.4 Evaluación del modelo de detección de objetos 21

4.4.1 Entrenamiento y testeo 21

4.4.2 Resultados obtenidos 21

5 Conclusiones y trabajo futuro **24**

5.1 En el plano académico 24

5.2 En el plano de investigación 25

5.3 Trabajo a futuro 26

Anexo **27**
Modelo de clasificación 27
Librerías empleadas 30
Librerías para la Adquisición y Preprocesamiento de Imágenes 30
Librerías para el Etiquetado de Datos 31
Librerías para el Entrenamiento del Modelo 31
Código usado 35
Configuración 35

Lista de acrónimos **36**

Bibliografía **37**

Índice de figuras

2.1 Resultado esperado de la primera detección. 11
2.2 Estructura de las R-CNN. 11
2.3 Criterio IoU. 12
2.4 Criterio NMS. 12
2.5 Criterio NMS. 13
2.6 Arquitectura EfficientDet. 13

4.1 Muestra aleatoria de imágenes del dataset 19
4.2 Matríz de confusión modelo de clasificación. 20
4.3 Gráficas de pérdida y precisión 21
4.4 Detección de residuos 22

5.1 Predicciones incorrectas 28
5.2 Predicciones correctas 29
5.3 Software empleado en el etiquetado. 34
5.4 Imagen previa al etiquetado. 34
5.5 Imagen después del etiquetado. 34

Índice de Código

3.1 Creación modelo 15
3.2 Cálculo probabilidades 16
4.1 Training 21
5.1 Bucle de entrenamiento y evaluación 28
5.2 config 35

Resumen

Nuestro proyecto surge de una profunda preocupación por el medio ambiente y un fuerte sentimiento ecologista. La idea que nos movilizó fue la de contribuir a la sociedad de una manera significativa y positiva, abordando el problema de la clasificación de residuos. Nuestro objetivo era desarrollar una solución que permitiese a las personas clasificar los residuos con solo tomar una fotografía.

Como primer enfoque de este desafío, decidimos explorar el uso de modelos de clasificación. Esta etapa inicial nos permitió comprender la complejidad del problema y adquirir los conocimientos fundamentales sobre el reconocimiento de objetos en imágenes. A medida que avanzábamos, nos dimos cuenta de la necesidad de abordar la detección de objetos de manera más precisa.

Por lo tanto, decidimos ampliar nuestro enfoque y explorar diversos modelos de detección de objetos. Esta decisión nos permitió desarrollar una solución más completa y versátil que podría beneficiar a la sociedad al proporcionar una herramienta para la clasificación de residuos de manera efectiva y sostenible.

En resumen, nuestro proyecto se originó por un profundo sentimiento ecologista y se enfocó en la creación y evaluación de modelos de detección de objetos para ayudar a la sociedad en la clasificación de residuos a partir de fotografías. A medida que avanzamos en este viaje, nuestro compromiso con la causa ecológica sigue siendo el motor que impulsa nuestros esfuerzos por contribuir a un futuro más sostenible.

Abstract

Our project arises from a deep concern for the environment and a strong ecological sentiment. The idea that motivated us was to contribute to society in a meaningful and positive way by addressing the issue of waste classification. Our goal was to develop a solution that would allow people to classify waste simply by taking a photograph.

As the initial focus of this challenge, we decided to explore the use of classification models. This initial stage allowed us to grasp the complexity of the problem and acquire fundamental knowledge about object recognition in images. As we progressed, we realized the need to address object detection more precisely.

Therefore, we decided to broaden our focus and explore various object detection models. This decision allowed us to develop a more comprehensive and versatile solution that could benefit society by providing an effective and sustainable waste classification tool.

In summary, our project originated from a deep ecological sentiment and focused on the creation and evaluation of object detection models to assist society in waste classification from photographs. As we continue on this journey, our commitment to the ecological cause remains the driving force behind our efforts to contribute to a more sustainable future.

Glosario

- Detección de objetos (*Object detection*)
- Modelo de detección de objetos (*Object detection model*)
- Bounding box (*Cuadro delimitador*)
- Non-Maximum Suppression (NMS)
- Anchor boxes (*Cajas ancla*)
- Feature map (*Mapa de características*)
- Predicción de clases (*Class prediction*)
- Regresión de coordenadas (*Coordinate regression*)
- Redes neuronales convolucionales (*Convolutional Neural Networks, CNN*)
- Aprendizaje profundo (*Deep learning*)
- YOLO (*You Only Look Once*)
- SSD (*Single Shot MultiBox Detector*)
- R-CNN (*Region-based Convolutional Neural Network*)
- Máscara de segmentación (*Segmentation mask*)
- Precisión de detección (*Detection accuracy*)

Capítulo 1

Introducción

Este capítulo introductorio ofrece una breve contextualización del proyecto, indicando los objetivos del mismo, así como una descripción de la estructura del documento.

1.1 Motivación

Desde nuestro grupo, hemos planteado un escenario de negocio en el que ONGs, agencias gubernamentales o cualquier empresa busca potenciar el ecologismo y una sociedad verde a partir de un reciclaje responsable por parte de los ciudadanos. Gracias al modelo que vamos a tratar de desarrollar en este trabajo, seremos capaces de dar a los usuarios una experiencia sencilla con la que poder fotografiar el residuo producido y saber con una fiabilidad alta en qué contenedor deben depositarlo.

1.2 Estructura de la memoria

Esta memoria está organizada en cinco capítulos, contando con el presente, que se describen brevemente a continuación:

- El **capítulo 1** ayuda a contextualizar el proyecto, describiendo los objetivos del mismo.
- El **capítulo 2** realiza una introducción a conceptos previos necesarios para entender el desarrollo del trabajo. En este capítulo se explican las distintas arquitecturas de redes neuronales, así como los posibles modelos para problemas de ”Object Detection”. Se trata de un breve estudio teórico del problema así como de los distintos enfoques que han servido para atajarlo.
- El **capítulo 3** describe en detalle la implementación de los distintos modelos llevados a cabo.

- El **capítulo 4** se dedica a los resultados experimentales. Comienza con una descripción del conjunto de datos y del entorno utilizado para posteriormente discutir los resultados obtenidos de las diferentes estrategias de modelización propuestas.
- El **capítulo 5** presenta las conclusiones obtenidas del trabajo, y también proporciona una breve descripción de posibles líneas de trabajo futuro.

Capítulo 2

Conceptos previos

En este capítulo se realiza una introducción a aquellas cuestiones directamente relacionadas con el trabajo que se llevó a cabo, y que son necesarias para entender posteriormente los desarrollos realizados.

2.1 Problemas de detección de objetos

El problema de detección de objetos es un problema de los más clásicos del deep learning y computer vision. En este tipo de problemas buscamos un algoritmo capaz de detectar una serie de objetos escogidos para imágenes dadas. Para un humano esto resulta un problema trivial, sin embargo comprendemos la complejidad computacional y estadística que presenta este problema al estudiarla desde un algoritmo de machine learning. Las imágenes actúan como una fuente de información no estructurada que debe ser tratada correctamente para obtener las regiones donde encontramos los objetos deseados para después pasarlos por una red neuronal convolucional (CNN) que los clasifique.

Abordando el primer de estos problemas, ¿cómo podemos asegurarnos de detectar correctamente dónde se halla el objeto en la imagen? La idea es conseguir dibujar un recuadro en torno a la entidad buscada.

