Decoupling Food Segmentation

Felipe Cisternas Alvarez.
Universidad Técnica Federico Santa Maria.
felipe.cisternasal@sansano.usm.cl.

Resumen

El Machine Learning ha sido uno de los mayores avances de los últimos tiempos, permitiendo analizar, automatizar y predecir data a escalas gigantescas. Este proyecto se basa en analizar imágenes de alimentos, para segmentarlas mediante el uso de Deep learning con modelos de Redes Neuronales y así clasificar y separar los distintos tipos de alimentos presentes un plato.

Palabras clave: Computer Vision, **Deep Learning**, **Imagen**, **Segmentación Semántica**, **Comida**, **Salud**

1. Introducción y Motivación

La salud y sobre todo la obesidad en estos últimos años ha ido en aumento debido a la gran cantidad de comida rápida y la excesiva cantidad de calorías en las mismas. Cuando alguien quiere bajar de peso, el factor mas decisivo es su alimentación, siendo mas influyente incluso que el ejercicio, es por esto que es muy importante saber que se esta consumiendo en cada comida. Mediante el análisis de la comida podemos obtener información muy útil y relevante, como la cantidad de calorías, la distribución de proteínas, carbohidratos, grasas, etc, para poder ayudarnos en la tarea que busquemos, ya sea bajar de peso, aumentar masa corporal, tener una vida mas saludable, podemos detectar cuando la comida se encuentra en descomposición o mal estado para prevenir enfermedades, entre otros. Es por eso que en este proyecto se investigó y desarrolló un modelo de Deep Learning implementado con Redes Neuronales con la arquitectura SOLO, para segmentación semántica de alimentos, proponiendo una serie de mejoras para la tarea planteada.

2. Problema

2.1 Definición del Problema

Para el caso de la clasificación de alimentos en un plato de comida, estamos frente a un problema de **detección y segmentación de objetos** en imágenes. La **segmentación semántica** es la tarea de clasificar y asignar partes similares de una imagen a una determinada clase, en específico asignar a cada píxel una clase en particular para que el grupo de píxeles conforme la detección del objeto en particular.

Figura 1: Segmentación Semántica de una foto de una casa con sus diferentes clases

Esto nos permite poder trazar los bordes de objetos y descomponer la imagen en los diferentes objetos que la componen, en este caso la segmentación semántica es clave para poder separar las distintas clases de alimentos, además de que al tener el área que ocupa determinada clase de alimento en el plato o la foto, se podría calcular aproximadamente la cantidad de comida y así mediante una base de datos alimenticia saber cuantas calorías aporta ese alimento.

Figura 2: Segmentación Semántica en un plato de alimentos

2.2 Estado del Arte Segmentación Semántica

El estado del Arte (SoTA) de la segmentación semántica, lo lideran modelos de **Redes Neuronales (Deep Learning)**, entre los cuales se encuentran:

- **U-Net** [10] :
  Es una red convolucional originalmente desarrollada para segmentación de **imágenes biomédicas**, como su nombre lo indica, la visualización de su arquitec-

tura tiene **forma de U** y se compone de **2 partes**, la izquierda, el **camino contractivo** y la derecha, el **camino expansivo**. El propósito de la contracción es capturar el **contexto de la imagen**, mientras que el propósito de la expansión es ayudar a mejorar la **precisión de la localización** de la segmentación. la arquitectura esta compuesta por capas de: **convolución, Max Pooling y ReLU**.

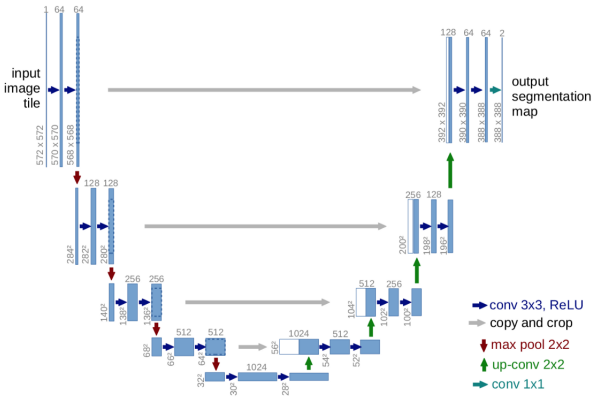


Figura 3: Arquitectura U-Net

- **DeepLab** [1] :
  Esta red se compone de una arquitectura Encoder-Decoder, la Versión 3 (V3) utiliza la arquitectura **ResNet-101** [4] pre-entrenada con el dataset **ImageNet** junto con **atrous convolutions**, una convolución donde el kernel utiliza dilataciones que omiten valores mediante el **dilation rate** que define el espacio que se saltara entre los valores.
  Como principal **Extractor de Características (Feature Extraction)**, Deeplab V3+ utiliza una versión modificada de la arquitectura **Xception** [2] como principal feature extractor.

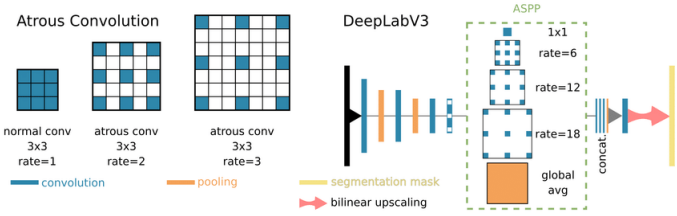


Figura 4: Atrous Convolutions

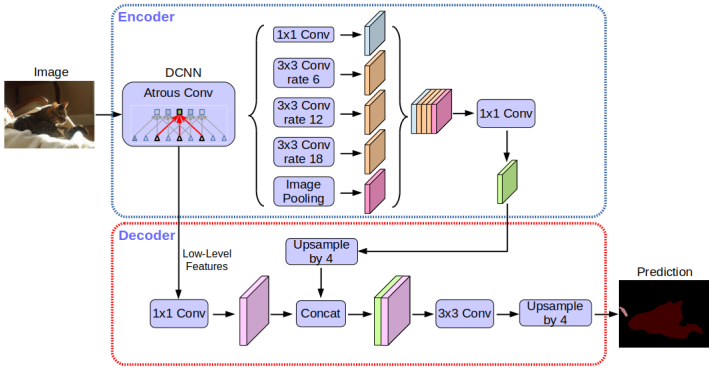


Figura 5: Arquitectura DeepLab

- **Mask R-CNN** [3] :
  Es una extensión de la arquitectura **Faster R-CNN**, la cual es una red convolución profunda que divide la imagen en regiones y utiliza un detector para clasificar cada región, Mask R-CNN utiliza segmentación y además utiliza **Bounding Boxes**, a cada región de interés se le aplica una máscara de segmentación y finalmente aplica una bounding box con el label correspondiente.

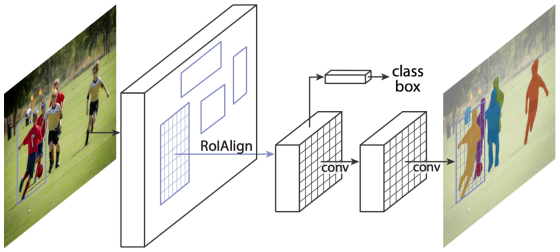


Figura 6: Arquitectura Mask R-CNN

- **FastFCN** [12] :
  Fast Fully Convolutional Network, en esta arquitectura se utiliza un módulo **JPU (Joint Pyramid Upsampling)**, que se utiliza para **sobremuestrear** y **aumentar la resolución** del **mapa de características (feature map)**, pasando de tener un mapa de baja resolución (low-resolution feature maps) a un mapa de características de mayor resolución (high-resolution feature maps) y por ende obtener mayor información. Este modulo reemplaza las **Dilated/Atrous Convolutions** ya que estas consumen mucho tiempo y memoria.

