UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DETECCIÓN Y ESTIMACIÓN DE PORCIÓN PARA INGREDIENTES GRANULARES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

Gonzalo Nicolás Hernández Álvarez

PROFESOR GUÍA:
Andrés Thomas Brigneti

MIEMBROS DE LA COMISIÓN
Iván Sipirán Mendoza
Juan Barrios Núñez
Carlos Navarro Clavería

Este trabajo ha sido parcialmente financiado por Kwali SPA

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: GONZALO NICOLÁS HERNÁNDEZ ALVAREZ
FECHA: 2023
PROF. GUÍA: ANDRÉS THOMAS BRIGNETI

DETECCIÓN Y ESTIMACIÓN DE PORCIÓN PARA INGREDIENTES GRANULARES

Los modelos basados en *IAs* han demostrado ser una herramienta poderosa en el ámbito del control de calidad en diversas industrias. Aun así, existen entornos más complejos a los que se ven enfrentados estos modelos, como lo puede ser el control de calidad sobre alimentos, en específico, a la forma en que se presentan los alimentos a estos modelos. Esto último puede ser un problema debido a varios factores como, por ejemplo, la variabilidad en la apariencia, cambios en la iluminación, superposición de ingredientes, oclusión, o texturas y colores similares.

En el presente trabajo, se busca implementar una metodología que permita detectar y estimar la porción de ingredientes granulares asociados a imágenes obtenidas desde las cocinas de *PizzaHut*, mediante cámaras instaladas por la *start up* chilena *Kwali*, *start up* bajo la cual se realiza este trabajo.

Se destaca la variedad de modelos utilizados a lo largo de esta experiencia, comenzando por modelos de detección de objetos, tanto *One-Stage* (*Yolov6*), como *Two-Stage* (*Faster R-CNN*), pasando por un modelo de segmentación *prompteable* denominado *SAM*, y finalizando con modelos de estimación de profundidad monoculares basados en *Transformers*, nombrados *MIM* y *BinsFormers*.

Luego de entrenar y evaluar, se tiene que el par (*Yolov6*, *BinsFormers*) obtiene el mejor desempeño en los *datasets* respectivos, sin embargo, se decide utilizar a *Faster R-CNN* por sobre *Yolov6*, debido en gran medida a que *Faster R-CNN* se encuentra arraigado en los *pipelines* de la empresa, sumado a que no existe una gran diferencia entre las salidas de estos 2 modelos. Como principal resultado se tiene que todos modelos poseen una alta capacidad de detectar ingredientes con una forma definida, tamaño promedio, y con un color que resalte en las imágenes, mientras que aquellos que no presentan estas cualidades, poseen bajo índice de detección y estimación.

Finalmente, se crea un algoritmo que detecta y estima la porción de diversos ingredientes presentes dentro de una imagen, obteniendo resultados prometedores, a pesar de lo simple de la metodología utilizada. Dicho algoritmo posee un amplio margen de mejora, si consideramos un aumento en los datos utilizados, mapas de profundidad más detallados, ampliación de elementos a detectar, y futuros modelos de detección *SOTA*.

Agradecimientos

Quiero partir agradeciendo a mi familia, pilar fundamental en mi crecimiento como persona ya que sin duda no sería el mismo sin cada uno de ustedes. En especial a mi mamá y a mi papá, por estar siempre preocupados de mi educación, a pesar de lo adversa de las condiciones y del sacrificio que implicara. Gracias por su amor y apoyo incondicional durante toda mi vida. Sin ustedes no estaría acá.

Agradezco infinitamente a mi novia, por complementarme, por estar a mi lado estos años de manera incondicional, siendo su amor, calidez y apoyo anímico parte fundamental de mis días.

Agradecer a mis amigos del liceo, que ya forman parte de mi familia, y por los que extraño los años de media. A los amigos y amigas de la U con los cuales compartí las primeras experiencias universitarias e hicieron más agradable los primeros años, con los que comparto hasta el día de hoy y espero seguir haciéndolo. A mi amigo y compañero de Eléctrica, con el cual sobrevivimos todos estos años a base de papas, gracias por hacer más amena esta etapa, no podría haber deseado un mejor amigo.

Agradezco también a mi gatita, quien me acompañó día y noche en mi regazo durmiendo por los dos, y alegrando mis días con su dualidad tierna y con cara de pocos amigos.

Por último, quiero agradecer a mi profesor guía, por bajarme el estrés con la memoria, tener paciencia con mis avances y guiarme a lo largo de este período.

Tabla de Contenido

Índice de ilustraciones

Índice de tablas

Capítulo 1

Introducción

El concepto de *comida* va más allá de la satisfacción de nuestras necesidades nutricionales o de su importancia en la cultura, ya que en la actualidad la tecnología está transformando radicalmente la industria alimentaria tal y como la conocemos. A medida que las empresas buscan optimizar sus procesos productivos y reducir costos, la tecnología se ha convertido en una parte esencial de la cadena de producción de alimentos. Si bien a menudo pensamos en la tecnología en el ámbito de la automatización de procesos o la gestión de inventario, su impacto en la manipulación directa de alimentos ha sido menos explorado hasta ahora.

Es en este contexto que surge *Kwali SPA*, una *start up* chilena que utiliza técnicas basadas en *deep learning* para ofrecer soluciones innovadoras a franquicias de alimentos. Su objetivo es apoyar a diversas franquicias de alimentos a reducir los costos operacionales por cada ítem, así como de medir la velocidad orden-servicio para cada cliente, entre muchas otras prestaciones.

Uno de los aspectos más interesantes de las soluciones ofrecidas por *Kwali SPA* es la detección de objetos granulares que, en este caso, se refiere a ingredientes granulares. Esto se logra mediante la captura de imágenes desde las cocinas de franquicias que se dedican a la producción y expendio de pizzas. Estas empresas a menudo enfrentan desafíos significativos en términos de costos operativos, donde un correcto porcionamiento de los ingredientes es uno de los puntos claves a tratar. La necesidad de optimizar y automatizar este proceso ha llevado a *Kwali* impulsar el desarrollo de un algoritmo de detección y estimación de porciones para estos ingredientes granulares.

Esta solución requerirá de un nuevo *dataset* RGB-D que brinde la profundidad de cada objeto granular, donde D es por *depth* o profundidad, a la par de modelos que estimen dicha profundidad, donde se utilizarán modelos *monoculares* que ocupan una única imagen, junto con métricas que nos ayuden a discernir entre ellos. La solución propuesta no solo tendrá el potencial de reducir los costos operativos al minimizar el desperdicio de ingredientes, sino que también puede mejorar la eficiencia en la preparación de alimentos. Además, brindará a las franquicias una mayor visibilidad sobre sus procesos de producción, lo que puede ayudarles a tomar decisiones más informadas sobre estos procesos, a la vez que influye en la satisfacción del cliente final, al estandarizar la presentación del producto y asegurando la calidad de este.

1.1. Objetivos

A continuación, se detallan los objetivos del presente trabajo:

1.1.1. Objetivo General

El objetivo general del presente trabajo, es desarrollar un algoritmo basado en *Deep Learning* que, dada una única imagen de una *pizza* como *input*, sea capaz de detectar y estimar la porción en *onzas* de los ingredientes presentes en esta imagen.

1.1.2. Objetivos Específicos

- Construir bases de datos con imágenes, tanto RGB-D para la estimación de profundidad, como RGB con los respectivos *bounding boxes* para la detección de objetos.
- Entrenar modelos de detección de objetos y de estimación de profundidad monocular.
- Evaluar los detectores utilizando métricas apropiadas a los contextos de detección de objetos y estimación de profundidad.
- Diseñar e implementar un módulo que reciba imágenes y utilice los modelos entrenados para detectar y estimar la porción de los objetos presentes en estas.

1.2. Metodología

El presente problema consta de cierta investigación, pero sobre todo de experimentación, debido a lo cual la metodología a seguir se centrara principalmente en probar distintos modelos para los dos principales tópicos. Por ello, definiremos una serie de pasos que definirán el *modus operandi* a seguir, pudiendo repetirse algunos en caso de que los resultados no sean los esperados.

- Investigar los modelos existentes, y elegir aquellos que se ajusten a las exigencias.
- Preparar los datos de entrenamiento, según el modelo seleccionado.
- Entrenar los modelos y analizar sus resultados.
- Utilizar los modelos entrenados para detectar y estimar la porción de objetos granulares.

1.3. Riesgos y restricciones de recursos

La mayor restricción durante el desarrollo del presente trabajo es el recurso del tiempo, dados los extensos periodos entrenamiento, lo que se hace más notorio al tener que reentrenar dichos modelos, hasta obtener uno que entregue resultados satisfactorios según las métricas utilizadas.

Ahora bien, no hay que dejar de lado en esta discusión a los recursos computacionales, los que por fortuna no presentan un problema mayúsculo, en el sentido que son variadas las alternativas para el entrenamiento de redes neuronales, desde una computadora personal, hasta servicios prestados por *Google* y *Amazon*, es decir, *Google Colaboratory* y *Amazon SageMaker* respectivamente.

1.4. Alcances

Los alcances del presente proyecto consideran, en un futuro cercano, la instalación en producción de este. El tiempo que tome dependerá directamente de la planificación estratégica que tome la empresa para este, el cual actualmente posee una prioridad media.

1.5. Estructura del informe

El informe se presenta bajo la siguiente estructura:

1. En el Capítulo 2 se ilustra el Marco Teórico, precisando las bases teóricas necesarias para la implementación del proyecto. Se presentan conceptos como Redes Neuronales, Convoluciones, *Transformers*, entre otros.
2. En el Capítulo 3, se detalla la Solución propuesta, describiendo los pasos a seguir para la implementación de esta.
3. En el Capítulo 4 se exponen los Resultados obtenidos.
4. En el Capítulo 5 se presenta el Análisis de los resultados obtenidos, argumentando los alcances de estos, a la vez de comentar las dificultades encontradas.
5. Finalmente, en el Capítulo 6 se detallan las Conclusiones del presente trabajo.

Capítulo 2

Marco Teórico

En este capítulo se establecerán las bases necesarias para entender cómo funcionan las redes neuronales profundas, en específico los detectores de objetos y segmentadores semánticos, para luego pasar a definir los optimizadores, y finalizando en las técnicas evaluativas de estas redes neuronales.

2.1. Redes Neuronales (NN)

Definición 2.1.1 *Un perceptrón se define como la función*

$$f(x) = \begin{cases} 1 & si \quad \omega \cdot x + b > 0 \\ 0 & si \qquad no \end{cases} \tag{2.1}$$

donde x, $\omega \in \Re^n$, $\cdot$ *es el producto punto, y* $b \in \Re$ *es el bias.*

Definición 2.1.2 *Una capa fully connected es un conjunto de perceptrones* $f(\cdot)$ *que reciben el mismo input* $x \in \Re^n$ *y computan en conjunto un vector de salida* $y \in \Re^m$ *tal que:*

$$y = f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \dots \\ f_m(x) \end{bmatrix} \tag{2.2}$$

2.2. Redes Neuronales Convolucionales (CNN)

Una *CNN* (*convolutional neural network*) es un tipo de red neuronal ampliamente utilizada en datos correlacionados espaciales y temporales, como lo son las imágenes y audio, entre otros. Está basada en funcionamiento de las neuronas ubicadas en la corteza visual de un cerebro biológico [1].

2.2.1. Convolución

El cálculo de la convolución para una imagen x y un *kernel* W es expresado como:

$$y_{i,j,k} = \sum_{m,n,p} x_{i-m,j-n,p} \cdot W_{m,n,p,k} \tag{2.3}$$

donde x, y son tensores de 3 dimensiones, mientras que W es un tensor de 4 dimensiones, i, j son las coordenadas de la imagen, los índices m, n recorren el *kernel*, mientras que el índice p y k representan la cantidad de canales de entrada y salida respectivamente.

Intuitivamente, la ecuación anterior puede interpretarse como, posicionar el *kernel* W sobre la imagen de entrada, multiplicar y sumar los elementos que se superponen, y luego mover el filtro las veces que sea necesario hasta cubrir todas las posiciones posibles. Dicho proceso es apreciable en la Figura 2.1.

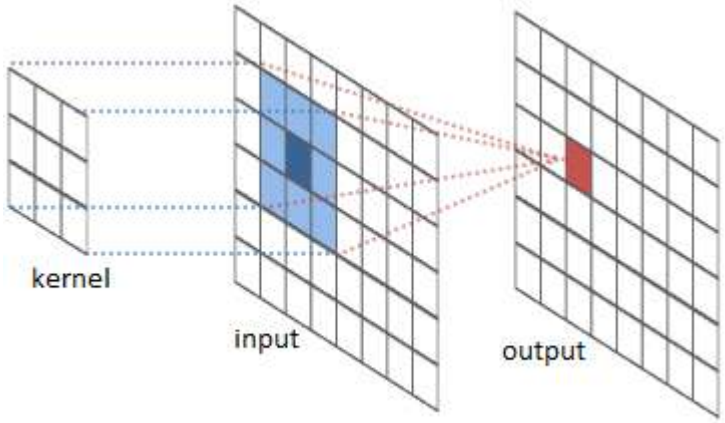


Figura 2.1: Convolución

2.2.2. Capas Convolucionales

Las capas convolucionales que conforman una CNN aplican diversas operaciones convolucionales a los datos de entrada, logrando así reducir el número de estos datos, a la vez que preserva la información espacial de estos, obteniendo así un mejor entendimiento de la imagen a nivel global. Un diagrama de una capa convolucional se presenta en [[2], Figura 2.2].

Actualmente, las capas convolucionales más populares están compuestas por filtros adaptativos, los que son ajustados durante el entrenamiento. Estas capas añaden un parámetro b llamado *bias*, con lo que la ecuación de convolución en una capa convolucional posee la siguiente forma:

$$y_{i,j,k} = \sum_{m,n,p} x_{i-m,j-n,p} \cdot W_{m,n,p,k} + b_k \tag{2.4}$$

donde b es un tensor de 1 dimensión.

Cada capa convolucional posee los siguientes hiperparámetros:

- **Tamaño del Filtro**: Corresponde a las dimensiones de los filtros a utilizar en la capa, generalmente cuadrados, definiendo así campo receptivo observado.
- **Profundidad de la Capa**: Corresponde al número de filtros a utilizar en la capa.
- ***Stride***: Corresponde al desplazamiento del filtro al momento de realizar la convolución.
- ***Padding***: Corresponde a la cantidad de ceros añadidos simétricamente a la imagen de entrada.

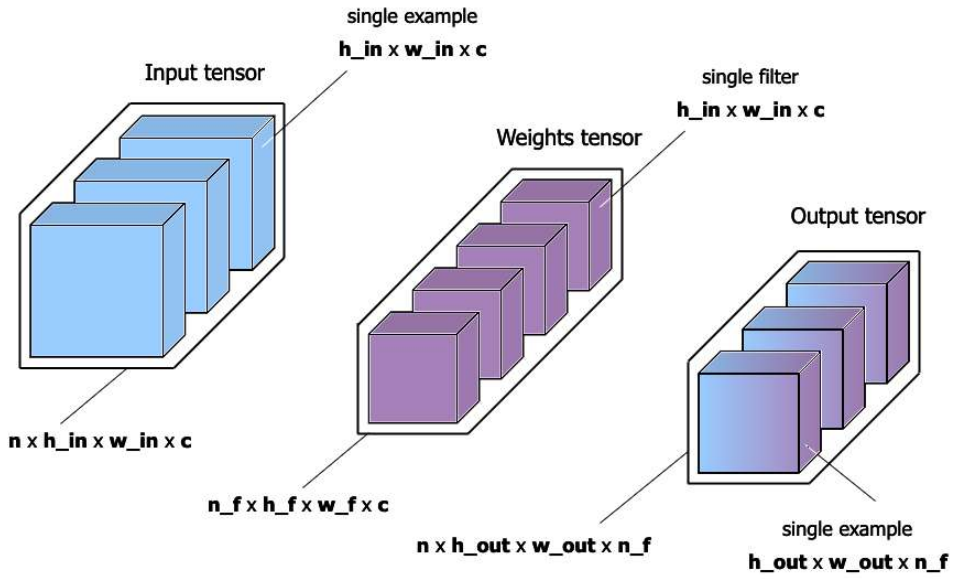


Figura 2.2: Capa Convolucional

2.2.3. Capas *Pooling*

Las capas *pooling* [3], son capas utilizadas para reducir el tamaño de la imagen de entrada, lo que conlleva una reducción en el número de parámetros, así como en el *overfit* y en el costo computacional.