Una solución que se planteó para este problema fue crear ventanas deslizantes que ”buscaran” iterando sobre la imagen. Sin embargo, encontramos muchos problemas en este planteamiento: ¿Cómo de grande debe ser esta ventana? ¿Cómo iteramos y obtenemos un buen tiempo de cómputo? ¿Podemos detectar una entidad múltiples veces? ¿Y si dos entidades son muy cercanas y son percibidas como una sola?

Para atajar estas cuestiones se creó una nueva métrica: la mAP (mean average precision) Esta proporciona una medida unificada de precisión que tiene en cuenta tanto la precisión de la localización de los objetos como la precisión de su clasificación. El mAP puede tomar valores en el rango de 0 a 1, donde un valor más cercano a 1 indica un mejor rendimiento del

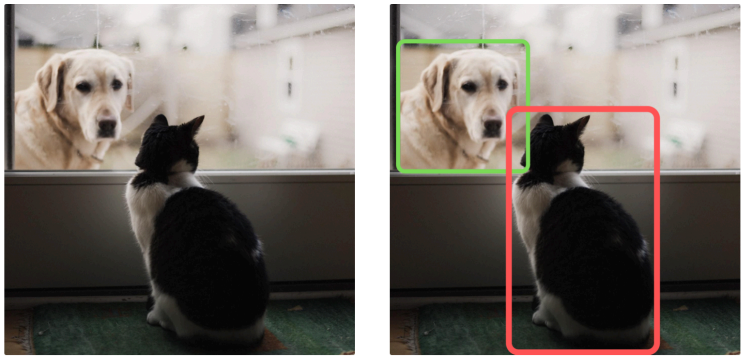

Figura 2.1: Resultado esperado de la primera detección.

modelo. Con esta medida surgen estrategias efectivas para atajar el problema.

En 2014 surgen las "Region Based Convolutional Neural Networks" (R-CNN) con la siguiente propuesta: primero determinar "regiones de interés" dentro de la imagen n y luego realizar la clasificación de imágenes sobre esas áreas usando una red pre-entrenada.

El proceso comienza con la aplicación de un algoritmo inicial en la imagen para identificar áreas de interés potenciales, las cuales pueden llegar a ser alrededor de 2000 regiones de diferentes tamaños. Estas regiones seleccionadas se someten a la red neuronal convolucional (CNN), que actúa como un clasificador binario para validar si pertenecen a las clases correctas y eliminar aquellas detecciones de baja confianza. Finalmente, se utiliza un regresor para ajustar la posición de las regiones seleccionadas.

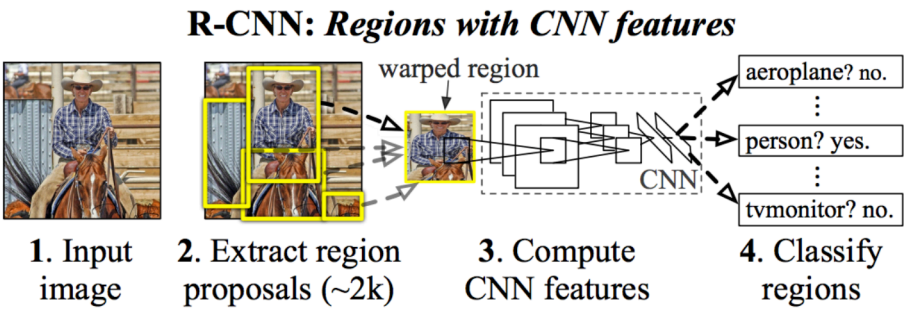

Figura 2.2: Estructura de las R-CNN.

Para evitar problemas de solapamiento de un objeto en múltiples regiones se usa el criterio IoU (Intersection over Union) y NMS (Non-Máximum-Supression). IoU actúa calculando la proporción de la intersección entre ambas regiones respecto a su unión.

NMS nos permite quedarnos de entre muchas cajas que detectaron al mismo objeto y se superponen, con la que mejor se ajusta al resultado mientras eliminamos el resto.

Con estas ideas se consiguió paliar los problemas anteriores, pero el tiempo de cómpu-

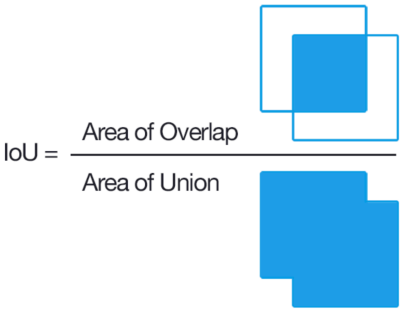

Figura 2.3: Criterio IoU.

Figura 2.4: Criterio NMS.

to era muy elevado por lo que se desarrollaron dos algoritmos mejorados: Fast R-CNN que mejora el algoritmo inicial haciendo reutilización de algunos recursos como el de las features extraídas por la CNN agilizando el entreno y detección de las imágenes y Faster R-CNN que incorpora la capacidad de proponer regiones de interés para mejorar la detección de estas.

En 2016 se da el siguiente gran paso en object detection, el algoritmo YOLO. El algoritmo YOLO (You Only Look Once) prescinde de las iteraciones y realiza la detección en un solo cálculo. Gracias a esto es un algoritmo mucho más rápido y permitió nuevos usos como aplicar este tipo de algoritmos a vídeo.

En YOLO se genera una rejilla sobre la imagen y en lugar de iterar a través de cada celda de la rejilla para realizar las predicciones, la red CNN aprende a generar detecciones basadas en la información extraída de la imagen completa combinando la rejilla con las regiones probables en las que se encuentra un objeto.

La arquitectura de la red YOLO se basa en una red convolucional GoogleNet de 24 capas convolucionales bautizada cono DarkNet. Se caracteriza porque da una única salida en la que se muestra la clasificación del objeto y el posicionamiento del mismo.

En 2019 el equipo de Google Brain publicó la familia de modelos EfficientDet. Esta familia es un conjunto escalable de modelos de detección de objetos. Se parte de un modelo inicial D0 y manteniendo la arquitectura se van incrementando la profundidad y el numero de capas creando los modelos D1 a D7. Cada modelo consigue resultados mas precisos que el anterior

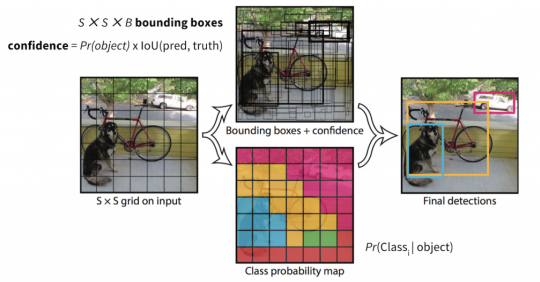

Figura 2.5: Criterio NMS.

pero a costa de realizar más operaciones y por tanto ser más lento. Así se obtiene un conjunto de modelos y se puede escoger el más adecuado según se quiera primar la precisión o la velocidad en función de la tarea u objetivo que se persiga.

La arquitectura EfficientDet consta de varias partes separadas. En la primera se utiliza una red EfficientNet, una red convolucional que genera mapas de características o features de distinto tamaño a partir de las imágenes. A continuación estas features pasan por una serie de capas BiFPN (Weighted Bi-directional Feature Pyramid Network), que realizan una combinación de las características. Por último, la salida de estas capas se le pasa tanto a una red encargada de predecir las categorías como a otra encargada de predecir los recuadros donde se encuentran los objetos.

