Universidad Zaragoza
1474

Trabajo Fin de Grado

Conteo Automático de Unidades en Palés de Picking mediante Redes Neuronales

Neural Network-Based Automatic Unit Counting on Picking Pallets

Autor

Alejandro Sanz del Río

Director

Daniel Forcen Esteban

Escuela de Ingeniería y Arquitectura

2025

Resumen

Este Trabajo de Fin de Grado aborda el desarrollo de un sistema de visión por computador basado en redes neuronales para el conteo automático de unidades en palés abiertos dentro de entornos logísticos. El objetivo principal es reducir el tiempo y los errores asociados al conteo manual, frecuente en tareas de *picking*. Para ello, se exploran y comparan distintos modelos de conteo existentes, entre ellos *Detectron2*, *Learning To Count Everything (LTCEverything)* y *Learning To Count Anything (LTCAnything)*, evaluando su rendimiento sobre *datasets* reales capturados en almacenes de Carreras Grupo Logístico.

Los experimentos se han diseñado para cubrir tres escenarios: conteo en productos individuales, generalización cruzada entre productos y combinación de categorías. Se aplican distintas estrategias como *fine-tuning* completo, entrenamiento de capas MLP externas y adaptación directa de modelos preentrenados. El análisis cuantitativo se realiza mediante las métricas MAE y RMSE, y se apoya en herramientas como `PyTorch`, `PyTorch Lightning`, `TorchMetrics` y despliegues en `OpenShift`, garantizando la trazabilidad y repetibilidad de los entrenamientos.

Los resultados muestran que el enfoque más eficaz combina el modelo preentrenado LTCA con una capa MLP entrenada externamente, alcanzando errores medios cercanos a una unidad incluso en escenarios complejos. Se concluye que este tipo de soluciones tiene un elevado potencial de aplicabilidad en procesos logísticos reales, ofreciendo automatización, robustez y reducción del esfuerzo manual asociado al conteo de unidades.

Abstract

This Bachelor's Thesis presents the development of a computer vision system based on neural networks for the automatic counting of units on open pallets in logistics environments. The main goal is to reduce the time and errors associated with manual counting, commonly required during picking operations. To achieve this, several state-of-the-art counting models are explored and compared —including *Detectron2*, *Learning To Count Everything (LTCEverything)*, and *Learning To Count Anything (LTCAnything)*— and evaluated using real datasets collected from warehouses of Carreras Grupo Logístico.

The experiments cover three scenarios: counting in single-product datasets, cross-product generalization, and mixed-product configurations. Strategies such as full *fine-tuning*, external MLP layer training, and direct adaptation of pretrained models are employed. The evaluation uses MAE and RMSE metrics, supported by tools like `PyTorch`, `PyTorch Lightning`, `TorchMetrics` and deployments on `OpenShift`, ensuring reproducibility and traceable results.

The findings show that the most effective configuration combines a pretrained LTCA model with an external MLP layer, achieving mean errors close to one unit even in challenging conditions. This demonstrates the potential of these systems to automate and improve accuracy in real-world logistics operations, reducing the need for manual intervention and increasing efficiency.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que conforman el grupo IT de Carreras por su cálido recibimiento y por darme la oportunidad de realizar mi proyecto en un entorno tan profesional. En especial, me gustaría mencionar a Daniel Forcen, Sergio Cantín, Fernando Gutiérrez y Ricardo Guillén, quienes, en el día a día, siempre se han preocupado por mí, por mis necesidades, y con quienes he podido contar para lo que hiciese falta.

También me gustaría agradecer a mi tutor, Eduardo Montijano, quien, a pesar de marcharse a realizar sus investigaciones a Estados Unidos, no dudó en tutorizarme este proyecto. Su experiencia y consejos han sido claves para guiarme en la toma de decisiones técnicas y en la planificación del proyecto.

Índice general

Resumen **I**

Abstract **II**

Agradecimientos **III**

Índice general **IV**

Índice de figuras **VII**

Índice de tablas **X**

1. Introducción **1**

1.1. Motivación y Contexto 1

1.2. Objetivos y alcance 2

1.3. Cronograma y secciones 4

2. Análisis del problema y estado del arte **5**

2.1. Detectron2 6

2.2. Learning To Count Everything 7

2.3. Learning To Count Anything 8

3. Sistema de conteo automático **11**

3.1. Conjunto de datos 11
3.1.1. Dataset Vinos Blancos 13
3.1.2. Dataset Salsas Basílico 14
3.1.3. Dataset Vinagres Variados 16
3.1.4. Dataset Conjunto 17
3.2. Baselines 18

4. Experimentos **25**
4.1. Experimentos propuestos 25
4.1.1. Experimento 1: Entrenamiento en un Dataset 25
4.1.2. Experimento 2: Entrenamiento sobre capacidad de generalización . 26
4.1.3. Experimento 3: Entrenamiento con varios productos 26
4.2. Herramientas 27
4.3. Métricas de evaluación 27

5. Evaluación y análisis de resultados **29**
5.1. Evaluación experimento en un dataset 29
5.2. Evaluación experimento sobre generalización 34
5.3. Evaluación experimento datasets conjuntos 37
5.4. Conclusión experimentos 38

6. Conclusión y trabajo futuro **40**
6.1. Conclusiones 40
6.2. Trabajo futuro 41

A. Anexos **43**
A.1. OpenShift y S3 Bucket AWS 44

A.2. Imagenes de conteo de modelos básicos 45

Bibliografía **50**

Índice de figuras

1.1. Ejemplo del entorno logístico con palés abiertos en un almacén. 1

1.2. Conteo automático de unidades en un palé de picking. 2

2.1. Esquema general del modelo Detectron2. 7

2.2. Esquema general del modelo Learning To Count Everything. 8

2.3. Esquema general del modelo Learning To Count Anything. 10

3.1. Setup de captura de imagenes. 12

3.2. Distribución del dataset Vinos Blancos. 13

3.3. Ejemplos de diferentes unidades de Vinos Blancos. 14

3.4. Ejemplos de 12 unidades de Vinos Blancos. 14

3.5. Distribución del dataset Salsas Basílico. 15

3.6. Ejemplos de diferentes unidades de Salsas Basílico. 15

3.7. Ejemplos de 24 unidades de Salsas Basílico. 16

3.8. Distribución del dataset Vinagres Variados. 17

3.9. Ejemplos de diferentes unidades de Vinagres Variados. 17

3.10. Distribución del dataset conjunto Vinos Blancos, Salsas Basílico y Vinagres Variados. 18

3.11. Adaptación directa del modelo Detectron2. 19

3.12. Fine-tuning de un MLP externo en Detectron2. 20

3.13. Adaptación directa del modelo LTCEverything. 21

3.14. Fine-tuning del CountRegressor del modelo LTCEverything. 22

3.15. Fine-tuning de un MLP externo en el modelo LTCEverything. 22

3.16. Adaptación directa del modelo LTCAnything. 23

3.17. Fine-tuning de un MLP externo en LTCAnything. 24

3.18. Fine-tuning completo en LTCAnything. 24

5.1. Ejemplos de diferentes interacciones con modelos básicos de Detectron2 en los datasets. 31

5.2. Ejemplos de diferentes interacciones con modelos básicos de LTCAnything en los datasets. 31

5.3. Ejemplos de diferentes interacciones con modelos básicos de LTCEverything en los datasets. 32

5.4. Comparativa de errores MAE y RMSE para cada modelo en los conjuntos de test y validación del dataset Vinos Blancos. Cuanto menor sea el valor, mejor rendimiento. . . 32

5.5. Comparativa de errores MAE y RMSE para cada modelo en los conjuntos de test y validación en el dataset Salsas Basílico. Cuanto menor sea el valor, mejor rendimiento. . 33

5.6. Comparativa de errores MAE y RMSE para cada modelo en los conjuntos de test y validación en el dataset Vinagres Variados. Cuanto menor sea el valor, mejor rendimiento. 33

5.7. Comparativa de errores MAE y RMSE para cada modelo en pruebas de generalización entre productos. La línea separa los experimentos entrenados con vinos y evaluados con salsas (izquierda), y viceversa (derecha). 35

5.8. Comparativa de errores MAE y RMSE para cada modelo en pruebas de generalización entre productos. La línea separa los experimentos entrenados con vinos y evaluados con vinagres variados (izquierda), y viceversa (derecha). 36

5.9. Comparativa de errores MAE y RMSE para cada modelo en pruebas de generalización entre productos. La línea separa los experimentos entrenados con salsas y evaluados con vinagres variados (izquierda), y viceversa (derecha). 37

5.10. Comparativa de errores MAE y RMSE para cada modelo en los conjuntos de test y validación en el dataset Conjunto. Cuanto menor sea el valor, mejor rendimiento. . . . 38

A.1. Ejemplos de diferentes interacciones con modelos básicos de Detectron2 en el dataset Vinos Blancos. 45

A.2. Ejemplos de diferentes interacciones con modelos básicos de LTCAnything en el dataset Vinos Blancos. 46

A.3. Ejemplos de diferentes interacciones con modelos básicos de LTCEverything en el dataset Vinos Blancos. 46

A.4. Ejemplos de diferentes interacciones con modelos básicos de Detectron2 en el dataset Salsas Basílico. 47

A.5. Ejemplos de diferentes interacciones con modelos básicos de LTCAnything en el dataset Salsas Basílico. 47

A.6. Ejemplos de diferentes interacciones con modelos básicos de LTCEverything en el Dataset Salsas Basílico. 48

A.7. Ejemplos de diferentes interacciones con modelos básicos de Detectron2 en el dataset Vinagres Variados. 48

A.8. Ejemplos de diferentes interacciones con modelos básicos de LTCAnything en el dataset Vinagres Variados. 49

A.9. Ejemplos de diferentes interacciones con modelos básicos de LTCEverything en el dataset Vinagres Variados. 49

Índice de tablas

1.1. Cronograma de trabajo del TFG. 4

Capítulo 1

Introducción

1.1. Motivación y Contexto

En un mundo cada vez más orientado a la automatización y eficiencia operativa, la logística tiene un papel esencial para garantizar que los productos lleguen al lugar adecuado en el momento justo. Sectores como la alimentación, la automoción o la gran distribución dependen de redes logísticas ágiles, inteligentes y robustas. Dentro de este ecosistema, los centros logísticos y almacenes se convierten en nodos críticos, donde cada segundo cuenta y cada error puede suponer un coste.

Uno de los procesos clave en la operativa de almacén es el ***picking***: la tarea de seleccionar y agrupar productos desde el almacén para conformar pedidos. Aunque esta operación pueda parecer rutinaria, puede ser compleja. La precisión en el conteo de unidades es vital para evitar descontrol del stock, retrasos en entregas o errores de facturación. Sin embargo, muchas empresas aún confían en el recuento manual cuando se trata de palés abiertos, lo que genera un cuello de botella, como se puede ver en la Figura 1.1.

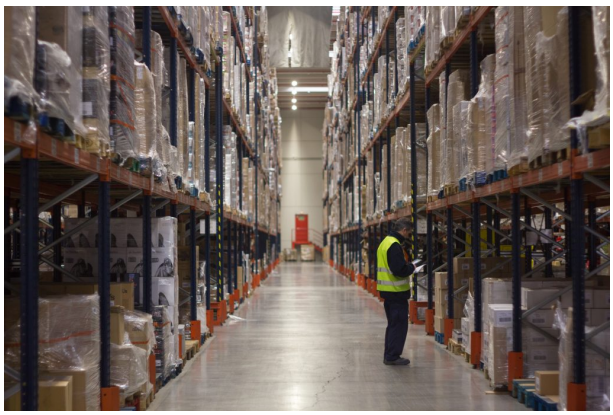

Figura 1.1: Ejemplo del entorno logístico con palés abiertos en un almacén.

En este contexto, aparece la oportunidad de poder aplicar soluciones tecnológicas basadas en **visión por computador e inteligencia artificial**. La automatización del conteo visual no solo permite reducir errores humanos, sino también liberar a los operarios para tareas de mayor valor añadido. Errores durante la carga o descarga, o unidades que se pierden por deterioro o defectos, todos estos errores tienen una solución basada en inteligencia artificial.

Este Trabajo de Fin de Grado tiene como finalidad ser una respuesta innovadora a un problema real de gran impacto: desarrollar un sistema de conteo automático basado en redes neuronales, entrenado con imágenes reales del entorno logístico, capaz de adaptarse a diferentes productos y condiciones visuales sin requerir anotaciones complejas ni procesos de configuración específicos para cada tipo de palé. Se puede ver un ejemplo en la Figura 1.2.

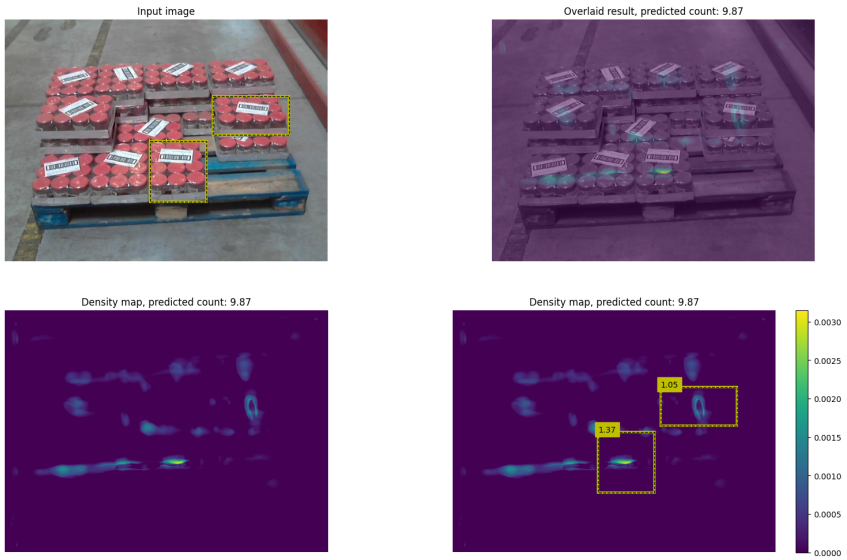

Figura 1.2: Conteo automático de unidades en un palé de picking.

Este es el punto de partida hacia una visión más inteligente y eficiente de la gestión de almacenes.

1.2. Objetivos y alcance

El objetivo del proyecto es desarrollar un sistema de visión por computador, basado en redes neuronales, que permita contar automáticamente el número de unidades visibles en palés abiertos utilizados en procesos de *picking* logístico. Se busca reducir el tiempo y los errores asociados al conteo manual.

El desarrollo del trabajo también tiene varios subobjetivos. Uno de ellos el estudio

del arte en redes neuronales para el problema de conteo de unidades. Para ello, en lugar de diseñar un modelo desde cero, se experimenta comparando diferentes configuraciones sobre arquitecturas existentes. Otro subobjetivo es el análisis y evaluación de los diferentes modelos con datos realistas.

El sistema debe ser capaz de generalizar a distintos tipos de productos y configuraciones de carga, y se abordan problemas como la oclusión parcial, la variabilidad en las condiciones de iluminación y la perspectiva de la cámara.

Este trabajo se encuentra dentro de ese reto real y concreto. La propuesta se ha desarrollado en colaboración con la empresa Carreras Grupo Logístico, una compañía líder en el sector de la logística y el transporte en España. Carreras opera en toda la cadena de suministro: transporte nacional e internacional, almacenaje, distribución capilar y servicios logísticos avanzados. Sus plataformas logísticas funcionan de forma ininterrumpida, lo que amplifica la importancia de mantener un control preciso y eficiente del inventario en tiempo real.

A nivel del alcance, la empresa no dispone de ningún sistema o modelo cuya funcionalidad sea la que se busca. Por lo tanto, se ha tenido que recurrir a sistemas o modelos externos sobre los que poder realizar la experimentación y evaluación. Se han modificado las arquitecturas de estos modelos, se ha realizado la captura y creación de un dataset en condiciones realistas en un almacén de Carreras y se ha diseñado un banco de pruebas para comparar todas las propuestas y satisfacer los objetivos.

Durante la realización del TFG se ha apoyado en el sistema de visión por computador *Detectron2* [1], el modelo de conteo visual *Learning To Count Everything* [2] y el modelo de conteo visual *Learning To Count Anything* [3].

Con el fin de llegar al objetivo se han empleado herramientas como `PyTorch`, que permite definir y entrenar redes neuronales personalizadas, así como para manejar el flujo de datos, la optimización y el cálculo de pérdidas. Herramientas como `TorchMetrics` han proporcionado métricas precisas para evaluar el rendimiento de los modelos, mientras que entornos como `Anaconda Prompt` y editores como `VSCode` han permitido lanzar y depurar los experimentos de forma flexible.

Desde el punto de vista metodológico, para llegar a cumplir con el objetivo se ha seguido una estructura experimental clara, incluyendo separación de conjuntos (*train*, *validation* y *test*), repetibilidad en los entrenamientos y evaluación objetiva mediante métricas estandarizadas como MAE y RMSE. Estas decisiones aseguran que los resultados obtenidos puedan compararse de forma justa y replicarse en condiciones similares.

1.3. Cronograma y secciones

En este apartado se describen brevemente el resto de secciones de la memoria y el correspondiente cronograma del TFG, ver Tabla 1.1:

- **Análisis del problema y revisión del estado del arte**: investigación sobre sistemas y modelos que mejor se adapten a nuestro problema, así como un estudio de sus arquitecturas.
- **Preparación del *dataset***: proceso de selección de los productos que se van a contar y del procedimiento de captura de imágenes.
- **Diseño experimental y entrenamiento/*fine-tuning***: se detallan las pruebas que se van a realizar con los diferentes sistemas y modelos seleccionados, incluyendo la adaptación del modelo base, el *fine-tuning* de un MLP externo o el *fine-tuning* completo del modelo.
- **Evaluación y análisis de resultados**: se lleva a cabo la evaluación de los diferentes modelos obtenidos en los experimentos con nuestros *datasets*, con el objetivo de comparar cuál de ellos ha obtenido mejores resultados y sacar conclusiones.
- **Desarrollo de memoria**: se redacta el documento que contiene el trabajo realizado y sus conclusiones.