Como se muestra en [[3], Figura 2.3], una capa *max pooling* reduce la dimensionalidad eligiendo el máximo por cada región de la imagen, mientras que una capa *average pooling* calcula el valor promedio por cada región de la imagen. En ambos caso, se utiliza un filtro de 2×2, y un stride de 2, definiendo así las regiones de colores.

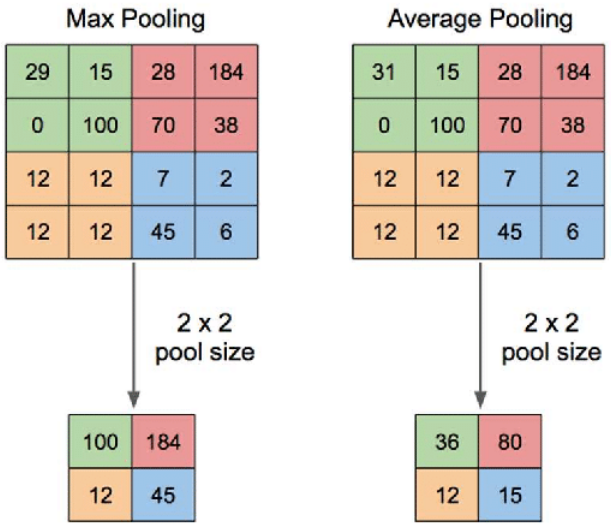


Figura 2.3: Capas *Max Pooling* y *Average Pooling*

2.3. Transformers

Los *transformers* son una familia de modelos de *deep learning*, introducidos por primera vez en el año 2017 en [4], donde se pudieron observar desempeños sobresalientes, especialmente en tareas de procesamiento del lenguaje natural, como la traducción automática, la clasificación de texto y la generación de texto.

Estos modelos emplean una arquitectura basada en *atención*, a fin de procesar secuencias de entrada de longitud variable. En lugar de usar redes neuronales recurrentes, como las utilizadas en muchos otros modelos de lenguaje, aquí se utilizan una serie de "transformaciones" para procesar las entradas y producir salidas. Estas transformaciones son implementadas mediante capas de atención y capas de proyección, y pueden ser entrenadas de manera efectiva usando las ya conocidas técnicas de optimización basadas en gradiente. La arquitectura principal es visible en [[4], Figura 2.4].

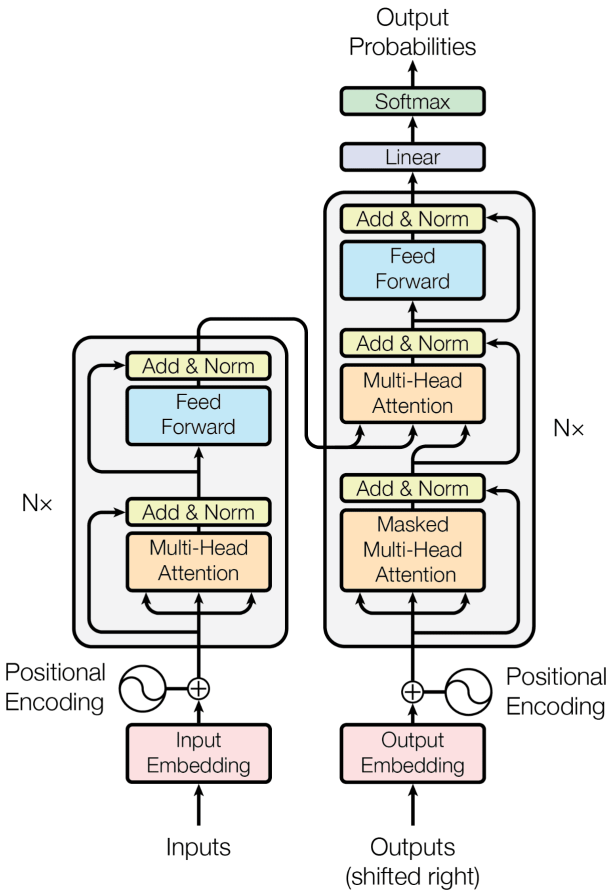


Figura 2.4: Arquitectura Transformer

2.3.1. Self-attention

También conocidos como mecanismos de atención, es una técnica usada en modelos de *deep learning* que permiten al modelo enfocarse en partes específicas del *input* cuando este es procesado. El *input* es primero transformado en un par *key-value*, que son luego usadas para calcular los pesos de atención para cada *key*, representando así la importancia de cada llave en el *input*. Estos pesos son luego multiplicados por el *value*, y sumados, obteniendo así una suma ponderada que es empleada como *input* de la siguiente capa. Este proceso permite al modelo calcular la importancia de diferentes partes del *input*, permitiendo así enfocarse en la información más relevante.

Los modelos que implementan estos mecanismos poseen variadas ventajas sobre modelos tradicionales. Un ejemplo de esto es la capacidad de capturar relaciones entre palabras que se encuentran a una larga distancia dentro del *input*, debido a que el significado de una palabra puede depender de otra que se encuentra delante o por detrás. Adicionalmente, permiten manejar *inputs* de secuencias variables, lo que es importante en tareas de NLP. En [[4], Figura 2.5] es visible la arquitectura de *self-attention*, así como del *multi-head attention*, que no es más que varios *self-attention* en paralelo, y que luego son concatenados y proyectados con una capa linear.

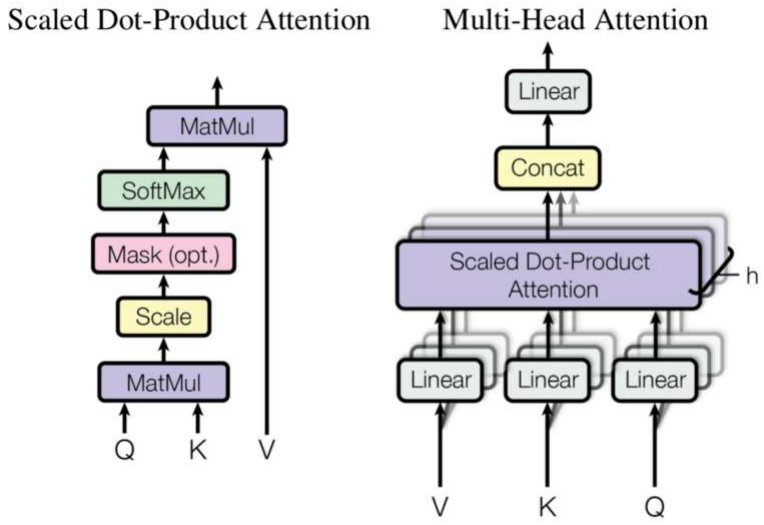


Figura 2.5: Arquitectura self-attention

2.3.2. Vision Transformers

Los *Vision Transformers* (*ViT*) son una variante de los *transformers*, los que permiten utilizarlos para el reconocimiento de imágenes, en lugar de procesar secuencias de lenguaje. Estos fueron introducidos en [5], de donde se desprende [[5], Figura 2.6]. En dichos modelos, en lugar de procesar imágenes directamente, los *ViT* dividen las imágenes en bloques pequeños, o *patches*, y procesan cada bloque como una secuencia de *tokens* de lenguaje. Esto permite a los *ViT* utilizar la arquitectura de atención para aprender a combinar información de diferentes partes de la imagen de manera efectiva. Los *ViT* han demostrado tener un rendimiento excepcional en tareas de reconocimiento de imágenes y han llegado a ser una pieza fundamental de muchas aplicaciones de visión por ordenador.

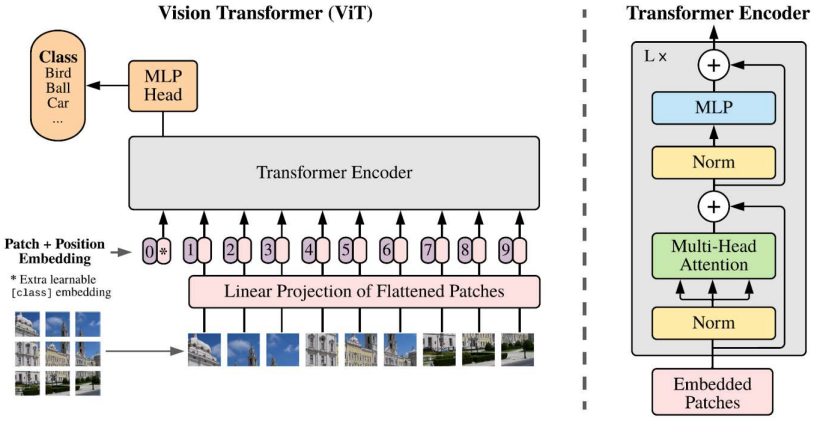


Figura 2.6: Arquitectura Vision Transformers

Aunque los *ViT* han demostrado tener un rendimiento excepcional en muchas tareas de

visión por ordenador, todavía existen algunos desafíos a los que se enfrentan. Por ejemplo, requieren una cantidad significativa de recursos computacionales para entrenar y utilizar, lo que puede hacer que sean menos prácticos para algunas aplicaciones. También pueden ser difíciles de interpretar y pueden tener dificultades para procesar ciertos tipos de información visual compleja. Finalmente, los *ViT* no pueden procesar imágenes de gran tamaño.

2.3.3. ViT vs CNN

Algunas de las ventajas de la *self-attention* en comparación con la convolución son:

1. Permite a las redes neuronales *mirar* a cualquier parte de la entrada a la vez, mientras que la convolución solo permite que se *vea* una ventana local de la entrada. Esto puede ser útil en tareas en las que es importante tener en cuenta la relación entre elementos distantes en la entrada.
2. Es más fácil de paralelizar y optimizar, ya que no requiere el uso de *kernels* ni de productos punto.
3. Es más fácil de entender y explicar, puesto que permite visualizar fácilmente qué partes de la entrada están siendo utilizadas por la red neuronal para realizar una tarea dada.

Sin embargo, también hay algunas desventajas en el uso de la *self-attention*:

1. Es más costosa en términos de cálculo, ya que requiere el cálculo del par *key-value* para cada elemento de la entrada y cada salida.
2. No es tan buena en el procesamiento de características locales como la convolución, ya que no utiliza un kernel para extraer características.
3. Puede ser menos robusta en tareas en las que la entrada está ruidosa o tiene errores, ya que no utiliza un kernel para aislar características relevantes.

2.4. Detectores de Objetos

Los detectores de objetos son arquitecturas especializadas en detectar y clasificar objetos, asignándole un *bounding box* al objeto detectado. Estas arquitecturas se pueden dividir en detectores *two-stage* y *one-stage*.

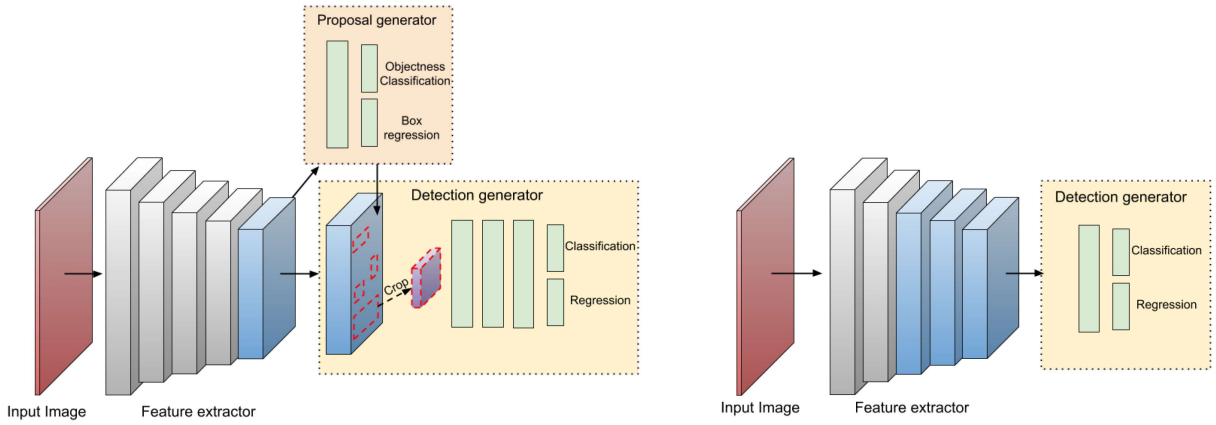
2.4.1. *Two-Stage*

Por un lado, los detectores de objetos del tipo *Two-Stage*, son métodos conocidos por ser relativamente lentos, pero con un alto nivel de desempeño, dado que estos poseen 2 etapas, las que son visibles en [[6], Figura 2.7 (a)], y corresponden a:

1. Generar regiones de interés, o *anchor boxes*, que puedan contener un objeto relevante.
2. Clasificar cada región de interés según las clases existentes, descartando esta en caso de que no se detecte ninguna clase.

2.4.2. *One-Stage*

Por otro lado, los detectores de objetos del tipo *One-Stage* son métodos conocidos por su rapidez, pero con un desempeño por debajo que su par de 2 etapas. Dicha rapidez se debe a que une las dos etapas en solo una, lo que es visible en [[6], Figura 2.7 (b)].



(**a**) Two-stage Faster R-CNN

(**b**) One-stage RetinaNet

Figura 2.7: Ejemplos de detectores *Two-Stage* y *One-Stage*

2.4.3. Faster R-CNN

Faster R-CNN (*Region-based Convolutional Neural Networks*) [7], es un modelo de detección de objetos del tipo *Two-Stage*, el cual tiene por modelos precursores R-CNN [8] y Fast R-CNN [9], y ocupa la tecnología de redes *CNN*. Este modelo se compone por 3 etapas, las cuales son:

1. Red CNN pre-entrenada sobre un *dataset*, generalmente *ImageNet* [10], la cual extraerá características sobre las imágenes dadas como *input*.
2. *Region Proposal Network* (*RPN*), la cual generara regiones de interés o *bounding boxes*, a partir de las características extraídas anteriormente, donde es probable que se encuentren los objetos.
3. Fast R-CNN, la cual clasifica el contenido de cada *bounding box*. En caso de encontrar un objeto, se ajusta el tamaño del *bounding box* en relación con el objeto, en caso contrario, se descarta el *bounding box*.

Cada una de las etapas descritas son visibles en [[11], Figura 2.8], donde son encerradas por regiones delimitadas por líneas punteadas. De izquierda a derecha vemos la red CNN pre-entrenada, seguida de la etapa *RPN* arriba y la etapa *Fast R-CNN* abajo.

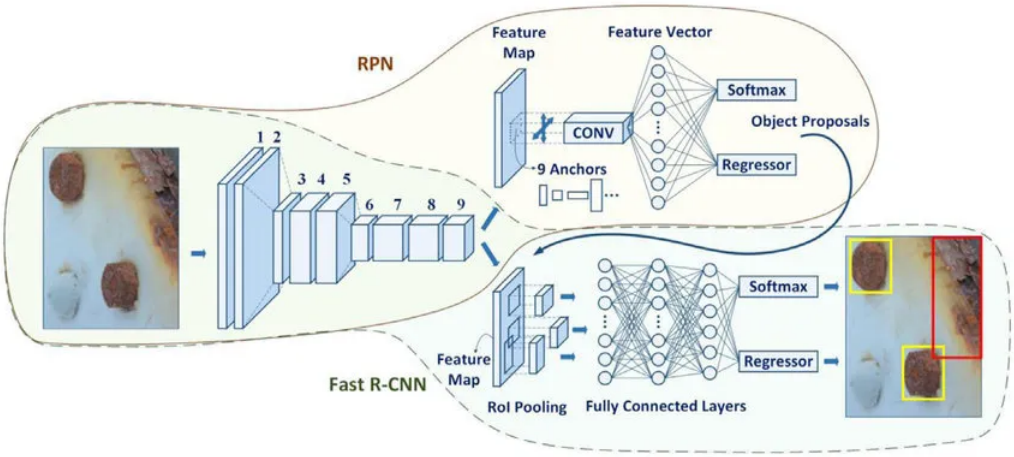


Figura 2.8: Arquitectura Faster R-CNN

2.4.4. Yolov6

Yolov6 [12] es parte de la popular familia de modelos de detección de objetos Yolo (*You Only Look Once*), que pertenecen al tipo de detectores *One-Stage*, y que es conocida por su excelente balance entre rapidez y *accuracy*. Este modelo cuenta con un nuevo diseño, comparando con sus predecesores, incorporando la tecnología de *transformers*, y considera los siguientes componentes:

2.4.4.1. Network Design

Se diseñó un *backbone* re-parametrizable denominado *EfficientRep*. Para los modelos pequeños, el componente principal del *backbone* es *RepBlock* para la etapa de entrenamiento, que consta de capas *RepVGG* [13], una capa de normalización, y una función de activación *ReLU*, mientras que al momento de la inferencia, cada *RepBlock* es reemplazado por *RepConv*, compuesto por capas convolucionales de 3×3 con una función de activación *ReLU*. Por otro lado, para los modelos medianos a grandes, se utilizaron *CSPStackRep Block* para construir el *backbone*. Este bloque está compuesto por 3 capas convolucionales de 1×1, sumado a sub-bloques de 2 *RepVGG blocks* o *RepConv*, para entrenamiento e inferencia respectivamente.