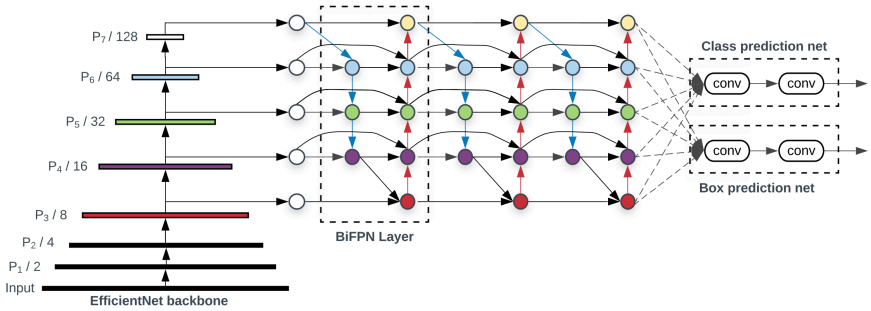

Figura 2.6: Arquitectura EfficientDet.

Estas son algunas de las mejores aproximaciones a los problemas de object detection, aunque cabe mencionar algunas otras redes que también han dado buenos resultados como el modelo Single Shot Detector que cuenta con una estructura piramidal en su CNN en la que las capas van disminuyendo gradualmente. También tenemos la RetinaNet que también tiene una estructura piramidal y añade una función de pérdida en sus cálculos. En 2019 aparece Google Spinet que rompe con la estructura piramidal y propone una arquitectura novedosa llamada “scale-permuted” en la que se alternan diversos tamaños en las convoluciones. Por último, mencionamos Facebook: DETR, que incorpora técnicas de NLP para problemas de

object detection.

En este trabajo se utiliza la arquitectura EfficientDet para el modelo de detección de objetos ya que es la que mejores valores de AP (Average Precision) consigue y se encuentra implementada en Tensorflow.

Capítulo 3

Implementación

En este capítulo se describen las implementaciones de los distintos modelos probados en este trabajo. Al final del capítulo se indica la referencia al repositorio donde se pueden encontrar todos los desarrollos y en el anexo se explican brevemente las librerías empleadas.

3.1 Modelo de clasificación

Como primera aproximación, tal y como se explicó en capítulos anteriores se implementó un modelo de clasificación de imágenes utilizando un módulo pre-entrenado de TensorFlow Hub.

El procesado de las imágenes comienza definiendo un marcador de posición para las imágenes codificadas. Se decodifican las imágenes en formato JPEG y se convierten en tensores de punto flotante, que se redimensionan al tamaño esperado por el módulo de imágenes. El resultado es un lote de imágenes preprocesadas. A continuación, se aplica el módulo de imágenes a estas imágenes preprocesadas para extraer las características de las mismas, que servirán como entrada para el modelo de clasificación.

A continuación la función que construye el modelo de clasificación. En este caso, se utiliza una única capa densa (lineal) para la clasificación.

```
def create_model(features):
  # Single linear layer
  layer = tf.layers.dense(inputs=features, units=NUM_CLASSES, activation=
    None)
  return layer
```

Listing 3.1: Creación modelo

En el siguiente fragmento de código se puede ver como se obtienen los'logits' del modelo antes de aplicar la función softmax y como se utilizan para calcular la pérdida de entropía cruzada. Esta pérdida se minimiza durante el entrenamiento utilizando el optimizador de descenso de gradiente estocástico (SGD).