Tarea	**Marzo**	**Abril**	**Mayo**	**Junio**
Análisis del problema y estado del arte	X	X		
Preparación del dataset		X	X	
Diseño experimental y entrenamiento			X	X
Evaluación y análisis de resultados			X	X
Desarrollo de la memoria				X

Tabla 1.1: Cronograma de trabajo del TFG.

Capítulo 2

Análisis del problema y estado del arte

En la empresa no hay ninguna estructura que se utilice para el conteo de unidades ni que emplee un sistema de visión por computador. Por ello, se parte de la observación del problema del conteo manual de unidades en palés abiertos y se plantea la hipótesis de que es posible estimar automáticamente el número de unidades mediante redes neuronales.

Se busca información sobre cómo se puede solucionar el problema y si existe alguna herramienta que pueda ayudar a cumplir el objetivo. Se encuentran los sistemas *Detectron2* [1] y *MediaPipe* [4], enfocados en visión por computador y visión multimodal, con los que se puede empezar a trabajar para evaluar su capacidad para realizar el conteo de unidades. Se decide empezar a trabajar con *Detectron2* debido a que dispone de diferentes modelos para la detección y segmentación de objetos, mientras que *MediaPipe* está más orientado a modelos preentrenados y optimizados para tareas como detección de manos, rostro, pose corporal o seguimiento de objetos.

Para realizar las primeras pruebas en los diferentes sistemas y modelos, lo que se hace es descargar un *dataset* de cajas de prueba. Este dataset se obtiene de los conjuntos de imágenes públicas en *Roboflow*, una plataforma para el desarrollo de modelos de visión por computador.

Durante las pruebas iniciales y en el proceso de búsqueda de información, se observa que el principal problema va a ser el conteo de las unidades que presentan oclusión parcial o total. Por ello, se buscan herramientas que sean capaces de generar una imagen, un mapa de densidad o algún mecanismo que permita contar las unidades con oclusión. Se encuentra *Depth Anything V2* [5], un modelo de estimación de profundidad monocular, y *SeGAN* [6] (*Segmenting and Generating the Invisible*), un modelo que combina segmentación semántica y generación de contenido con el objetivo de predecir y reconstruir las partes ocultas de los objetos en una imagen.

Antes de probar estos dos modelos a ciegas, se buscan proyectos que realizasen conteo de unidades apoyándose en mapas o anotaciones de densidad. Realizando esta búsqueda

concreta, se encuentran los modelos *Learning To Count Everything* [2] y *Learning To Count Anything* [3], enfocados al conteo de unidades.

Una vez se tiene toda la información anterior, se decide con qué sistemas y modelos se va a trabajar. Se elige *Detectron2* por encima de *MediaPipe* debido a dos factores. El primero de ellos es que *MediaPipe* está más enfocado al reconocimiento de personas, sus expresiones, sus poses... Y el segundo es que *Detectron2* ofrece una mayor variedad de modelos de detección, segmentación, *keypoints* y más. Respecto a los modelos de estimación de profundidad, se decide trabajar solamente con los enfocados al conteo de unidades, ya que ahorran la implementación de la parte del conteo. A continuación, en las siguientes secciones se abordan más en detalle los sistemas y modelos seleccionados.

2.1. Detectron2

Detectron2 [1] es un *framework* de código abierto desarrollado por Facebook AI Research (FAIR) para realizar tareas de detección de objetos, segmentación de instancias, *keypoints* y más. Está construido sobre `PyTorch` y se utiliza para entrenar y evaluar modelos de visión por computador de alto rendimiento.

Dentro de *Detectron2* existen varios modelos dependiendo de la tarea que se quiera realizar. Para la tarea de detección existen diferentes modelos, como *Faster R-CNN*, *RetinaNet* o *RPN and Fast R-CNN*. Todos estos modelos están preentrenados sobre el *dataset* COCO (*Common Objects in Context*), que contiene más de 33 000 imágenes, 80 categorías de objetos y anotaciones de *bounding boxes*, máscaras de segmentación y *keypoints* humanos. Todo esto permite que obtengan un rendimiento inicial robusto y puedan servir como base para *fine-tuning*[1].

A la hora de elegir modelo, se elige uno de *Faster R-CNN* debido a que es más preciso que los otros, aunque sea más lento. Dentro de este se elige el modelo `faster_rcnn_R_50_FPN_3x`, que es el modelo recomendado por defecto. Utiliza *ResNet-50*, que es más rápida; la arquitectura sobre la que se construye la cabeza del detector es *Feature Pyramid Network* (FPN), que combina múltiples escalas ofreciendo mejor rendimiento en objetos de diferente tamaño; y el `3x` indica la duración del entrenamiento, el triple que `1x`, lo que conlleva una mayor precisión.

Este modelo tiene la arquitectura que se muestra en la Figura 2.1, compuesta por tres partes:

- **Backbone**: red convolucional que extrae mapas de características de la imagen. En este modelo se trata de una *ResNet-50 + FPN*. *ResNet-50* es una red residual de

[1]Proceso de reentrenamiento de un modelo preentrenado utilizando un conjunto de datos más pequeño y específico para una tarea particular.

50 capas que actúa como extractor de características profundas a distintos niveles, a la que se le suma *FPN*, que añade una pirámide de características multiescala (de P2 a P5). Esto será útil a la hora de extraer los *features* del *backbone* para los experimentos.

- **Proposal Generator**: red que genera regiones candidatas donde podrían estar los objetos. Toma los *feature maps* del *backbone*, en este caso *FPN*, y genera cajas candidatas donde puede haber objetos. Cada propuesta tiene una puntuación de probabilidad de contener objetos.
- **ROI Heads**: módulos que clasifican y ajustan las regiones propuestas para obtener las detecciones finales. A partir de las propuestas genera *bounding boxes*[2] finales con la clase del objeto junto a una puntuación de confianza.

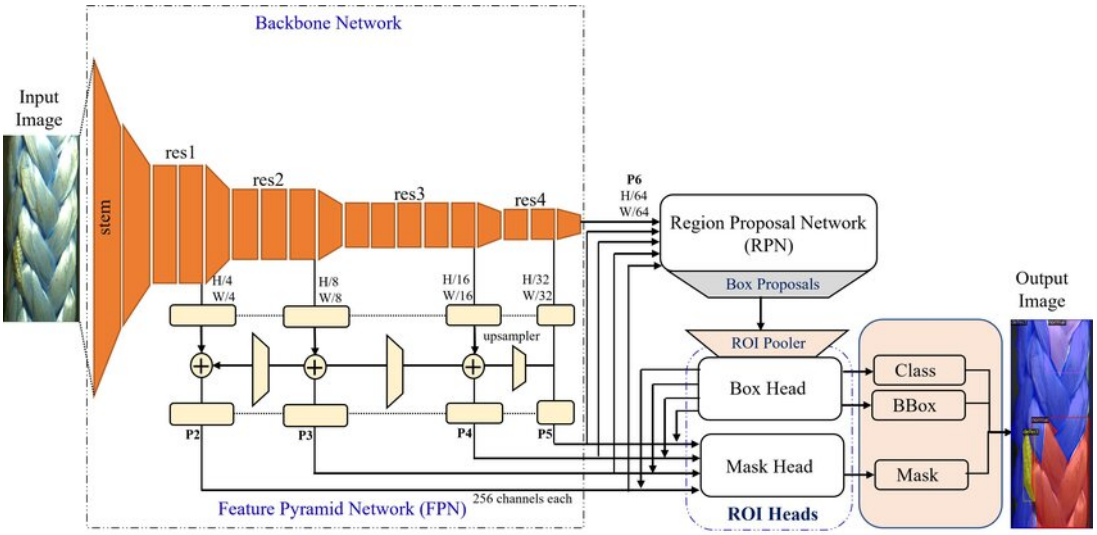

Figura 2.1: Esquema general del modelo Detectron2.

2.2. Learning To Count Everything

Learning to Count Everything (LTCEverything) [2] es un enfoque de *few-shot counting* que propone contar objetos de cualquier categoría visual, pasándole solamente unas pocas instancias de ejemplo dentro de una imagen. El objetivo es generalizar el conteo a nuevas clases con mínima supervisión.

FamNet (*Few-Shot Adaptation and Matching Network*) es la arquitectura propuesta por *LTCEverything*. Está diseñada para funcionar con cualquier categoría de objeto gracias a

[2]Cajas de referencia que se utilizan para delimitar la ubicación de un objeto dentro de una imagen o video.

su capacidad de adaptación y modularidad. El objetivo es minimizar el MSE,

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(c_i - \hat{c}_i)^2\,, \tag{2.1}$$

entre el mapa de densidad predicho y el real (generado a partir de puntos usando una gaussiana adaptativa). También se adapta en el testeo, mejorando la predicción usando dos pérdidas: *Min-Count Loss*, que fuerza a que cada caja de ejemplo tenga al menos un objeto, y *Perturbation Loss*, que compara la densidad predicha en la zona del ejemplar con una gaussiana para mejorar la precisión espacial.

En la implementación se ofrece un modelo preentrenado con el *dataset FSC-147* (147 clases y más de 6.000 imágenes con anotaciones de puntos y cajas).

Este modelo tiene la arquitectura que se muestra en la Figura 2.2, que consta de dos partes:

- **Backbone:** red convolucional que extrae mapas de características de la imagen. En esta arquitectura es *ResNet-50* preentrenado en *ImageNet*. Se usan los bloques 3 y 4, cuyas salidas son congeladas y se utilizan como *multi-scale features*[3].
- **Count Regressor:** módulo de predicción a través de la densidad. Recibe mapas de correlación entre las *features* de la imagen y de los *exemplars* (las pequeñas regiones de la imagen que contienen ejemplos del objeto que se desea contar y se pasan al modelo), y produce un mapa de densidad 2D donde la suma de los valores representa el conteo total.

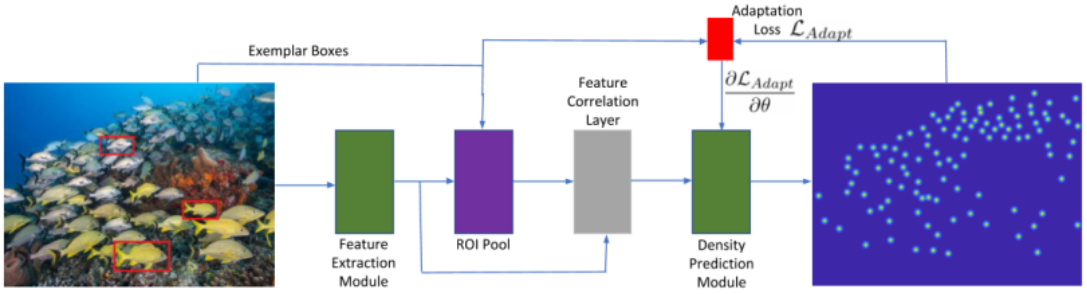

Figura 2.2: Esquema general del modelo Learning To Count Everything.

2.3. Learning To Count Anything

Learning To Count Anything (LTCAnything) [3] es un modelo de conteo de objetos de clases desconocidas sin necesidad de imágenes de referencia ni anotaciones de puntos.

[3]Extracción de información de diferentes tamaños o escalas en una imagen.

Es *reference-less* (el modelo no necesita ejemplos concretos, como imágenes con cajas de referencia, durante la inferencia para contar objetos) y *class-agnostic* (el modelo puede contar objetos de cualquier categoría sin estar limitado a clases específicas), con un enfoque débilmente supervisado (aprendizaje con anotaciones incompletas o imprecisas). Se basa en que contar objetos puede abordarse como una tarea de reconocimiento de repeticiones sin conocimiento previo de qué se cuenta.

El modelo es *Reference-less Class-agnostic Counter* (RCC), y el entrenamiento solo usa la cuenta total (*count*) como supervisión. Se entrena comparando predicciones entre un *teacher* (*crops* globales) y un *student* (*crops* locales), minimizando la divergencia entre sus distribuciones de clases predichas. Los *crops* son recortes de imagen extraídas de la imagen original. La pérdida, en este caso, se calcula con el *Absolute Percentage Error*(APE),

$$APE = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{c_i - \hat{c}_i}{c_i} \right| \times 100, \tag{2.2}$$

que sirve para no penalizar excesivamente los errores en imágenes con conteos bajos. En la implementación se ofrecen los pesos de un modelo entrenado con el *FSC-133*, un dataset derivado del *FSC-147* que elimina imágenes duplicadas, solapamientos entre *splits* y errores de conteo.

Este modelo tiene la arquitectura que se muestra en la Figura 2.3, que consta de tres partes:

- **Backbone**: red convolucional que extrae mapas de características de la imagen. En esta arquitectura es ViT-S (*Vision Transformer Small*), inicializado con pesos auto-supervisados del método DINO [7][4]. Utiliza múltiples cabezas de atención para construir una representación global de la imagen.
- **Linear Count Projection**: capa lineal que se aplica sobre los *embeddings* (representación densa y continua de una imagen o recorte en un espacio vectorial que permite que el modelo entienda similitudes semánticas entre objetos o regiones de una imagen) del ViT-S, proyectando el espacio obtenido a un número escalar que será el conteo final.
- **Localisation head** (opcional): bloques Conv-ReLU-Upsample que transforman la predicción del conteo en mapas de densidad.

[4]El método DINO de Caron et al. es una técnica de aprendizaje auto-supervisado para entrenar redes neuronales, especialmente transformadores como ViT (Vision Transformer), sin necesidad de etiquetas.

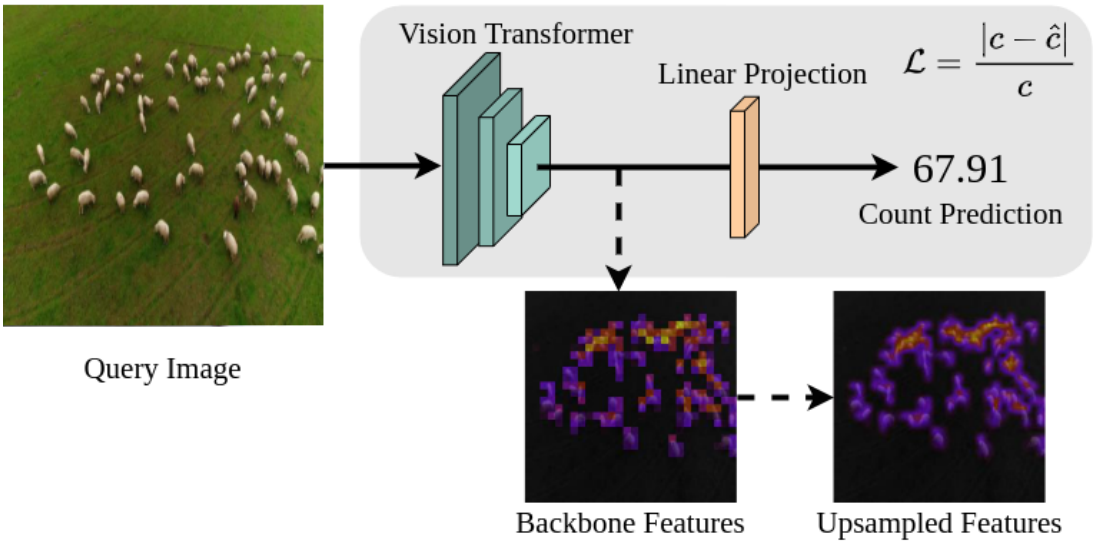

Figura 2.3: Esquema general del modelo Learning To Count Anything.

Capítulo 3

Sistema de conteo automático

Durante la realización de este trabajo se han realizado dos aportaciones a la solución del conteo automático de unidades. La primera de ellas consiste en un conjunto de datos procedentes de un almacén logístico que permite entrenar a los modelos con datos de un entorno real. La segunda de las aportaciones es sobre los modelos, ya que se han propuesto mejoras en sus arquitecturas con la finalidad de que el conteo sea más preciso.

3.1. Conjunto de datos

La calidad de las imágenes es clave en el rendimiento de los modelos de conteo visual. Un buen *dataset* aumenta las probabilidades de que un entrenamiento sea efectivo y se devuelvan resultados fiables en validación y test. Para obtener este buen conjunto de datos y que sea útil a la hora del entrenamiento, es necesario contar con un volumen considerable de imágenes. Por este motivo, se buscó la manera de generar el mayor número posible de imágenes en el menor tiempo posible. Como los modelos han sido entrenados previamente el número de imágenes no necesita ser tan elevado ya que los modelos ya han aprendido representaciones generales útiles para la tarea de conteo de unidades.

Carreras cuenta con un almacén en el que hay movimiento en todo momento. Puesto que no se puede interrumpir todo el flujo de movimiento de productos, se construye una zona de captura de datos dentro de la nave con condiciones equivalentes a las del flujo real. Se elige una submuestra de productos representativa del conjunto de productos que se mueven en el almacen.

Como el entrenamiento de vision por computador en la empresa esta en desarrollo, se intenta simplificar la situación. Para ello, se elige una muestra de productos que sea generica y extrapolable (productos embalados con forma de caja), ya que abarcan un alto porcentaje del total de productos que se manejan en el almacen. Y en la disposición del

palé, solo existan dos orientaciones (vertical y horizontal). Teniendo en cuenta todo esto, se eligen dos productos principales: cajas de vino blanco y cajas de salsa basílico. A estos, se les suma un cojunto de productos de vinagres.