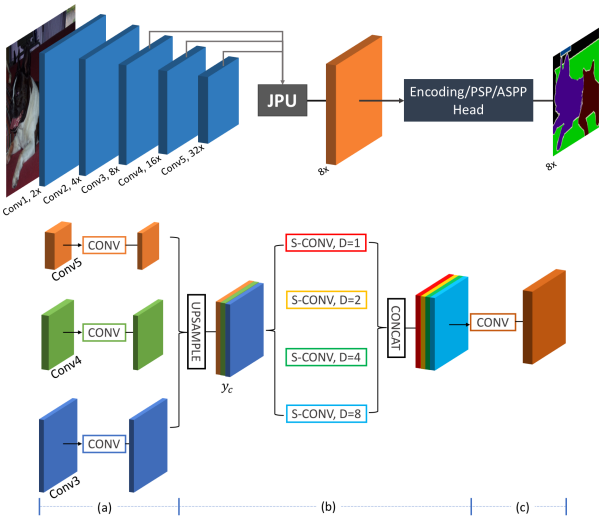


Figura 7: Aqruitectura FastFCN y Modulo JPU

El estado del Arte es muy variado en esta tarea porque dependiendo el dataset, una arquitectura da mejores resultados que otras, hay que tener en cuenta que la segmentación semántica tiene variados ámbitos de aplicación como:

- Reconocimiento de escritura (Handwriting Recognition)
- Fotos con modo retrato (portrait mode)
- conducción autónoma (Self-driving cars)

- diagnostico de exámenes médicos (Medical image diagnostics)

entre otras aplicaciones. Es por esto que cada tarea cuenta con distintas arquitecturas en el estado del arte, variando en cada caso sus hiperparametros, funciones de perdida y de optimización.

3. Estado del Arte en la clasificación de alimentos

Como se pudo observar existe una amplia variedad de arquitecturas para la segmentación semántica pero ahora el estudio se centrara en la tarea de reconocer y clasificar alimentos, para esto revisaremos las arquitecturas propuestas por el estudio realizado por la **universidad de Singapur.** [13].
Dentro de este estudio se analizaron diversas arquitecturas el estado del arte en la segmentación de alimentos, entre las cuales están:

- **CCNet:** Criss-Cross Attention Net [5], una red neuronal que aplica mecanismos de atención, en específico para cada píxel de la imagen aplica atención con las lineas de píxeles que lo cruzan, esta operación se aplica recurrente-mente para obtener un contexto general de toda la imagen, al aplicar criss-cross attention, la operación se realiza entre un píxel y ciertos píxeles de la imagen, reduciendo la cantidad de memoria que necesita el modelo y reduciendo la complejidad de las operaciones a realizar.

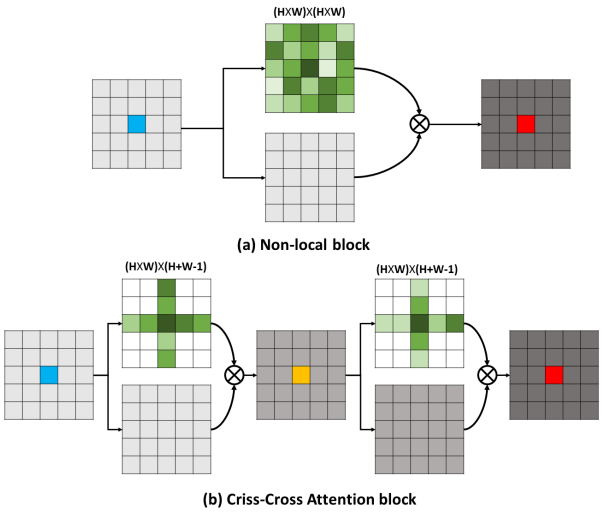


Figura 8: Criss-Cross Attention

- **FPN**: Feature Pyramid Network [6] es una arquitectura que se utiliza generalmente como extractor de características, es muy similar a la JPU, en este caso las convoluciones van reduciendo el tamaño de la imagen y capturando las características mas importantes de esta, para después hacer la operación inversa, aumentar el tamaño mediante las convoluciones para ir generando predicciones a distintos niveles de características y así generar una predicción general.

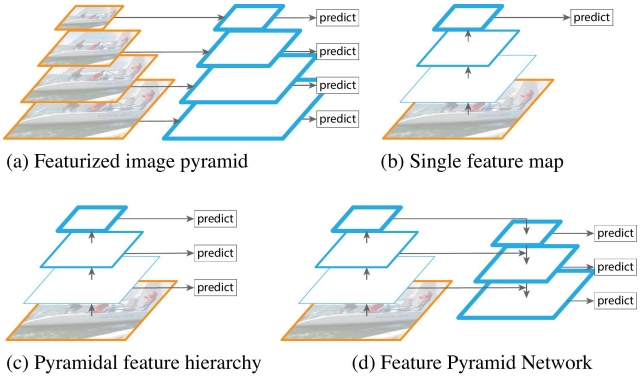


Figura 9: Feature Pyramid Network

- **SeTR**: Segmentation Transformer [14] es una arquitectura basada en **Transformers**, el encoder del transformer trata a una imagen como una secuencia de parches de imágenes a los cuales les aplica mecanismos de atención, para poder aprender características relevantes (Embeddings) según el contexto, una vez aprendidas las características con el encoder, el decoder del transformer se utiliza para reconstruir la imagen con la resolución original, con este enfoque no se pierde resolución espacial.

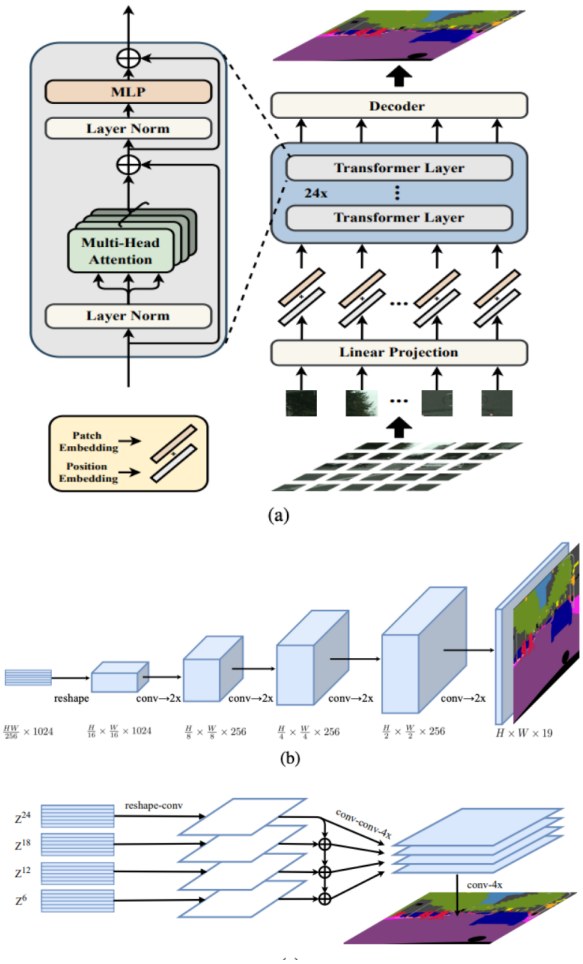


Figura 10: Segmentation Transformer

4. Data

4.1 Obtención de la Data

Para este proyecto se utilizara un Dataset de imágenes de segmentación de alimentos, compuesto por **39.962** imágenes con **76.491** anotaciones repartidas a través de **498** clases distintas.
la data se obtuvo del challenge **Food Recognition Benchmark 2022** a través de la pagina de AICrowd

4.2 Características de la Data

la data se compone de mas de cincuenta mil imágenes de comida, donde la dimensión promedio de cada imagen es de **653x651 píxeles**, con una desviación estándar de **295 píxeles para la altura** y **287 píxeles para el ancho**, los casos bordes de la data tienen un mínimo de **183 píxeles de altura** y **182 píxeles de ancho**, y de máximo tienen **4096 píxeles de altura** y **4068 pieles de ancho**.
Como existen tantas clases no todas van a tener la misma proporción de muestras en el dataset, es por esto que se aplicaran métodos de **Data Augmentation** mediante rotaciones y transformaciones geométricas para balancear las clases y que el modelo aprenda mejor la generalización.