Para evaluar el modelo, se aplicará la función softmax a los logits y de esta manera se obtienen las probabilidades de clase. Se calculará la clase predicha como aquella con la probabilidad más alta. La precisión del modelo se mide mediante la proporción de predicciones correctas en el conjunto de datos.

```
logits = create_model(features)
labels = tf.placeholder(tf.float32, [None, NUM_CLASSES])

cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits,
    labels=labels)
cross_entropy_mean = tf.reduce_mean(cross_entropy)

optimizer = tf.train.GradientDescentOptimizer(learning_rate=LEARNING_RATE)
train_op = optimizer.minimize(loss=cross_entropy_mean)

probabilities = tf.nn.softmax(logits)

prediction = tf.argmax(probabilities, 1)
correct_prediction = tf.equal(prediction, tf.argmax(labels, 1))

# To eval on our test set.
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Listing 3.2: Cálculo probabilidades

3.2 Modelo de detección

Para conseguir este modelo se ha realizado un fine-tuning de un modelo previamente entrenado disponible en Tensorflow. El fine-tuning consiste en partir de un modelo general pre-entrenado con una gran cantidad de imágenes y entrenarlo con un dataset pequeño para lograr un modelo más específico. Para la creación de este modelo es necesario la API 'Tensorflow Object Detection API' y 'Tensorflow 2' anteriormente descritas. El código utilizado para la creación de dicho modelo se puede encontrar en: Object Detection Train model.ipynb. Los pasos que hemos realizado para la creación de este modelo son los siguientes:

3.2.1 Instalación

En un paso inicial, debemos de crear el entorno de desarrollo e incorporar todas las herramientas necesarias para el buen funcionamiento de todos los procesos posteriores. Inicialmente descargamos e instalamos en nuestro entorno de ejecución todas las librerías necesarias, de la que cabe destacar 'Tensorflow', con la que tendremos acceso a la API de detección de objetos. La version de Tensorflow utilizada es la 2.8.0 mientras que la de la API de object detection es la última disponible y se obtiene clonando el repositorio de Github.

3.2.2 Preparación de los datos

Para este modelo es necesario un dataset diverso, además de tener todas las imágenes de este etiquetadas. Este proceso de etiquetación manual de las imágenes ya se ha explicado anteriormente, por lo que tenemos nuestro dataset inicial con las imágenes clasificadas. Con este dataset inicial creamos los conjuntos de train y test. Para ello, utilizamos un script que separa las imágenes en train y test de forma aleatoria en una proporción 80/20.

Una vez que las imágenes están separadas hay que transformar los datos a formato binario, los cuales son necesarios para el entrenamiento del modelo de Tensorflow. Esta transformación de los datos se realizó mediante la ejecución de un script externo, obteniendo los ficheros .record de train y test.

Finalmente es necesario un fichero 'labelmap'. Este fichero contiene las diferentes categorías de objetos que el modelo debe clasificar asignando un número a cada una. En este caso, vamos a trabajar con las siguientes etiquetas: 'cardboard', 'glass', 'metal', 'paper' y 'plastic'.

3.2.3 Configuración del modelo

En el entrenamiento se ha partido de un modelo EfficientDet D0 preentrenado con el dataset COCO 2017. Este dataset contiene más de 200 000 imágenes y 80 categorías diferentes de objetos y constituye el marco principal utilizado para evaluar modelos de detección de objetos. Para este pre-entrenamiento se utilizaron 300 epochs y un batch size de 1200, obteniendo un resultado de AP (Average Precision) en test de 34.6. El resultado del entrenamiento son los valores de los 3.9 millones de parámetros de la red entrenada. Estos están disponibles en forma de fichero checkpoint para ser utilizados como punto de partida en el proceso de fine-tuning en la web de Tensorflow. En este caso se conservan todos los valores de los parámetros del modelo excepto los correspondientes a las ultimas capas del mismo donde se predicen las categorías y los recuadros donde se encuentran los objetos. Para estas capas, por tanto, el entrenamiento se hace partiendo desde cero. Los parámetros de la red que genera features y las capas BiFPN sí se conservan ya que al haber sido entrenadas con muchas imágenes con objetos de muchas categorías distintas son generales y sirven también para la detección de residuos.

Para controlar las opciones de entrenamiento se utiliza el archivo .config disponible en la web de Tensorflow correspondiente al modelo EfficientDet D0. Este archivo se tunea para indicar la ubicación de los archivos train.record y test.record con los que entrenar. También se establecen el numero de clases, el batch size, numero de steps y checkpoint de partida utilizando código para la configuración.

Capítulo 4

Evaluación Experimental

Esta sección se dedica a la evaluación experimental de las propuestas anteriores. El capítulo comienza con una descripción del dataset y entorno de pruebas, para después mostrar los resultados obtenidos. El principal objetivo de este capítulo es evaluar los resultados obtenidos con los diferentes modelos propuestos.

4.1 Datos de prueba

El dataset [4.1] que hemos empleado para entrenar los modelos contiene un total de 2503 imágenes de residuos. Los objetos que se muestran en dichas imágenes pueden pertenecer a las siguientes 6 categorías: 'batteries', 'cardboard', 'glass', 'metal', 'paper', 'plastic'. Se ha escogido dicho dataset dado que cuenta con una amplia variedad de imágenes. Además, dichas imágenes aportan gran cantidad de información a la fase de entrenamiento dado que utilizan distintos objetos para representar la misma clase y distintas perspectivas del mismo.

4.2 Entorno de desarrollo

Se ha empleado Google Colab, un entorno de desarrollo en la nube basado en Google Drive que permite a los usuarios crear y ejecutar cuadernos interactivos de tipo Jupyter Notebook en línea. Este entorno ofrece acceso a recursos de hardware como unidades de procesamiento gráfico (GPU) y unidades de procesamiento tensorial (TPU), lo que facilita la ejecución de cálculos intensivos, especialmente en el ámbito del aprendizaje automático. Para el entrenamiento del modelo de Object Detection se ha precisado una GPU Nvidia T4.

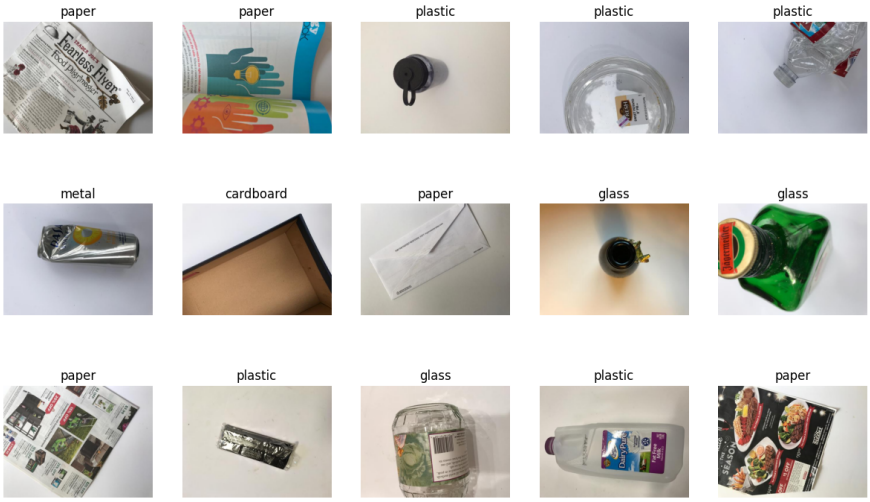

Figura 4.1: Muestra aleatoria de imágenes del dataset

4.3 Evaluación del modelo de clasificación

4.3.1 Entrenamiento y testeo

El proceso comienza con la definición de los parámetros clave, como son el número de pasos de entrenamiento, el tamaño de lote y la frecuencia de evaluación.

Se procede entonces a la inicialización de una sesión de TensorFlow, seguida por la ejecución de un bucle de entrenamiento [5.1] que se repite un número predeterminado de veces. Durante cada iteración del bucle, se obtiene un lote aleatorio de ejemplos de entrenamiento, se calcula la pérdida correspondiente al entrenamiento, se efectúa la actualización de los parámetros del modelo y se mide la precisión del entrenamiento.

Con intervalos regulares, el código evalúa el rendimiento del modelo en un conjunto de prueba independiente. Se realiza el cálculo de la pérdida y la precisión en este conjunto de datos de prueba.

Finalmente, se registran las métricas de rendimiento, como la precisión y la pérdida, para su análisis posterior.

4.3.2 Resultados obtenidos

Como se puede ver en la gráfica [4.3] la precisión aumenta a medida que avanza el entrenamiento, pasando de un 24.85% en el paso 0 a un 70.71% en el paso 99. Esta mejora indica que el modelo está aprendiendo y ajustándose mejor a medida que se le proporciona más entrenamiento.

En términos generales, una precisión del 70.71% se considera aceptable, pero su utilidad

depende del contexto y de los requisitos específicos de la aplicación. También es importante tener en cuenta la matriz de confusión [4.2], que revela el rendimiento del modelo en la clasificación de clases individuales. La matriz de confusión muestra que algunas clases, como ”paper” y ”cardboard”, tienen puntajes F1 más altos (0.82 y 0.78, respectivamente), lo que indica una mejor capacidad del modelo para identificar estas clases en comparación con otras, como ”plastic” y ”glass”, que tienen puntajes F1 más bajos (0.56 y 0.6, respectivamente).

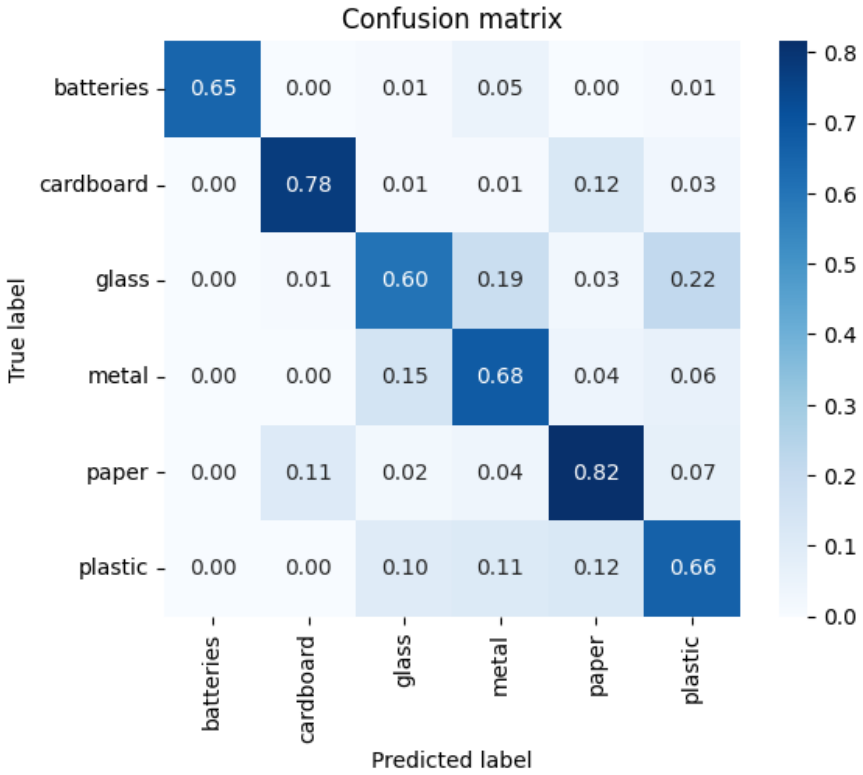

Figura 4.2: Matríz de confusión modelo de clasificación.

Los resultados obtenidos son buenos, aunque existe margen de mejora. Se ha logrado una precisión del 70.71% pero podría optimizarse más.

Para lograr mejoras, se pueden explorar diversas estrategias. En primer lugar, se podría probar con otros modelos de TensorFlow Hub, como MobileNet V2 o Inception V3.

Otra opción sería añadir capas ocultas. También se podrían ajustar los hiperparámetros como el número de pasos de entrenamiento, la tasa de aprendizaje y el tamaño del lote para encontrar las configuraciones más óptimas.

Además, se podría experimentar con distintos optimizadores, como Adagrad, para evaluar si tienen un impacto positivo en el proceso de entrenamiento.

Estas estrategias pueden ayudar a refinar el modelo y alcanzar un mejor rendimiento de

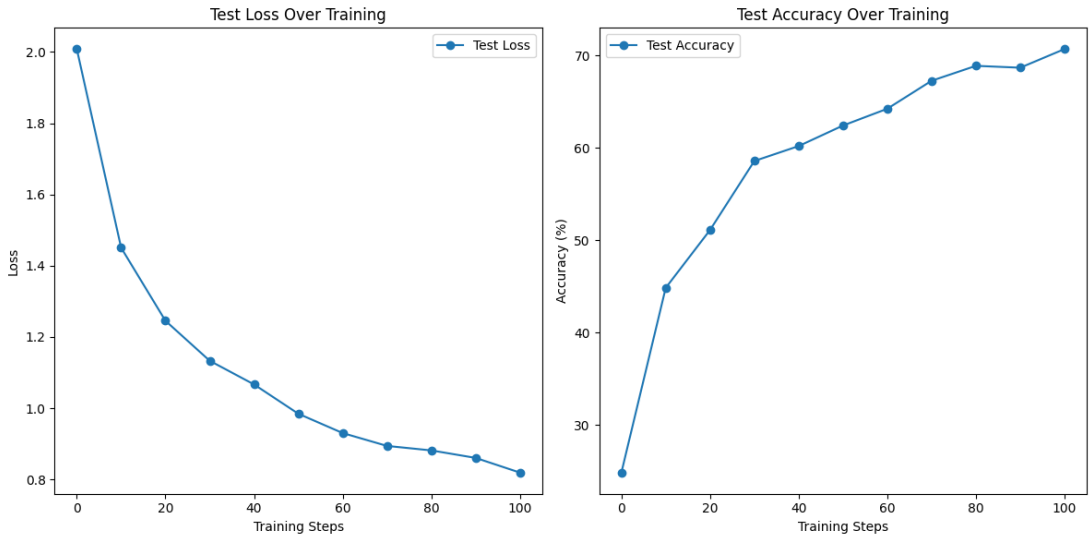

Figura 4.3: Gráficas de pérdida y precisión

acuerdo con los objetivos y las restricciones específicas de la aplicación.

4.4 Evaluación del modelo de detección de objetos

4.4.1 Entrenamiento y testeo

El entrenamiento se realiza utilizando un batch size de 8 y 40 épocas. Para realizarlo se ejecuta el siguiente script:

```
!python /content/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={pipeline_config_path} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --sample_1_of_n_eval_examples=1 \
    --num_eval_steps={num_eval_steps}
```

Listing 4.1: Training

A medida que el modelo es entrenado los valores de los parámetros obtenidos se guardan en archivos checkpoint. Tomando el ultimo archivo se puede construir el modelo ya entrenado con las imágenes de entrenamiento del dataset de residuos.

4.4.2 Resultados obtenidos

Una vez entrenado, el modelo se puede utilizar para detectar objetos en las imágenes de test utilizando la API de Object Detection de Tensorflow. En la figura 4.4 se presentan algunos

de los resultados obtenidos.

Figura 4.4: Detección de residuos

Utilizando las métricas usuales para modelos de detección se obtienen mAP@0.5IoU = 0.799 y mAP@0.50:0.95 = 0.59 En cuanto a los resultados de mAP separados por categorías, aparecen en la tabla.

Categoría	mAP
Cartón	0.40
Vidrio	0.65
Metal	0.67
Papel	0.61
Plástico	0.59

En el estado del arte de la detección de objetos existen modelos que consiguen valores cercanos a 0.5 de AP. Sin embargo, estos resultados corresponden al dataset COCO, en el que hay que detectar objetos de 80 categorías distintas tan diferentes entre si como personas, medios de transporte, señales de trafico, animales, objetos cotidianos, muebles, tipos de comida, muebles... Por esta razón los resultados no son comparables ya que en este caso el modelo entrenado solo distingue 5 clases diferentes de residuos.

Capítulo 5

Conclusiones y trabajo futuro

En este apartado se describen las conclusiones extraídas tras la realización de este trabajo desde dos puntos de vista diferentes: en el plano académico, el conocimiento adquirido gracias al trabajo realizado y, en el plano de investigación, las conclusiones extraídas de los distintos modelos explorados.

5.1 En el plano académico

En primer lugar, hemos profundizado en las técnicas de aprendizaje profundo (deep learning) aplicadas a la detección de objetos. Como equipo, hemos estudiado arquitecturas de redes neuronales convolucionales (CNN) y las hemos implementado en Python utilizando bibliotecas como TensorFlow y PyTorch. Esto nos ha proporcionado una comprensión sólida de cómo funcionan estos modelos y cómo se pueden entrenar para reconocer y localizar objetos en imágenes.

Además, hemos tenido la oportunidad de trabajar con conjuntos de datos tanto para detección como para clasificación de objetos. Esto nos ha permitido aprender a preprocesar y etiquetar imágenes de manera efectiva para el entrenamiento de modelos. También hemos reconocido la importancia de los conjuntos de datos en ambas tareas y cómo pueden afectar el rendimiento del modelo.

En términos prácticos, hemos mejorado nuestras habilidades de programación en Python y nos hemos familiarizado con bibliotecas específicas para la visión por ordenador, como OpenCV. Hemos aprendido a utilizar herramientas de entrenamiento y evaluación de modelos, lo que ha sido esencial para medir el rendimiento y la precisión de los algoritmos que hemos implementado.

Sin embargo, una distinción clave entre la detección y la clasificación de objetos es que, en la detección, además de reconocer los objetos, también debemos localizar su posición en la imagen. Esto añade un nivel adicional de complejidad al problema. Mientras que en la clasi-

ficación, el objetivo principal es asignar una etiqueta a una imagen, en la detección, debemos identificar la presencia y ubicación de múltiples objetos dentro de una imagen.

Nuestra colaboración y esfuerzo como equipo nos ha preparado para abordar desafíos tanto en la detección como en la clasificación de objetos. Estamos ansiosos por seguir aprendiendo y mejorando nuestras habilidades en ambas áreas y por las oportunidades futuras que se presentarán en este campo en constante evolución.

5.2 En el plano de investigación

Desde el punto de vista de los resultados obtenidos en nuestro proyecto de detección de objetos utilizando distintas redes neuronales en imágenes estáticas, hemos extraído valiosas conclusiones que arrojan luz sobre las capacidades y limitaciones de estas arquitecturas en este contexto específico.

En primer lugar, hemos observado que la elección de la arquitectura de red tiene un impacto significativo en la precisión de la detección de objetos en imágenes estáticas. Al comparar diferentes arquitecturas, como MobileNet, ResNet y YOLO, hemos notado que cada una tiene sus propias fortalezas y debilidades en términos de precisión y velocidad. Por ejemplo, MobileNet, siendo una red más ligera, logra un equilibrio aceptable entre precisión y velocidad, lo que lo hace adecuado para aplicaciones en tiempo real.

En contraste, en la clasificación de objetos, donde el objetivo principal es asignar una etiqueta o categoría a una imagen, hemos observado que las redes neuronales convolucionales (CNN) tradicionales tienden a funcionar de manera eficaz. Estas redes están diseñadas para extraer características significativas de las imágenes y, cuando se entrenan adecuadamente, pueden lograr altos niveles de precisión en la clasificación.

Sin embargo, cuando comparamos la tarea de clasificación de objetos con la detección de objetos, que implica identificar y localizar objetos dentro de una imagen, notamos diferencias fundamentales. En la detección de objetos, no solo estamos interesados en la etiqueta o categoría de los objetos, sino también en su ubicación precisa en la imagen. Esta tarea agrega un nivel adicional de complejidad, ya que requiere modelos que puedan realizar ambas tareas simultáneamente.

En resumen, nuestras conclusiones destacan las diferencias clave entre la clasificación de objetos y la detección de objetos, con énfasis en cómo las arquitecturas de redes neuronales deben adaptarse para abordar con éxito cada una de estas tareas. Cada enfoque tiene sus propias ventajas y desafíos, y comprender estas diferencias es esencial para tomar decisiones informadas en futuros proyectos de visión por ordenador.

5.3 Trabajo a futuro

Este proyecto marca el punto de partida de un esfuerzo continuo que tiene el potencial de culminar en un producto valioso y comercializable. En primer lugar, sería beneficioso explorar y aplicar estrategias de cooperación más avanzadas entre los componentes de detección y clasificación. Por ejemplo, considerar la compartición de información relevante a nivel global entre estos componentes podría mejorar la eficacia general del sistema.

Además, para llevar este proyecto al ámbito comercial, sería crucial optimizar su rendimiento y eficiencia. Esto incluiría la búsqueda de formas de mejorar la precisión y la velocidad de detección y clasificación. La optimización de la arquitectura de red neuronal utilizada, la selección de hiperparámetros y la implementación de técnicas de transferencia de aprendizaje podrían ser áreas de enfoque.

Otro aspecto fundamental es la escalabilidad y la capacidad de manejar grandes conjuntos de datos y cargas de trabajo. Esto podría requerir la implementación de técnicas de procesamiento distribuido o la adopción de servicios de nube escalables.

Además, es importante considerar la facilidad de uso y la integración con otras herramientas y sistemas. El desarrollo de una interfaz de usuario intuitiva y la capacidad de integrarse con aplicaciones y plataformas existentes podría aumentar la utilidad y la adopción del producto.

Finalmente, la robustez y la seguridad son factores críticos a medida que se avanza hacia la producción comercial. Se deberán realizar pruebas exhaustivas, incluida la evaluación de la resistencia a ataques y la detección de posibles vulnerabilidades.

En resumen, la transición de este proyecto de investigación a un producto comercializable requerirá un enfoque en la mejora de la precisión, la eficiencia, la escalabilidad, la facilidad de uso, la integración y la seguridad. El trabajo futuro podría incluir la exploración de estas áreas para cumplir con los estándares y requisitos comerciales.

Anexo

Este capítulo está destinado a incluir información adicional, como códigos y datos complementarios que, debido a las restricciones de espacio o formato, no se incluyeron en los capítulos principales del documento. Este anexo proporciona una visión más completa y detallada del trabajo realizado, permitiendo así acceder a recursos adicionales para comprender y replicar los experimentos.

Modelo de clasificación

Fragmento del código empleado para el entrenamiento y análisis del modelo de clasificación [5.1]. Ejemplo de imágenes clasificadas de forma incorrecta [5.1] y correcta [5.2].