2.4.4.2. Label Assignment

Este componente es responsable de asignar las clases a cada región de interés, dentro de los cuales se encuentran métodos basados en *IoU*, o en el *gound-truth*.

2.4.4.3. Loss Functions

La detección de objetos contiene 2 sub-tareas; clasificación y localización, lo que nos conduce a 2 funciones de pérdida, llamados *classification loss* y *box regression loss*, los que serán descritos brevemente a continuación:

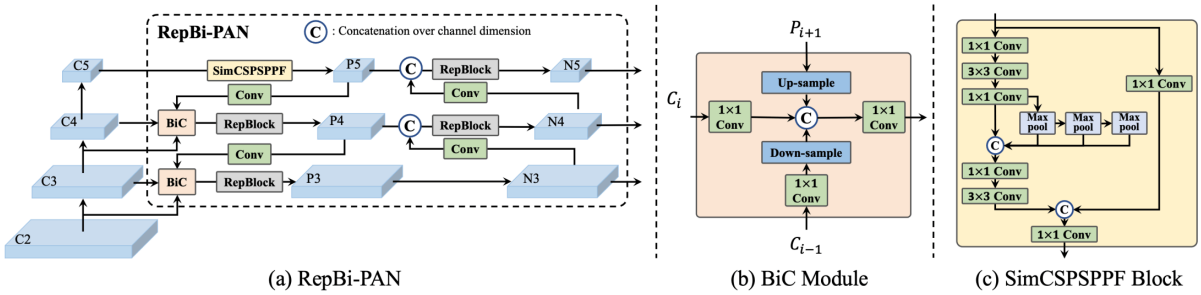
- Classification loss: se utiliza la función *VariFocal Loss* (VFL) [14], que se basa en la función *FocalLoss* [15], pero que manipula los ejemplos positivos y negativos de forma

asimétrica, lo que le permite balancear las señales aprendidas de cada lado.

- Box regression loss: En trabajos pasados de la familia Yolo se utilizaba la función de pérdida L1, en cambio en este caso se utiliza la función de pérdida *SIoU* [16].

2.4.4.4. Quantization and Deployment

La arquitectura de este modelo es apreciable en [[12], Figura 2.9].



(a) RepBi-PAN (b) BiC Module (c) SimCSPSPPF Block

Figura 2.9: Arquitectura Yolov6

Cabe mencionar que este modelo fue actualizado recientemente, agregando las siguientes características:

- Se renovó el *neck* del detector con un módulo *Bi-directional Concatenation* (BiC), a fin de proveer de una localización de objetos mucho más precisa. Así mismo, el módulo SPPF es simplificado, formando el módulo *SimCSPSPPFBlock*, trayendo un mayor desempeño a cambio de un despreciable costo en tiempo.
- Se utiliza una nueva estrategia de entrenamiento, denominada *anchor-aided training* (AAT).
- Se agregan nuevas etapas, tanto en el *backbone* como en el *neck*.

2.5. Segmentadores Semánticos

Los segmentadores semánticos son arquitecturas especializadas en *particionar* una imagen de entrada en múltiples regiones, lo que a su vez puede entenderse como una forma de clasifición a nivel de pixel, ya que cada uno de estos es asignado a una clase. Formalmente, dada una imagen de entrada I, se desea calcular una máscara de segmentación M donde, además de las clases existentes, se considera la clase *ground* para aquellas regiones o píxeles que no pertenecen a ninguna clase. Un ejemplo de una imagen junto a su máscara de segmentación es visible en [[17], Figura 2.10].

Figura 2.10: Ejemplo de Segmentación Semántica

2.5.1. SAM

SAM (Segment Anything Model) [18], es un nuevo modelo para segmentación sobre imágenes que ha sido específicamente diseñado y entrenado para ser *promptable*, es decir, que sea capaz de entender que y como se desea el *output* del modelo. Esta característica se traduce en la capacidad del modelo a adaptarse a nuevas tareas, incluso sin previo entrenamiento sobre estas últimas. Este modelo provee máscaras, calidad de la predicción sobre estas, así como mascarás de baja resolución que pueden ser utilizadas en iteraciones posteriores. El diagrama de este modelo es apreciable en [[18], Figura 2.11].

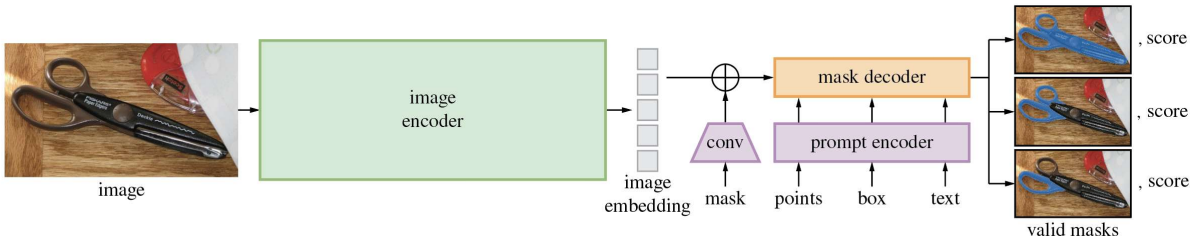


Figura 2.11: Diagrama SAM

En cuanto al modo de uso de este modelo, para obtener una máscara de un objeto, se pueden entregar tanto uno como múltiples puntos, los cuales deben estar en el formato (x, y), además de asignarle a estos puntos las etiquetas 1 o 0, que simbolizan un punto a segmentar o uno a excluir, respectivamente. Adicionalmente, se puede entregar como *input* un *bounding box* en el formato $xyxy$, que en conjunto a puntos, permiten al modelo una selección mucho más precisa al momento de realizar una segmentación sobre una imagen nunca antes vista, lo que se conoce como *zero-shot*. Esto es visible en [[18], Figura 2.12], donde se logra segmentar únicamente el neumático del camión, en vez de la rueda en su totalidad.

Figura 2.12: Ejemplo de múltiples tipos de *inputs*

2.6. Estimadores de Profundidad Monoculares

La estimación de profundidad monocular es una tarea en visión computacional en la que se predice la profundidad de una escena, dada una única imagen (monocular). En otras palabras, se estima la distancia de cada pixel, dentro los objetos involucrados en una escena, relativa a la cámara del punto de vista.

2.6.1. MIM

Masked Image Modeling (*MIM*) [19], es un modelo cuyo funcionamiento consta de enmascarar ciertas porciones de la imagen de entrada, para luego intentar predecir estas porciones enmascaradas. Este modelo consta de 4 componentes primordiales:

1. *Masking strategy*: Dada una imagen de *input*, este componente selecciona el área a enmascarar, además de como realizar este proceso.
2. *Encoder architecture*: Extrae un vector de características a la imagen enmascarada, que luego es usado para predecir las regiones enmascaradas. El *encoder* entrenado es preferible que sea transferible para diversas tareas de visión computacional, por lo que se consideran arquitecturas como *ViT* y *Swin Transformer*.
3. *Prediction head*: Este componente será aplicado al vector de características para intentar predecir la región enmascarada.
4. *Prediction target*: Este componente define la forma de la región a predecir, que puede ser tanto los valores iniciales de cada pixel, así como alguna transformación de estos píxeles. Adicionalmente, define el tipo de *loss*, que típicamente incluye del tipo *cross-entropy*, y valores de perdida l_1 o l_2.

Un ejemplo de un modelo *MIM* es visible en [[20], Figura 2.13]. En las siguientes subsecciones se explicarán las opciones típicamente utilizadas por cada componente.

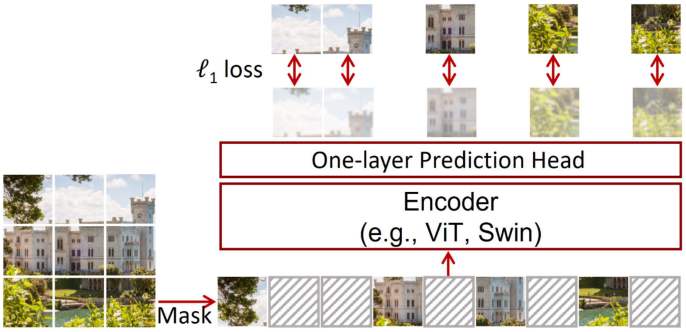


Figura 2.13: Modelo SimMIM

2.6.1.1. Masking Strategy

Para realizar la transformación del área enmascarada para las imágenes de entrada se utiliza un vector entrenable denominado *mask token*. Para la selección del área, se utilizan distintas estrategias, ilustradas en [[20], Figura 2.14]:

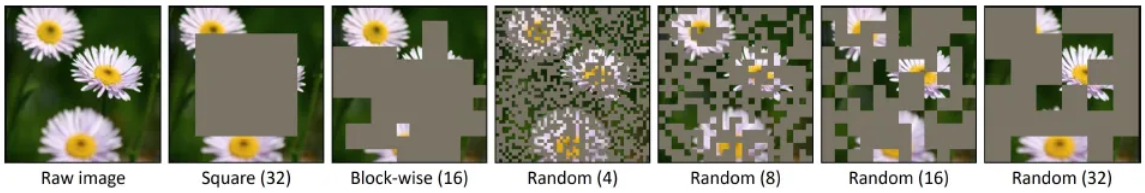


Figura 2.14: Estrategias de enmascarado

- *Patch-aligned random masking*: Los bloques de imágenes son utilizados ampliamente por los *ViT*, y es por ello que es conveniente operar el enmascarado de imágenes en bloque, de forma que el bloque está completamente enmascarada o no. Generalmente, se utilizan bloques de tamaño 32×32.
- Otras estrategias de enmascarado son; enmascarar de forma aleatoria, o bloques *block-wise* complejos.

2.6.1.2. Prediction Head

Este componente puede ser de forma y capacidad arbitraria, siempre y cuando el *output* cumpla con la imagen objetivo. En este caso se pueden utilizar tanto *auto-encoders*, capas *MLP*, o incluso un *Swin-T* inverso.

2.6.1.3. Prediction Targets

Es en este componente donde se atribuye un valor de profundidad a los mapas de características entregados por el componente anterior. Dado que los píxeles pueden tomar valores continuos en el espacio de colores, una opción es predecir los valores iniciales de los píxeles enmascarados usando regresión. Generalmente, las arquitecturas de visión computacional

producen mapas de características de una resolución baja utilizando submuestreo, del tipo 16× o 32×. Para predecir totalmente los píxeles en la resolución original, se *mapea* el vector de características a la resolución original, permitiendo a este vector encargarse de la predicción de cada pixel. El $l_1 - loss$ es aplicado sobre los píxeles enmascarados:

$$L = \frac{1}{\Omega(x_M)} \|y_M - x_M\|_1 \tag{2.5}$$

donde x, y $\in \mathbb{R}^{3HW \times 1}$ es la imagen RGB de *input* y *output* respectivamente, M denota el grupo de píxeles enmascarados, $(\cdot)$ es el número de elementos.

2.6.2. BinsFormers

BinsFormer [21], es un modelo que consta principalmente de 3 componentes esenciales, visibles en [[21], Figura 2.15]; un primer módulo a nivel de píxeles (*Per-pixel module*), un segundo módulo que hace uso de *transformers* (*Transformer module*), y un último módulo que estima la profundidad (*Depth prediction module*).

Este modelo funciona de la siguiente manera; dada una imagen RGB, el módulo a nivel de píxeles se encarga de extraer características de esta, para luego decodificarlas en características *multi-scale* **F**, y una representación por cada pixel f_p. Luego, las características **F** son entregadas como *query* al modelo *transformer*, y su salida proyectada por *MLPs* independientes, obteniendo así predicciones y representaciones de *bins*, siendo respectivamente **b** y f_b. Finalmente, el módulo de estimación de profundidad combina la salida de los módulos previos, entregando un mapa de profundidad, calculando una distribución de probabilidad P, junto a los centros de los *bins* $c(\mathbf{b})$, mediante combinaciones lineares.

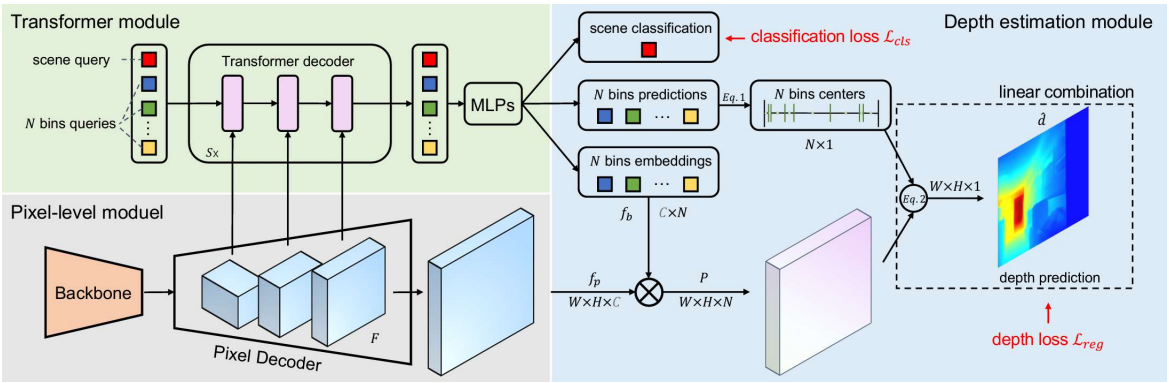


Figura 2.15: Arquitectura BinsFormer

2.6.2.1. Per-pixel module

Toma una imagen de tamaño $H \times W$ como *input*. Un *backbone* es aplicado a esta para extraer un conjunto de características. Luego, usando un *decoder* se aumentan las características, a fin de generar una representación por pixel $f_p \in \mathbb{R}^{H \times W \times C}$, donde C es la dimensión de representación. Esto produce S mapas de características $\mathbf{F} = \{f_i\}_{i=1}^{S}$, donde f_i indica el i-ésimo mapa de características.

2.6.2.2. Transformer module

Aplica el *decoder* estándar, transformando N representaciones usando los mecanismos de *self-attention*. Estas representaciones sirven como *queries* para interactuar con las características $\mathbf{F}$, y son transformadas en representaciones $Q \in \mathbb{R}^{C_q \times N}$ usando el *decoder*. Luego, se les aplica un *perceptron* linear con *softmax* para obtener N largos de *bins* $\mathbf{b} = \{b_i\}_{i=1}^{N}$. Asimismo, se utilizan 3 capas de *perceptron* con una activación *ReLU* para obtener N representaciones de *bins* $f_b \in \mathbb{R}^{C \times N}$, a fin de calcular la similitud con representaciones f_p en el siguiente módulo.

2.6.2.3. Depth prediction module

Junta la salida de los anteriores módulos, a fin de estimar una profundidad. Dado los largos de los *bins* $\mathbf{b}$ obtenidos desde el módulo *transformer*, usamos la siguiente conversión:

$$c(b_i) = d_{min} + (d_{max} - d_{min}) \left(\frac{b_i}{2} + \sum_{j=1}^{i-1} b_j \right) \tag{2.6}$$

donde $c(b_i)$ es la profundidad central del i-ésimo *bins*, d_{max} y d_{min} son los valores máximos y mínimos del *dataset*, respectivamente.

