Segmentación de imágenes usando la arquitectura U-net

Carlos Garavito[1]*, Juan Sebastián Cortés[2]

Resumen

Se presenta la arquitectura de redes convolucionales U-net, la cual es usada para realizar segmentación semática de imágenes. Se da cuenta de la implementación mediante el uso de pytorch y se aplica a un conjunto de datos de imágenes de ciudad con 35 categorías diferentes, evidenciando que con 100 epocas de entrenamiento, la red es capaz de predecir todas las clases con una precisión del 72%.

Keywords
Machine learning — Deep learning — Convolutional neural networks — Image segmentation

[1, 2] *Escuela de ingenieria, ciencia y tecnología; Universidad del Rosario; Bogotá, Colombia.*

***Corresponding author**: carloss.garavito@urosario.edu.co

Introducción

En procesamiento de imágenes, usualmente se resuelven los denominados problemas de clasificación y localización de objetos. El método de segmentación semántica aborda justamente el problema de clasificación de objetos, pero la hace a nivel de píxel individual. De esta manera, además de identificar una clase, es posible identificar los píxeles específicos que pertenecen a esta. En la imagen 1 se observa la comparación entre las tareas antes descritas.

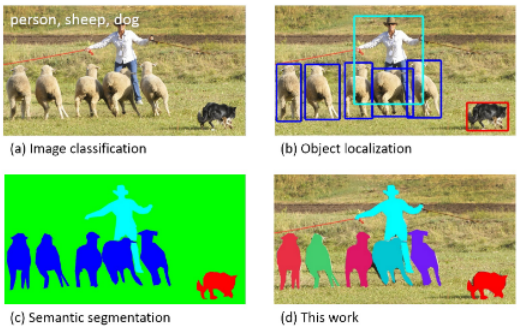

Figura 1. Comparación entre clasificación, localización y segmentación de objetos en imágenes. [1]

Algunas de las arquitecturas convolucionales para realizar segmentación en imágenes son U-net, FastFCN, Gated-SCNN, DeepLab y Mask R-CNN [2]. Las aplicaciones de este tipo de soluciones son diversas, algunos ejemplos son su uso en vehículos autónomos, robótica y sistemas de detección de daños [3]. Dado que una de las arquitecturas mas usadas es la de U-net, el objetivo de este trabajo es el de implementar esta arquitectura para la segmentación de diferentes clases de objetos en fotografías de una ciudad. En la primera parte, se muestra el fundamento teórico de la red; en la segunda parte se comparten la metodología usada y finalmente se discuten los resultados obtenidos.

1. Marco teórico

En contraste con una red neuronal, una red convolucional es capaz de recibir en la entrada tensores. Esto es, es posible manipular información que tiene más grados de dimensionalidad, como es el caso de las imágenes que pueden ser representadas como tensores de dimensión $[C, A, L]$, donde C corresponde al canal que puede ser R, G ó B, A el ancho y L el largo de la imagen, como se muestra en la imagen 2.

Figura 2. Canales de color en una imagen. [4]

En particular, en los procesos de entrenamiento de redes convolucionales es usual realizar este proceso por lotes de datos. De esta manera, al tensor de imágenes inicial se le incluirá una dimensión adicional que corresponderá al número del lote

en el cual se esta trabajando. Así, la imagen será ahora un tensor de 4 dimensiones, $[B,C,A,L]$, donde B hace referencia al batch o lote de entrenamiento. Una forma intuitiva de entender este tratamiento con tensores se puede visualizar en la imagen 3, en donde cada imagen es un cubo y cada batch de imágenes es un vector de cubos.

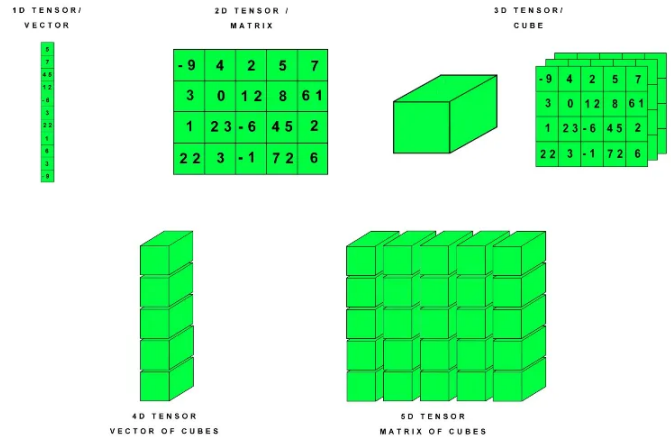

Figura 3. Visualización gráfica de tensores de distinto rango [5]

.

Ahora bien, aprovechando el tratamiento tensorial de los datos de entrada, la red convolucional toma su nombre debido a que a los tensores de entrada, y a lo largo de la profundidad de la red, se aplican funciones denominadas kernels de convolución que modifican las dimensiones del tensor y capturan caracteristicas de los datos de entrada. Una representación de un kernel se muestra en la figura 4.

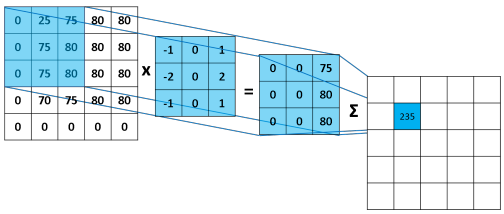

Figura 4. Ejemplo de una convolución aplicada a un tensor de rango 2 [6].