```
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(NUM_TRAIN_STEPS):
    train_batch = get_batch(batch_size=TRAIN_BATCH_SIZE)
    batch_images, batch_labels = get_images_and_labels(train_batch)
    train_loss, _, train_accuracy = sess.run(
        [cross_entropy_mean, train_op, accuracy],
        feed_dict={encoded_images: batch_images, labels: batch_labels})
    is_final_step = (i == (NUM_TRAIN_STEPS - 1))
    if i % EVAL_EVERY == 0 or is_final_step:
      test_batch = get_batch(batch_size=None, test=True)
      batch_images, batch_labels = get_images_and_labels(test_batch)
      test_loss, test_accuracy, test_prediction, correct_predicate = sess.run(
        [cross_entropy_mean, accuracy, prediction, correct_prediction],
        feed_dict={encoded_images: batch_images, labels: batch_labels})
      print('Test accuracy at step %s: %.2f%%' % (i, (test_accuracy * 100)))
      test_loss_values.append(test_loss)
      test_accuracy_values.append(test_accuracy)
```

Listing 5.1: Bucle de entrenamiento y evaluación

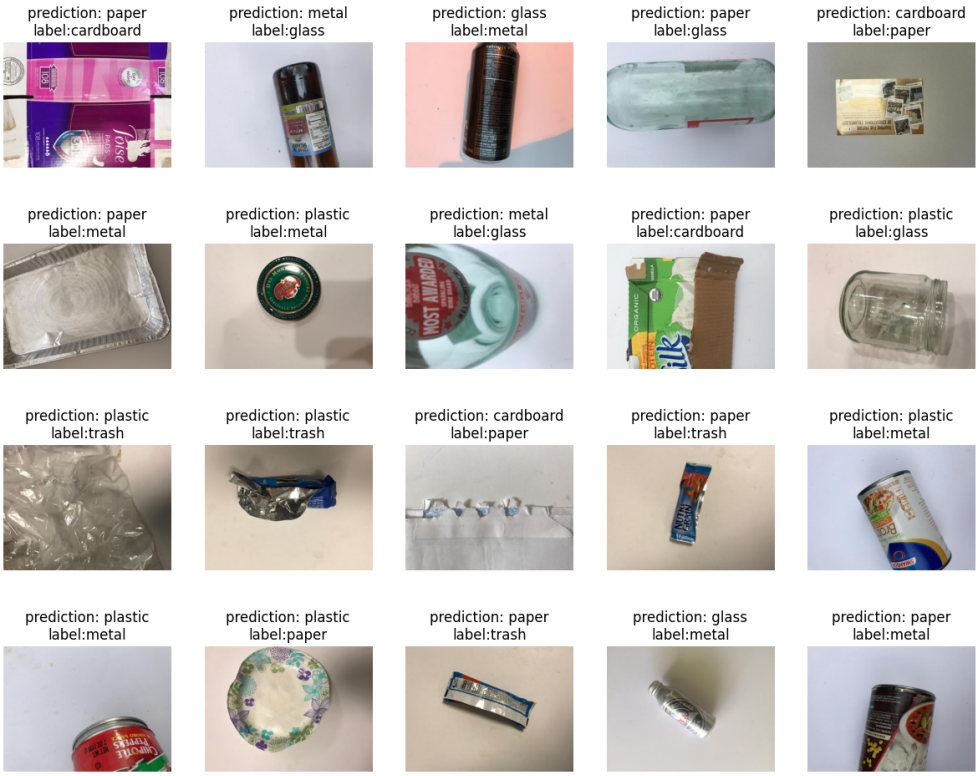

Figura 5.1: Predicciones incorrectas

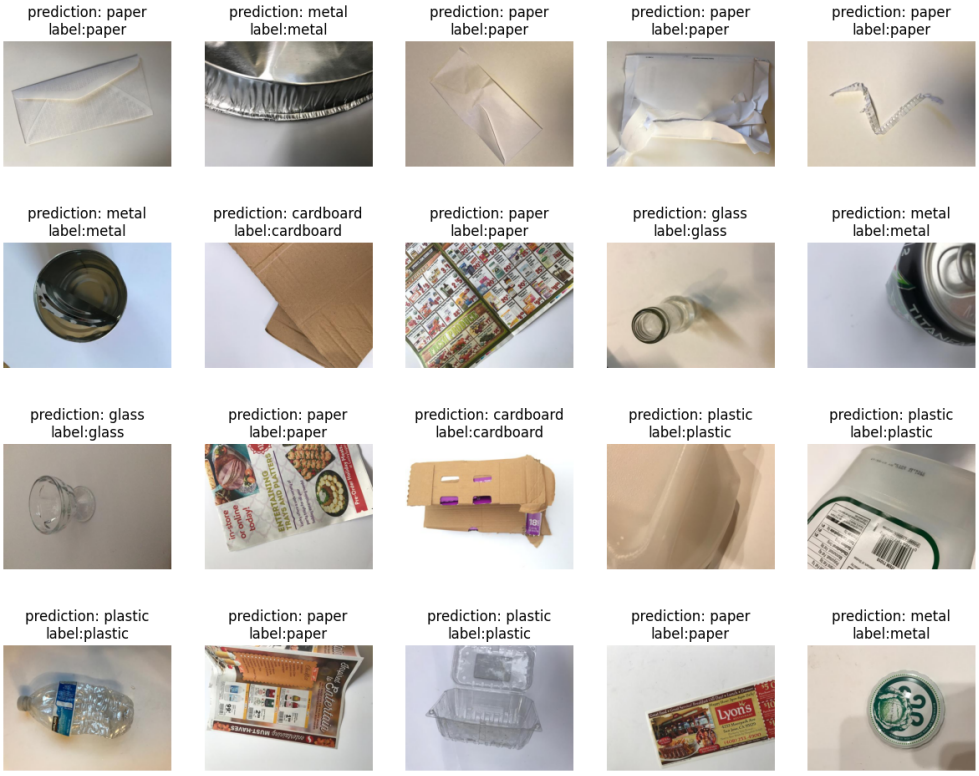

Figura 5.2: Predicciones correctas

Librerías empleadas

En el desarrollo y ejecución de cualquier proyecto de detección de imágenes, especialmente aquellos orientados a la clasificación automatizada de objetos, la elección y utilización de las librerías adecuadas desempeña un papel fundamental en la consecución de resultados precisos y eficientes. En esta sección, se presentarán y analizarán detalladamente las librerías empleadas en las distintas etapas del proyecto de detección de imágenes para la clasificación de basura. Dichas librerías se agruparán en tres categorías principales: adquisición y preprocesamiento de imágenes, etiquetado de datos y entrenamiento del modelo de detección. Para cada categoría, se abordarán tanto los aspectos técnicos como los implicaciones en el contexto empresarial, destacando su relevancia y contribución al logro de los objetivos del proyecto.

Librerías para la Adquisición y Preprocesamiento de Imágenes

cv2 (OpenCV)

OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto ampliamente utilizada en aplicaciones de visión por computadora y procesamiento de imágenes. En el contexto de este proyecto, OpenCV se emplea para la captura y manipulación de imágenes, proporcionando herramientas para la carga, visualización y transformación de imágenes. Desde el punto de vista empresarial, OpenCV agiliza la manipulación de imágenes en bruto, lo que contribuye a una adquisición eficiente y a un preprocesamiento consistente de las imágenes de basura, mejorando la calidad de entrada para el proceso de entrenamiento del modelo.

uuid (Universally Unique Identifier)

La librería uuid se emplea para generar identificadores únicos para cada imagen adquirida. Estos identificadores son cruciales para el seguimiento de imágenes individuales a lo largo del flujo de trabajo, desde la adquisición hasta la evaluación del modelo. En términos de negocio, la asignación de identificadores únicos facilita la trazabilidad y la gestión de datos, lo que a su vez contribuye a la fiabilidad y calidad de los resultados obtenidos.

os (Operating System Interface)

La librería os proporciona una interfaz para interactuar con el sistema operativo subyacente. En este proyecto, se utiliza para gestionar la creación y organización de directorios de almacenamiento de imágenes y otros datos. Desde un enfoque empresarial, esta librería contribuye a una estructura ordenada de los datos, lo que simplifica la administración del proyecto y favorece la colaboración entre equipos.

time

La librería time se emplea para medir y registrar el tiempo necesario para llevar a cabo ciertas operaciones dentro del proyecto, como la adquisición y el preprocesamiento de imágenes. Desde la perspectiva de negocio, el seguimiento y análisis del tiempo invertido en diferentes etapas proporciona información valiosa sobre la eficiencia del proceso y puede llevar a mejoras en la productividad.

Librerías para el Etiquetado de Datos

qt5py3

Qt es una plataforma de desarrollo de interfaces gráficas de usuario (GUI) que facilita la creación de aplicaciones con interfaces intuitivas y atractivas. En este proyecto, la librería qt5py3 se emplea para desarrollar una herramienta de etiquetado de datos, permitiendo a los usuarios marcar y definir áreas de interés en las imágenes. En el ámbito empresarial, una interfaz de etiquetado amigable y eficiente agiliza el proceso de creación del conjunto de datos anotados, lo que a su vez impulsa la precisión del modelo final.

lxml

lxml es una librería de procesamiento de XML y HTML utilizada en este proyecto para gestionar y manipular los datos de etiquetado generados por la herramienta de marcado. Desde una perspectiva de negocio, el uso de lxml garantiza la integridad y consistencia de los datos etiquetados, lo que es esencial para el entrenamiento efectivo del modelo de detección.