Mientras tanto, se obtiene un mapa de similitud usando el producto punto entre las representaciones de píxeles f_p y las representaciones de *bins* f_b, y si luego se aplica una función *softmax* obtenemos un mapa de distribución de probabilidad $P \in \mathbb{R}^{H \times W \times N}$. Finalmente, por cada pixel, el valor de profundidad $\hat{d}$ es calculado a partir de una combinación lineal de la distribución de probabilidad y las profundidades centrales de *bins* $c(b)$ como sigue:

$$\hat{d} = \sum_{i=1}^{N} c(b_i) p_i \tag{2.7}$$

Luego de estimar los mapas de profundidad, se aplica la siguiente función de perdida:

$$\mathcal{L}_{reg} = \alpha \sqrt{\frac{1}{T} \sum_i g_i^2 - \frac{\lambda}{T^2} (\sum_i g_i)^2} \tag{2.8}$$

donde $g_i = log - logd_i$, con d_i la profundidad objetivo y la profundidad estimada. T denota el número de píxeles que poseen valores válidos de profundidad objetivo.

2.7. Optimizadores

Al momento del entrenamiento de una red, se calcula un valor de *loss* o pérdida que se desea minimizar, buscando que el modelo tenga un desempeño óptimo. Para este fin, se ocupan los llamados optimizadores, los que poseen distintos parámetros y propiedades, pero que comparten el ser algoritmos iterativos basados en los gradientes de cada uno de los pesos de la red respecto al valor de *loss*.

2.7.1. SGD

SGD (*Stochastic Gradient Descent*) [22] es un algoritmo de optimización basado en gradientes de primer orden, para funciones estocásticas. Utiliza la siguiente ecuación:

$$\theta := \theta - \alpha \nabla_\theta f(\theta_{t-1}, X, Y) \tag{2.9}$$

donde θ son los pesos de la red, α es el *learning rate*, y f es la función de pérdida que se calcula a partir de los pesos de la red, el batch de datos X y sus etiquetas respectivas Y.

2.7.2. *ADAM*

ADAM (*ADAptive Moment estimation*) [23] es un algoritmo de optimización el cual, al igual que el anterior, está basado en gradientes de primer orden, para funciones estocásticas. Este método calcula *learning rates* de forma individual para distintos parámetros, donde se destacan los *momentum* para el estimador del primer momento del gradiente (esperanza del gradiente), y además para el segundo momento del gradiente (varianza sin centrar). Estos quedan definidos como sigue:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta f_t(\theta_{t-1}, X, Y) \tag{2.10}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla_\theta^2 f_t(\theta_{t-1}, X, Y) \tag{2.11}$$

Dado que estos estimadores se inicializan en 0, y que los valores de β tienden a ser 1, se hace la siguiente corrección:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad , \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.12}$$

Finalmente, los pesos de la red se actualizarán siguiendo la fórmula:

$$\theta := \theta - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t \tag{2.13}$$

2.8. Técnicas de evaluación

2.8.1. *Intersection Over Union* (IOU)

Es el resultado de calcular la proporción entre el área de la intersección de los *bounding boxes* a comparar (detectado y *ground truth*), y el área de la unión de estos. Dicho resultado nos ayudará a determinar la calidad de la detección, así como en el cálculo del AP y mAP, que serán definidos más adelante. Se calcula como se indica en [[24], Figura 2.16].

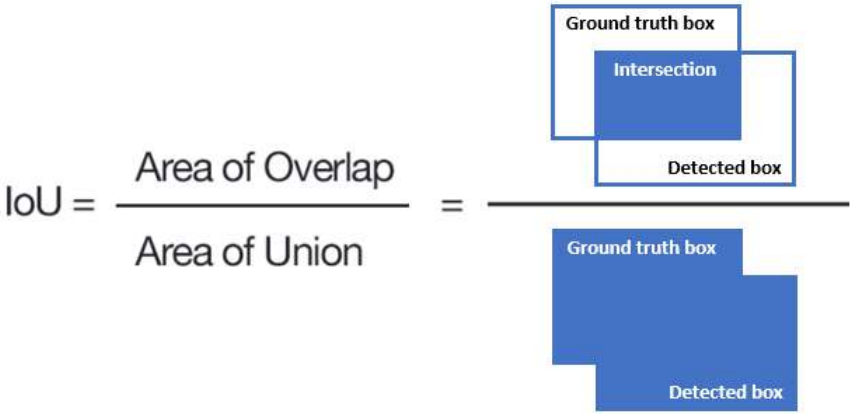


Figura 2.16: Fórmula *Intersección Over Union*

2.8.2. Tipos de Predicción

Los tipos de resultado de una predicción, entregada por un modelo de detección de objetos, puede ser clasificada como una de estas 4 categorías, las que se determinan según un umbral de IOU, que denominaremos IOU_t.

- ***True Positive* (TP)**: El modelo predice que en una región de la imagen se encuentra un objeto perteneciente a una clase, el IOU entre la predicción y el *ground truth* es mayor a IOU_t y las clases concuerdan.
- ***False Positive* (FP)**: El modelo predice que en una región de la imagen se encuentra un objeto perteneciente a una clase, pero el IOU entre la predicción y el *ground truth* es menor a IOU_t, o bien, es mayor a IOU_t pero ya existe un TP con una confianza mayor.
- ***True Negative* (TN)**: El modelo predice que en una región de la imagen **no** se encuentra un objeto perteneciente a una clase.
- ***False Negative* (FN)**: El modelo predice que en una región de la imagen se encuentra un objeto perteneciente a una clase, pero no hay una predicción con IOU mayor a IOU_t, o se equivoca en la clase.

2.8.3. *Precision*

Corresponde a la cantidad de detecciones correctas (*True Positives*) sobre el total de detecciones. Queda entonces definido por la siguiente fórmula:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{total\ de\ detecciones} \tag{2.14}$$

Este valor se puede entender como qué tan preciso es el modelo al momento de detectar.

2.8.4. *Recall*

Corresponde a la cantidad de detecciones correctas sobre el total de *ground truth* presentes.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{total\ de\ ground\ truths} \tag{2.15}$$

Este valor se puede entender como qué tan bueno es el modelo detectando todo lo que debiera detectar.

2.8.5. F_1 *score*

Representa la media armónica entre los valores de *recall* y *precision*. Considera tanto falsos positivos como falsos negativos, con lo cual funciona de buena manera en *datasets* con ejemplos no balanceados.

$$F_1 = \frac{2 \cdot (precision \cdot recall)}{precision + recall} \tag{2.16}$$

2.8.6. Curva *precision-recall*, AP y mAP

La curva *precision-recall* muestra la relación entre *precision* y *recall* para diferentes IOU_t. Para la creación de esta curva, se debe seguir el siguiente procedimiento:

1. Obtener las detecciones sobre el *test set*, para luego ordenarlas de mayor a menor, según el valor de *confidence score* asociado.
2. Calcular el *IOU* entre las detecciones y los *ground truths*, para así clasificar la detección según los tipos de predicción detallados en 2.8.2.
3. Según el orden obtenido en 1., calcular *precision* y *recall* para cada detección, ocupando los TP y FP acumulados hasta el momento del cálculo.

Con esto, obtendremos una curva similar a la visible de color azul en la Figura 2.17. Así mismo, podemos introducir el valor de AP (*Average Precision*) [25], que representa el área bajo la curva de la curva *precision-recall*, y que se expresa como sigue.

$$AP = \int_0^1 p(r)dr \tag{2.17}$$

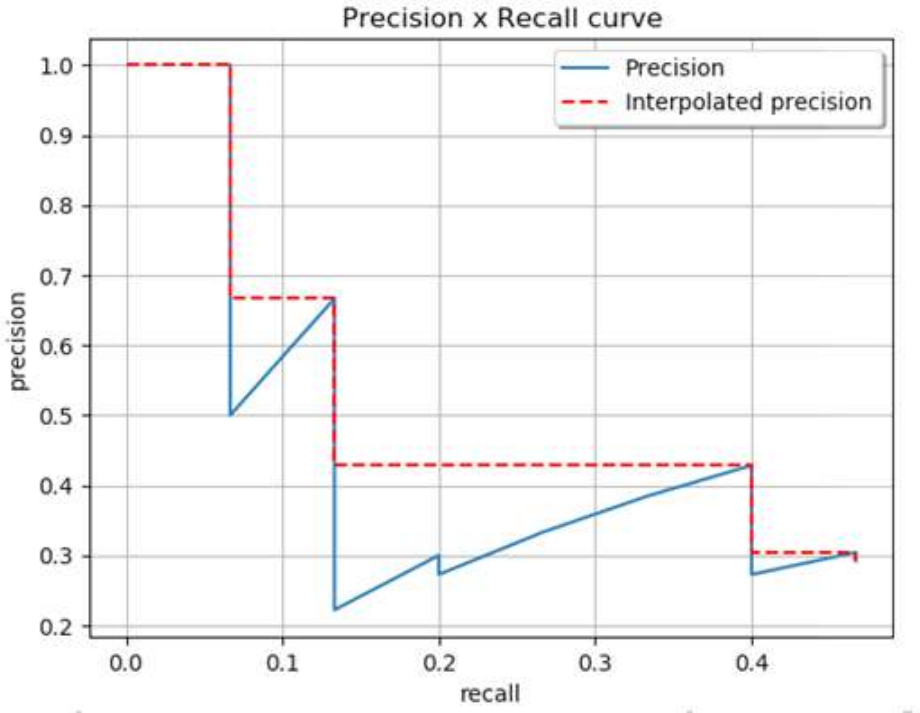


Figura 2.17: Curva *precision-recall*

Sin embargo, en la práctica se usa una suma finita sobre ventanas de *recall*, presentada en la Ecuación 2.18, donde se interpola la *precision*, tomando la máxima *precision* sobre todos los *recall* mayores al actual. Con esto, obtendremos una curva similar a la visible de color rojo en [[25], Figura 2.17].

$$AP = \sum_{n=0} (r_{n+1} - r_n) p_{interp}(r_{n+1}) \ , \quad con \quad p_{interp}(r_{n+1}) = máx_{\hat{r};\hat{r} \geq r_{n+1}} p(\hat{r}) \tag{2.18}$$

Con esta definición, se definen las siguientes métricas:

- AP: Utilizando la Ecuación (2.18), se promedian los AP a distintos valores de IoU^t. Dichos valores se toman en el intervalo [0.5, 0.95], a un paso de 0.05.
- AP50: Se calcula el AP con $IoU^t = 0.5$.
- AP75: Se calcula el AP con $IoU^t = 0.75$.

Además, si se computa el AP por cada clase existente, se puede tomar el promedio de todos los AP, obteniendo así el *Mean Average Precision* (mAP)[25]:

$$mAP = \frac{1}{|C|} \sum_{c_i \in C} AP_{c_i} \tag{2.19}$$

donde AP_{c_i} corresponde al AP calculado para la i-ésima clase.

2.8.7. Métricas entre máscaras de profundidad

Para el caso de la evaluación de las máscaras de profundidad, se utilizaron protocolos de evaluación estándar definidos en [26], los que dependen de las siguientes matrices, definidas por:

$$thresh = max(\frac{Mask_{gt}}{Mask_{pred}}, \frac{Mask_{pred}}{Mask_{gt}}) \tag{2.20}$$

$$diff = Mask_{pred} - Mask_{gt} \tag{2.21}$$

$$diff\_log = log(Mask_{pred}) - log(Mask_{gt}) \tag{2.22}$$

donde $Mask_{gt}$ es la máscara de referencia, y $Mask_{pred}$ es la máscara entregada por el modelo respectivamente. Luego, se tiene métricas para el error, que mientras más cercanas a 0 mejor, tales como el error cuadrático medio (rmse), error cuadrático medio logarítmico (rmse_log), entre otros. Adicionalmente, tenemos métricas para el *accuracy*, las cuales son las medidas $\delta_i < 1{,}25^i$, para $i \in \{1, 2, 3\}$, y que deben aspirar a valores cercanos a 1.

$$\delta_1 = \sum_{i \in N} \sum_{j \in M} \mathbb{1}_{\mathbb{R}<1{,}25}(a_{ij}) \cdot \frac{1}{N \cdot M} \tag{2.23}$$

$$\delta_2 = \sum_{i \in N} \sum_{j \in M} \mathbb{1}_{\mathbb{R}<1{,}25^2}(a_{ij}) \cdot \frac{1}{N \cdot M} \tag{2.24}$$

$$\delta_3 = \sum_{i \in N} \sum_{j \in M} \mathbb{1}_{\mathbb{R}<1{,}25^3}(a_{ij}) \cdot \frac{1}{N \cdot M} \tag{2.25}$$

$$abs\_rel = \sum_{i \in N} \sum_{j \in M} \frac{|b_{ij}|}{d_{ij}} \cdot \frac{1}{N \cdot M} \tag{2.26}$$

$$sq\_rel = \sum_{i \in N} \sum_{j \in M} \frac{b_{ij}^2}{d_{ij}} \cdot \frac{1}{N \cdot M} \tag{2.27}$$

$$rmse = \sqrt{\sum_{i \in N} \sum_{j \in M} b_{ij}^2 \cdot \frac{1}{N \cdot M}} \tag{2.28}$$

$$rmse\_log = \sqrt{\sum_{i \in N} \sum_{j \in M} c_{ij}^2 \cdot \frac{1}{N \cdot M}} \tag{2.29}$$

$$log10 = \sum_{i \in N} \sum_{j \in M} |log_{10}(\frac{d_{ij}}{e_{ij}})| \cdot \frac{1}{N \cdot M} \tag{2.30}$$

$$silog = \sqrt{\sum_{i \in N} \sum_{j \in M} c_{ij}^2 \cdot \frac{1}{N \cdot M} - 0{,}5 \cdot (\sum_{i \in N} \sum_{j \in M} c_{ij} \cdot \frac{1}{N \cdot M})^2} \tag{2.31}$$

donde a_{ij}, b_{ij}, d_{ij}, c_{ij}, y e_{ij} son los coeficientes de las matrices $thresh$, $diff$, $diff\_log$, $Mask_{gt}$, y $Mask_{pred}$ respectivamente.

2.9. Estado del Arte

A fin de hablar del estado del arte en torno al trabajo realizado, debemos dividir este en los 2 tópicos más importantes, es decir, la detección por un lado, y la estimación de volumen por otro. Para cada uno de estos, señalaremos ciertos trabajos realizados con anterioridad, filtrando aquellos que utilizan más de una imagen, así como métodos no basados en *deep learning*.

2.9.1. Detección de Ingredientes en Comidas

Podemos mencionar el trabajo de Jingjing Chen and Chong-Wah Ngo [27], donde se proponen diversas arquitecturas basadas en *deep learning* para un aprendizaje simultáneo entre la detección de ingredientes y la categoría de la comida, ambas obtenidas desde una imagen. Utilizando el *dataset Vireo-Food172*, se obtuvieron valores de F_1 *score* de un 47,18 %.

Otro modelo es el propuesto por Bolaños et al. [28], donde se utiliza una *CNN* para actuar como un predictor *multi-label* para el aprendizaje de recetas, en términos de una lista de ingredientes obtenidas desde una imagen. Utilizando el *dataset Recipe5k*, se obtuvieron valores de F_1 *score* de un 47,51 %.

Vemos además algunos modelos matemáticos, con base en *deep learning*, como los métodos formulados por J. Chen et al. [29], donde se proponen métodos basados en DCNN para reconocimiento de ingredientes. Utilizando el *dataset Vireo-Food172*, se obtuvieron valores de F_1 *score* de un 61,74 %.

2.9.2. Estimación de Volumen en Comidas

En este apartado podemos mencionar a Yang et al. [30], quien estima el volumen al computar el producto punto entre un vector de volumen y un vector de probabilidad, calculado a partir de una red *MobileNetV2* [31] modificada. Utilizando el *dataset Real Food*, se obtuvieron errores volumétricos promedio cercanos a un 20,1 %.