Se procede a la captura de imágenes. Para obtener las fotos se utiliza una webcam Logitech y se desarrolla una aplicación que toma una imagen cada 10 segundos, un intervalo suficiente para permitir el movimiento de cajas y evitar que las imágenes fueran idénticas. Este *script* organiza las imágenes en carpetas cuyo nombre es el número de unidades en el palé, de manera que al finalizar el proceso se sabe perfectamente cuántas unidades hay en cada imagen. El nombre de cada imagen también contiene el número de unidades correspondiente. Como medida de comodidad, se realiza un aviso sonoro los tres segundos previos a la captura para dar tiempo al usuario a que la imagen contenga la máxima cantidad de información relevante.

Es importante anticiparse a los posibles problemas relacionados con las imágenes. Estos incluyen la oclusión parcial, la variabilidad en las condiciones de iluminación y la perspectiva de la cámara. Para mitigar estos factores y asegurar que el entrenamiento contemple distintas condiciones, se emplean tres perspectivas de cámara diferentes, evitando que todas las imágenes tengan el mismo ángulo. Las fotografías se toman dentro del propio almacén para mantener unas condiciones de iluminación similares a las de un caso real. También se considera la altura a la que los operarios podrían capturar imágenes con su PDA (dispositivo móvil utilizado por los operarios), por lo que se apilan palés de madera hasta alcanzar una altura media y se coloca el portátil con la webcam sobre ellos (ver Figura 3.1).

Una vez finaliza el proceso de captura, se suben las imágenes a la plataforma *Roboflow*, que permite dividir automáticamente las imágenes en los conjuntos de *train*, *val* y *test*. Para añadir la etiqueta `count`, que representa el número de unidades presentes en la imagen, se utiliza un *script* en `Python` que aprovecha el nombre de los archivos para extraer el número de unidades y añadirlo como anotación. Se obtiene una carpeta con las imágenes ya divididas en sus *splits*, junto con sus correspondientes anotaciones en formato COCO.

A continuación, se describe cómo están divididos los distintos *datasets*.

Figura 3.1: Setup de captura de imagenes.

3.1.1. Dataset Vinos Blancos

En este *dataset* se busca que el producto tenga una forma geométrica sencilla para ver como evaluan los modelos. Este *dataset* cumple con este requisito ya que su forma es un prisma quadrangular, se puede ver en las Figuras 3.3 3.4.

Este *dataset* está formado por 594 imágenes del palé que contiene cajas del producto de vino blanco. En este palé hay 22 unidades de producto, por lo cual el conteo abarca valores de 0 a 22. Se incluye también el caso de palé vacío para fines de experimentación, aunque en la práctica, si un palé se vacía, se repone con uno nuevo. La división del *dataset* realizada mediante *Roboflow* genera la distribución que se puede observar en la Figura 3.2.

De cada número de productos se comienza tomando unas 30 imágenes, reduciendo esa cantidad cuando el número de unidades es menor o igual que cinco. Además, del palé vacío se capturan 10 imágenes. Las cajas se iban moviendo entre capturas, aunque siempre se intentó mantener dos orientaciones principales (horizontal y vertical), se puede ver en las Figuras 3.3 3.4.

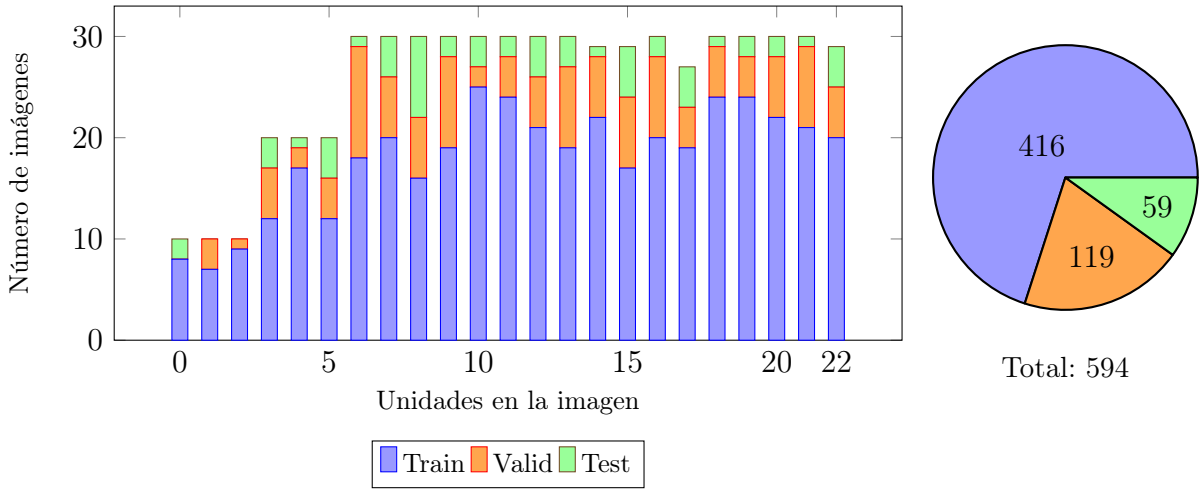

Figura 3.2: Distribución del dataset Vinos Blancos.

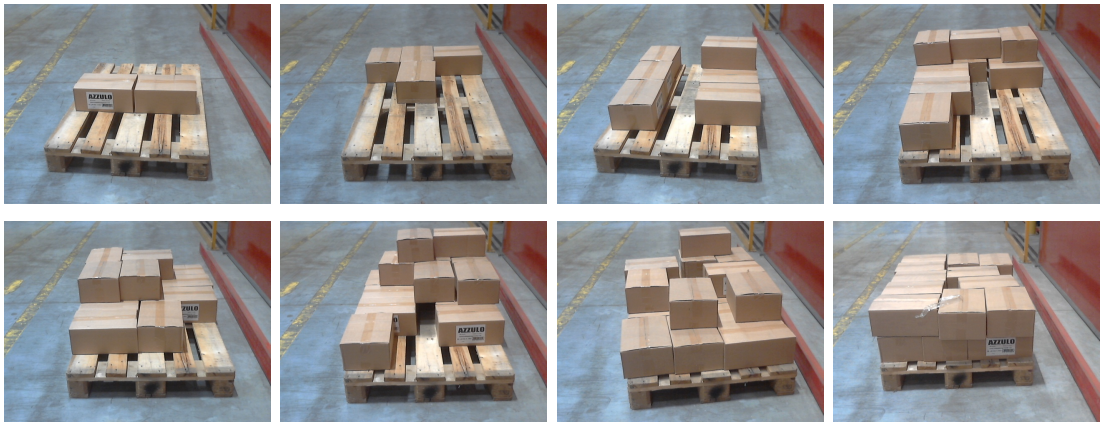

Figura 3.3: Ejemplos de diferentes unidades de Vinos Blancos.

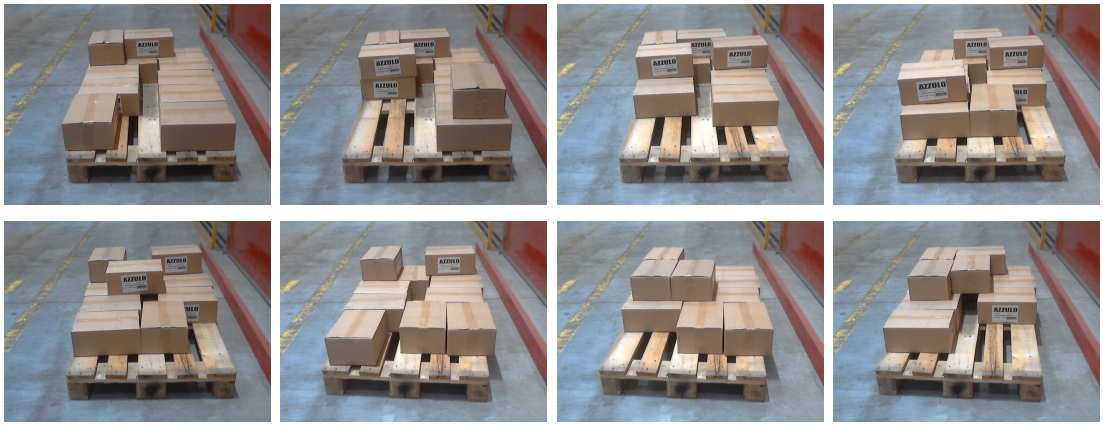

Figura 3.4: Ejemplos de 12 unidades de Vinos Blancos.

3.1.2. Dataset Salsas Basílico

Para el segundo *dataset* se busca que la forma geométrica sea un poco mas complicada de reconocer, así se puede evaluar a los modelos en formas más complejas. En este *dataset* la forma de las unidades sigue siendo un prisma quadrangular pero la caja tiene un plástico transparente en el que se pueden ver a través los tarros de la salsa que se asemejan a cilindros, se puede ver en las Figuras 3.6 3.7.

Inicialmente el palé contiene 94 unidades pero para realizar pruebas y entrenamientos con *datasets* que contengan un número similar de imagenes se decide reducir las unidades. Se rebaja el numero de unidades a 48, un poco más del doble que el *dataset* Vinos Blancos, y así al realizar unas 20 fotografías por conjunto de unidades quedan dos *datasets* de la misma magnitud. De igual forma que el otro *dataset* cuando quedan pocas unidades en el palé se reducen el número de fotos capturadas, tambien se cuenta el caso del palé vacio. Al terminar el proceso de captura el *dataset* contiene 804 imagenes en total. La división

del *dataset* se realiza mediante *Roboflow* y genera la distribución que se puede obsevar en la Figura 3.5

Figura 3.5: Distribución del dataset Salsas Basílico.

Figura 3.6: Ejemplos de diferentes unidades de Salsas Basílico.

Figura 3.7: Ejemplos de 24 unidades de Salsas Basílico.

3.1.3. Dataset Vinagres Variados

Este *dataset* tiene como objetivo ser utilizado en los experimentos para comprobar como evaluan los modelos con conjuntos de datos más pequeños. Por ello, contiene menos imágenes que los dos anteriores y en lugar de tener un solo producto, incluye dos. Estos dos productos son dos tipos diferentes de vinagre: uno de ellos tiene una forma de prisma rectangular de color blanco (ver Figura 3.9) para comprobar cómo evaluan los modelos con imagenes en las que las sombras son más notorias. El otro vinagre es un pack de 12 botellas de cristal (ver Figura 3.9), cuya forma no es tan sencilla de evaluar como los productos de los *datasets* anteriores.

El objetivo es obtener imágenes de 50 unidades de cada producto. Como solo se toman tres fotos por unidad, esto permite evaluar cómo se comportan los modelos con un número reducido de imágenes. En este caso, se ha descartado la opción de tener el palé vacío. La división del *dataset*, realizada mediante *Roboflow*, genera la distribución que se puede observar en la Figura 3.8.

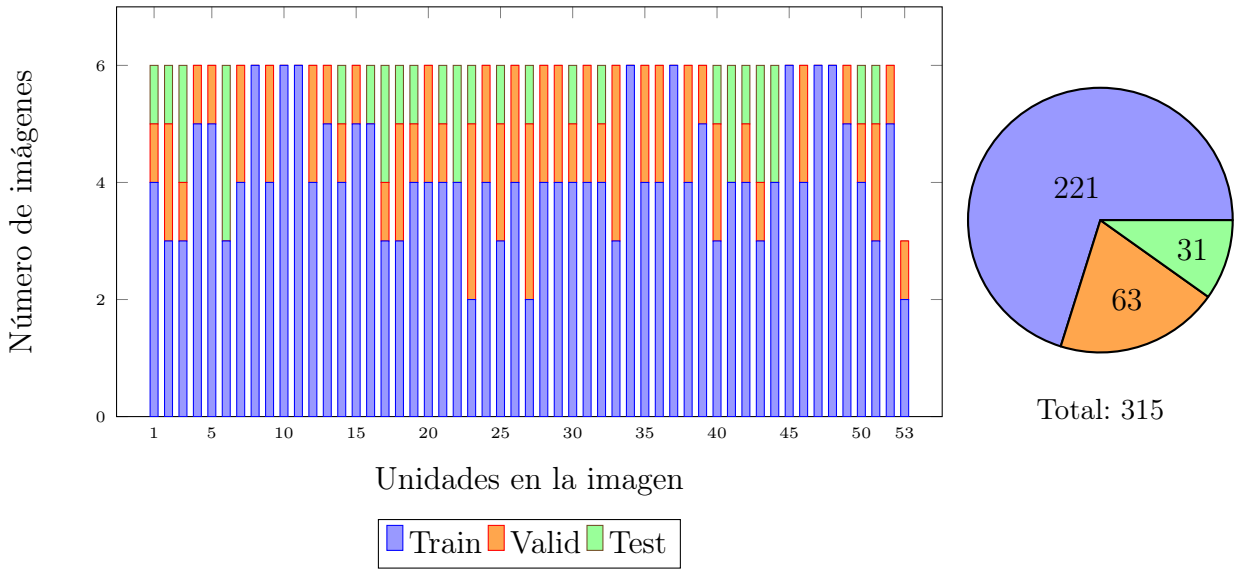

Figura 3.8: Distribución del dataset Vinagres Variados.

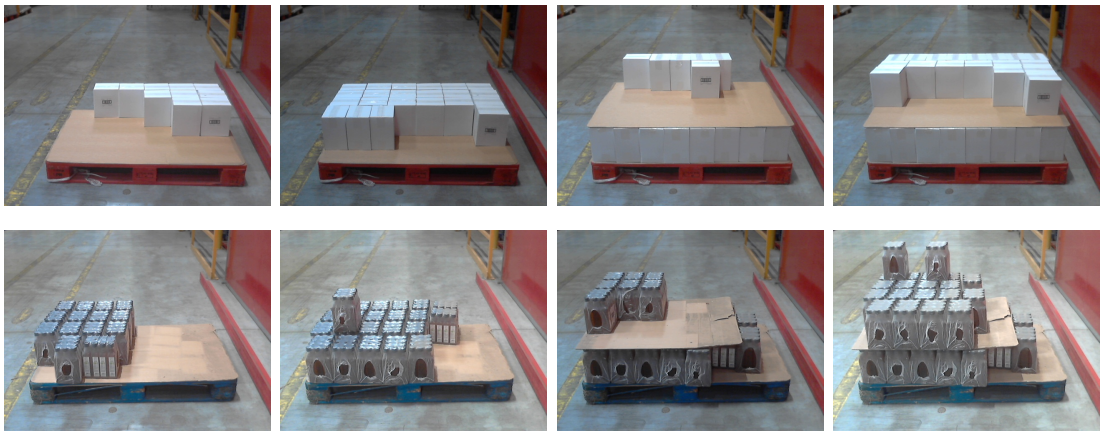

Figura 3.9: Ejemplos de diferentes unidades de Vinagres Variados.

3.1.4. Dataset Conjunto

Para este *dataset* lo que se hace es juntar las imagenes de los *datasets* anteriores con la finalidad de poder realizar pruebas y entrenamientos cuando el conjunto contiene varios productos diferentes. En este *dataset* se van a realizar una distribucion juntando los *datasets* de Vinos Blancos, Salsas Basílico y Vinagres Variados.

La distribución sumando los tres *datasets* es de 1713 imagenes en total. Se suben todas las imagenes a *Roboflow* y desde ahí se realiza una nueva división que genera la distribución que se puede ver en la Figura 3.10

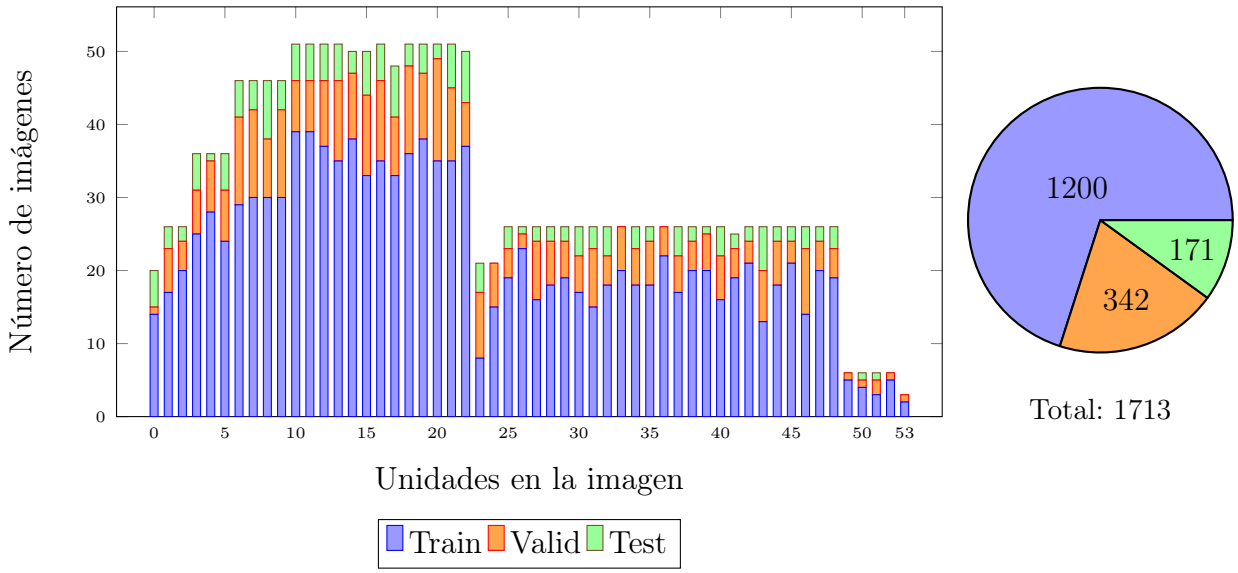

Figura 3.10: Distribución del dataset conjunto Vinos Blancos, Salsas Basílico y Vinagres Variados.