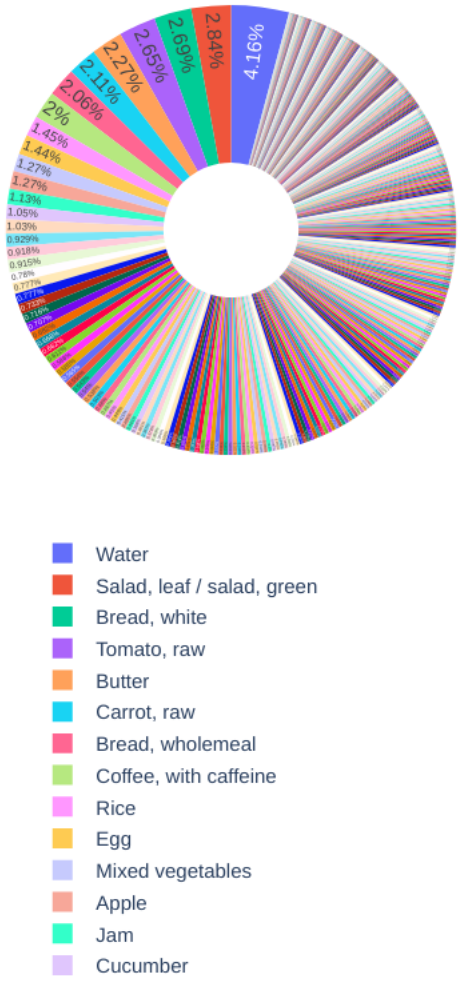


Figura 11: RoundPlot con el porcentaje de imágenes de cada Clase del Dataset y Labels de las clases con mayor cantidad de imágenes

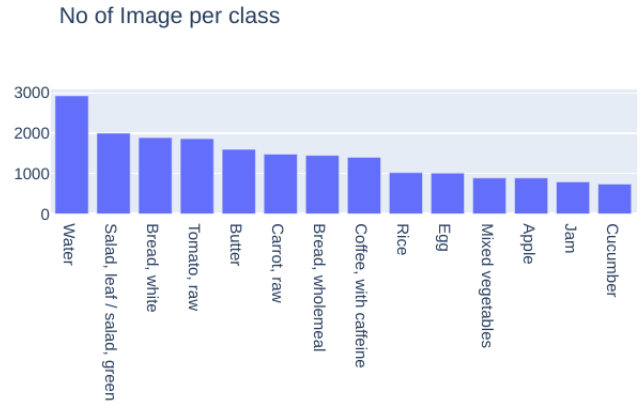


Figura 12: Histograma de las clases con mayor numero de muestras del Dataset de alimentos

5. Arquitectura SOLO

Segmenting Objects by Locations es una arquitectura para segmentación semántica y es la arquitectura que se utilizara en esta investigación, es una arquitectura muy simple y ligera que da buenos resultados. La idea central de SOLO es dividir el problema de segmentación en 2 sub-problemas simultáneos, la clasificación y la segmentación, es por esto que se divide la imagen en una grilla uniforme, si el centro de un objeto cae dentro de alguna celda de la grilla, esa celda se encargara de predecir la categoría y segmentar el objeto. SOLO divide la imagen en una grilla uniforme para que la reciba como input un modulo FCN (Fully Convolutional Network), el output de este modulo FCN se divide en dos ramas, donde una rama se encargara de predecir la categoría del objeto en cuestión y la otra rama por su parte se encargara de detectar la mascara de segmentación de cada objeto, cada una de estas ramas utiliza una FPN (Feature Pyramid Network).

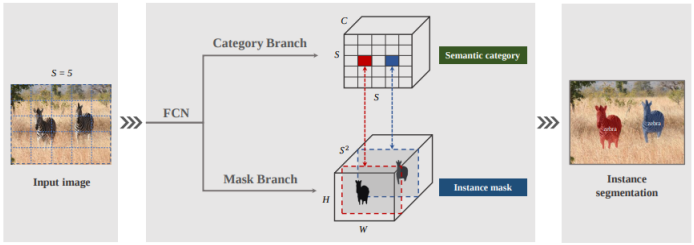


Figura 13: SOLO Network

5.1 Semantic Category

Para cada celda de la grilla, SOLO predice un vector C-dimensional que indica las probabilidades semánticas de clase donde C es el numero de clases. La grilla se divide en SxS celdas, por lo tanto el espacio de salida de esta rama sera SxSxC, este diseño esta basado en el supuesto de que cada celda de la grilla SxS pertenece a una sola clase, a su vez perteneciendo solo a 1 categoría semántica.

5.2 Instance Mask

En paralelo con la predicción de la categoría semántica, cada celda positiva, (es decir que esa celda contenga parte del objeto) generara una mascara de instancia. Como la

grilla se divide en SxS celdas, existirán S^2 mascaras de instancias en total, estas mascaras se codificaran como tensores de tres dimensiones, tal que la salida de las mascaras será de H_1xW_1xS^2, también se aplica **CoordConv**, una técnica para obtener invarianza espacial, para se agregan las coordenadas de los píxeles que se normalizan entre [-1,1] y este tensor es concatenado al tensor de features, de esta forma se agrega información espacial a la convolución, obteniendo invarianza espacial con un tensor de HxWx$(D+2)$, donde los últimos dos canales son las coordenadas x e y.

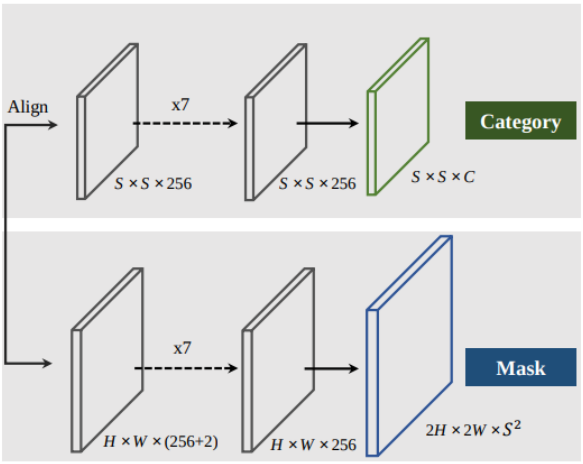


Figura 14: SOLO Heads para una imagen con 256 canales como ejemplo

5.3 SOLO Learning

Para la rama de la predicción de la categoría, la red le asigna al objeto una probabilidad de categoría por cada celda SxS en la grilla, una celda es considerada positiva si cae dentro de la región del centro de cualquier mascara de verdad entre las distintas clases, de otra forma es considerada negativa, esta técnica llamada **Center Sampling** es altamente efectiva y se utiliza en trabajos recientes relacionados con la detección de objetos, dado el centro de masa (C_x, C_y), el largo w y la altura h de la mascara de verdad, la región central es controlada por constantes $\epsilon : (C_x, C_y, \epsilon w, \epsilon h)$, en el caso de solo $\epsilon = 0.2$ y se generan en promedio 3 celdas positivas por cada mascara de verdad de cada clase. Además para cada etiqueta de las clases se genera una segmentación binaria por cada mascara para cada celda positiva.