Así mismo, se destaca el trabajo de Graikos et al. [32], donde se genera una nube de puntos tridimensionales a partir de un mapa de profundidad, una máscara de segmentación, y parámetros de la cámara. Esto, para aproximar el volumen mediante un algoritmo *cloud-to-volume*. Utilizando el *dataset EPIC-KITCHENS and their own food video*, se obtuvieron errores volumétricos promedio cercanos a un 36,9 %.

Finalmente, se señala a Koichi Okamoto [33], quien en su trabajo estima las calorías de un plato fotografiado, al usar objetos de referencia para determinar regiones de comida, así como de estimar una curva cuadrática sobre los ingredientes en la imagen. Utilizando el *dataset 20 kinds of Japanese Foods*, se obtuvieron errores volumétricos promedio cercanos a un 21,3 %.

Las propuestas señaladas anteriormente no dan solución al problema planteado en este trabajo, ya que estas se centran en la detección y segmentación de ingredientes no granulares, lo que añade una capa adicional de complejidad, dada la escala en la que estos ingredientes son captados en cámara, pero que, sin embargo, entregan ciertas señales sobre hacia donde encaminar el presente trabajo.

Capítulo 3

Solución Propuesta

En este capítulo se detallará la solución propuesta para llevar a cabo la presente experiencia. Dicha solución especificará la configuración experimental utilizada, la que contempla los modelos y sus configuraciones, el hardware utilizado, los *datasets* utilizados y sus propiedades, los tratamientos aplicados a los datos, y finalmente el entrenamiento de los modelos. Se presenta además la metodología propuesta para la detección y estimación de porción de ingredientes granulares.

3.1. Modelos a utilizar

Para la elección de los diversos modelos utilizados durante esta experiencia, se hizo uso de los *rankings* suministrados en [34], buscando aquellos modelos con un equilibrio entre desempeño y su número de parámetros, o más bien, su tamaño en memoria, a fin de cumplir los requisitos computacionales disponibles en la empresa. Con ello, en este trabajo se utilizará tanto *Yolov6* como *Faster R-CNN* para el caso de detección de objetos, mientras que para la segmentación semántica se utilizara *SAM* en *zero-shot*. Finalmente, la estimación de profundidad monocular se obtendrá utilizando tanto *BinsFormer* como *MIM*.

3.2. Entorno de Entrenamiento y Pruebas

Para los diversos experimentos se utilizará la configuración de hardware que provee *Google Colaboratory*, y que se muestra en la Tabla 3.1, donde el hardware corresponde a una instancia de máquina virtual en la nube. Es importante recalcar la importancia de utilizar en los experimentos una GPU (*Graphics Processing Unit*), debido a su capacidad de paralelismo, permitiendo así entrenar un modelo más rápido que en una CPU (*Central Processing Unit*).

	CPU	GPU	RAM	Disco
Google Colab	Xeon 2,3 GHz	Tesla K80 12GB VRAM	12,6 GB	33 GB
Local	i5 13.600K 5,1 GHz	RTX 4070 Ti 12GB VRAM	32 GB	100 GB

Tabla 3.1: Hardware utilizado

3.3. *Dataset*

El *dataset* a utilizar durante el entrenamiento será proporcionada por *Kwali SPA*, la empresa donde se realiza la presente experiencia. Dicho *dataset* contiene más de 2.000 imágenes de pizzas extraídas directamente desde las cocinas de *PizzaHut*®, y que son repartidas entre 8 especialidades. Estas especialidades corresponden a: *pepperoni*, *italiana*, *milano*, *española*, *meaty*, *american_meat*, *españolísima*, *napolitana* y *veggie*.

Cada imagen del *dataset* posee un ID único, así como un archivo *.json* con el mismo ID, el que contiene la información completa de los *bounding boxes* que identifican a los todos ingredientes presentes en la imagen, como lo es:

- ***filename***: contiene el nombre de la imagen etiquetada.
- ***boxes***: contiene la información de los puntos *bounding boxes* dentro de una lista. El formato utilizado es

  $$[x1, y1, x2, y2]$$

  donde (x1, y1) corresponden a las coordenadas de la esquina superior izquierda, mientras que (x2, y2) a la esquina inferior derecha.
- ***labels***: contiene la información de la clase etiquetada en los *bounding boxes* dentro de una lista.
- ***area***: contiene la información del área envuelta por cada uno de los *bounding boxes* dentro de una lista.

Un ejemplo de imagen con sus respectivos *bounding boxes* es visible en la Figura 3.1.

Figura 3.1: Ejemplo de etiquetado de *bounding boxes*

Adicionalmente, en la Figura 3.2 es posible ver el conteo por clase dentro del *dataset* completo.

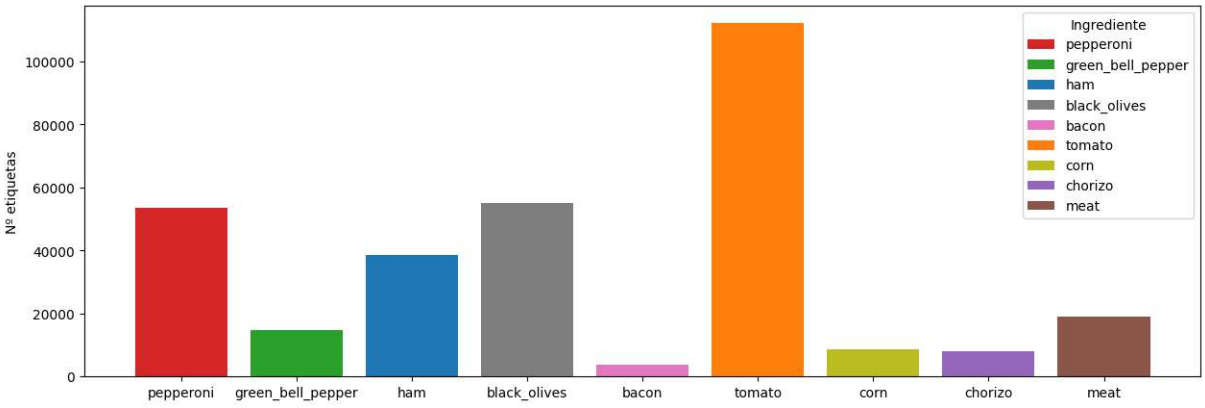


Figura 3.2: Conteo de instancias por clases

3.3.1. *Dataset* sintetico

Dada la escasez de *datasets* de profundidad RGB-D ligados al ambiente de preparaciones culinarias, sumado al costo relacionado a realizar una segmentación para más de 2,000 imágenes, se decide realizar un *dataset* sintético de máscaras de profundidad para las distintas pizzas contenidas en el *dataset* original. Para ello, debemos tener algunos valores en consideración, como la distancia entre la cámara y la tabla de corte, las alturas promedio, tanto de una pizza, así como de los diversos ingredientes, sumado a la posibilidad de que ciertos ingredientes sean cubiertos por otros.

Para el caso de la distancia entre la cámara, solo tomaremos lo ajeno a la pizza como distancia máxima (255). En cuanto a las alturas promedio, estas fueron suministradas por la empresa donde se realiza esta experiencia, y son mostrados en la Tabla 3.2. Otro elemento a considerar es la posibilidad de que ciertos ingredientes sean cubiertos por otros, a lo que recurriremos a los manuales de preparación de pizzas, donde determinados ingredientes se agregan primero que otros, como son los *pepperonis* y *chorizos*.

Ingrediente	pepperoni	green_bell_pepper	ham	black_olives	bacon	tomato	corn	chorizo	meat
Altura media [cm]	0.1	0.8	0.1	0.4	0.1	0.5	0.5	0.1	0.8

Tabla 3.2: Altura promedio por Ingrediente

Finalmente, tomando en cuenta que la altura promedio de una pizza ronda los 1,5$[cm]$, y que el tamaño máximo de las *pizzas* no debe sobrepasar los 3,5$[cm]$, que es la altura comercial de las cajas de *pizza*, utilizaremos la siguiente expresión para convertir de $[cm]$ a valores de píxeles en escala de grises.

$$Depth_{gray} = \left\lceil \frac{depth - 0{,}004}{0{,}0138} \right\rceil \tag{3.1}$$

donde al evaluar un valor de 3,5 [cm] da un resultado cercano a 255, que es lo que buscabamos.

Con estas consideraciones, un ejemplo de una máscara de profundidad de una imagen es visible en la Figura 3.3. Cabe mencionar que para la creación de este *dataset* no se consideraron los bordes de la pizza, dado que queda fuera del alcance de esta experiencia, la cual se centra únicamente en los ingredientes presentes en las imágenes suministradas.

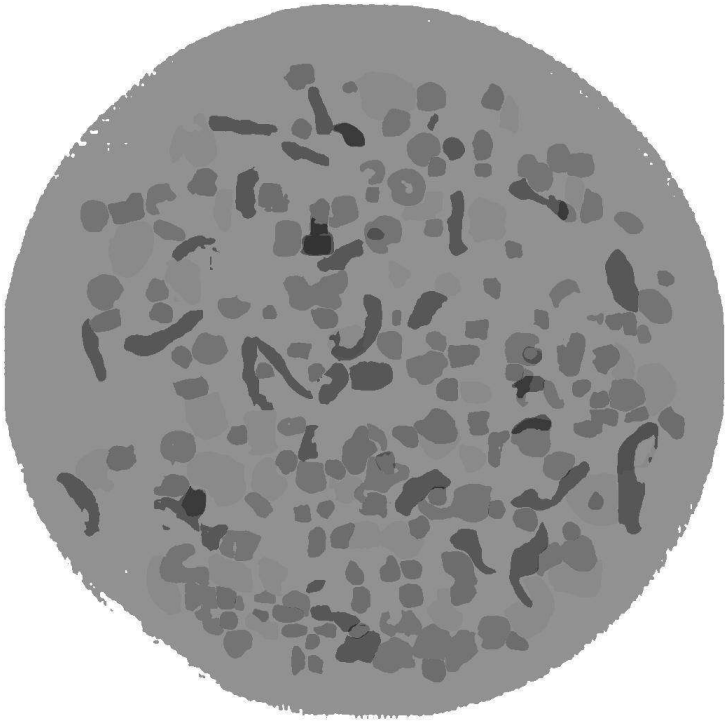


Figura 3.3: Ejemplo de máscara de profundidad sintética

3.4. Tratamiento de Datos

Para el correcto uso de los diversos modelos, es necesario realizar una serie de pasos.

3.4.1. Division del *dataset*

Para los distintos entrenamientos de los modelos previamente enunciados, se debió realizar una división de los *datasets* utilizados, a fin de obtener *sub-datasets* de entrenamiento, validación, y de prueba. Los porcentajes de data distribuida en cada unos de estos fue 70 %, 20 % y 10 %, lo que se traduce en cerca de 1400, 400 y 200 imágenes respectivamente.

3.4.2. Formato *YOLO*

En primera instancia, por cada imagen en el *dataset* debemos crear un archivo *.txt* con la información de los *bounding boxes* presentes en la imagen. Dichos archivos deben estar contenidos en la misma carpeta que las imágenes, y donde cada una de sus líneas deben estar escritas de la siguiente forma:

$$< class\_id > < x_{center} > < y_{center} > < width > < height >$$

Todos los valores, a excepción de *class_id*, deben estar normalizados según las dimensiones de la imagen, por lo que estos pertenecerán al conjunto (0, 1]. Estos valores son:

- ***class_id*** corresponde al id asociado a la clase.
- x_{center}, y_{center} corresponde al centro del *bounding box*. Se calculan como:

$$x_{center} = \frac{x_1 + x_2}{2} \cdot \frac{1}{W} \quad , \quad y_{center} = \frac{y_1 + y_2}{2} \cdot \frac{1}{H} \tag{3.2}$$

- ***width*** y ***height*** corresponden a las dimensiones del *bounding box*. Se calculan como:

$$width = \frac{x_2 - x_1}{W} \quad , \quad height = \frac{y_2 - y_1}{H} \tag{3.3}$$

3.4.3. Formato *Nyu*

En segunda instancia, por cada subconjunto del *dataset* debemos crear un archivo *.txt* con la información de los pares de imágenes (RGB, profundidad). Dichos archivos deben estar contenidos en la carpeta anexa a los datos originales, y donde cada una de sus líneas deben estar escritas de la siguiente forma:

$$< RGB\_path > < depth\_path >$$

con *RGB_path* y *depth_path* las direcciones de las imágenes RGB y de profundidad respectivamente.

3.5. Entrenamiento de modelos neuronales

3.5.1. Faster R-CNN

En este modelo descargaremos los pesos pre-entrenados, a fin de aprovechar los conocimientos adquiridos anteriormente. Hecho esto, se da inicio al entrenamiento con el comando:

```
python trainval_net.py --dataset <data_path> --net res101 -bs 32 --nw 4
                       --lr 5e-3 --lr_decay_step 5e-4 --cuda
```

3.5.2. Yolov6

Deberemos crear un archivo *.yaml* que contenga la siguiente información:

- ***path***: ruta al directorio donde se encuentran todos los archivos.
- ***train***: nombre de la carpeta de *train set*.
- ***val***: nombre de la carpeta de *validation set*.
- ***test***: nombre de la carpeta de *test set*.
- ***nc***: número de clases.
- ***names***: lista con los nombres de las clases.

Una vez configurado este archivo, se da inicio al entrenamiento con el comando:

```
python tools/train.py --epochs 50 --conf configs/yolov6s_finetune.py --batch 32 --workers 4
                      --data <yaml_file>
```

3.5.3. SAM

Dado que el código de entrenamiento no ha sido compartido por el repositorio de *MetaAI*, a la par de los buenos resultados obtenidos en segmentación *zero-shot* vistos en [18], es que opta por utilizar este modelo sin previo entrenamiento sobre el *dataset* utilizado durante la experiencia.

3.5.4. MIM

Una vez configurado este archivo, nos resta descargar los pesos pre-entrenados, a fin de aprovechar los conocimientos adquiridos anteriormente. Hecho esto, se da inicio al entrenamiento con el comando:

```
python train.py --epochs 5 --dataset nyudepthv2 --scale_size 640 --batch_size 1 --workers 1
                --max_depth 255 --max_depth_eval 255 --backbone swin_base_v2 --crop_h 480
                --crop_w 480 --depths 2 2 18 2 --num_filters 32 32 32 --deconv_kernels 2 2 2
                --window_size 30 30 30 15 --pretrain_window_size 12 12 12 6 --log_dir logs/
                --use_shift True True False False --shift_size 2 --drop_path_rate 0.3
                --layer_decay 0.9 --pretrained <weigths_file> --data_path <data_path>
```

3.5.5. Binsformers

En este caso deberemos únicamente modificar el parámetro *data_root* del archivo *configs/_base_/datasets/kitti.py* con la ruta al *dataset*. Hecho esto, se da inicio al entrenamiento con el comando:

```
python tools/train.py configs/binsformer/binsformer_swinl_22k_w7_kitti.py
```

3.6. Detección y estimación de porción de ingredientes granulares

Primeramente, comenzamos con la detección de los ingredientes, para lo cual utilizaremos uno de los modelos de detección de objetos presentados, obteniendo así una lista de *bounding boxes* con sus respectivas clases asociadas. Para la estimación de porción, emplearemos un modelo de estimación de profundidad, a fin de obtener un mapa de profundidad de la imagen.

Seguido de ello, hacemos uso de los *bounding boxes* obtenidos anteriormente, junto al modelo *SAM*, para crear un mapa de segmentación de cada ingrediente, y con ello poder segmentar la región asociada en el mapa de profundidad. Sin embargo, se debe tener cuidado al manipular dichos mapas de profundidad, dado que existen regiones donde 2 ingredientes se sobreponen unos a otros, debiendo en estos casos excluir estas regiones, y trabajar con la profundidad *pura* de cada ingrediente, dado que por esta oclusión no se puede conocer el tamaño real de estos.