3.2. Baselines

Una vez ya se tienen preparados los *datasets* y se han elegido los modelos con los que se va a trabajar, se puede comenzar a realizar pruebas y entrenamientos. El entrenamiento está condicionado por las circunstancias del entorno: la empresa esta en desarrollo de proyectos de estas características, por lo que hay ausencia de herramientas habitualmente presentes en sistemas de visión por computador. No se cuenta con un ordenador con GPU ni con servidores que contengan tarjetas gráficas, lo que influye directamente en que los entrenamientos deben realizarse desde la CPU, incrementando notablemente su duración.

Por ello, se plantean varios experimentos que no impliquen un coste computacional elevado ni duraciones de varios días. Durante las pruebas con los modelos se intenta adaptar las arquitecturas base para poder trabajar con nuestros *datasets* y con *features* extraídos de los *backbones*. La decisión de trabajar con estos *features* se toma porque permite reducir considerablemente los tiempos, ya que evita reentrenar todo el modelo.

Los *features* se extraen desde la última capa del *backbone* para capturar representaciones de alto nivel que resumen la semántica completa de la imagen. Esta capa proporciona información más abstracta, ideal para tareas como el recuento de objetos. Los *features* obtenidos se pasan a una MLP externa, que se construye y entrena por separado. Únicamente se entrena el MLP lo que reduce el tiempo total necesario para el entrenamiento.

Durante esta fase se aplican varias técnicas para evitar el *overfitting*, como el uso de *early stopping* utilizando parte del *dataset* como validación, la inclusión de capas de

dropout en la MLP o la reducción del número de épocas.

El entorno de trabajo para los modelos *Detectron2*, *Learning To Count Everything* y *Learning to Count Anything* se configura mediante `Anaconda` en un sistema basado en `Python`. Las dependencias vienen especificadas por los autores y se instalan de forma manual o mediante archivos `environment.yml` proporcionados por los autores. El entorno se ejecuta completamente en CPU, sin necesidad de GPU, y se utilizan herramientas como `PyTorch Lightning` y `TorchMetrics` para el entrenamiento y la evaluación de los modelos. La estructura de los proyectos se mantiene según el repositorio original, adaptando únicamente rutas y configuraciones específicas para el conjunto de datos propio.

A continuación están los modelos referencia, los cuales se toman como puntos de partida.

Adaptación directa de Detectron2

En este *baseline* se busca adaptar directamente el modelo de *Detectron2* para evaluar su rendimiento sobre nuestras imágenes. Este modelo devuelve las *bounding boxes* y puntuaciones de confianza asociadas a los tipos de objetos para los que fue entrenado. La estructura se puede ver en la Figura 3.11.

Para adaptarlo a nuestro objetivo de conteo automático, se modifica el código del *ROI Head* con el fin de contar el número de *bounding boxes* detectadas y devolver dicho recuento como predicción. Se valora filtrar las *bounding boxes* por debajo de un porcentaje de confianza mínimo antes de realizar el recuento pero se decide no hacerlo, ya que se considera importante incluir también las instancias con puntuaciones bajas, que pueden aportar al conteo total pese a su menor certeza.

Dado que una de las clases del conjunto de datos COCO con la que fue entrenado el modelo es *box*, el modelo resulta ser capaz de detectar adecuadamente las cajas visibles sin oclusión.

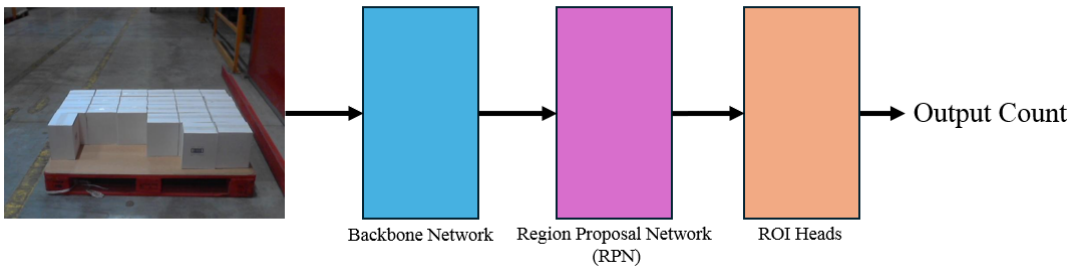

Figura 3.11: Adaptación directa del modelo Detectron2.

Fine-tuning de un MLP externo en Detectron2

En este *baseline*, se procede a extraer los *features* del *backbone* del modelo, utilizando las imágenes propias como entrada. Estos *features* se extraen de la capa `p6` del *Feature Pyramid Network* (FPN) aprovechando su mayor nivel de abstracción. A partir de estos *features*, se entrena una red neuronal *MLP* (Multi-Layer Perceptron) sencilla cuya tarea es predecir el número de unidades visibles en cada palé. La estructura se puede ver en la Figura 3.12.

El entrenamiento se lleva a cabo utilizando supervisión basada en conteo total por imagen, sin necesidad de anotaciones precisas a nivel de píxel o segmentación. Este enfoque, dentro del *weakly supervised learning*, permite reducir significativamente los costes de anotación. Se utiliza un conjunto de validación independiente que permite monitorizar la generalización del modelo durante el entrenamiento y ajustar *early stopping* o regularización si fuera necesario.

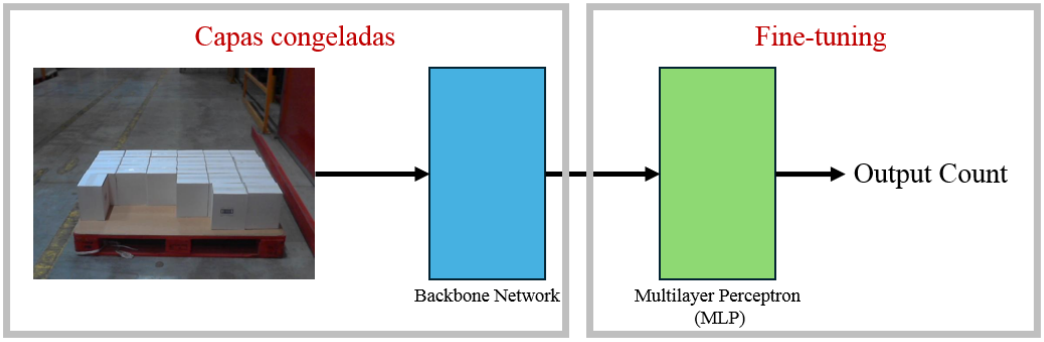

Figura 3.12: Fine-tuning de un MLP externo en Detectron2.

Adaptación directa de LTCEverything

Este *baseline* tiene como objetivo inicial evaluar la capacidad del modelo para reconocer las imágenes propias sin necesidad de entrenamiento adicional. El modelo *LTCEverything* genera como salida un mapa de densidad que indica la distribución espacial de los objetos en la imagen. A partir de este mapa, se obtiene el recuento total mediante la suma de todos los valores de densidad. La estructura se puede ver en la Figura 3.13.

Al modelo se le proporciona una imagen de referencia junto con las coordenadas de las cajas en dicha imagen, actuando como ejemplos de los objetos de interés. Se selecciona una única instancia del objeto en orientación vertical y otra en orientación horizontal, asi el sistema tiene dos referencias de las que puede aprender las variaciones más representativas con una supervisión mínima. Esta configuración permite que el modelo generalice la tarea de conteo a otras regiones de la imagen o a nuevas imágenes sin ajustes en los pesos del

modelo.

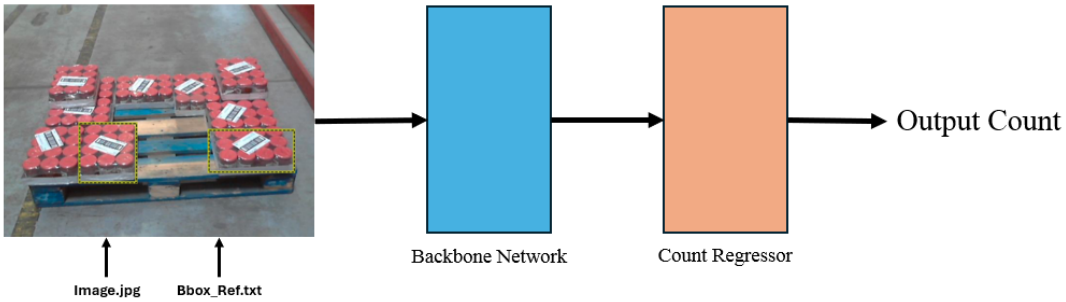

Figura 3.13: Adaptación directa del modelo LTCEverything.

Fine-tuning del CountRegressor en LTCEverything

En este *baseline*, el foco se encuentra en la segunda parte de la arquitectura del modelo: el *CountRegressor*. Dado que se pretende reutilizar este módulo manteniendo congelado el extractor de características (*backbone*), es necesario proporcionarle como entrada los *features* exactamente en el mismo formato en que los recibe durante la inferencia habitual. La estructura se puede ver en la Figura 3.14.

A diferencia del MLP externo que se esta utilizando en otros *baselines*, el *CountRegressor* no espera una entrada vectorial aplanada, ni se aplica ninguna operación de *pooling* que reduzca las dimensiones espaciales. El diseño del módulo requiere preservar la estructura bidimensional del mapa de características, ya que internamente aplica convoluciones para generar el mapa de densidad a partir del cual se realiza el conteo final.

El principal inconveniente de este *baseline* esta en que en el conjunto de datos propio no tiene anotaciones detalladas, como *bounding boxes* para cada instancia. Únicamente se dispone del valor total de unidades por imagen, lo que limita la supervisión y la capacidad del modelo para aprender correspondencias espaciales entre las regiones del mapa y los objetos presentes.

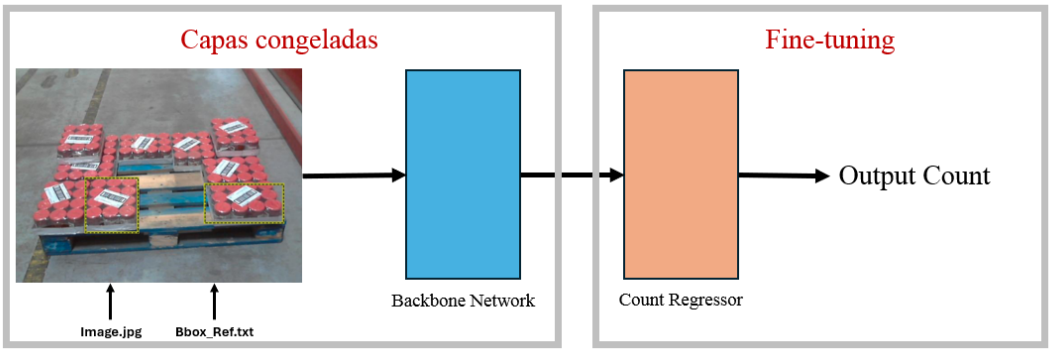

Figura 3.14: Fine-tuning del CountRegressor del modelo LTCEverything.

Fine-tuning de un MLP externo en LTCEverything

Este *baseline* es similar al anterior, con la diferencia de que, en lugar de entrenar el *CountRegressor*, se ha optado por entrenar un MLP sencillo que fue creado previamente para el *baseline* con el modelo *Detectron2*. La estructura se puede ver en la Figura 3.15. El objetivo de este enfoque es evaluar si un perceptrón multicapa independiente es capaz de aprender a estimar el número de objetos en la imagen a partir de los *features* extraídos del *backbone* del modelo. Para extraer los *features*, se utiliza la capa `map4` del *backbone*, ya que proporciona un buen compromiso entre nivel de detalle espacial y contenido semántico.

En este caso, al extraer los *features* del *backbone*, sí se ha aplicado un *pooled* para reducir las dimensiones espaciales del mapa de características y así poder introducirlos directamente al MLP. Este tipo de reducción resulta apropiado cuando se trabaja con arquitecturas que esperan entradas aplanadas y evita un coste computacional elevado.

Este enfoque permite comparar directamente la capacidad de generalización del MLP frente al *CountRegressor*, observando cómo distintas formas de procesar los mismos *features* afectan al rendimiento del modelo.

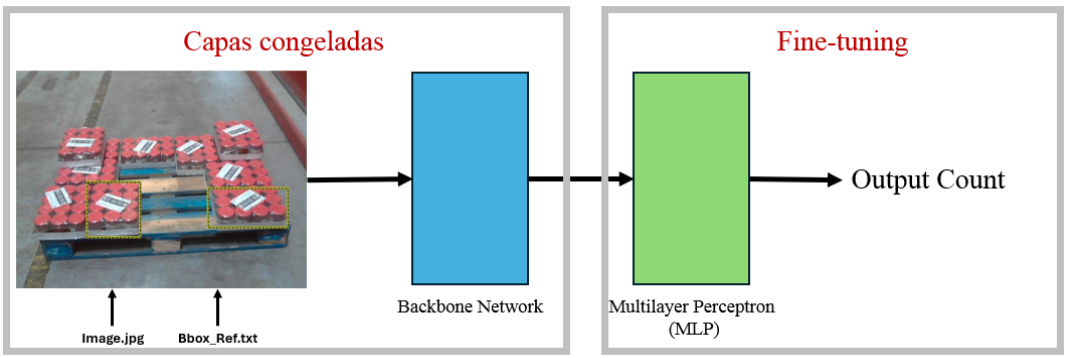

Figura 3.15: Fine-tuning de un MLP externo en el modelo LTCEverything.

Adaptación directa de LTCAnything

Para esta primer *baseline* del *LTCAnything* se ha seguido el mismo planteamiento que en los modelos anteriores: evaluar el rendimiento del modelo sobre nuestro conjunto de datos sin realizar ajustes en los pesos. El modelo devuelve como salida un mapa de densidad desde el cual se obtiene la predicción final de conteo total, sumando todos los valores del mapa. La estructura se puede ver en la Figura 3.16.

No ha sido necesario modificar la arquitectura del modelo, sino que las adaptaciones se han centrado exclusivamente en los módulos auxiliares del sistema. Se han editado las funciones de la carpeta `utils` para adaptar las rutas de entrada, el formato de las anotaciones y la estructura del dataset a los requisitos esperados por *LTCAnything*.

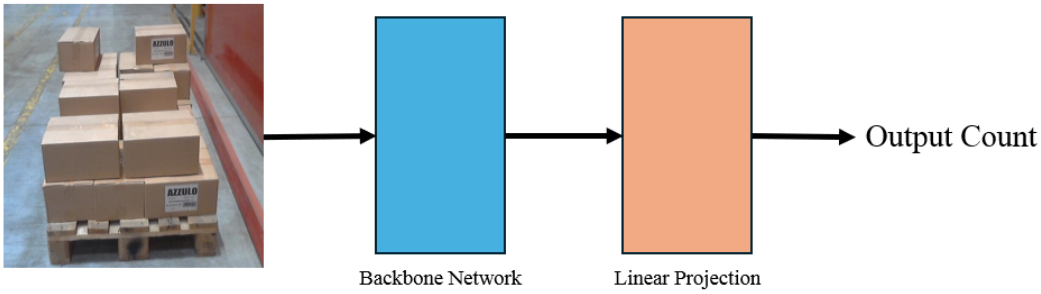

Figura 3.16: Adaptación directa del modelo LTCAnything.

Fine-tuning de un MLP externo en LTCAnything

En este *baseline*se han extraído los *features* del *backbone* y se han pasado a un MLP sencillo que ha sido entrenado para predecir el número de unidades presentes en la imagen. La estructura se puede ver en la Figura 3.17. Para ello, se ha aplicado una operación de *pooled* que reduce las dimensiones espaciales del mapa de características, facilitando así su uso como entrada de una red densa. Este MLP, previamente diseñado para otro *baseline* del *Detectron2*, actúa como regresor directo, aprendiendo a mapear representaciones visuales de alto nivel a un valor escalar de conteo.

El conjunto de datos se divide en entrenamiento y validación, manteniendo así un control sobre la generalización del modelo y evitando el *overfitting*. El modelo se entrena durante 300 épocas, y se guarda automáticamente cuando se detecta una mejora en el error de validación.

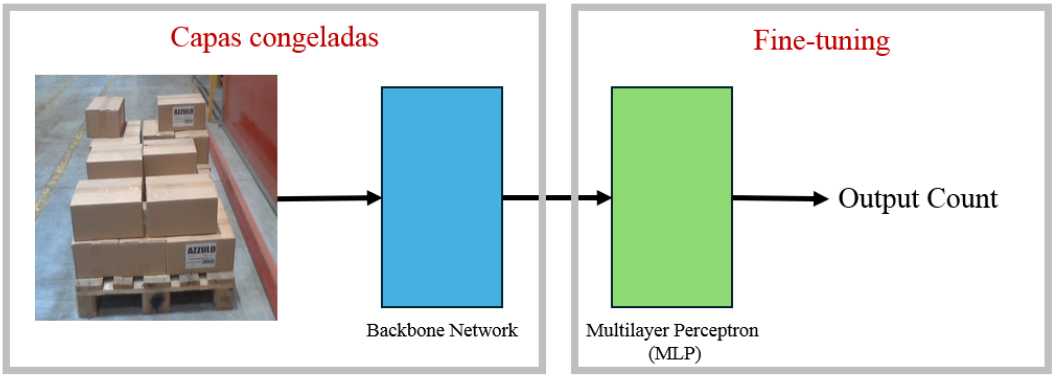

Figura 3.17: Fine-tuning de un MLP externo en LTCAnything.