5.4 Losses

La función de perdida de SOLO se define como

$$L = L_{cate} + \lambda L_{mask}$$

donde L_{cate} es la **Focal Loss**

$$FL(P_{true}) = -\alpha(1 - P_{true})^{\gamma} log(P_{true})$$

para la predicción de las categorías y L_{mask} es la **Dice Loss**

$$DL = 1 - \frac{2 * \sum_{x,y}(p_{x,y} \cdot q_{x,y})}{\sum_{x,y} p_{x,y}^2 + \sum_{x,y} q_{x,y}^2 + \epsilon}$$

para la predicción semántica de las mascaras, donde $p_{x,y}$ y $q_{x,y}$ hacen referencia al valor del píxel en las coordenadas (x, y) tanto de la mascara de verdad como de la predicción.
Para finalizar el optimizador que utiliza SOLO es **SGD** (Stochastic gradient descent), con un **Learning Rate** = 0.001, **Momentum** = 0.9 y un **Weight decay** = 0.0001, además se utilizan parámetros de **Warm Up** para que la red comience poco a poco a incrementar el learning rate durante determinadas iteraciones, esto se realiza para evitar el *Early Overfitting*, en SOLO el **Warmup** es de tipo lineal con un **Warmup iters** = 500 y **Warmup Ratio** = 1/3, eso significa que el learning rate va a comenzar con un tercio del valor original e ira aumentando durante las 500 primeras iteraciones hasta llegar al learning rate original .

5.5 Inferencia

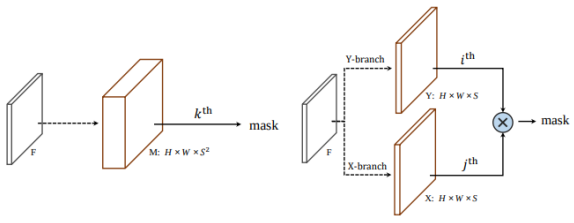
la inferencia de solo es bastante simple, dada una imagen como input esta pasa a través de la FPN y se obtienen los valores de la predicción de las categorías y las mascaras correspondientes, después se utiliza un threshold de 0.1 para filtrar las predicciones con baja confidencia, después se seleccionan las 500 mascaras con valores mas altos y se pasan a través del operador **NMS** (non-maximum-suppression) para seleccionar las mascaras con mayor probabilidad, después se utiliza un threshold de 0.5 para convertir las mascaras a mascaras binarias, por ultimo se calcula la **Maskness** que representa la calidad y confidencia de la predicción de la mascara y cada valor de la clasificación se multiplica por la maskness como una calificación final para la confidencia de la clasificación y segmentación.

5.6 Decoupled SOLO

Decoupled SOLO es una mejora de la red ya que muchas veces las predicciones son redundantes ya que los objetos se encuentran dispersos en la imagen.
En Decoupled SOLO el tensor de salida $M \in \mathbb{R}^{HxWxS^2}$ es reemplazado por dos tensores de salida, $X \in \mathbb{R}^{HxWxS}$ e $Y \in \mathbb{R}^{HxWxS}$ correspondiente a cada eje respectivamente, así el espacio de salida se reduce de HxWxS^2 a HxWx2S. la predicción de la mascara del objeto se define como la multiplicación de los dos canales

$$m_k = x_j \otimes y_i$$

donde x_j e y_i son el j-esimo e i-esimo canal de X e Y después de pasar por la operación sigmoide



(a) Vanilla head (b) Decoupled head

Figura 15: Decoupled Heads

Esta versión desacoplada tiene mejores resultados y además es mas eficiente y requiere una cantidad consid-

erable menos de memoria de la GPU, siendo la principal característica por la que se eligió esta arquitectura.

6. Innovación y Mejoras Propuestas

Para la mejora de la red SOLO se propusieron una serie de cambios basado en la literatura y papers recientes para obtener mejores resultados

6.1 AdamW

Para comenzar vamos a cambiar el optimizador, si bien SGD es un buen optimizador, con el pasar del tiempo han salido nuevos y mejores optimizadores, el optimizador mas famoso hasta la fecha es **Adam**, pero en esta ocasión vamos a presentar una versión modificada de este optimizador llamado **AdamW** [7] donde la gran diferencia es que desacopla el paso de *Weight Decay*, la motivación sobre este cambio es que se analizo que el weight decay en Adam no es el mismo que en SGD por lo que esta versión AdamW resuelve este problema, así como **Decoupled SOLO** es una mejora para SOLO, en esta investigación se propone el uso de **Decoupled Adam**, un optimizador reciente que varias otras arquitecturas comienzan a utilizar como es el famoso modelo *Megatron Turing* [11] un modelo de lenguaje de 530 billones de parámetros creado por una colaboración entre Nvidia y Microsoft.
En tareas de **Computer Vision** como clasificación y segmentación de imágenes, SGD obtenía mejores resultados que Adam, por lo que en 2019 Ilya Loshchilov y Frank Hutter proponen AdamW, esto basándose en que la regularización L_2 con gradientes adaptativos no es equivalente a weight decay ya que al comparar Adam, los pesos de los gradientes se regularizan menos de lo que deberían si se utilizara SGD con weight decay. Otro punto que evidencian los autores es que la regularización L_2 no es efectiva en Adam ya que como mencionamos anteriormente esta regularización esta mal implementada por eso es que SGD obtiene mejores resultados en tareas de clasificación. Weight Decay por otro lado si es efectiva en Adam y acá hay que hacer la distinción que en el caso de Adam Weight decay $\neq$ Regularización L_2. El weight decay óptimo depende del numero total de batches de actualización de los pesos, es decir que mientras mas largo numero de batches menor es el weight decay óptimo. Por ultimo Adam se puede beneficiar en gran manera por el uso de un **scheduled learning rate multiplier** (multiplicador de la tasa de aprendizaje programado) global como puede ser *cosine annealing*. Como se puede observar la implementación de la regularización L_2 en Adam es

$$g_t = \nabla f(\theta_t) + w_t \theta_t$$

donde w_t es el porcentaje de cambio del weight decay en un tiempo t.
En cambio AdamW ajusta el termino weight decay para que aparezca en la actualización del gradiente

$$\theta_{t+1,i} = \theta_{t,i} - \eta(\frac{1}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t + w_{t,i}\theta_{t,i}), \forall t$$

Este pequeño cambio aplica adecuadamente weight decay en Adam.