Realizados estos pasos, tendremos tanto el área de cada ingrediente, representado en el mapa de segmentación, junto a su profundidad, representado en el mapa de profundidad, y con ello, el volumen de cada ingrediente. Para traducir estos valores a unidades de medida espacial haremos uso de las siguientes Ecuaciones:

$$Area_{ing} = \sum_{i \in N} \sum_{j \in M} \mathbb{1}_{\mathbb{R}=1}(m_{ij}) \cdot \left(\frac{100}{ppm}\right)^2 [cm^2] \tag{3.4}$$

$$Depth_{ing} = 0{,}0138 \cdot depth + 0{,}004[cm] \tag{3.5}$$

donde m_{ij} denota la máscara asociada al ingrediente, *ppm* (*pixel per meter*) representa la medida de píxeles por metro, y *depth* es el valor de profundidad asociado al ingrediente dentro del mapa de profundidad. Bastaría entonces multiplicar ambos valores, obteniendo así el volumen de cada ingrediente, y con ello el valor en onzas asociado, tomando en cuenta que 1 $[oz]$ equivale a 29,5735 $[cc]$.

Capítulo 4

Resultados

A continuación se presentan los resultados del entrenamiento de los modelos. Estos son obtenidos luego de evaluar sobre el conjunto de prueba de los *datasets* definidos anteriormente, aplicando las métricas detalladas en la Sección 2.8.

4.1. Detectores de Objetos

Comenzaremos evaluando los modelos entrenados, ocupando como métrica principal AP para distintos umbrales, junto a las curvas *precision-recall* correspondientes, además de mostrar algunos ejemplos de las salidas obtenidas. Cabe destacar que dichas métricas por clase son obtenidas al aplicar los modelos sobre las imágenes en su totalidad, es decir, con la presencia de las demás clases, lo que nos ayudara a discernir entre un modelo y otro fácilmente.

Modelo	Faster R-CNN				Yolov6			
Clase	AP_{50}	AP_{75}	mAP	F_1	AP_{50}	AP_{75}	mAP	F_1
all	0.479	0.366	0.33 ± 0.01	0.517	0.546	0.383	0.352 ± 0.009	0.586
pepperoni	0.866	0.781	0.67 ± 0.035	0.909	0.915	0.804	0.698 ± 0.033	0.893
green_bell_pepper	0.603	0.407	0.371 ± 0.035	0.741	0.693	0.421	0.4 ± 0.032	0.718
ham	0.694	0.525	0.46 ± 0.037	0.799	0.76	0.547	0.492 ± 0.035	0.752
black_olives	0.6	0.438	0.392 ± 0.037	0.742	0.727	0.486	0.453 ± 0.0346	0.764
bacon	0	0	0 ± 0	0	0.026	0.023	0.018 ± 0.003	0.136
tomato	0.348	0.171	0.184 ± 0.028	0.515	0.435	0.186	0.213 ± 0.027	0.549
corn	0.020	0.057	0.087 ± 0.018	0.384	0.085	0.026	0.037 ± 0.012	0.29
chorizo	0.629	0.576	0.487 ± 0.037	0.743	0.681	0.594	0.515 ± 0.035	0.715
meat	0.557	0.34	0.326 ± 0.034	0.679	0.598	0.366	0.346 ± 0.032	0.642

Tabla 4.1: Desempeño *Average precision* por clase

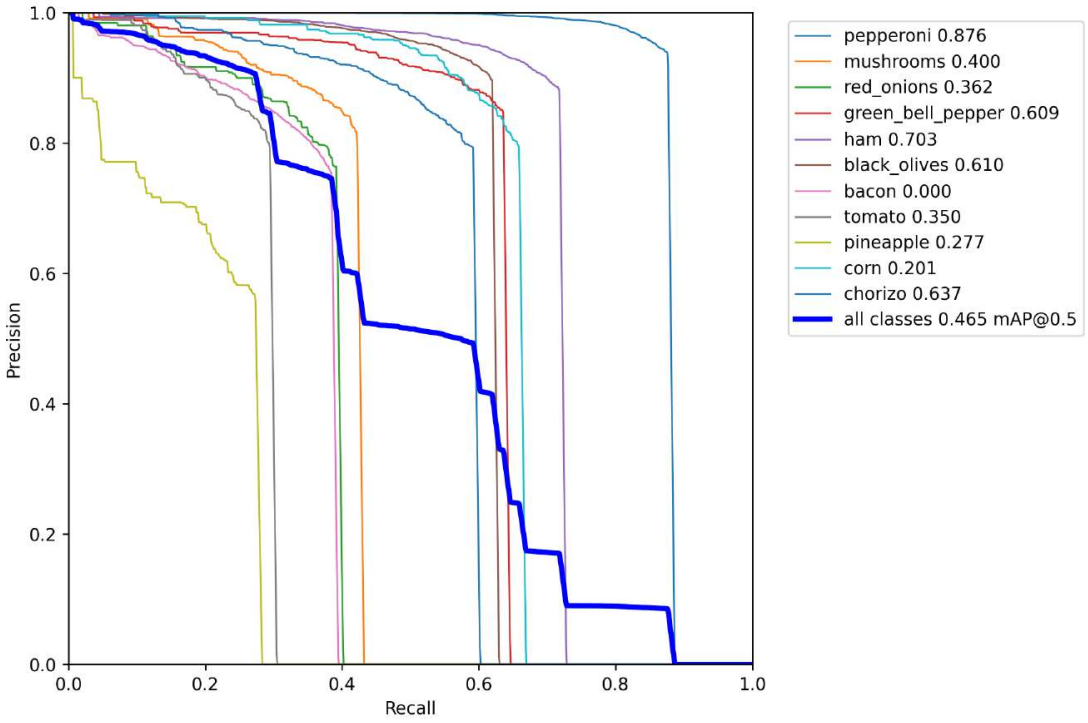


Figura 4.1: Curva PR Faster R-CNN

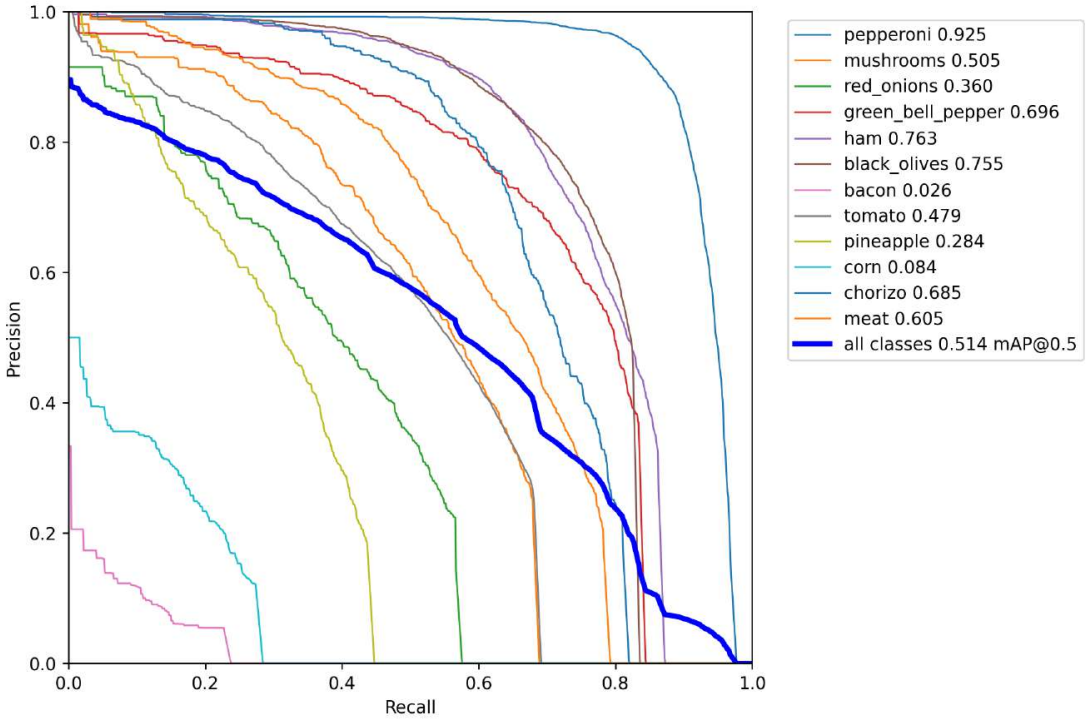
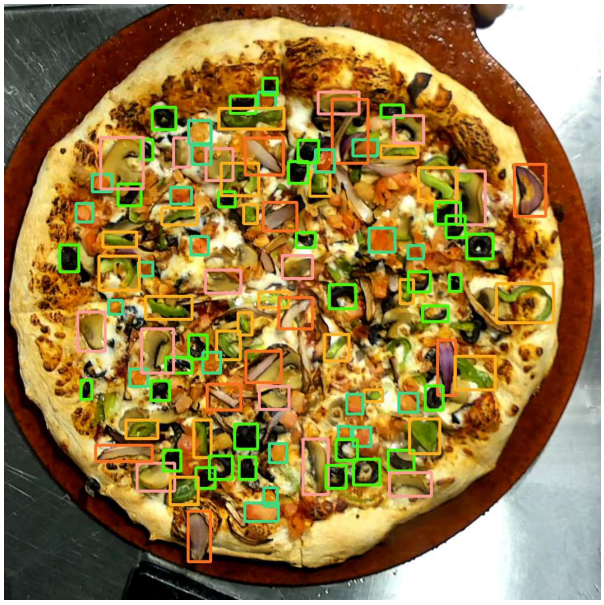
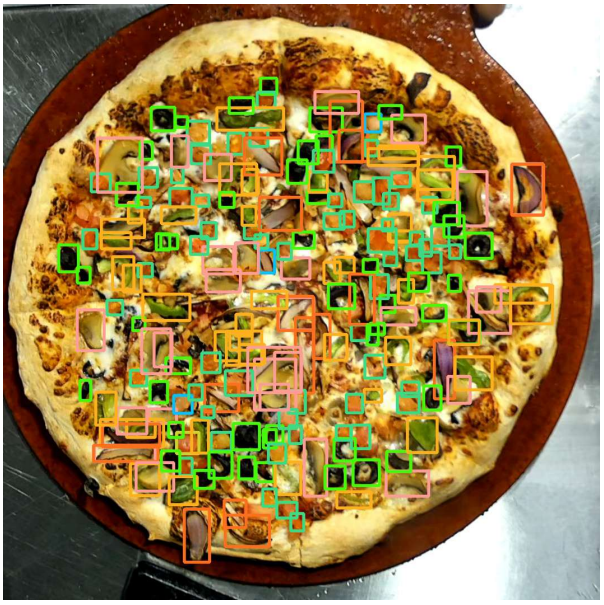


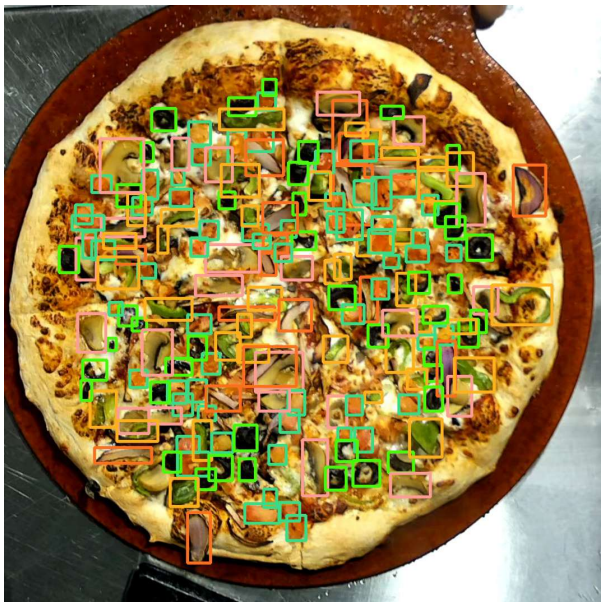
Figura 4.2: Curva PR Yolov6



(a) Faster-RCNN



(b) Yolov6



(c) Ground Truth

Figura 4.3: Ejemplo 1 de *bounding boxes*, usando ambos modelos, para una misma imagen, junto a su *ground truth*

(a) Faster-RCNN

(b) Yolov6

(c) Ground Truth

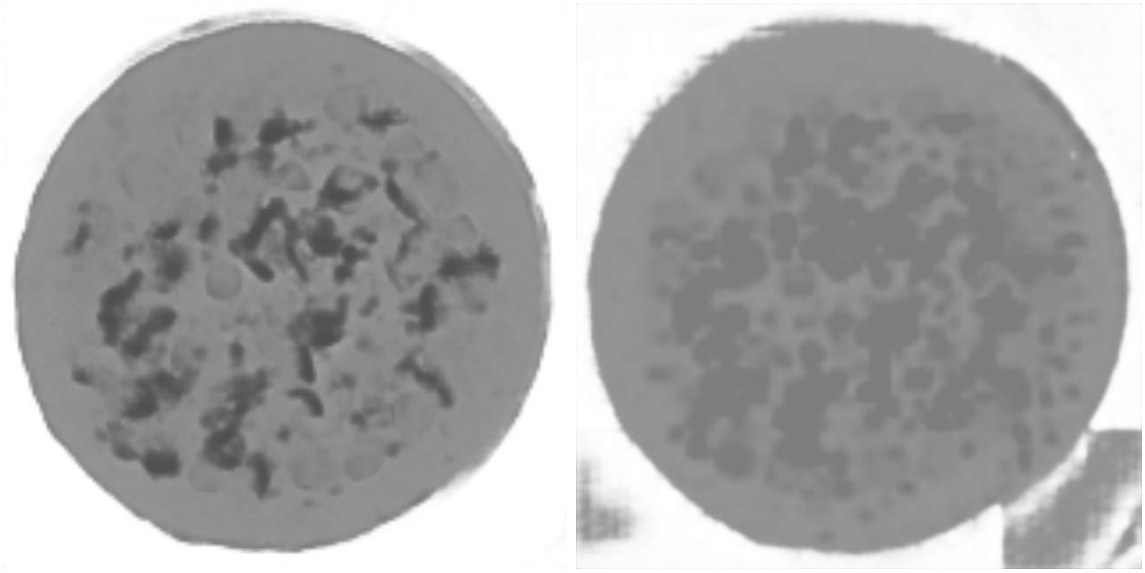
Figura 4.4: Ejemplo 2 de *bounding boxes*, usando ambos modelos, para una misma imagen, junto a su *ground truth*

4.2. Estimadores de Profundidad Monoculares

En este caso, evaluaremos los modelos ocupando las métricas definidas en la subsección 2.8.7, además de mostrar algunos ejemplos de las salidas obtenidas.