Fine-tuning completo del modelo LTCAnything

Dado que este modelo no requiere anotaciones de *bounding boxes* u otras anotaciones para entrenar, y considerando los buenos resultados obtenidos en el *baseline* anterior, se ha optado por realizar un *fine-tuning* completo del modelo con el dataset propio. La estructura se puede ver en la Figura 3.18.

Esta estrategia permite ajustar todos los parámetros del modelo original a las características específicas del nuevo dominio, lo que puede mejorar la precisión del conteo en escenarios con variaciones visuales relevantes. El entrenamiento se realiza directamente sobre los mapas de densidad generados por el modelo.

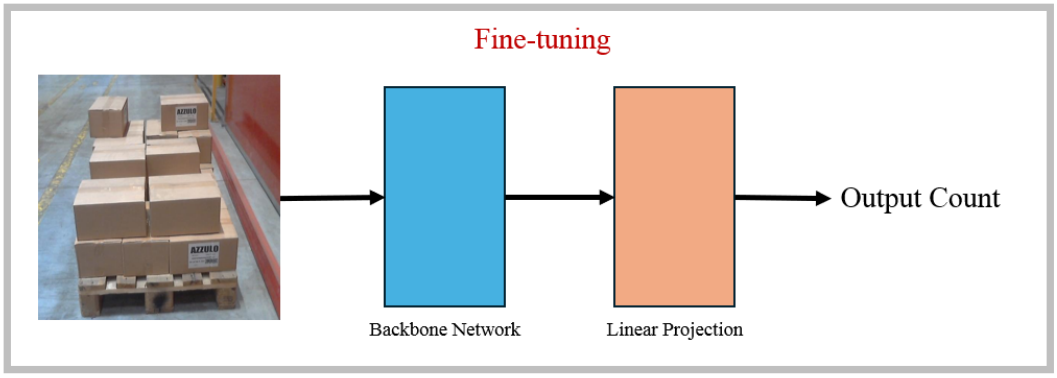

Figura 3.18: Fine-tuning completo en LTCAnything.

Capítulo 4

Experimentos

En este apartado se comentan los diferentes experimentos propuestos, las herramientas utilizadas y las métricas de evaluación, con el objetivo de que los resultados obtenidos puedan compararse y replicarse en condiciones similares.

4.1. Experimentos propuestos

Partiendo de los *baselines* definidos previamente, así como de los conjuntos de datos, se procede a establecer el entorno experimental sobre el que se desarrollarán los siguientes estudios. Todos los experimentos se ejecutan en un entorno de trabajo controlado a través de `Anaconda Prompt`, lo que facilita la gestión de dependencias, la ejecución reproducible de *scripts* y el lanzamiento eficiente de cada uno de los *baselines*.

Para los experimentos cada conjunto de datos ha sido dividido en tres particiones: 70 % para entrenamiento (train), 20 % para validación (valid) y 10 % para prueba (test), con el objetivo de evaluar adecuadamente el rendimiento del modelo en diferentes fases del proceso de aprendizaje.

4.1.1. Experimento 1: Entrenamiento en un Dataset

El objetivo de este primer experimento es analizar cómo se comporta cada uno de los *baselines* cuando se entrenan y evalúan sobre un conjunto de datos que contiene únicamente instancias de un único tipo de producto. También se comprobará cómo se comportan cuando el conjunto de datos tiene pocas instancias y contiene un par de productos. Este enfoque permite observar el rendimiento de los modelos en un escenario más controlado y homogéneo, donde además existen pequeñas variaciones en la orientación del objeto a

contar.

Se han seleccionado dos *dataset*: Vinos Blancos 3.1.1 y Salsas Basílico 3.1.2, que presentan una cantidad razonable de imágenes por categoría. Estos conjuntos permiten entrenar modelos robustos sin introducir variabilidad excesiva, facilitando así una evaluación más precisa del comportamiento base de cada modelo. Tambien se entrenará y evaluará con el *dataset* de Vinagres Variados 3.1.3 que contiene la mitad de imágenes que los dos *datasets* anteriores.

4.1.2. Experimento 2: Entrenamiento sobre capacidad de generalización

Este experimento tiene como finalidad evaluar la capacidad de los modelos para generalizar el conteo de objetos más allá del dominio específico en el que fueron entrenados. En entornos reales, los modelos no siempre se enfrentan a imágenes del mismo tipo de producto que vieron durante el entrenamiento, por lo que resulta fundamental analizar su comportamiento ante productos no vistos previamente.

Para ello, se realiza una prueba cruzada entre *datasets*. Se utiliza los *datasets* de Vinos Blancos 3.1.1 y Salsas Basílico 3.1.2, entrenando el modelo con una parte del *dataset* de uno de los productos y evaluándolo con una parte del otro. Esta configuración permite analizar el grado de transferencia entre productos visualmente distintos pero con ciertas similitudes estructurales. Se hace lo mismo con el *dataset* de Vinagres Variados 3.1.3 y los *datasets* anteriores. Esta prueba busca comprobar hasta qué punto el modelo es capaz de mantener un rendimiento aceptable cuando se enfrenta a escenarios más abiertos y diversos.

4.1.3. Experimento 3: Entrenamiento con varios productos

Este experimento tiene como objetivo analizar el rendimiento de los modelos cuando el conjunto de imágenes contiene múltiples tipos de productos. Se busca evaluar su capacidad de conteo en escenarios más complejos y realistas, donde conviven objetos con diferentes formas, tamaños y texturas. Se realiza la siguiente prueba: se combinan los *datasets* de Vinos Blancos 3.1.1, Salsas Basílico 3.1.2 y Vinagres Variados 3.1.3 en un *dataset* nuevo llamado *Dataset* Conjunto 3.1.4. Este *dataset* contiene una nueva distribución de train, valid y test con el objetivo de evaluar el rendimiento de los modelos en contextos en los que existen varios productos.

4.2. Herramientas

Para el desarrollo, entrenamiento y evaluación de los modelos se han empleado diversas herramientas y entornos. A continuación, se resumen brevemente las más relevantes junto a su función en la implementación:

- **PyTorch:** biblioteca de código abierto para el desarrollo de modelos de aprendizaje profundo. Permite definir redes neuronales de forma flexible y ejecutar operaciones sobre GPU para acelerar el entrenamiento. Se ha utilizado para definir y entrenar redes neuronales personalizadas, como el *MLP*, así como para manejar el flujo de datos, la optimización y el cálculo de pérdidas.
- **PyTorch Lightning:** *framework* que abstrae la estructura de entrenamiento en PyTorch, facilitando la organización del código y la ejecución eficiente de experimentos con soporte para validación y *callbacks*. Se ha empleado en los experimentos con el modelo *LearningToCountAnything* en funciones como el *early stopping* o el guardado automático del mejor modelo.
- **TorchMetrics:** biblioteca para calcular métricas comunes en aprendizaje automático. Se ha utilizado para evaluar de forma consistente el rendimiento de los modelos durante el entrenamiento y validación.
- **Visual Studio Code (VSCode):** editor de código fuente ligero y extensible. Ha sido el entorno principal de desarrollo para escribir y depurar *scripts* en Python.
- **Anaconda Prompt:** terminal de Anaconda que permite gestionar entornos virtuales y ejecutar scripts en entornos controlados. Se ha utilizado para lanzar los experimentos y gestionar dependencias.
- **OpenShift:** plataforma de orquestación de contenedores basada en Kubernetes. Se ha utilizado para desplegar las primeras pruebas de*Detectron2* debido a que los entrenamientos son muy extensos. Para ver más detalle sobre el proceso se puede ver en el Anexo A.1
- **Bucket S3 en AWS:** almacenamiento en la nube utilizado para guardar *datasets*, modelos y resultados. Permite un acceso eficiente y seguro desde diferentes entornos de ejecución. Se ha utilizado para guardar los *checkpoints* y pesos de los modelos entrenados en OpenShift.

4.3. Métricas de evaluación

Para la comparación de los resultados de los experimentoss se utilizan dos métricas comunes en trabajos previos relacionados con el conteo de objetos [2] [3]: MAE y RMSE.

- **MAE (Mean Absolute Error)**: mide la media de los errores absolutos entre los valores predichos y los reales. Es una métrica intuitiva que indica, en promedio, cuánto se desvía la predicción del valor real.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **RMSE (Root Mean Squared Error)**: mide la raíz cuadrada de la media de los errores al cuadrado. Penaliza más los errores grandes y es útil cuando se desea dar más peso a desviaciones importantes.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Donde en las métricas:

- n: número total de imágenes.
- y_i: conteo real en la imagen i.
- $\hat{y}_i$: conteo predicho por el modelo en la imagen i.

Capítulo 5

Evaluación y análisis de resultados

En este capítulo se presentan los resultados obtenidos a partir de los diferentes modelos y experimentos realizados. Con la comparación de estos resultados se pretende analizar cómo evalúan los modelos sobre los distintos *datasets* y cuáles rinden mejor en los tres experimentos.

Los resultados obtenidos por el *baseline* que trata del *fine-tuning* del *CountRegresor* del *LTCAnything* no se ven reflejados en las Figuras de este capítulo. Este *baseline* requiere anotaciones detalladas, como *bounding boxes* para cada instancia, que el conjunto de datos propio no tiene. Como consecuencia, al evaluar el rendimiento mediante métricas estándar como el error medio absoluto (MAE) o la raíz del error cuadrático medio (RMSE), se observan valores significativamente más elevados que en otros *baselines*. Por esta razón, los resultados obtenidos en este *baseline* no se tendrán en cuenta en el análisis final, aunque ofrecen información valiosa sobre las limitaciones del entrenamiento exclusivamente con conteos globales.

5.1. Evaluación experimento en un dataset

En este apartado se muestran los resultados de las pruebas utilizando únicamente un *dataset*. Se utilizan los *dataset* de Vinos Blancos 3.1.1, de Salsas Basílico 3.1.2 y de Vinagres Variados 3.1.3. A continuación, se comentarán los resultados de las Gráficas 5.4 5.5 5.6.

Para todos los modelos, se observa una mejora cuando se realiza cualquier *fine-tuning* en comparación con el modelo básico, a excepción de *Detectron2* en el *dataset* de Salsas Basílico (ver Figura 5.5). El modelo de *Detectron2* no varía sus resultados post entrenamiento, seguramente porque el modelo básico ha sido preentrenado con el *dataset* de COCO, que no contenía imágenes parecidas al producto de las Salsas Basílico, a diferencia

del *dataset* de Vinos Blancos y Vinagres Variados, que sí contienen cajas similares a las del *dataset* de COCO.

A simple vista, se puede ver que los resultados de los *dataset* Salsas Basílico y Vinagres Variados, ver Figuras 5.5 5.6, han sido peores que los obtenidos con el *dataset* de Vinos Blancos, Figura 5.4. Esto es justo lo que se quería al buscar *datasets* cuyas unidades tuvieran una forma geométrica más complicada o que dentro del *dataset* haya dos productos diferentes.

Los *fine-tuning* no permiten mostrar la imagen debido a que se realizan a partir de los *features*, pero se puede observar cómo funcionan las imágenes con los modelos básicos y asumir una mejora con el entrenamiento. Se comentan las imagenes de los tres *dataset* de forma general, si se quiere ver más imagenes de cómo evaluan los modelos ver Anexo A.2.

Se observa que los modelos básicos no son capaces de contabilizar las unidades que tienen oclusión total o parcial, ver Figura 5.1 5.2 5.3. Se puede ver la diferencia entre los modelos: *Detectron2* casi no es capaz de reconocer más de 3 o 4 unidades. Esto, sumado a que hay *datasets* que contienen el doble de unidades máximas, tiene sentido que en estos conjuntos de datos el error sea mayor que el doble. *LTCEverything* afina mucho más y se puede ver en los mapas de densidad (ver Figura 5.3) cómo es capaz de reconocer casi todas las unidades visibles. *LTCAnything* también es capaz de reconocer bastante bien las unidades visibles y, como se puede observar, aunque no tiene tanta actividad en el mapa de calor, marca con precisión los objetos que tiene claro que son unidades, ver Figura 5.2. Además estos dos últimos son capaces de entender que cuando el recubrimiento de la caja es transparente, lo que tienen que contar es la unidad que contiene los frascos, botellas... y no los frascos o botellas en sí.

El modelo básico con mejores resultados es el de *LTCEverything*, seguramente porque, aunque no está entrenado con el conjunto de datos propio, se le pasa la imagen de referencia con las coordenadas de las unidades de ejemplo. Destacar también que *LTCAnything* es el peor modelo básico de los tres, a pesar de que se había preentrenado previamente para realizar conteos.

Como se puede ver, el *baseline* que mejor resultados obtiene es el de realizar un *fine-tuning* de una capa MLP externa desde el modelo *LTCAnything*. Esta prueba acierta el número de unidades con un error más o menos cercano a 1, lo cual, en comparación al resto de pruebas, es una mejora notoria. Cabe destacar que, aunque en el *dataset* Salsa Basílico es más complicado realizar el conteo, obtiene mejores resultados que en la evaluación del *dataset* Vinos Blancos para la misma prueba, ver Figura 5.5. De tal manera, los resultados del *dataset* Vinagres Variados, ver Figura 5.6, han sido peores en comparación con los otros dos *datasets*. Estos resultados son totalmente normales, ya que Vinagres Variados contiene la mitad de las imágenes y tiene dos productos diferentes.

El porqué este baseline tiene mejores resultados que los otros baselines que incluyen también un MLP externo puede deberse a su backbone ViT-S, que, gracias al preentre-

namiento con DINO, extrae representaciones visuales buenas y generalizables.

El mejor rendimiento del entrenamiento con MLP externa frente al *fine-tuning* completo de *LTCAnything* puede deberse a que este enfoque conserva intactas las representaciones generales del modelo preentrenado. Al entrenar únicamente una capa superior sencilla, se aprovechan mejor las características ya aprendidas sin modificar la estructura interna del modelo base.

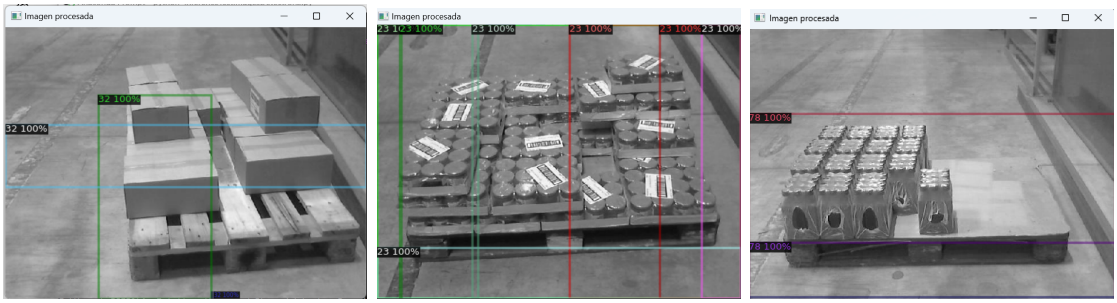

Figura 5.1: Ejemplos de diferentes interacciones con modelos básicos de Detectron2 en los datasets.

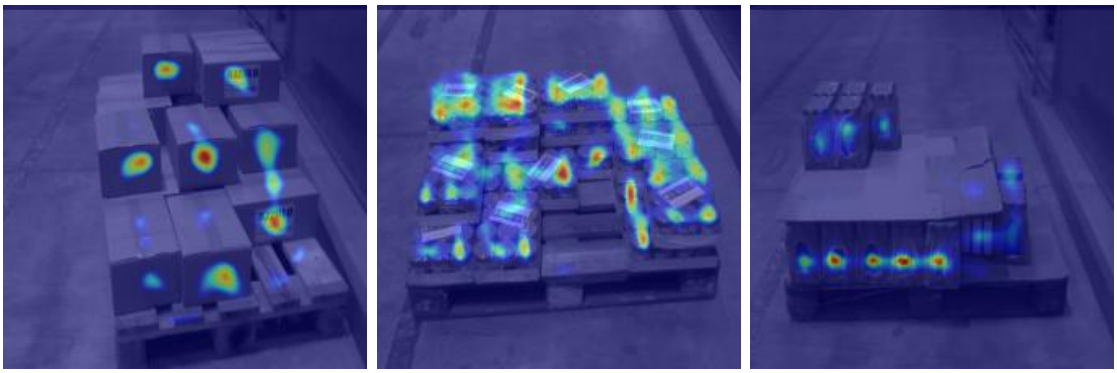

Figura 5.2: Ejemplos de diferentes interacciones con modelos básicos de LTCAnything en los datasets.