Librerías para el Entrenamiento del Modelo

tensorflow

TensorFlow es un marco de trabajo de código abierto ampliamente reconocido para el desarrollo y entrenamiento de modelos de aprendizaje automático. En este proyecto, TensorFlow se utiliza para construir, entrenar y evaluar el modelo de detección de basura. Desde un punto de vista técnico, TensorFlow proporciona una amplia gama de herramientas para la construcción y optimización de modelos de detección de objetos. En términos de negocio, el uso de TensorFlow permite la implementación eficiente de un sistema de detección automatizada de basura, lo que puede tener un impacto positivo en la gestión de residuos y la sostenibilidad.

protobuf

Protocol Buffers (protobuf) es una metodología de serialización de datos eficiente y escalable utilizada en este proyecto para definir la estructura de los datos de entrenamiento y evaluación. Desde una perspectiva empresarial, protobuf acelera la transmisión y almacenamiento de datos, lo que es esencial cuando se trabaja con grandes conjuntos de datos en un proyecto de detección de imágenes.

matplotlib y pyplot

Matplotlib es una librería de visualización de datos en 2D que incluye el módulo pyplot para la creación de gráficos y visualizaciones. En el contexto de este proyecto, matplotlib y pyplot se emplean para visualizar las métricas de rendimiento del modelo, lo que facilita la interpretación de los resultados tanto para los desarrolladores como para las partes interesadas empresariales.

object_detection, utils y builders

La librería ”object_detection” es un componente central en el proceso de construcción y entrenamiento de modelos de detección de objetos. Ofrece una serie de herramientas y funciones que simplifican la implementación de modelos de detección personalizados. Esta librería proporciona algoritmos y arquitecturas predefinidas que pueden ser adaptadas y ajustadas para abordar tareas específicas, como la detección de basura en este proyecto.

El módulo ”utils” en ”object_detection” incluye funciones de utilidad que son esenciales para varias tareas, como la conversión de formatos de datos, el procesamiento de anotaciones y la evaluación del modelo. Estas funciones auxiliares agilizan la preparación de los datos de entrenamiento y permiten una evaluación precisa del rendimiento del modelo en términos de métricas como la precisión y el recuerdo.

Por otro lado, el módulo ”builders” en ”object_detection” se encarga de simplificar la construcción y configuración de modelos de detección. Proporciona interfaces y herramientas para definir la arquitectura del modelo, ajustar los hiperparámetros y establecer las configuraciones de entrenamiento. Esto es esencial para facilitar la experimentación con diferentes configuraciones de modelos y mejorar la eficiencia en la creación de arquitecturas de detección personalizadas.

Desde una perspectiva empresarial, el uso de la librería ”object_detection” y sus componentes asociados acelera el proceso de desarrollo del modelo de detección de basura. Permite a los ingenieros y científicos de datos centrarse en la creación de modelos efectivos sin tener que construir todas las partes desde cero. Esto reduce significativamente el tiempo y los recursos requeridos para la creación de un modelo de detección de objetos de alta calidad.

cv2, numpy, pandas, opencv-python, tensorflowjs

Estas librerías se emplean en varias etapas del proceso de entrenamiento, incluyendo el procesamiento de imágenes, manipulación de datos y exportación del modelo. Desde una perspectiva técnica, cv2 (OpenCV) se utiliza para el procesamiento de imágenes, mientras que numpy y pandas se emplean para el manejo y análisis de datos. OpenCV-Python facilita la integración de las capacidades de procesamiento de imágenes en el flujo de trabajo de entrenamiento. TensorFlow.js habilita la implementación del modelo entrenado en entornos web y aplicaciones JavaScript. Desde una perspectiva de negocio, estas librerías permiten una manipulación eficiente de datos y una implementación versátil del modelo, maximizando la calidad y el alcance del sistema de detección de basura.

En conjunto, las librerías presentadas en esta sección desempeñan un papel crucial en el desarrollo y éxito del proyecto de detección de imágenes para la clasificación de basura. Su contribución técnica y empresarial garantiza la eficiencia, precisión y utilidad del sistema de detección, lo que a su vez puede tener un impacto significativo en la gestión sostenible de los residuos y en la promoción de prácticas ambientales responsables. Con esto ya tenemos las librerías que usaremos para entrenar nuestro modelo de detección de objetos.

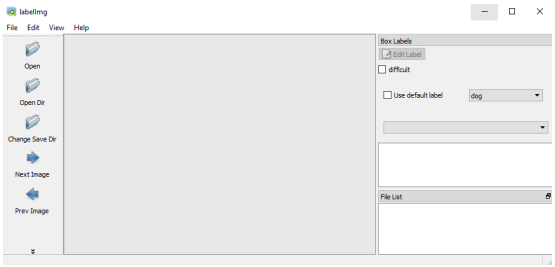

Figura 5.3: Software empleado en el etiquetado.

Figura 5.4: Imagen previa al etiquetado.

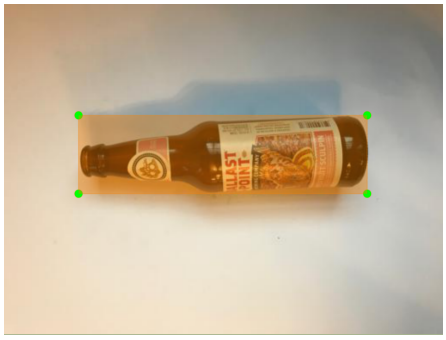

Figura 5.5: Imagen después del etiquetado.

Código usado

Configuración