Algorithm 1 SGD with L_2 regularization and SGD with decoupled weight decay (SGDW), both with momentum

given initial learning rate $\alpha \in \mathbb{R}$, momentum factor $\beta_1 \in \mathbb{R}$, weight decay/L_2 regularization factor $\lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow$ SelectBatch$(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\eta_t \leftarrow$ SetScheduleMultiplier(t) ▷ can be fixed, decay, be used for warm restarts
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + \eta_t \alpha \boldsymbol{g}_t$
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \boldsymbol{m}_t - \eta_t \lambda \boldsymbol{\theta}_{t-1}$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow$ SelectBatch$(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow$ SetScheduleMultiplier(t) ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Figura 16: SGD y AdamW

6.2 AMSGrad

AMSGrad [9] es un método de optimización estocástico que busca arreglar y mejorar la convergencia con los optimizadores basados en Adam, AMSgrad utiliza el máximo de los gradientes pasados v_t en vez de la media exponencial para actualizar los parámetros. Generalmente Adam adapta automáticamente un learning rate separado para cada parámetro en el problema de optimización, esto genera una limitación ya que si bien es bueno que Adam disminuya el learning rate cuando se acerca al óptimo, esto puede generar que se aumenten las iteraciones necesarias para llegar a ese óptimo lo cual es malo. AMSGrad es una extensión de Adam que mantiene un máximo del vector del segundo momento y lo utiliza para corregir el bias del gradiente para actualizar los parámetros, en vez de utilizar el vector del momento en si mismo, esto ayuda a prevenir la convergencia prematura (**Premature Convergence**) que ralentiza la optimización.

$$m_t = \beta_1 m_{t_1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t_1} + (1 - \beta_2) g_t^2$$
$$\hat{v_t} = max(v_{t-1}, v_t)$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

6.3 Mixed Precision Training

Precisión Mixta o **Mixed Precision** [8] es una técnica donde los pesos, las activaciones y los gradientes son almacenados en formato IEEE de punto flotante con precisión media (16 bits necesarios para almacenar un numero) en vez de utilizar precisión simple (32 bits necesarios para almacenar un numero), esto quiere decir que se utilizan menos bits para almacenar estos valores, provocando que la red pierda un poco de información pero gane velocidad de computo y reduciendo el tamaño en memoria que utiliza ya que los valores de los pesos, activaciones y gradientes utilizan la mitad del tamaño original. Se crea una copia

de los pesos que acumulan los gradientes después de cada paso del optimizador, esta copia es redondeada a **Half-precision format** durante el entrenamiento, se procede a escalar la función de perdida apropiadamente para poder trabajar este nuevo formato. Utilizando este enfoque se puede reducir la memoria y el tiempo necesarios para entrenar redes neuronales profundas hasta casi el doble del valor original.

6.4 Data Augmentation

aumento de datos o **Data Augmentation** es una técnica para aumentar artificialmente la cantidad de datos con los que se va a entrenar la red. al aumentar el numero de datos y variedad de estos mismos en nuestro dataset, conseguimos que la red pueda aprender de una forma mas óptima el problema requerido, reduciendo el **overfitting**.

6.4.1 Gaussian Noise

Aplicar distintos tipos de ruido a la data de entrenamiento es una técnica muy utilizada en tareas de *Computer Vision*. Cada vez que el modelo recibe una imagen, a esta se le aplica un ruido de forma aleatoria, haciendo que cada imagen sea distinta a la anterior incluso si recibiera la misma imagen dos veces seguidas. Existen muchos tipos de ruido y transformaciones que se le pueden aplicar a las imágenes, desde cambios de color, inversiones, etc, en este caso se aplico un ruido de tipo Gaussiano a los datos.

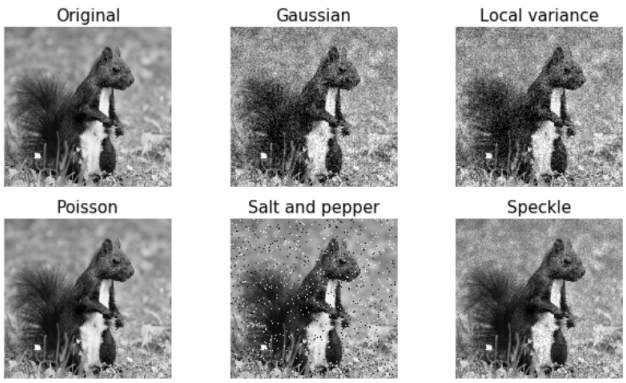


Figura 17: Distintos tipos de ruidos aplicados a imágenes

Para SOLO se utilizo un ruido gaussiano aleatorio para cada imagen mediante una pipeline en el entrenamiento.

6.4.2 Spatial Transforms

En imágenes se puede aplicar transformaciones espaciales, como lo son la rotación, reflexión, recortes, etc, para aumentar la cantidad de datos disponibles.

Figura 18: Spatial Transforms, de una imagen se pueden obtener 9 sub-imágenes o mas para el entrenamiento

Para la red SOLO se utilizaron varias transformaciones, en especifico se utilizo un *random_flip*=(0.3, 0.3, 0.3) eso quiere decir que se voltea la imagen horizontal, vertical y diagonalmente según los valores indicados, por lo que cada transformación tiene 30 por ciento de probabilidad de ocurrir, por otro lado también se escalo la imagen a distintas dimensiones entre las cuales están 852, 512, 480, 448, 416, 384 y 352 píxeles.

6.5 Gradient Clipping

Cuando estamos trabajando con redes muy profundas, funciones de activación *Squash* (funciones acotadas en un rango de valores) y sobre todo con mixed precision training, comenzamos a tener problemas al calcular los gradientes, ya que estos comienzan a explotar o desvanecerse (**vanishing/exploding gradient problem**) esto quiere decir que tomar valores muy altos o muy bajos los cuales dificultan y empeoran el aprendizaje de la red. Para solucionar estos problemas se utiliza una técnica llamada **Gradient Clipping** que consiste en recortar y acotar a cierto rango el valor que puede tomar los gradientes calculados por la red para mejorar la optimización, para que cuando el gradiente sea muy alto o muy bajo, este pueda ser reajustado a un valor mas razonable. este método se puede realizar de varias formas, una de ellas es simplemente recortar el gradiente antes de la actualización de parámetros, otra forma es recortar la norma $||g||$ del gradiente g antes de la actualización de los parámetros.

$$\text{if } ||g|| > v, \text{ then } g \leftarrow \frac{gv}{||g||}$$

Para la implementación de SOLO se utilizo un gradient clipping con $Max\_norm = 35$ y $norm\_type = 2$, lo que significa que la norma máxima de los gradientes es 35 y se utiliza la norma 2 de vectores.

7. Métricas de Desempeño

7.1 MaP

Mean Average Precision es una métrica muy popular para medir el **Accuracy** en tareas de detección de objetos. Average precision calcula como su nombre lo dice la precisión media para el **Recall** entre [0,1]. El Recall mide cuan buenas fueron las predicciones positivas.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

8. IoU

Intersection over union o también conocido como **Jaccard Index** mide la superposición entre 2 bounding boxes, se utiliza esta métrica para medir cuanto se suponer pone la predicción con la anotación real.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

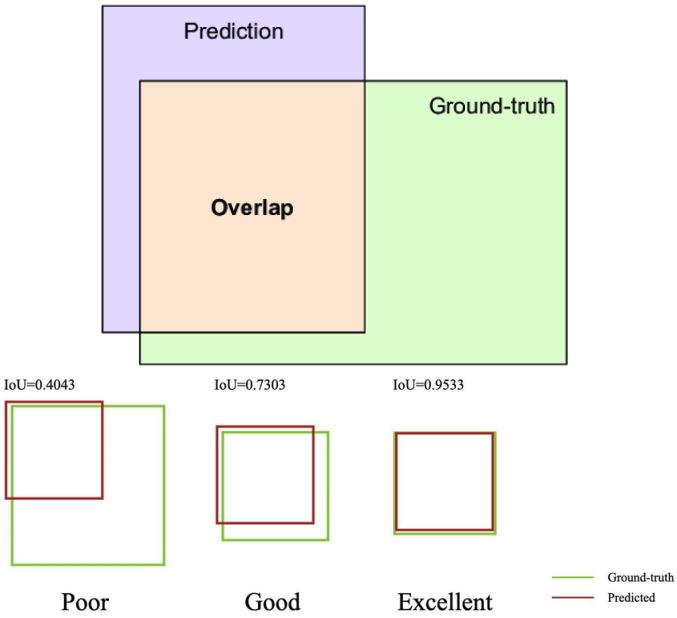


Figura 19: IoU

8.1 Dice Coefficient

También conocido como **F1 Score**, **Dice Coefficient** es 2 veces el área de superposición dividido por el numero total de píxeles en las bounding boxes.