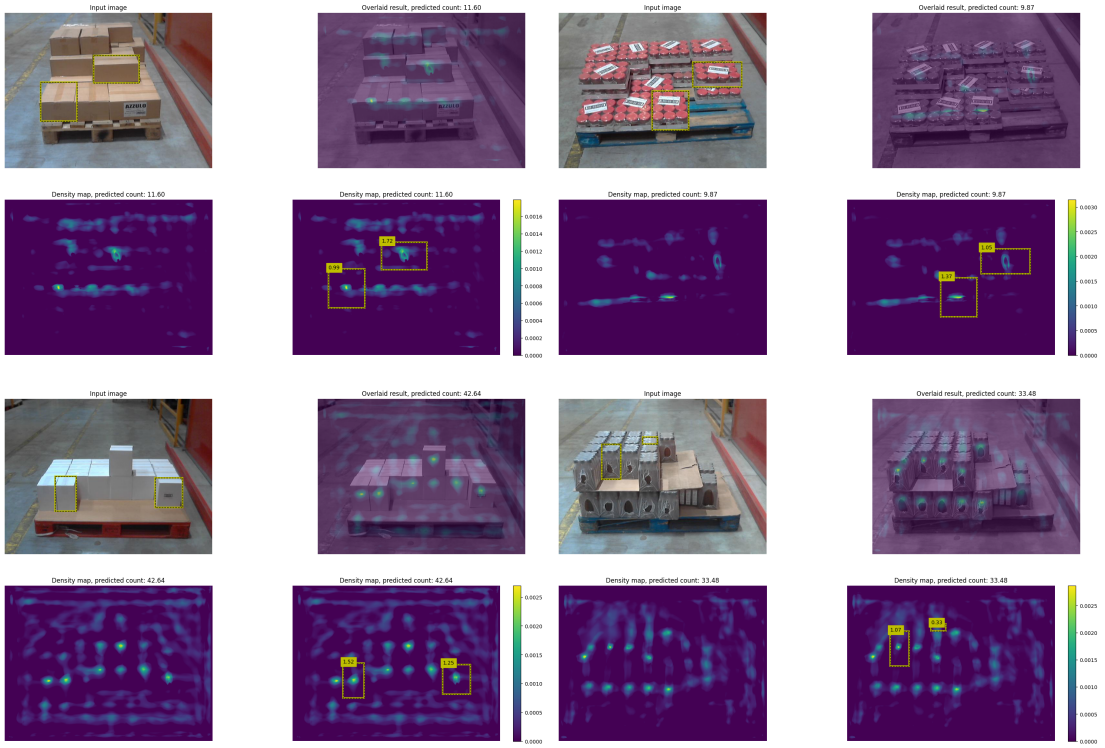

Figura 5.3: Ejemplos de diferentes interacciones con modelos básicos de LTCEverything en los datasets.

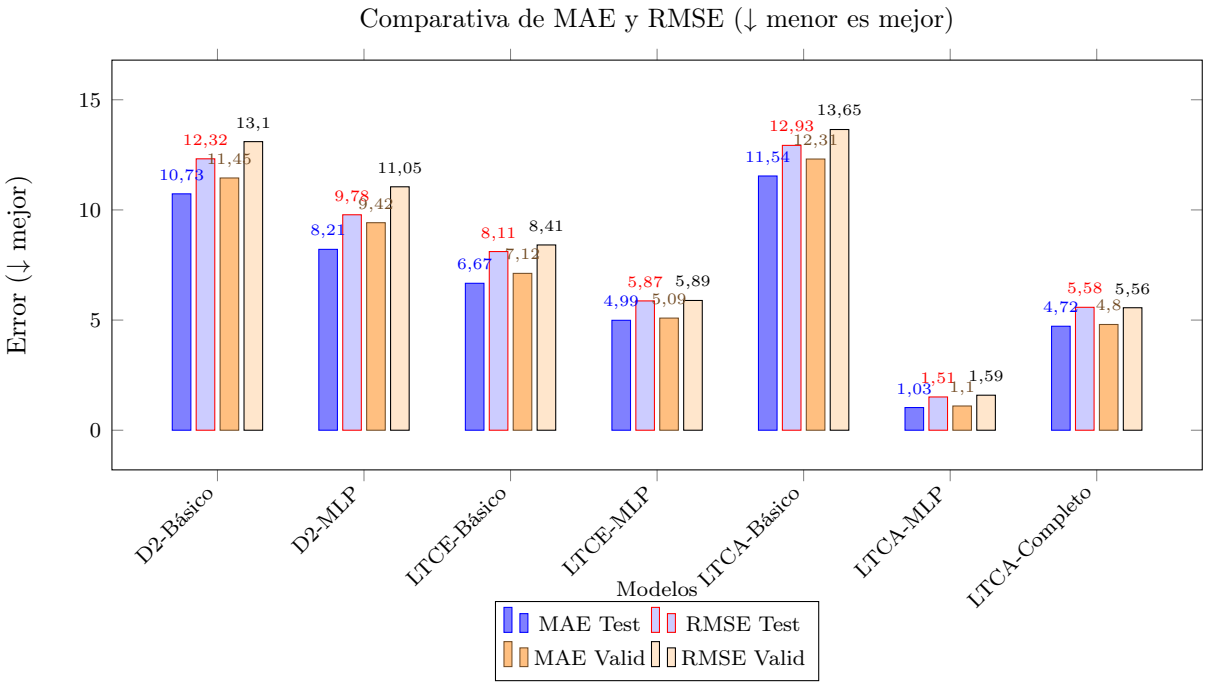

Figura 5.4: Comparativa de errores MAE y RMSE para cada modelo en los conjuntos de test y validación del dataset Vinos Blancos. Cuanto menor sea el valor, mejor rendimiento.

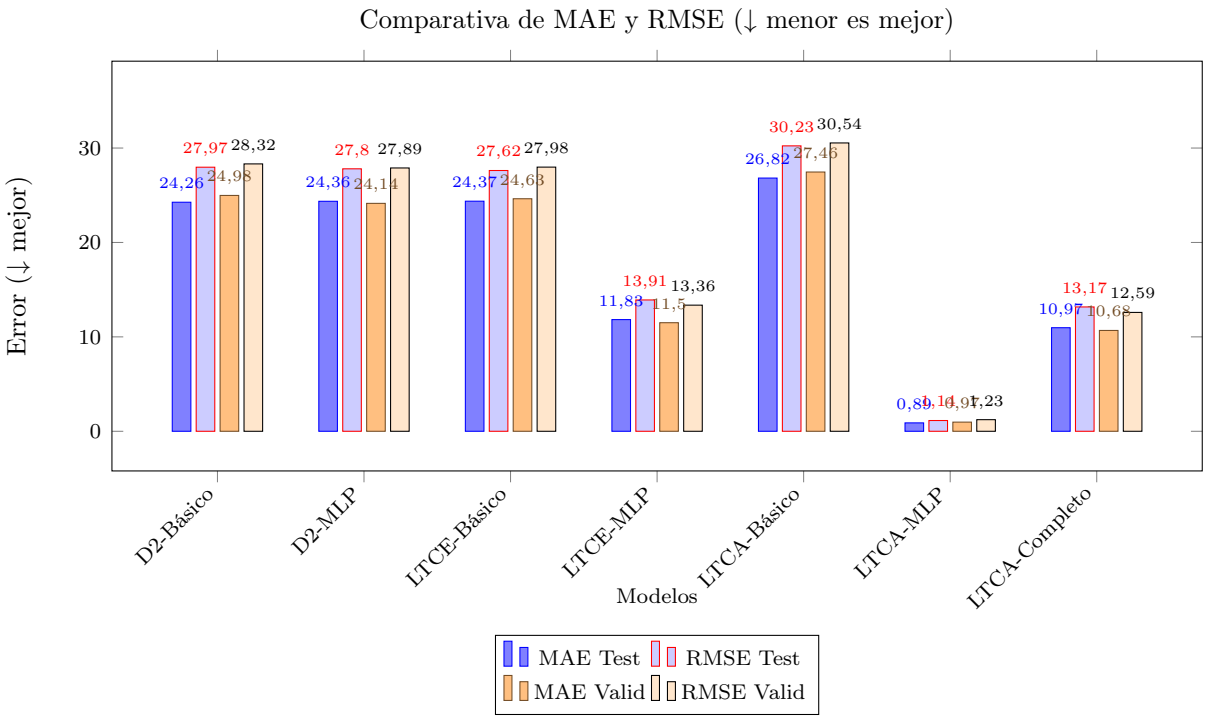

Figura 5.5: Comparativa de errores MAE y RMSE para cada modelo en los conjuntos de test y validación en el dataset Salsas Basílico. Cuanto menor sea el valor, mejor rendimiento.

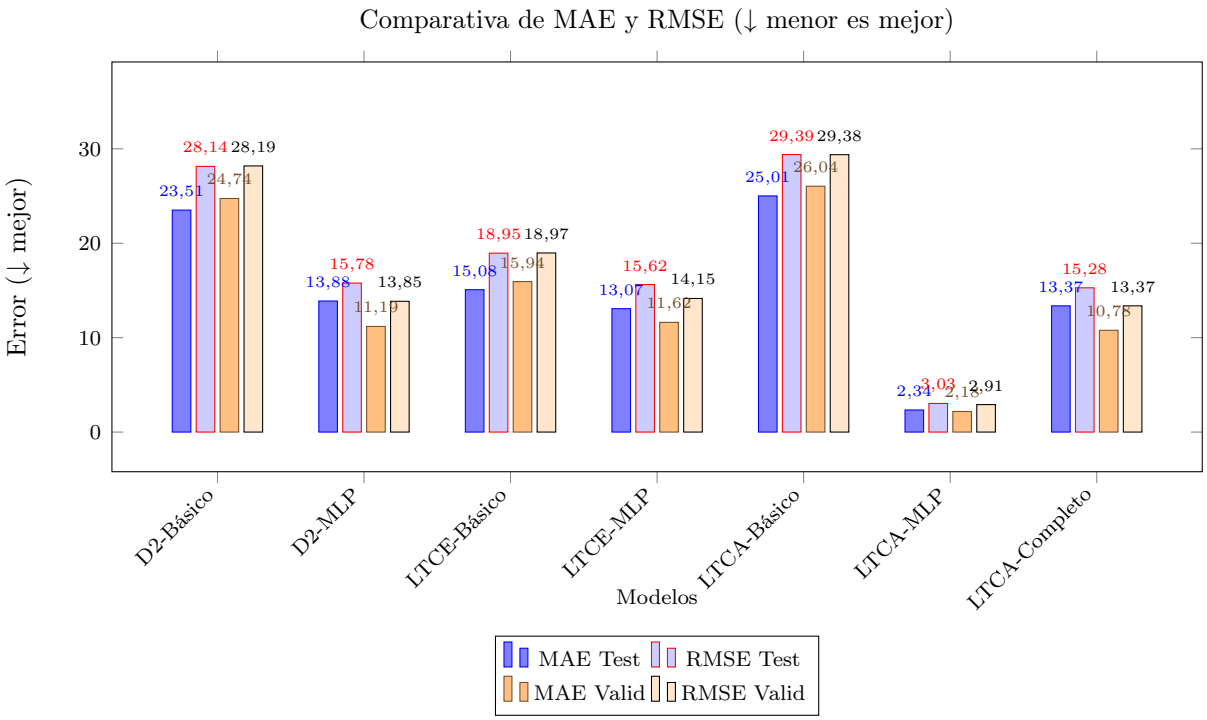

Figura 5.6: Comparativa de errores MAE y RMSE para cada modelo en los conjuntos de test y validación en el dataset Vinagres Variados. Cuanto menor sea el valor, mejor rendimiento.

5.2. Evaluación experimento sobre generalización

En este apartado se muestran los resultados de las pruebas utilizando parte de un *dataset* para entrenar y evaluados con parte de otro *dataset* diferente. Se utilizan los *dataset* de Vinos Blancos 3.1.1, de Salsas Basílico 3.1.2 y de Vinagres Variados 3.1.3. Este experimento tiene el objetivo de ver la capacidad de generalización de los modelos con el conjunto de datos de un entorno real, para ello se prueba cada combinación diferente posible.

Como lo interesante son los modelos ya entrenados con parte de los *datasets* y no tendría sentido evaluar los modelos básicos, se van a excluir de la tabla de resultados. También se excluye de las Figuras el experimento 2: *Fine-tuning* del *CountRegressor* del modelo *LCTEverything* debido a que los resultados son de una magnitud mayor que el resto de los obtenidos.

Como se puede observar en las Figuras 5.7 5.8 5.9, no hay ninguna combinacion de *baseline* y conjunto de datos que sea capaz de generalizar de manera correcta.

Se observa que todos los resultados son similares. Puede destacar los resultados obtenidos con el *fine-tuning* de la MLP externa tanto en *Detectron2* como en *LTCAnything* cuando se entrena con el *dataset* de Salsas Basílico y se evalua el *dataset* de Vinos Blancos, ver Figura 5.7. Aun así, los errores aún se mantienen por encima de los mejores obtenidos en las evaluaciones realizadas directamente sobre los *datasets* de entrenamiento.

Se puede hacer una hipótesis de que entrenar con un *dataset* que tiene complejidad estructural de sus unidades, en este caso el *dataset* de Salsas Basílico proporciona una base más robusta para la generalización. Esto obliga al modelo a aprender representaciones más complicadas que luego pueden aplicarse a otros productos visualmente más simples como los Vinos Blancos. También se puede pensar que evaluar sobre productos con estructura simple puede llevar a mejores resultados.

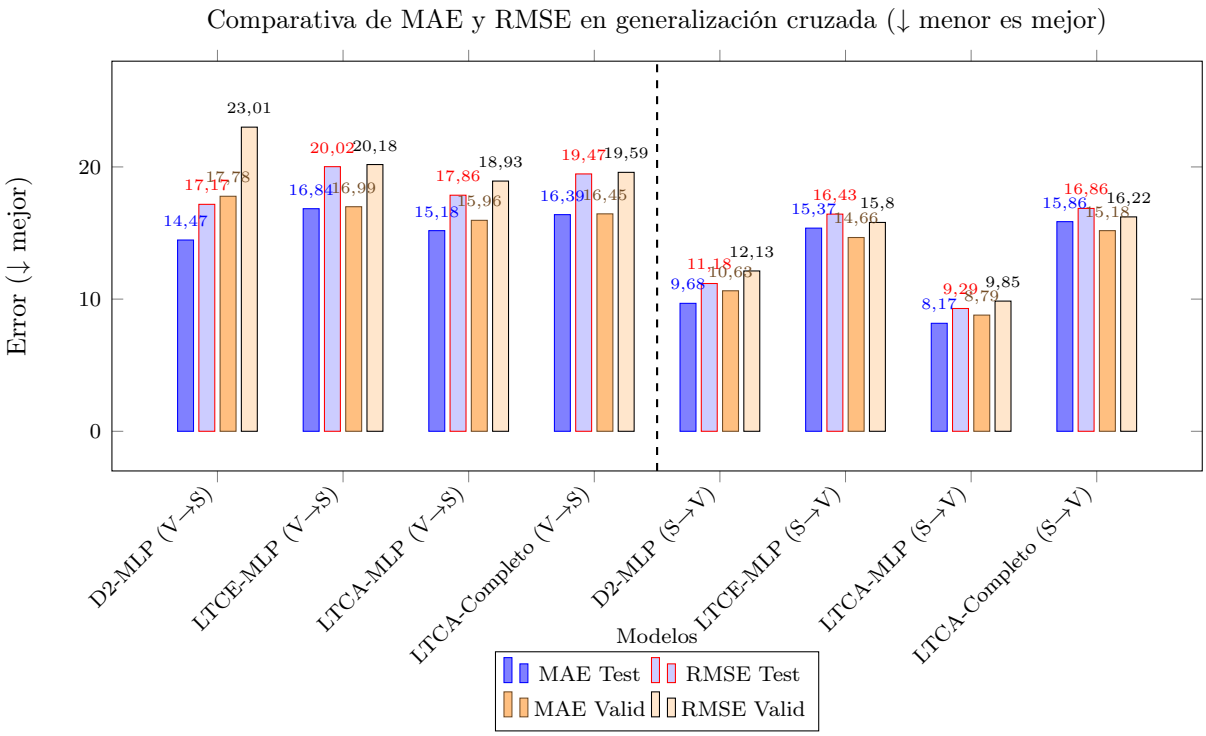

Figura 5.7: Comparativa de errores MAE y RMSE para cada modelo en pruebas de generalización entre productos. La línea separa los experimentos entrenados con vinos y evaluados con salsas (izquierda), y viceversa (derecha).

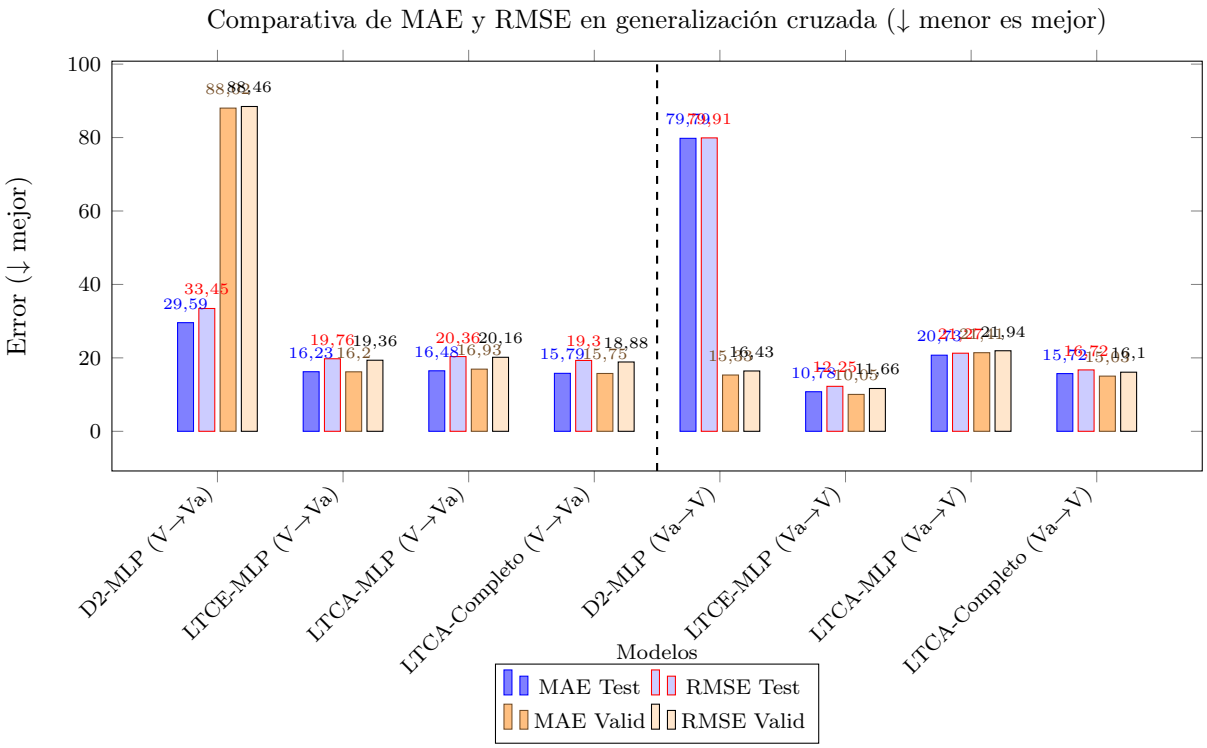

Figura 5.8: Comparativa de errores MAE y RMSE para cada modelo en pruebas de generalización entre productos. La línea separa los experimentos entrenados con vinos y evaluados con vinagres variados (izquierda), y viceversa (derecha).