```
import re

with open(base_config_path) as f:
    config = f.read()

with open('model_config.config', 'w') as f:

  # Set labelmap path
  config = re.sub('label_map_path: ".*?"',
             'label_map_path: "{}"'.format(labelmap_path), config)
  # Set fine_tune_checkpoint path
  config = re.sub('fine_tune_checkpoint: ".*?"',
                  'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint),
    config)
  # Set train tf-record file path
  config = re.sub('(input_path: ".*?)(PATH_TO_BE_CONFIGURED/train)(.*?")',
                  'input_path: "{}"'.format(train_record_path), config)
  # Set test tf-record file path
  config = re.sub('(input_path: ".*?)(PATH_TO_BE_CONFIGURED/val)(.*?")',
                  'input_path: "{}"'.format(test_record_path), config)
  # Set number of classes.
  config = re.sub('num_classes: [0-9]+',
                  'num_classes: {}'.format(len(classes_names)), config)
  # Set batch size
  config = re.sub('batch_size: [0-9]+',
                  'batch_size: {}'.format(batch_size), config)
  # Set training steps
  config = re.sub('num_steps: [0-9]+',
                  'num_steps: {}'.format(num_steps), config)
  # Set fine-tune checkpoint type to detection
  config = re.sub('fine_tune_checkpoint_type: "classification"',
             'fine_tune_checkpoint_type: "{}"'.format('detection'), config)
  # Set fine-tune checkpoint type to detection
  config = re.sub('metrics_set: "coco_detection_metrics"\n',
                  'metrics_set: "coco_detection_metrics"\n
    include_metrics_per_category: true\n', config)
  f.write(config)
```

Listing 5.2: config

Lista de acrónimos

- R-CNN (*Region-based Convolutional Neural Network*)
- YOLO (*You Only Look Once*)
- SSD (*Single Shot MultiBox Detector*)
- IoU (*Intersection over Union*)
- ROI (*Region of Interest*)
- NMS (*Non-Maximum Suppression*)
- CNN (*Convolutional Neural Network*)
- SVM (*Support Vector Machine*)
- K-NN (*k-Nearest Neighbors*)
- MLP (*Multilayer Perceptron*)
- CNN (*Convolutional Neural Network*)
- RNN (*Recurrent Neural Network*)
- DNN (*Deep Neural Network*)
- PCA (*Principal Component Analysis*)
- LDA (*Linear Discriminant Analysis*)
- ROC (*Receiver Operating Characteristic*)
- F1-Score

Bibliografía

[1] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 770-778).

[3] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[4] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 580-587).

[5] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision (ICCV)* (pp. 2980-2988).

[6] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)* (pp. 1097-1105).

[7] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 1-9).

[8] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., … & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision, 115(3)*, 211-252.

[9] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision, 88(2)*, 303-338.

[10] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521(7553)*, 436-444.