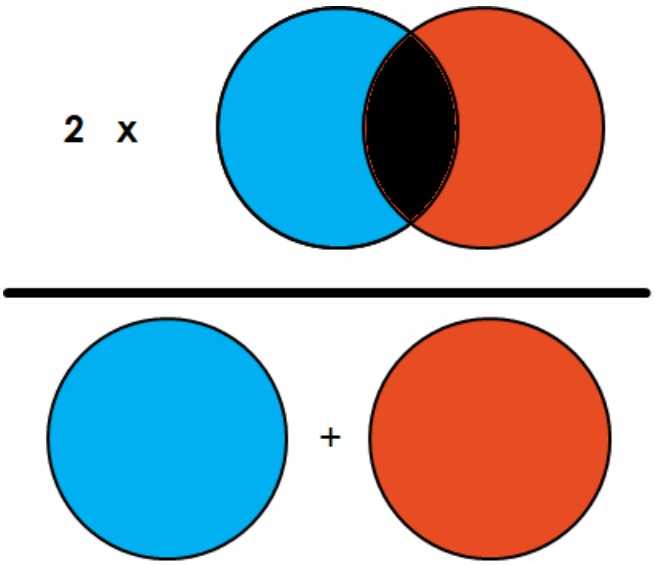


Figura 20: Dice Coefficient

Dice Coefficient es muy parecido a IoU, ambos están en el rango [0,1] y están correlacionados positivamente.
Para evaluar los distintos modelos se utilizo mAP, ya que la segmentación tiene áreas muy precisas, no es un simple cuadrado como las bounding boxes para *Objetc Detection*, si no que detecta todo el contorno del objeto por lo que al calcular IoU y Dice Coefficient no entrega resultados tan expresables como lo es mAP.

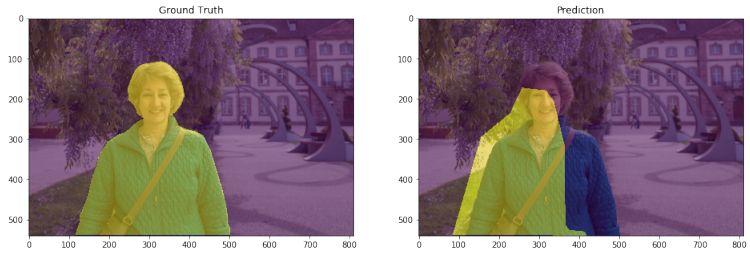


Figura 21: IoU en Segmentación, se puede observar como las áreas generadas no son simples cuadrados si no formas mas complejas

9. Entorno de Desarrollo y Parámetros de entrenamiento

9.1 Hardware

Todos los entrenamientos se realizaron en la misma maquina, entrenando la red SOLO variando los parámetros propuestos en la innovación y mejora.

- **GPU:** RTX 3070 Mobile, 8GB VRAM y 5120 CUDA Cores.
- **Procesador:** Ryzen 7 5800H, 8 Nucleos y 16 Hilos a 3.2 Ghz.
- **RAM:** 16GB a 3200 Mhz.
- **SO:** Linux Pop!_OS 22.04.
- **Software:** Mmdetection, Pytorch y Python.

9.2 SOLO

Como arquitectura se utilizo **decoupled_solo_light_r50_fpn_3x** implementado en Mmdetection, un software open source para detección de objetos basado en Pytorch.

- **Epochs:** 36
- **Iteraciones por epoch:** 6650
- **Channels:** 256
- **Stacked Convs:** 4
- **Strides:** (8,8,16,32,32)
- **Grids:** (40,36,24,16,12)

9.3 Optimizadores

Para los **optimizadores** tanto AdamW, AdamW con AMSGrad y SGD, se utilizaron los mismos parámetros comunes, cabe destacar que hay parámetros únicos de cada optimizador como lo es el **Momentum** para SGD y **eps** para AdamW y sus variaciones.

- **Learning Rate** = 0.001
- **weight_decay** = 0.0001
- **Momentum** = 0.9 $\parallel$ *Momentum en SGD*
- **eps** = $1e^{-4}$ $\parallel$ *epsilon en AdamW*

9.4 FocalLoss

- **Gamma** = 2.0 ‖ *gamma utilizado en FocalLoss*
- **Alpha** = 0.25 ‖ *alpha utilizado en FocalLoss*
- **loss_weight** = 1.0 ‖ *peso de la función de perdida*
- **reduction** = mean ‖ *el método para reducir la función de perdida*

9.5 DiceLoss

- **loss_weight** = 3.0 ‖ *peso de la función de perdida*
- **eps** = $1e^{-3}$ ‖ *epsilon para evitar divisiones por cero*
- **naive_dice** = False ‖ *utilizar Naive Diceloss*
- **reduction** = mean ‖ *el método para reducir la función de perdida*

10. Resultados

Como se menciono anteriormente cada modelo tanto la versión original de SOLO, como la versión con AdamW y la versión de AdamW + AMSGrad se entrenaron durante 36 epochs, que eran las epochs predeterminadas de la versión original.

10.1 funciones de Perdida

Vamos a comparar las distintas funciones de perdida a lo largo de las iteraciones en el entrenamiento, tanto la general como la de cada cabezal de las dos ramas de la arquitectura SOLO, donde *loss* hace referencia a la función de perdida general.

$$L = L_{cate} + \lambda L_{mask}$$

por otro lado *loss_cls* hace referencia a L_{cate} y *loss_mask* hace referencia a L_{mask}