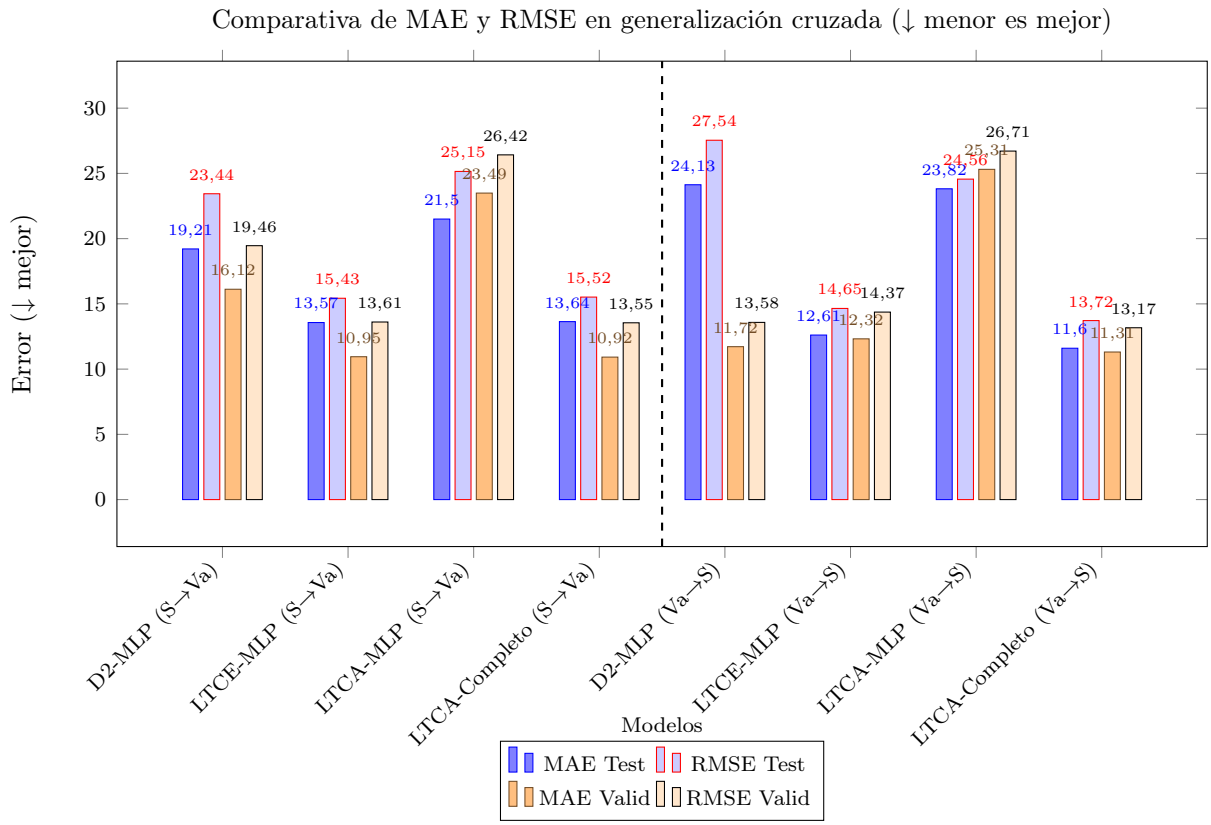

Figura 5.9: Comparativa de errores MAE y RMSE para cada modelo en pruebas de generalización entre productos. La línea separa los experimentos entrenados con salsas y evaluados con vinagres variados (izquierda), y viceversa (derecha).

5.3. Evaluación experimento datasets conjuntos

En este apartado se muestran los resultados de las pruebas utilizando los *datasets* de Vinos, Salsas y Vinagres en conjunto para entrenar y evaluar con el objetivo de evaluar los *baselines* cuando existen varios productos. Se utiliza el *dataset* Conjunto 3.1.4. Se excluye de la tabla el experimento 2: *Fine-tuning* del *CountRegressor* del modelo *LCTEverything* debido a que los resultados son de una magnitud mayor que el resto de los obtenidos.

No se añaden comentarios adicionales sobre los modelos básicos, ya que el comportamiento que presentan es el mismo que lo analizado en los apartados dedicados a la evaluación sobre un *dataset* (Seccion 5.1).

Como se puede ver en la Figura 5.10, el *fine-tuning* de la MLP externa partiendo de *Detectron2* empeora los resultados. Aunque no se puede afirmar el motivo, la pérdida de precisión puede deberse a una mayor variabilidad al combinar los *datasets*, o también a una falta de capacidad de adaptación por parte de *Detectron2* frente a dominios más

diversos.

El mejor resultado lo vuelve a dar el *fine-tuning* de una capa MLP externa desde el modelo *LTCAnything*. Este *baseline* vuelve a predecir el número de unidades con gran precisión, manteniendo un error medio cercano a 1. Esto da confianza a su robustez y adaptabilidad como una solución eficaz para tareas de conteo con mínima supervisión.

Se observa cómo los resultados son parecidos a los anteriores. Esto es positivo, ya que indica que, aunque aumente el número de productos, algunos con tres imágenes por unidad o con formas más complejas, el modelo sigue obteniendo resultados similares.

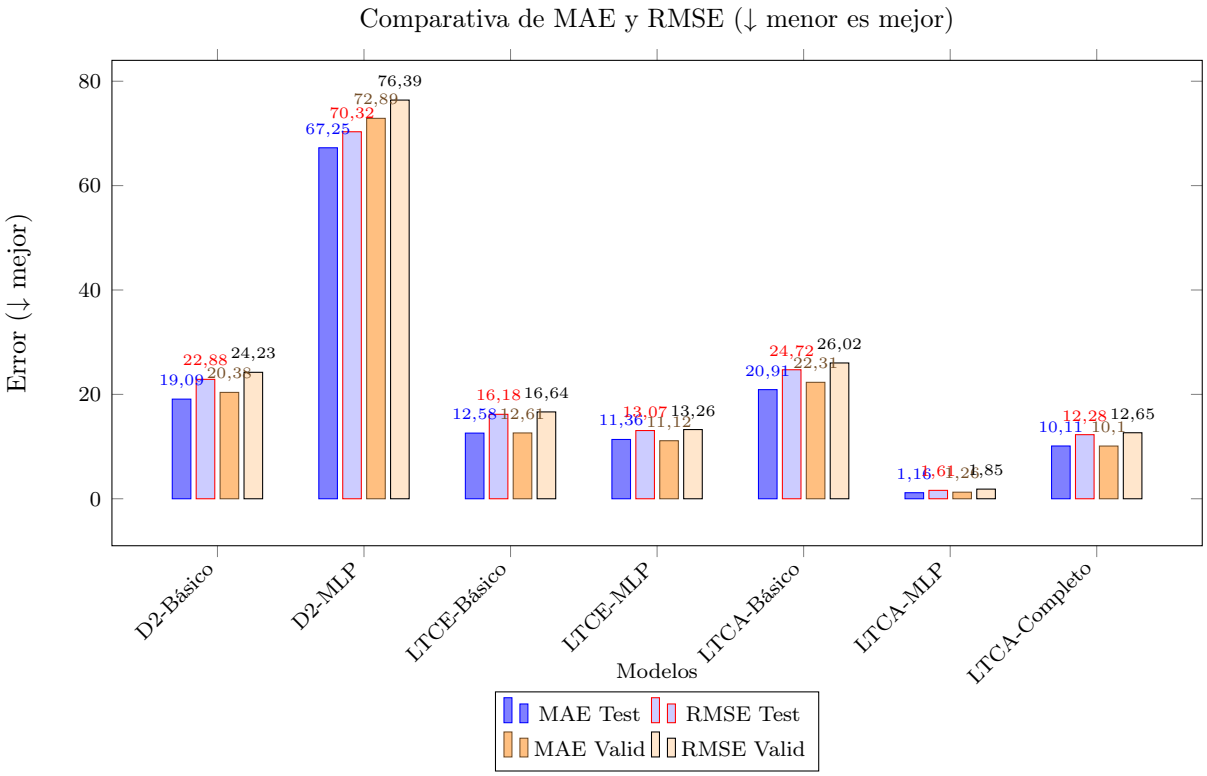

Figura 5.10: Comparativa de errores MAE y RMSE para cada modelo en los conjuntos de test y validación en el dataset Conjunto. Cuanto menor sea el valor, mejor rendimiento.

5.4. Conclusión experimentos

En este apartado se comenta si algún modelo puede ser útil para el objetivo principal del proyecto, lo cual ha sido el caso; a continuación, se hablará de dicho modelo.

El modelo que ha demostrado ser el más eficaz en todos los experimentos realizados ha sido una combinación de *Learning To Count Anything* (LTCAnything) y una capa MLP externa. Este modelo ha mostrado una diferencia notable en rendimiento respecto a los

demás enfoques, alcanzando resultados con un error cercano a 1.

Una de las principales razones por las que *LTCAnything+MLP* ha superado a los otros modelos es su capacidad para adaptarse a diferentes configuraciones de productos, aún cuando los conjuntos de datos de entrenamiento contienen pocas imágenes. El modelo *LTCAnything*, previamente entrenado utilizando el método DINO [7] (un enfoque de aprendizaje auto-supervisado para preentrenar redes neuronales sin necesidad de etiquetas), ya tiene una buena representación de las características visuales, lo que facilita que la capa MLP externa tenga mayor precisión. El *baseline* no se involucra en la predicción de mapas de densidad, lo que permite al modelo centrarse en la predicción del conteo, sin complicaciones adicionales.

Aunque *LTCAnything+MLP* es efectivo cuando se enfrenta a imágenes limitadas por producto, su capacidad para generalizar a nuevos productos ha mostrado ser más limitada en los experimentos de generalización. El modelo no ha logrado mantener el mismo nivel de rendimiento al ser evaluado en productos diferentes a los que fue entrenado. Esto hace que, cuando haya que expandir a nuevos productos, se necesite realizar un *fine-tuning* para que el modelo aprenda sobre los nuevos productos y pueda evaluarlos correctamente.

Capítulo 6

Conclusión y trabajo futuro

6.1. Conclusiones

Uno de los procesos clave en la operativa de almacén es el *picking*: la tarea de seleccionar y agrupar productos desde el almacén para conformar pedidos. Muchas empresas aún confían en el recuento manual cuando se trata de palés abiertos. Por ello, aparece la oportunidad de aplicar soluciones tecnológicas basadas en visión por computador e inteligencia artificial. Esto me ha motivado a realizar este Trabajo de Fin de Grado, que tiene como finalidad desarrollar un sistema de conteo automático basado en redes neuronales, que permita contar automáticamente el número de unidades en palés abiertos. Se busca reducir el tiempo y los errores asociados al conteo manual.

Este proyecto tiene una serie de objetivos que se han ido logrando a medida que ha transcurrido el proyecto. Primero, había que estudiar el estado del arte de las redes neuronales en el conteo de unidades; para ello, se han investigado y analizado diferentes modelos. Gracias a esto, durante el TFG se ha utilizado el sistema de visión por computador *Detectron2* [1] y los modelos de conteo visual *Learning To Count Everything* [2] y *Learning To Count Anything* [3].

En segundo lugar, otro objetivo era evaluar el rendimiento de estos modelos con datos de un entorno real. Para ello, se han capturado varios conjuntos de imágenes de un almacén logístico, resultando en la recolección de tres *datasets*: Vinos Blancos, Salsas Basílico y Vinagres Variados.

En tercer lugar, con la finalidad de alcanzar el objetivo principal del proyecto, se han propuesto mejoras en las arquitecturas de los modelos para que el conteo sea más preciso. Se han aplicado distintas estrategias como *fine-tuning* completo, entrenamiento de capas MLP externas y adaptación directa de modelos preentrenados.

Finalmente, se han realizado una serie de experimentos. Estos experimentos tienen

la intención de, además de evaluar la capacidad de conteo de los modelos, analizar su capacidad de generalización, así como ver cómo actúan cuando en el conjunto de datos hay múltiples productos. Como resultado de esta evaluación, se demuestra que *Learning To Count Anything*, sumado a una capa MLP externa, obtiene los mejores resultados. Además, se demuestra que ningún modelo tiene buena capacidad de generalización.

Para llegar al objetivo, se ha seguido una metodología que permite que los resultados obtenidos puedan compararse y replicarse en condiciones similares. Se han utilizado diversas herramientas descritas, experimentos con diferentes finalidades y métricas de evaluación, como MAE y RMSE, con este fin. Gracias a la implementación de un sistema bien estructurado y con una base de pruebas, se ha podido medir el rendimiento de los modelos y compararlos, lo que respalda la viabilidad y eficiencia del enfoque propuesto.

6.2. Trabajo futuro

El modelo *LTCAnything+MLP* es el más efectivo en los experimentos realizados hasta el momento, obteniendo los mejores resultados en términos de error de conteo. Uno de los principales desafíos que tiene es la capacidad de generalización del modelo cuando se enfrenta a nuevos productos. El siguiente paso es seguir añadiendo imágenes de diferentes productos al modelo, evaluando cómo se adapta a medida que se incorporan más datos de productos variados.

Para la captura de imágenes de todos los productos habría dos opciones. La primera de ellas es seguir el proceso que se ha seguido en este trabajo de realizar, de seguido, una cantidad suficiente de imágenes del palé, lo que puede causar algún inconveniente si los operarios necesitan utilizar ese palé. O la segunda, que sería realizar un proceso en el cual, cada cierto tiempo, se recorren todos los palés y se captura una fotografía. El inconveniente que tiene esta opción es que cabe la posibilidad de que a un producto nunca se le consiga hacer fotos en un rango determinado de unidades.

Una vez que el modelo haya sido entrenado con la mayoría de los productos del conjunto, se deberá evaluar cómo se comporta en términos de generalización. Esto con la intención de que, si en un futuro llegan nuevos productos, no se tenga que realizar un *fine-tuning* por producto.

Si se observa que el modelo comienza a estancarse o a perder rendimiento a medida que se agregan más productos, se podría considerar la opción de entrenar una red neuronal más compleja, como un *MLP* más profundo o con mayor capacidad, algo que ya se había planteado desde el principio en caso de que los resultados iniciales no fueran buenos.

El paso final será realizar este conteo como un servicio al cual la PDA (dispositivo móvil utilizado por los operarios) pueda acceder, de modo que puedan utilizarlo directamente en el entorno de trabajo. La PDA permitirá a los operarios realizar la evaluación de las

imágenes de forma rápida y eficiente. El flujo de trabajo consistiría en capturar la imagen del palé, llamar al servicio, procesarla para que realice el conteo, y de forma automática actualizar en la base de datos el número estimado de unidades.

Apéndice A

Anexos

A.1. OpenShift y S3 Bucket AWS

En este anexo se explica el porqué y el procedimiento de realizar el entrenamiento en un contenedor de `OpenShift`. En la empresa no existe ninguna estructura destinada al conteo de unidades ni que emplee un sistema de visión por computador. Además, no se cuentan con herramientas comunes en proyectos de este tipo, como un ordenador con GPU o servidores con tarjetas gráficas, lo que implica que los entrenamientos se realizan desde la CPU, lo que incrementa notablemente su duración.

En el caso de *Detectron2*, en las primeras pruebas, el entrenamiento ha tenido una duración superior a un día, lo que supone un inconveniente. Para solventarlo, la empresa me ha facilitado un contenedor en `OpenShift` para realizar el entrenamiento. Debido a la duración prolongada del proceso, como medida de precaución, se ha optado por subir a la plataforma `Bucket S3` de AWS los distintos *checkpoints* y el peso final del modelo. De esta manera, si por alguna razón el entrenamiento se interrumpe, ya sea por la caída del *pod* o cualquier otro motivo, se dispone de una copia de seguridad en AWS.

El proceso es el siguiente: a través de un `Dockerfile`, creo una imagen de `Miniconda` e instalo todas las dependencias necesarias para *Detectron2*. Además, incluyo el código del entrenamiento que quiero ejecutar, especificando que, cada ciertas iteraciones, se suba una copia del modelo al `bucket S3 de AWS` mediante un `hook`. Luego, subo este `Dockerfile` a `Amazon Elastic Container Registry (ECR)` de `AWS`. En el *deployment* de `OpenShift`, configuro que utilice la imagen previamente subida al `ECR`. Una vez lanzado el `pod`, el modelo comienza el entrenamiento, y cuando este finalice, los pesos del modelo serán almacenados en el `bucket S3`.

A.2. Imagenes de conteo de modelos básicos

En este anexo se encuentran más imagenes de como actuan los modelos básicos. Se comentan las imagenes de los tres *dataset* de forma específica.