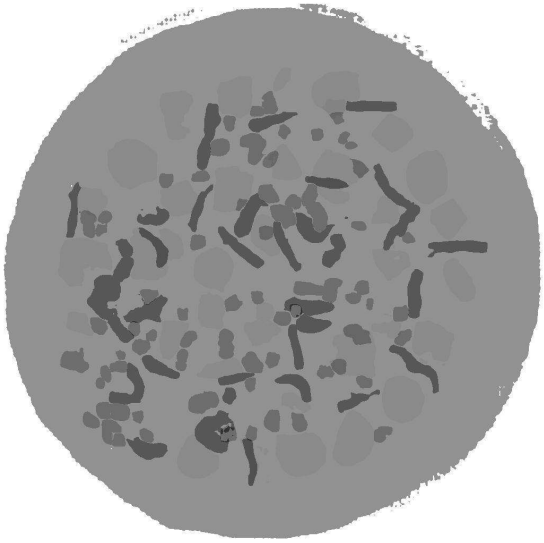
	Cercano a 1 mejor			Cercano a 0 mejor					
Modelo	δ_1	δ_2	δ_3	abs_rel	sq_rel	rmse	rmse_log	log10	silog
MIM	0.9220 ± 0.0063	0.9612	0.9997	0.0606	3.0684	23.3137 ± 1.0157	0.1264	0.0271	0.1246
Binsformer	0.9619 ± 0.0045	0.9901	0.9999	0.0489	0.0055	0.0548 ± 0.0029	0.0822	0.021	0.0795

Tabla 4.2: Desempeño Estimadores de Profundidad Monoculares



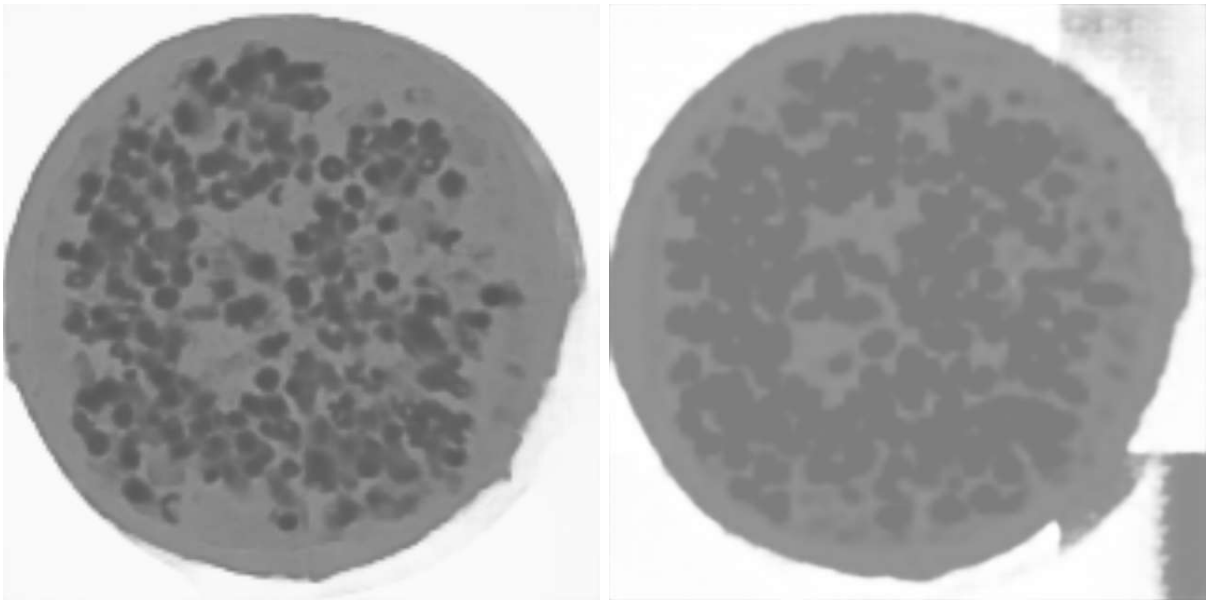
(a) BinsTransformer

(b) MIM



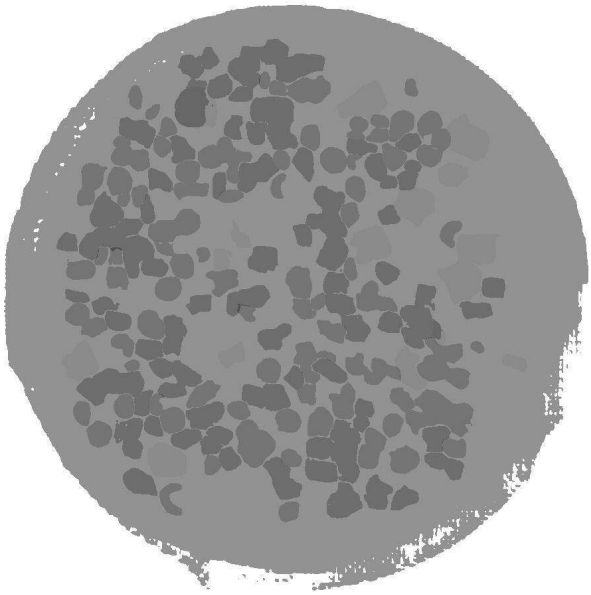
(c) Ground Truth

Figura 4.5: Ejemplo 1 de mapas de profundidad, usando ambos modelos, para una misma imagen, junto a su *ground truth*



(a) BinsTransformer

(b) MIM



(c) Ground Truth

Figura 4.6: Ejemplo 2 de mapas de profundidad, usando ambos modelos, para una misma imagen, junto a su *ground truth*

4.3. Estimación de porción de ingredientes granulares

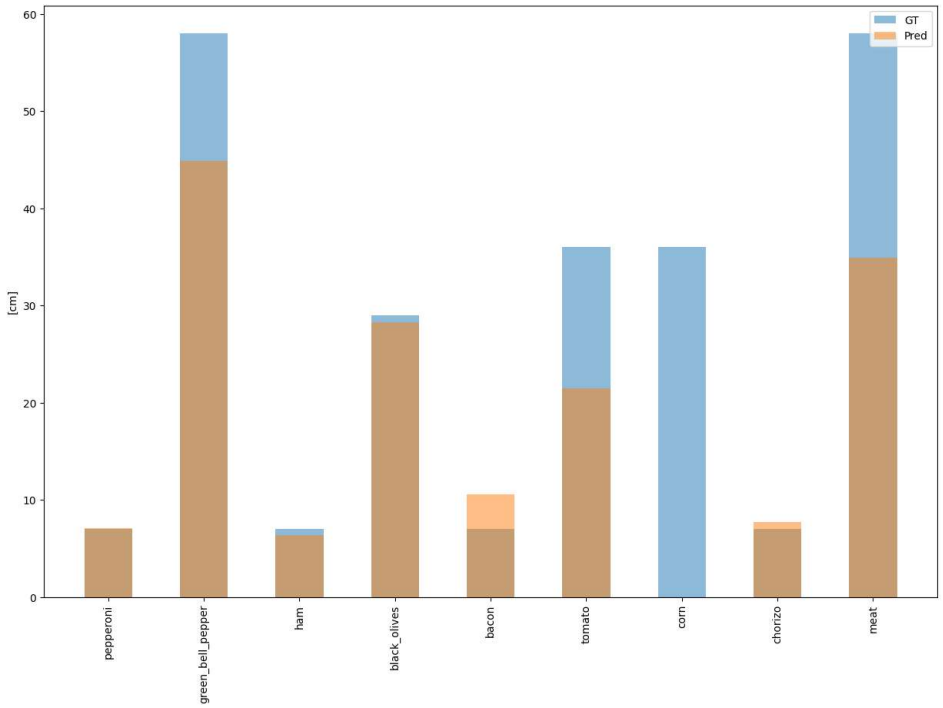


Figura 4.7: Profundidad promedio por clases

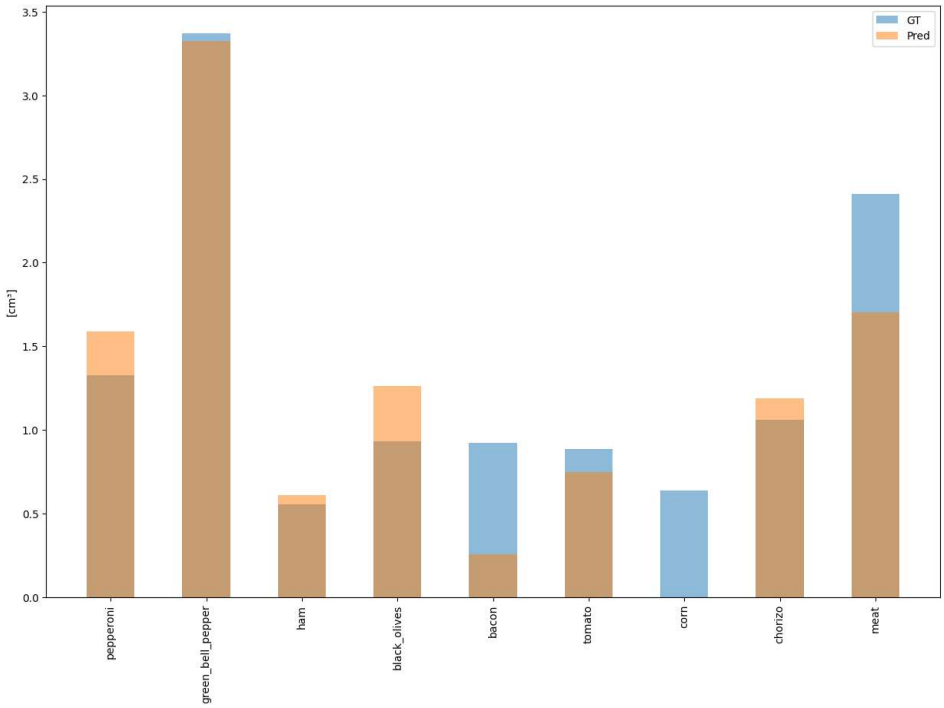
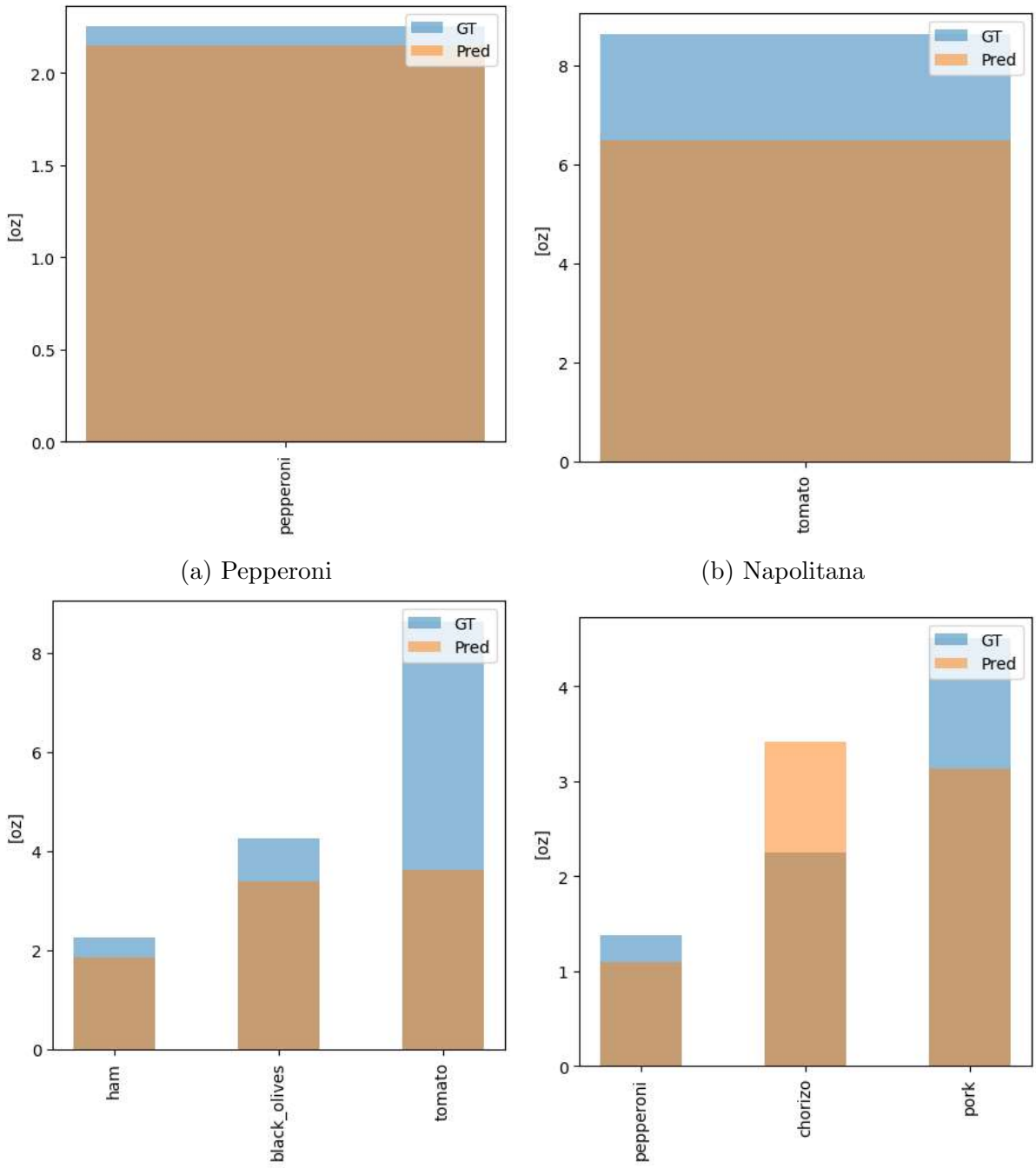


Figura 4.8: Volumen promedio por clases



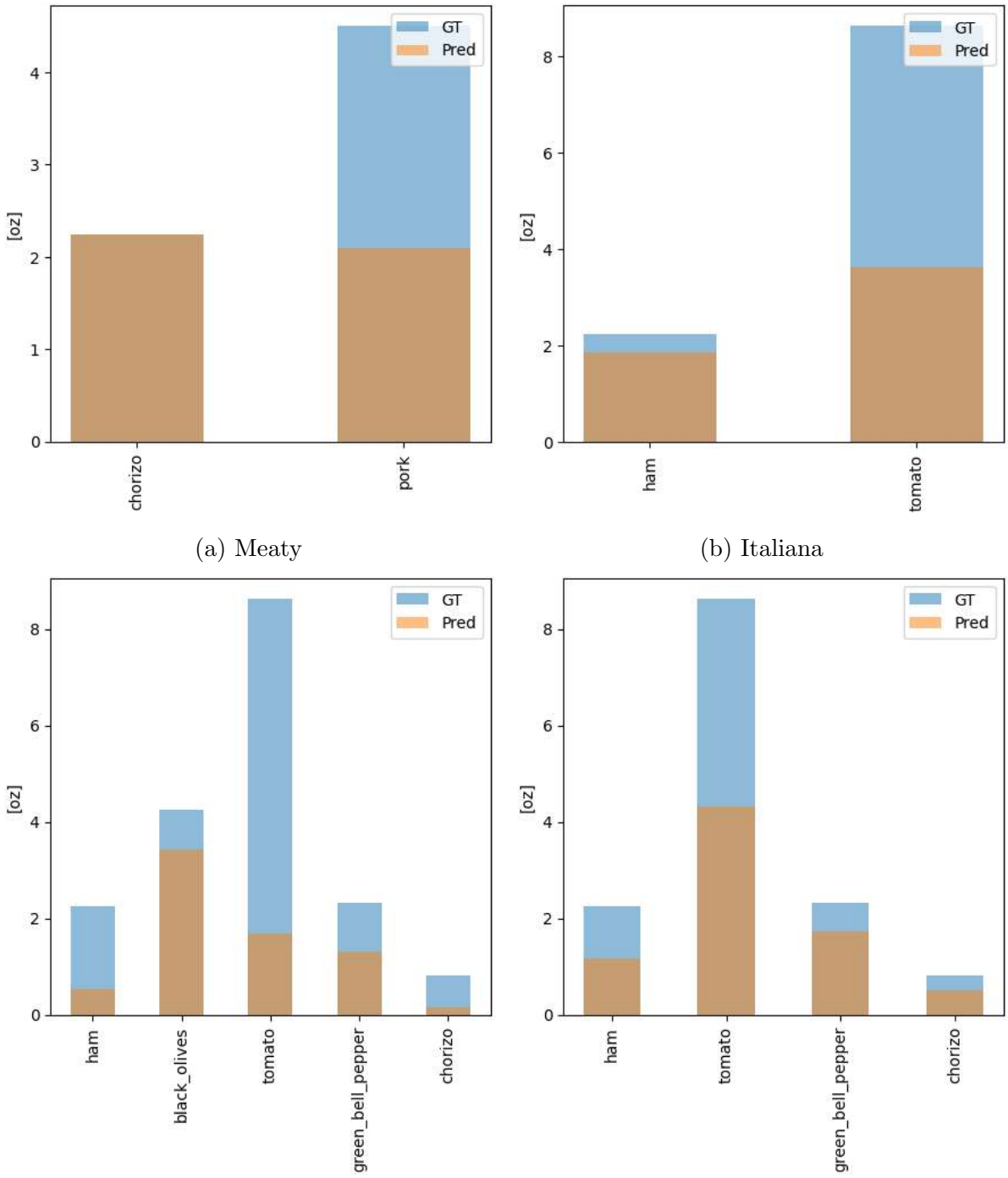
(a) Pepperoni

(b) Napolitana

(c) Milano

(d) American Meat

Figura 4.9: Porción promedio de cada clase en cada especialidad, parte 1



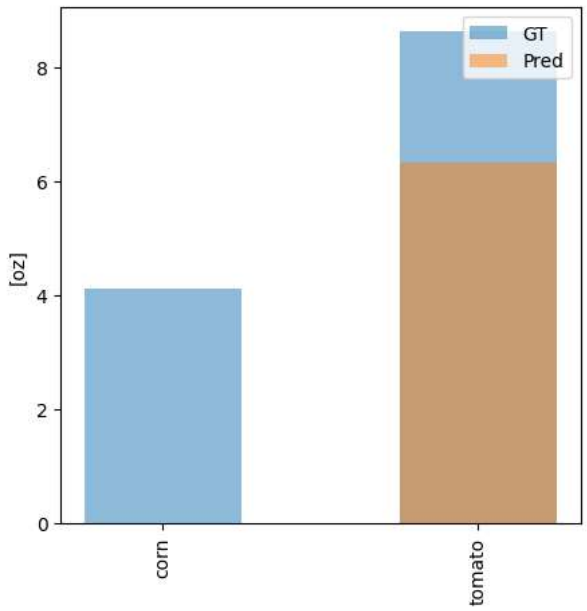
(a) Meaty

(b) Italiana

(c) Españolisima

(d) Española

Figura 4.10: Porción promedio de cada clase en cada especialidad, parte 2



(a) Veggie

Figura 4.11: Porción promedio de cada clase en cada especialidad, parte 3

En las Figuras anteriores podemos notar la clase *pork*, la cual se define como la suma de las clases *bacon* y *meat*. Finalmente, podemos ver dos tablas resumen, reflejando la porción promedio estimada en la Tabla 4.3, y la porción promedio de referencia en la Tabla 4.4.