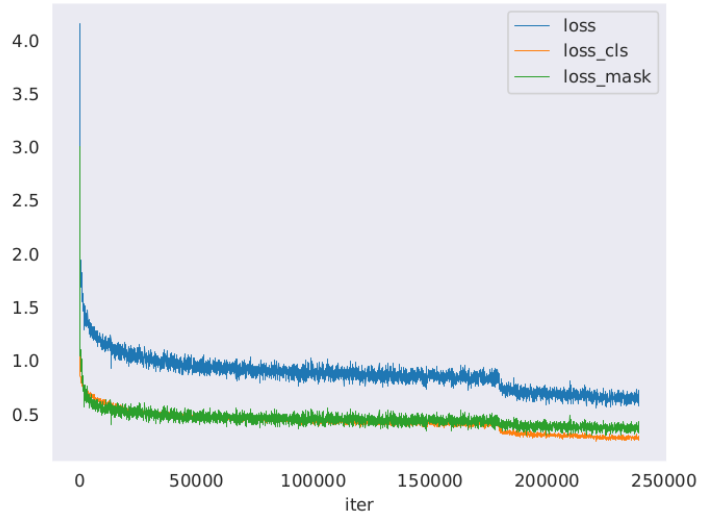


Figura 22: Funciones de perdida para la versión SOLO original

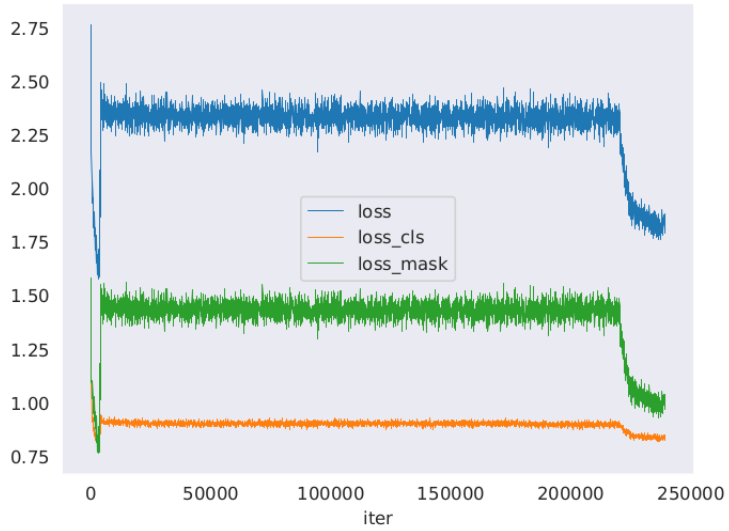


Figura 23: Funciones de perdida para la versión SOLO con optimizador AdamW

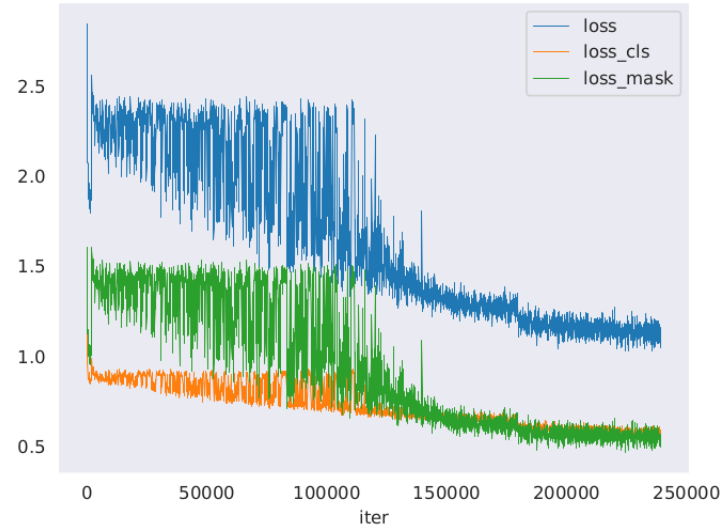


Figura 24: Funciones de perdida para la versión SOLO con optimizador AdamW con norma AMSGrad

10.2 mAP

Para comparar el mAP de cada modelo, vamos a comparar el average precision a distintos niveles de threshold de IoU, exactamente a 0.5, 0.75 y también compararemos el mAP según el tamaño de los objetos, tanto grandes (Large) como medianos (Medium) y no habrán pequeños (Small) ya que la comida en las fotos abarca gran porción de la imagen.

Model	mAP	mAP_{50}	mAP_{75}	mAP_m	mAP_l
SOLO	25.1	33	28.3	9.4	27.1
AdamW	0	0	0	0	0
AMSGrad	2.1	3.3	2.2	0.2	2.2

Tabla 1: Tabla Comparativa de mAP para los distintos modelos entrenados.

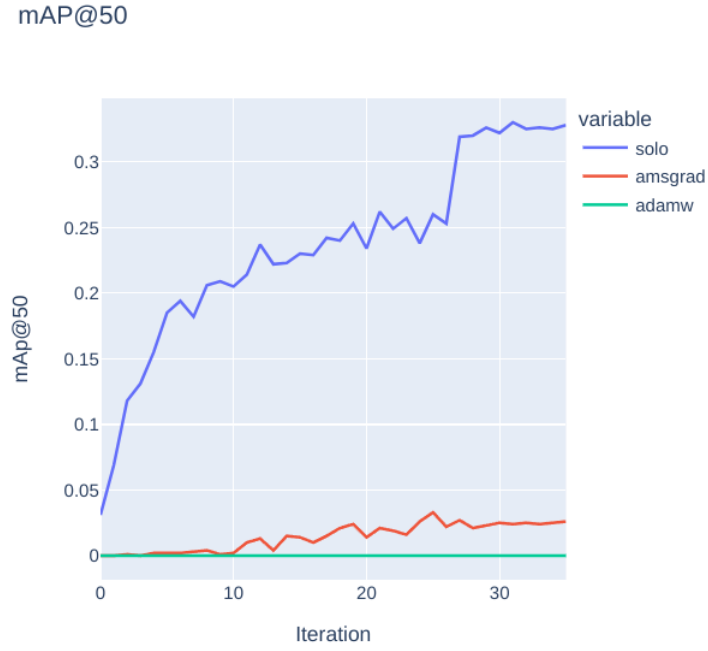


Figura 26: mAP_m y mAP_l Scores

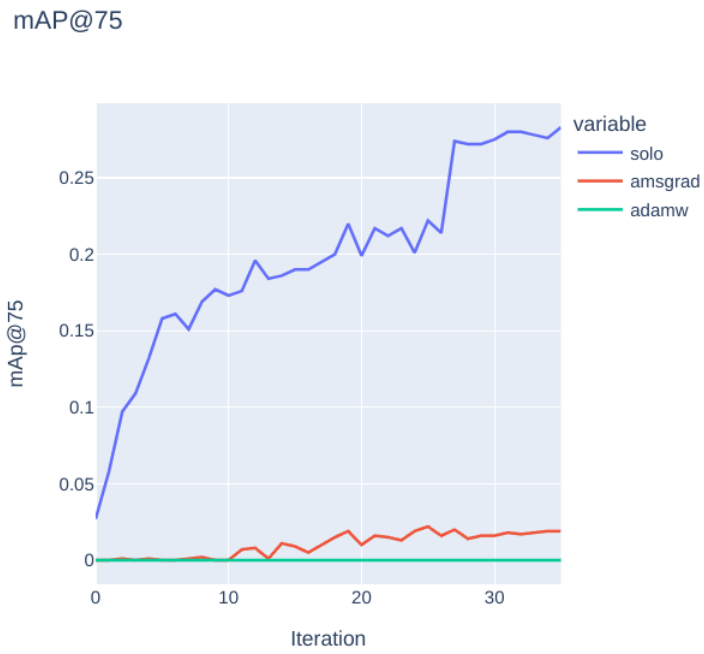


Figura 25: mAP, mAP@50 y mAP@75 Scores

11. Inferencias

Para las inferencias se mostraran a continuación fotos con la clasificación y seguntino hecha por cada modelo, estarán en el mismo orden siendo el primer modelo SOLO-Vanilla, después AMSGrad y por ultimo AdamW.