Se comentan las imagenes de Vinos Blancos: se observa que los modelos básicos no son capaces de contabilizar las cajas que tienen oclusión total o parcial, ver Figuras A.1 A.2 A.3. Se puede ver la diferencia entre los modelos: *Detectron2* casi no es capaz de reconocer más de 3 o 4 cajas. *LTCEverything* afina mucho más y se puede ver en los mapas de densidad (ver Figura A.3) cómo es capaz de reconocer casi todas las cajas visibles. *LTCAnything* también es capaz de reconocer bastante bien las cajas visibles y, como se puede observar, aunque no tiene tanta actividad en el mapa de calor, marca con precisión los objetos que tiene claro que son cajas, ver Figura A.2.

A continuación, las imagenes de Salsas Basílico. Se observa que, al igual que en el *dataset* anterior, los modelos básicos no son capaces de contabilizar las unidades que tienen oclusión parcial o total, ver Figuras A.4 A.5 A.6. *Detectron2* sigue sin ser capaz de contabilizar más de 3-4 unidades (ver Figura A.4), lo que, sumado a que hay el doble de unidades máximas, tiene sentido que el error sea mayor que el doble. En los mapas de densidad de *LTCEverything* y *LTCAnything*, ver Figuras A.6 A.5, se ve que son capaces de entender que lo que tienen que contar es la unidad que contiene los frascos cilíndricos y no los frascos en sí. Aun así, el error es muy alto, ya que solo cuentan las unidades visibles en su totalidad.

Finalmente las imagenes de Vinagres Variados. Se observan resultados similares a los anteriores siguiendo los modelos básicos sin ser capaceces de contabilizar con oclusión parcial o total, ver Figuras A.7 A.8 A.9. Lo interesante en estas imágenes es como tratan los modelos cuando hay dos productos diferentes. Se ve en la Figura A.9 como el modelo *LTCEveything* pierde precisión respecto a donde hay unidades, sus mapas de densidad tienen más puntos con menos confianza.

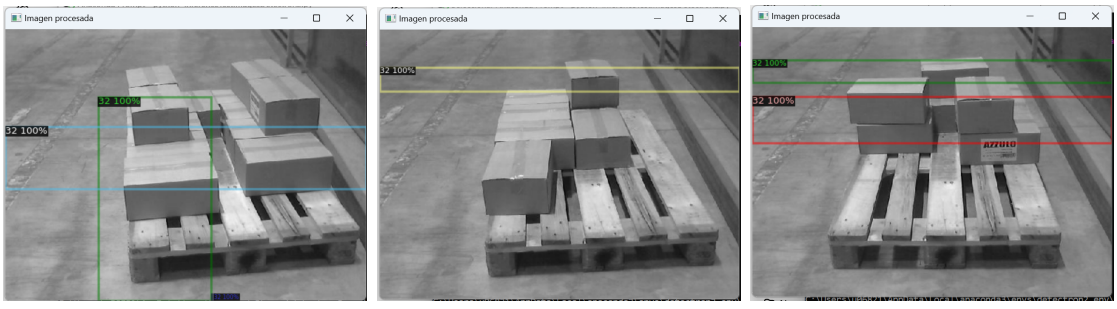

Figura A.1: Ejemplos de diferentes interacciones con modelos básicos de Detectron2 en el dataset Vinos Blancos.

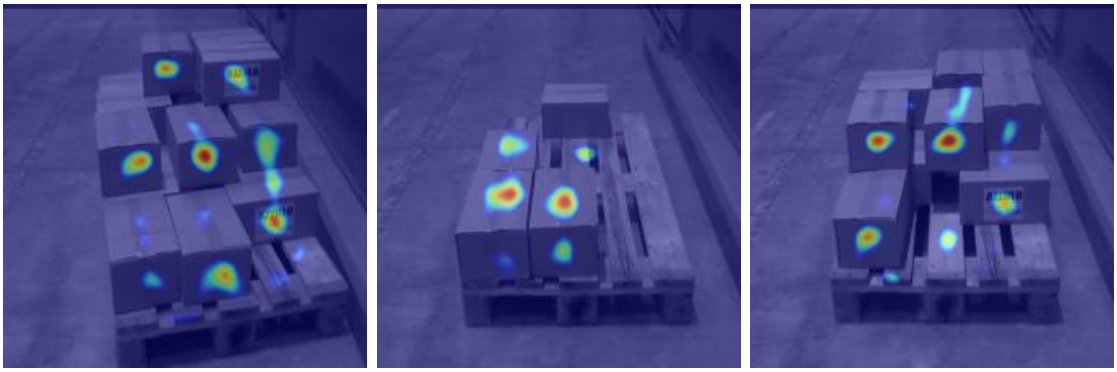

Figura A.2: Ejemplos de diferentes interacciones con modelos básicos de LTCAnything en el dataset Vinos Blancos.

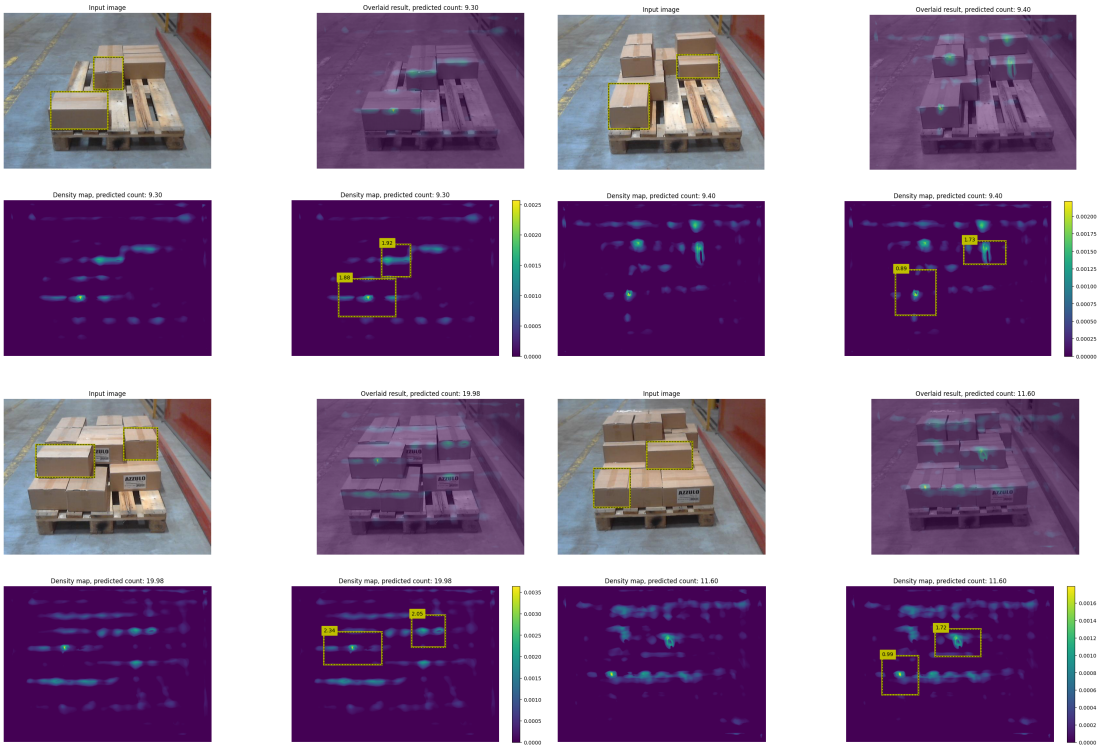

Figura A.3: Ejemplos de diferentes interacciones con modelos básicos de LTCEverything en el dataset Vinos Blancos.

Figura A.4: Ejemplos de diferentes interacciones con modelos básicos de Detectron2 en el dataset Salsas Basílico.

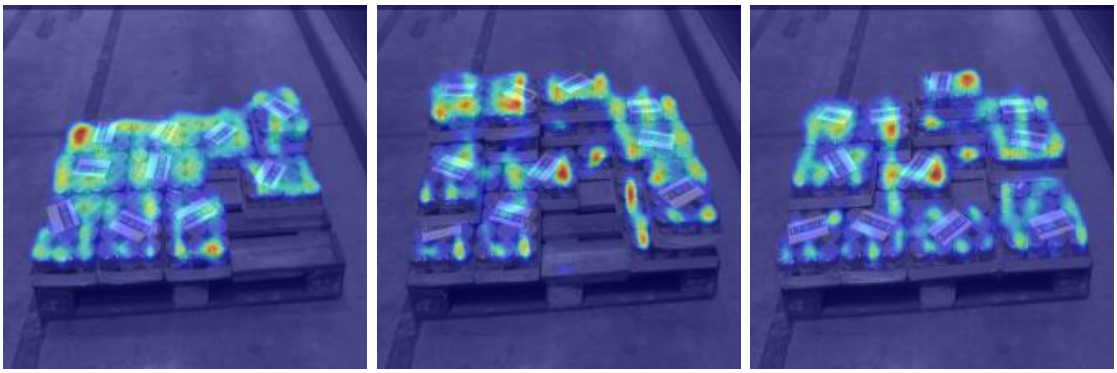

Figura A.5: Ejemplos de diferentes interacciones con modelos básicos de LTCAnything en el dataset Salsas Basílico.

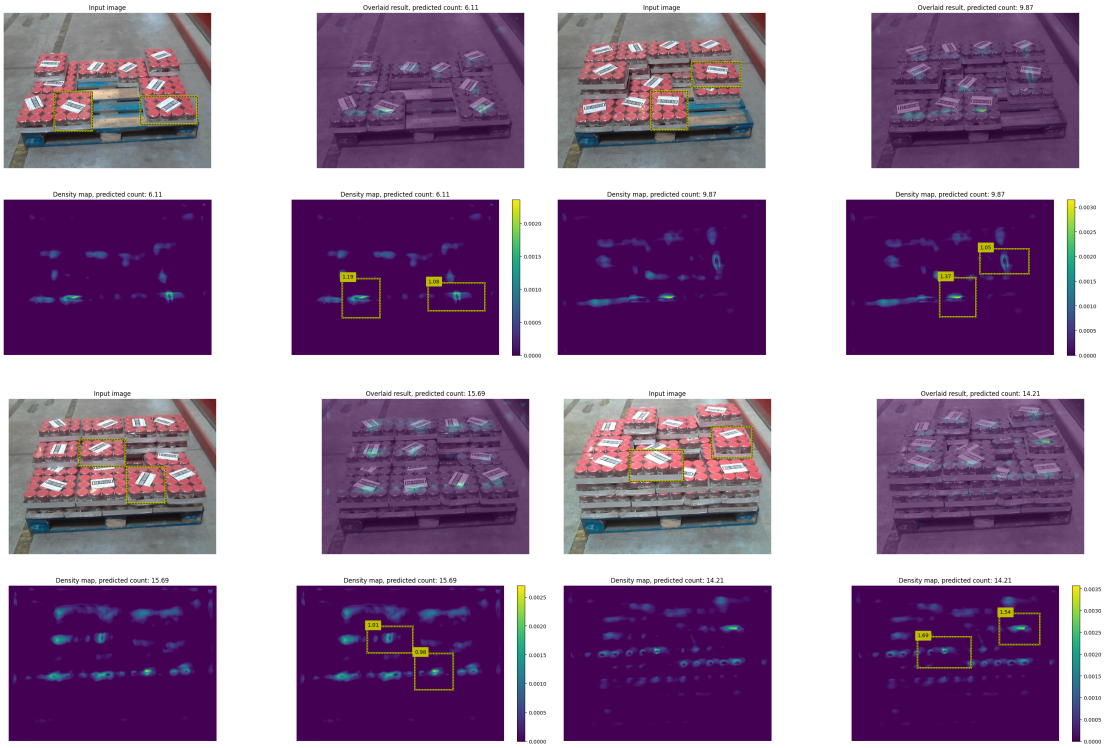

Figura A.6: Ejemplos de diferentes interacciones con modelos básicos de LTCEverything en el Dataset Salsas Basílico.

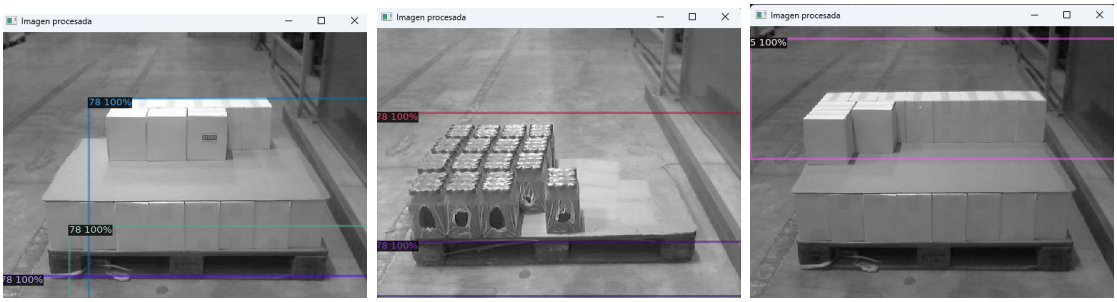

Figura A.7: Ejemplos de diferentes interacciones con modelos básicos de Detectron2 en el dataset Vinagres Variados.

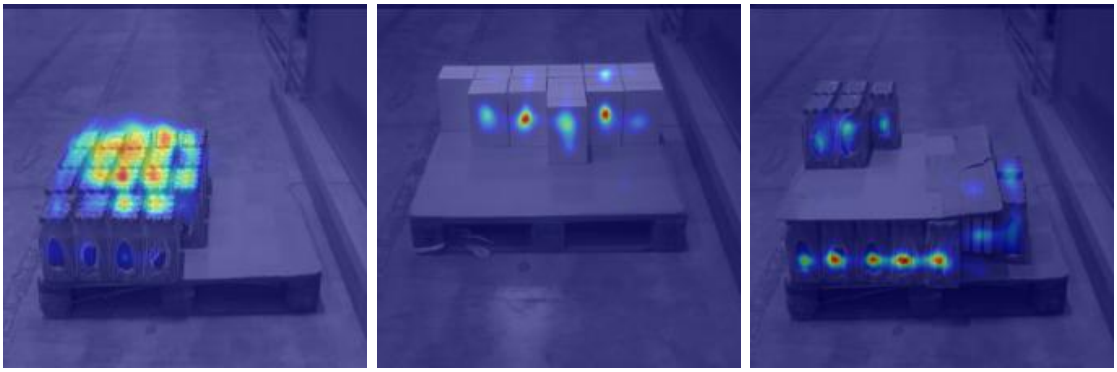

Figura A.8: Ejemplos de diferentes interacciones con modelos básicos de LTCAnything en el dataset Vinagres Variados.

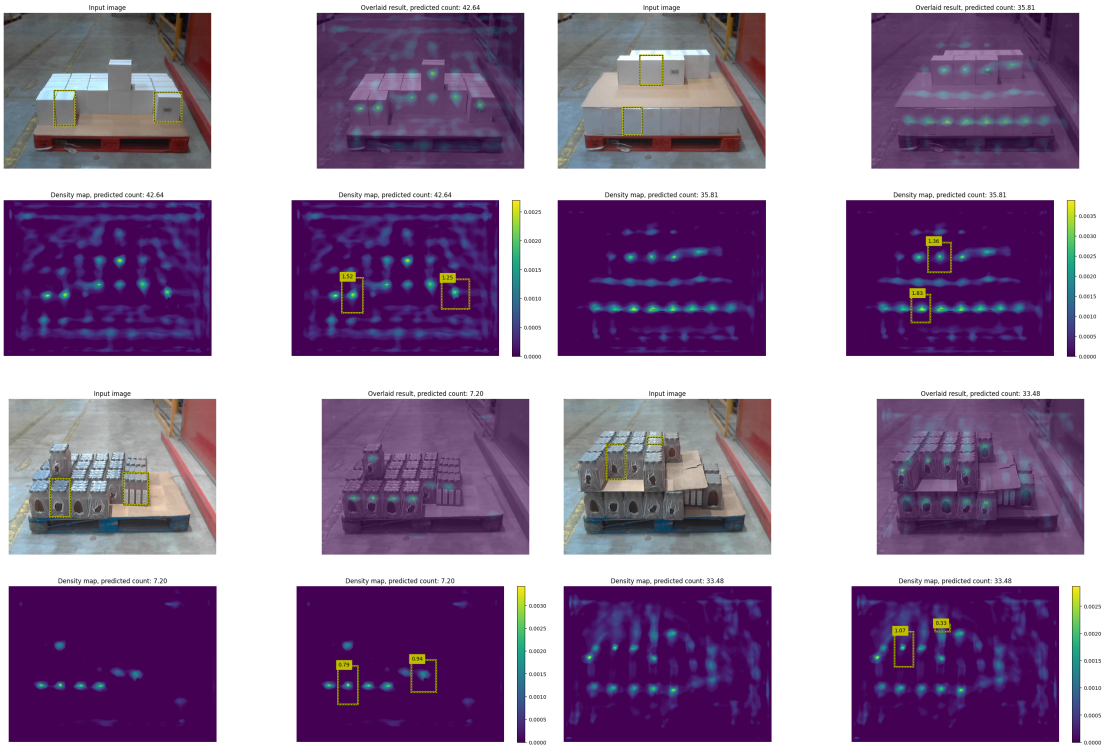

Figura A.9: Ejemplos de diferentes interacciones con modelos básicos de LTCEverything en el dataset Vinagres Variados.

Bibliografía

[1] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.

[2] V. Ranjan, U. Sharma, T. Nguyen, and M. Hoai, "Learning to count everything," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[3] M. Hobley and V. Prisacariu, "Learning to count anything: Reference-less class-agnostic counting with weak supervision," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[4] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "Mediapipe: A framework for building perception pipelines," *ArXiv*, vol. abs/1906.08172, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195069430

[5] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth anything v2," *arXiv:2406.09414*, 2024.

[6] K. Ehsani, R. Mottaghi, and A. Farhadi, "Segan: Segmenting and generating the invisible," in *CVPR*, 2018.

[7] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," 2021. [Online]. Available: https://arxiv.org/abs/2104.14294