	pepperoni	green_bell_pepper	ham	black_olives	bacon	tomato	corn	chorizo	meat
Pepperoni	2.118	-	-	-	-	-	-	-	-
Napolitana	-	-	-	-	-	6.494	-	-	-
Milano	-	-	2.030	3.394	-	2.729	-	-	-
American Meat	1.089	-	-	-	0.577	-	-	1.230	2.505
Meaty	-	-	-	-	0.387	-	-	2.25	1.677
Italiana	-	-	1.864	-	-	3.617	-	-	-
Españolisima	-	1.298	0.525	3.332	-	1.676	-	0.156	-
Española	-	1.736	1.159	-	-	4.330	-	0.518	-
Veggie	-	-	-	-	-	6.324	0	-	-

Tabla 4.3: Tabla resumen porción promedio estimada de cada clase en cada especialidad

	pepperoni	green_bell_pepper	ham	black_olives	bacon	tomato	corn	chorizo	meat
Pepperoni	2.25	-	-	-	-	-	-	-	-
Napolitana	-	-	-	-	-	8.625	-	-	-
Milano	-	-	2.25	4.25	-	8.625	-	-	-
American Meat	1.375	-	-	-	1.5	-	-	0.8125	2.9375
Meaty	-	-	-	-	1.5	-	-	2.25	2.9375
Italiana	-	-	2.25	-	-	8.625	-	-	-
Española	-	2.3125	2.25	-	-	8.625	-	0.8125	-
Españolisima	-	2.3125	2.25	4.25	-	8.625	-	0.8125	-
Veggie	-	-	-	-	-	8.625	4.125	-	-

Tabla 4.4: Tabla resumen porción promedio referencia de cada clase en cada especialidad

Capítulo 5

Análisis

5.1. Detectores de Objetos

Al momento de analizar los resultados de los cálculos presentados en la Tabla 4.1, podemos notar que el modelo *Yolov6* posee un mejor desempeño en todas las clases existentes, en comparación al modelo *Faster R-CNN*. Este comportamiento se observa con mayor detalle al comparar las salidas de estos modelos, visibles en las Figuras 4.3a y 4.3b, con el *groun truth* de la Figura 4.3c, donde se ven regiones donde no presentan detecciones de ingrediente alguno para el caso de *Faster-RCNN*. Este comportamiento se remarca al analizar las curvas PR por clase de las Figuras 4.1 y 4.2.

Dentro de estas figuras, junto a la Tabla 4.1, podemos diferenciar algunas clases que son difíciles de detectar en ambos escenarios, los que corresponden a las clases *bacon* y *corn*, y que se puede explicar por su tamaño relativo a los demás ingredientes presentes en las imágenes, sumado a una posible similitud de colores de otras clases mucho más reconocibles. Por otro lado, destacan algunas clases más diferenciables, como lo son *pepperoni*, *chorizo*, *ham*, y *black_olives*, debido a sus colores y tamaños relativos, sumado a que presentan formas estándar a lo largo de todas las imágenes.

Un caso curioso ocurre con la clase *tomato*, dado que resulta ser la más representada dentro del *dataset*, basándonos en la Figura 3.2, pero que, sin embargo, no destaca como una clase sencilla de detectar. Esto puede deberse a la similitud en color tanto a la salsa de tomate propio de las *pizzas*, como a la clase *pepperoni*, la cual es la segunda más representada, y sobre la cual los modelos se desempeñan de mejor manera.

En condiciones normales, nos decantaríamos a ciegas por el modelo *Yolov6*, pero aun bajo sus mejores resultados, se decide utilizar el modelo *Faster R-CNN*, dado que su rendimiento no dista en demasía con *Yolov6*, dado los visto en la tabla 4.1, pero además porque es el modelo arraigado en los *pipeline* de trabajo de la empresa donde se realizó esta experiencia, pudiendo de esta forma implementar más fácilmente la detección de porciones de ingredientes granulares en el entorno de trabajo. Se deja entonces como trabajo futuro el intercambio de estos modelos en dichos *pipelines* de trabajo.

5.2. Estimadores de Profundidad Monoculares

A partir de los resultados presentados en la Tabla 4.2, podemos observar que el modelo *BinsTransformer* presenta una ventaja, sobre el modelo *MIM*, presentando valores más próximos a 1 en las métricas δ_i, y valores más cercanos a 0 en las métricas restantes.

Paralelamente, podemos realizar una comparación visual entre los mapas de profundidad visibles en la Figura 4.5, donde vemos una mayor similitud, tanto en formas como colores, entre el mapa de profundidad entregado por *BinsTransformer*, y el *ground truth*, mientras que en el mapa entregado por *MIM* es difícil reconocer más de 2 colores a simple vista, sin mencionar que aparecen regiones anexas a la que nos interesa.

Aun así, podemos notar que ciertos ingredientes no tienen una forma notoriamente definida, lo que nos puede conllevar a unas variaciones importantes en los valores extraídos a partir de la máscara de profundidad. Basándonos en lo mencionado anteriormente, se decide utilizar el modelo *BinsTransformer* para generar los mapas de profundidad de esta experiencia.

5.3. Estimación de porción de ingredientes granulares

Si observamos la Figura 4.8, relativa a los volúmenes promedio obtenidos mediante la metodología descrita en la Subsección 3.6, se puede concluir que globalmente estamos ante resultados prometedores, donde vemos una diferencia porcentual del 32,8 %, entre los valores del *ground truth* y los entregados por los modelos.

Se puede notar que aquellas clases con peor desempeño son aquellas donde justamente los modelos de detección de objetos fallaban, es decir, las clases *bacon* y *corn*, donde si excluimos estas clases, pasamos desde una diferencia porcentual del 32,8 % al 17,6 %. Así mismo, vemos un desempeño decente en las clases restantes, presentando algunas sobreestimaciones de los volúmenes en las clases *pepperoni*, *ham*, *black_olives* y *chorizo*, a la vez que se encuentran subestimaciones en las clases *green_bell_pepper*, *tomato* y *meat*.

Estas fluctuaciones en las estimaciones podemos atribuírselas, en mayor parte, a las máscaras de profundidad predichas de cada ingrediente, ya que, como se mencionó en la Subsección anterior, estas no presentaban formas notoriamente definidas, lo que lleva a que estos ingredientes contengan una mezcla de valores de profundidad, aun cuando no hay presencia de solapamiento alguno, obteniendo entonces valores distintos a los esperados, tal y como se observa en la Figura 4.7.

Finalmente, a partir de las Figuras 4.9 y 4.10, podemos notar que el algoritmo propuesto tiende a subestimar las porciones de la mayoría de las clases a lo largo de todas las especialidades. Este comportamiento se puede explicar, por un lado, en que aquellas clases que se subestiman los volúmenes, se continúan subestimando en las porciones. Sin embargo, lo interesante es analizar aquellas clases en que se sobreestimaba el volumen, ya que aun pasando esto, se subestiman sus porciones, lo que se puede explicar en mayor medida a la cantidad de ingredientes encontrados por el modelo de detección, el cual detecta menos ingredientes que los que debería en las imágenes suministradas, afectando directamente a la suma total de los volúmenes de cada ingrediente.

Capítulo 6

Conclusiones

Durante el transcurso de la presente experiencia, se buscó un *aproach* que condujera a una estimación acertada de las porciones para distintos ingredientes granulares. El *aproach* propuesto tiene en consideración distintos ámbitos, como lo son el tipo de ingrediente, su profundidad relativa, así como el área neta que ocupa este dentro de la imagen. Considerando los alcances establecidos en la Sección 1.1, se puede afirmar que se lograron con éxito, y que existe un margen de mejora a considerar.

Se destaca además la creación, de forma manual, de un *dataset* sintético propio a partir de los datos etiquetados suministrados por *Kwali SPA*, empresa donde se originó la problemática a resolver, *dataset* que fue crítico a la hora de resolver la estimación de profundidad para los distintos ingredientes presentes en las imágenes, así como las herramientas utilizadas para crearla, dejándolas a disposición de la empresa para un futuro.

En relación con la etapa de entrenamientos de los distintos modelos, podemos mencionar que fue la que requirió mayor esfuerzo en ser completada. Esto, debido a distintos factores, entre los que se destacan los tratamientos que se debían realizar a los datos, la variación de distintos hiperparámetros, a fin de obtener los mejores resultados posibles, y pruebas fallidas en modelos *vanilla* puestos a prueba.

Paralelamente, podemos destacar los resultados obtenidos por los modelos entrenados, los cuales, al aplicar las métricas correspondientes, se obtuvieron valores cercanos con respecto a los presentados en el estado del arte con mejor desempeño, con valores de F_1 del $58,6\,\%$ obtenido versus un $61,74\,\%$ esperado, y errores volumétricos del $32,8\,\%$ obtenido versus $20,1\,\%$ esperado.

En cuanto a la metodología planteada, esta permitió cumplir exitosamente con el objetivo principal, desde obtención de los *bounding boxes* de cada ingrediente, pasando por la predicción del mapa de profundidad de la imagen, su posterior segmentación ingrediente por ingrediente, y finalmente la estimación de porción por cada clase de estos ingredientes.

Ahondando en los resultados obtenidos, podemos mostrar bastante confianza en que la metodología propuesta nos conduce a una solución válida de ser considerada para el problema general, dado que los volúmenes promedios estimados para cada ingrediente, no dista en demasía con los valores de referencia establecidos, por lo que, aunque los valores de las porciones estimadas no se acercan a los estándares que se desearían, muestran que con un modelo de detección de ingredientes con una mejor tasa de detección, mejorarían de forma radical los valores estimados de porción por cada ingrediente.

Finalmente, basándonos en lo mencionado anteriormente, podemos concluir que se cumplió con el objetivo general planteado de detectar y estimar ingredientes granulares dado una imagen. Dicho proceso se logró cumplir al seguir con los objetivos específicos establecidos, junto a una metodología estructurada y definida, lo que denota la importancia de establecer los pasos a seguir en cada uno de los experimentos en los que nos veremos insertos.

6.1. Trabajo futuro

En lo relativo al trabajo futuro, se consideran distintos tópicos basados en los problemas mencionados previamente, dentro de los que se encuentran:

- **Probar modelos *State-of-the-Art* futuros**, que permitan mejorar de sobremanera el desempeño mostrado por aquellos probados en esta experiencia.
- **Aumentar el volumen de datos**, a fin de una mayor abstracción por parte de los modelos, con especial énfasis en modelos basados en *Transformers*, recordando que necesitan grandes cantidades de datos para entrenar correctamente.
- **Etiquetados detallados de mapas de profundidad**, dado que un mayor detalle en estos conllevaría a una reducción en los errores en la estimación de estos mapas, y con ello, de cada ingrediente. Eso, debido a la utilización de un *dataset* sintético, sumado a todos los problemas que esto pueda acarrear.
- **Añadir más clases**, lo que permitiría utilizar la metodología de detección y estimación sobre un mayor número de ingredientes, ampliando el foco del objetivo de esta experiencia.
- **Generalización**, pudiendo resolver este problema no únicamente en el dominio de las *pizzas*, sino también sobre una mayor gama de preparaciones culinarias.

Bibliografía

[1] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*, pp. 267–285, Springer, 1982.

[2] P. Skalski, "Let's code convolutional neural network in plain numpy," 2020.

[3] M. Yani, S. Irawan, and C. Setianingsih, "Application of transfer learning using convolutional neural network method for early detection of terry's nail," *Journal of Physics: Conference Series*, vol. 1201, p. 012052, 05 2019.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[6] M. Carranza-García, J. Torres-Mateo, P. Lara-Benítez, and J. García-Gutiérrez, "On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data," *Remote Sensing*, vol. 13, no. 1, p. 89, 2020.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[9] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[11] A. Khazri, "Faster rcnn object detection," 2019.

[12] C. Li, L. Li, Y. Geng, H. Jiang, M. Cheng, B. Zhang, Z. Ke, X. Xu, and X. Chu, "Yolov6 v3.0: A full-scale reloading," 2023.

[13] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13733–13742, 2021.

[14] H. Zhang, Y. Wang, F. Dayoub, and N. Sunderhauf, "Varifocalnet: An iou-aware dense object detector," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8514–8523, 2021.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[16] Z. Gevorgyan, "Siou loss: More powerful learning for bounding box regression," *arXiv preprint arXiv:2205.12740*, 2022.

[17] S. Li, "deeplabv3plus on mapillary vistas." `https://github.com/parachutel/deeplabv3plus_on_Mapillary_Vistas`, 2018.

[18] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.*, "Segment anything," *arXiv preprint arXiv:2304.02643*, 2023.

[19] J. H. Z. Z. H. H. Y. C. Zhenda Xie, Zigang Geng, "Revealing the dark secrets of masked image modeling," *arXiv preprint arXiv:2205.13543*, 2022.

[20] Z. Xie, Z. Zhang, Y. Cao, Y. Lin, J. Bao, Z. Yao, Q. Dai, and H. Hu, "Simmim: A simple framework for masked image modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9653–9663, 2022.

[21] Z. Li, X. Wang, X. Liu, and J. Jiang, "Binsformer: Revisiting adaptive bins for monocular depth estimation," 2022.

[22] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] baeldung, "Intersection over union for object detection," 2022.

[25] R. Padilla, S. L. Netto, and E. A. Da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 international conference on systems, signals and image processing (IWSSIP)*, pp. 237–242, IEEE, 2020.

[26] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Advances in neural information processing systems*, vol. 27, 2014.

[27] J. Chen and C.-W. Ngo, "Deep-based ingredient recognition for cooking recipe retrieval," in *Proceedings of the 24th ACM international conference on Multimedia*, pp. 32–41, 2016.

[28] M. Bolaños, A. Ferrà, and P. Radeva, "Food ingredients recognition through multi-label learning," in *International Conference on Image Analysis and Processing*, pp. 394–402, Springer, 2017.

[29] J. Chen, B. Zhu, C.-W. Ngo, T.-S. Chua, and Y.-G. Jiang, "A study of multi-task and region-wise deep learning for food ingredient recognition," *IEEE Transactions on Image Processing*, vol. 30, pp. 1514–1526, 2020.

[30] Z. Yang, H. Yu, S. Cao, Q. Xu, D. Yuan, H. Zhang, W. Jia, Z.-H. Mao, and M. Sun, "Human-mimetic estimation of food volume from a single-view rgb image using an ai system," *Electronics*, vol. 10, no. 13, p. 1556, 2021.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[32] A. Graikos, V. Charisis, D. Iakovakis, S. Hadjidimitriou, and L. Hadjileontiadis, "Single image-based food volume estimation using monocular depth-prediction networks," in *International Conference on Human-Computer Interaction*, pp. 532–543, Springer, 2020.

[33] K. Okamoto and K. Yanai, "An automatic calorie estimation system of food images on a smartphone," in *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management*, pp. 63–70, 2016.

[34] R. Stojnic, R. Taylor, M. Kardas, and G. Cucurull, "Papers with code," 2018.