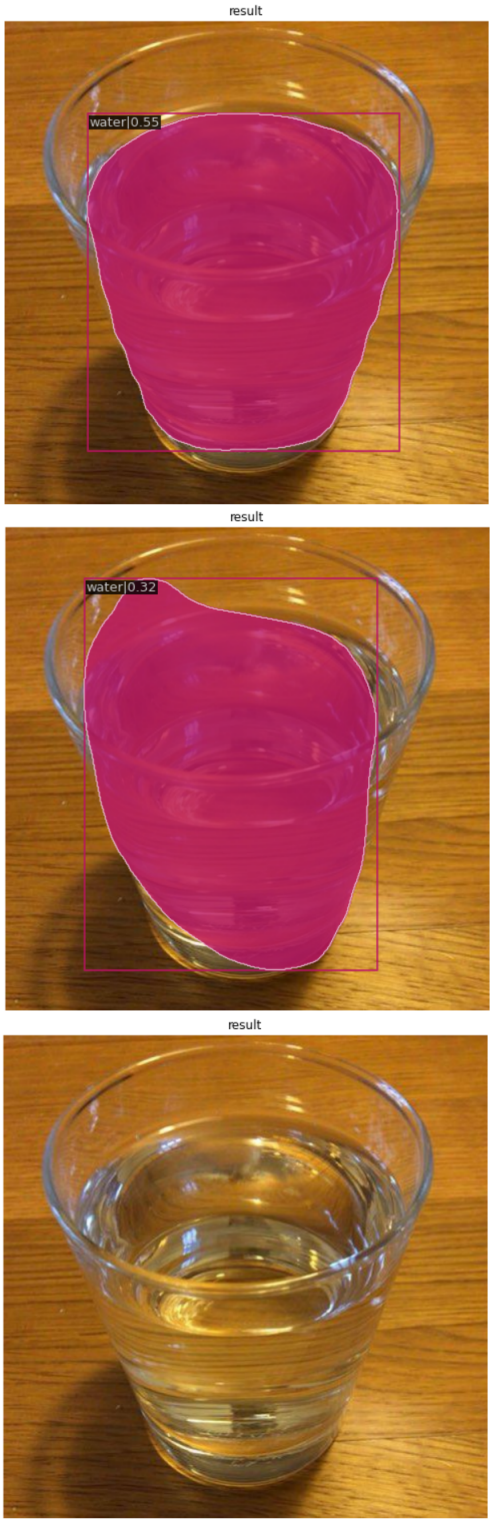


Figura 27: Test sobre una imagen de un vaso de agua para los modelos SOLO, AMSGrad y AdamW

Se puede observar como la mejor inferencia es la del Modelo SOLO, por otra parte AdamW no es capaz si quiera de reconocer y segmentar el vaso de agua.

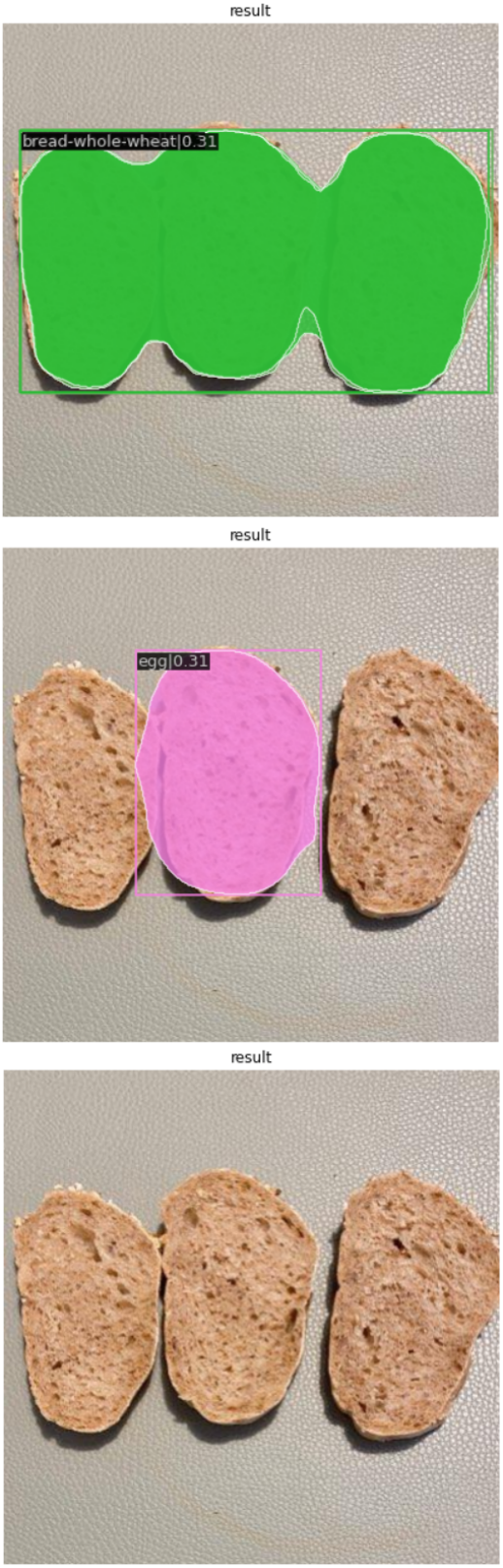


Figura 28: Test sobre una imagen de panes para los modelos SOLO, AMSGrad y AdamW

En esta inferencia se puede notar como AMSGrad confunde los trozos de pan con la clase de huevos, mientras que AdamW otra vez no es capaz de reconocer y segmentar los alimentos, siendo SOLO en su versión **Vanilla** (Original) el modelo que mejor desempeño presenta a la hora de clasificar y segmentar alimentos.

12. Conclusiones

Como se pudo observar en la sección de Resultados la incorporación del optimizador AdamW para la arquitectura SOLO no es de gran ayuda, de hecho es todo lo contrario, da peores resultados en comparación al optimizador SGD, pero cabe destacar que los experimentos se realizaron a **36 Epochs**, un numero bajo en el mundo del DeepLearning, quizás con un numero mayor de epochs el modelo pueda aprender de mejor manera y quizás superar a SGD con AdamW. Por otra parte cabe resaltar como la norma **AMSGrad** mejora los resultados al aplicar AdamW en segmentación de alimentos con la red SOLO, si bien al principio la función de perdida es bastante inestable, a la mitad del entrenamiento comienza a dar muy buenos resultados.

13. Trabajo Futuro

La idea de este proyecto es poder crear un modelo que pueda detectar y clasificar los distintos tipos de alimentos presentes en una imagen mediante un enfoque de **Computer Vision** con una simple foto proveniente por ejemplo de un celular. Este proyecto tiene muchas áreas de expansión siendo la principal el aumentar el tiempo de entrenamiento del modelo, otras formas pueden ser el poder contar las calorías presentes en la imagen de un plato, para así llevar una cuenta mas precisa de lo que consumimos a lo largo del día, si bien existen métodos de calcular calorías estas se hacen manualmente ingresando el tipo de alimento, la cantidad, etc, con este modelo SOLO, mediante el área de segmentación se podría **calcular la cantidad (en gramos) de alimento** y así mediante consultas a una base de datos alimenticia, **calcular la ingesta calórica de un plato de comida**. Otros trabajos futuros trabajos pueden realizarse tales como: **agregar mas clases de alimentos**, **detectar si la comida se encuentra en buen estado o en descomposición**, entre otros.

14. Bibliografía

Referencias

[1] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2016.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2016.

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Zilong Huang, Xinggang Wang, Yunchao Wei, Lichao Huang, Humphrey Shi, Wenyu Liu, and Thomas S. Huang. Ccnet: Criss-cross attention for semantic segmentation, 2018.

[6] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016.

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2017.

[8] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2017.

[9] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, 2019.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[11] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2019.

[12] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yizhou Yu. Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation, 2019.

[13] Xiongwei Wu, Xin Fu, Ying Liu, Ee-Peng Lim, Steven C. H. Hoi, and Qianru Sun. A large-scale benchmark for food image segmentation, 2021.

[14] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip H. S. Torr, and Li Zhang. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers, 2020.