Así una red convolucional típica, consiste en diferentes etapas de convolución hasta finalizar con una etapa de clasificación lineal, como se observa en la figura 5.

U-net

La arquitectura de red U-net es una arquitectura de redes neuronales convolucionales y esquema se muestra en la figura 6. Se puede observar que visualmente tiene forma de U, precisamente por eso su nombre. Esta red tuvo sus primeras aplicaciones en clasificación de imágenes médicas con clasificación binaria. Sin embargo, su uso se ha extendido a

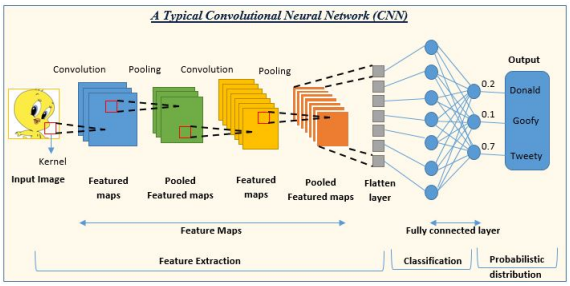

Figura 5. Representación de una red convolucional típica [7]

diversas ramas y también con la posibilidad de clasificación multiclase.

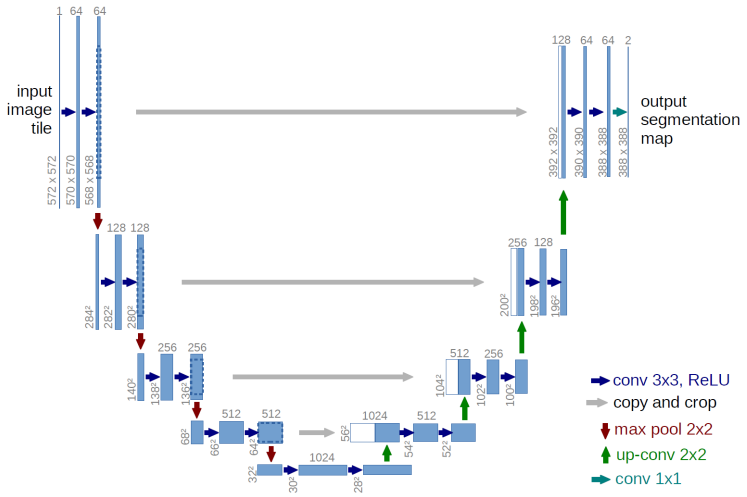

Figura 6. Arquitectura u-net [8]

.

Observe que en la arquitectura, mientras se avanza en la primera parte, el tamaño de la imagen se va reduciendo a la mitad pero la cantidad de propiedades se duplica. Esto es, hasta alcanzar un volumen de $30 \times 30 \times 1024$ en la mitad de la arquitectura.Varios autores denominan a esta primera parte la etapa de codificación. En la segunda etapa de la red, se observa que ocurre lo contrario, las dimensiones de la imagen se duplican y sus características se reducen a la mitad. Sin embargo, en esta etapa note que hay una concatenación representada con una linea de color gris. Esto es, en la etapa de decodificación, se usa compara información proveniente de la etapa de codificación con la etapa de decodificación. Así, observe que la etapa de decodificación finaliza con un volumen de $2 \times 388 \times 388$, en donde la dimensión igual a 2 representa las clases de clasificación. De tal manera, que si fuera un problema de más clases, esta dimensión sería igual al numero de clases. Este es justamente el escenario de la aplicación que se presenta en este informe.

2. Métodos

La metodología para realizar este proyecto se resumen de la siguiente manera:

- **Hardware** Uso de GPU con límites de versión gratuita de Google Colab.
- **Software** Framework pytorch.
- **Data** Acceso libre por publicación en huggin face pub.

Antes de programar la red, dentro del esquema de pytorch es necesario crear una clase `ataset y \verb` ataloader. La primera, permitirá el cargue de las imágenes en forma de tensores y la segunda, permitirá usar de forma amigable iteradores sobre lotes de entrenamiento. En particular, ambas clases deben ser construidas.
La estrategia para la programación de la red, consistió en identificar etapas secuenciales de la arquitectura. Así, según la representación gráfica en 6, cabe resaltar las siguientes etapas:

Double convolution Note que las flechas azules indican una convolución con un kernel $3x3$, con padding y stride iguales a uno. Se denominará double convolution a dos convoluciones de esta naturaleza. Observe que en particular, una double convolution esta presente al inicio y al final de la arquitectura.

Down convolution De forma análoga, se observa que en el segundo nivel de la red hay una etapa de max pool $2x2$, seguido de una doble convolución. Es de resaltar que la doble convolución de esta etapa tiene parámetros similares a la mencionada anteriormente, en donde lo diferente es el número de canales. Finalmente, se resalta que esta etapa de Down convolution se repite cuatro veces a lo argo de la red.

Up convolution Luego de llegar al último nivel de profundidad en la red al finalizar las secuencias de down convolution, hay una etapa de up-conv seguida de una doble convolución. De manera análoga al caso anterior, esta etapa de Up convolution se repite y nivel a nivel, varia el número de canales de salida.

3. Resultados

La implementación del notebook de Colab puede ser consultada en enlace `https://colab.research.google.com/drive/16f7lwNyWiZVVCuI5lGso_-_6FG_WQ2jy?usp=sharing`. Al hacer el entrenamiento con un conjunto total de 500 imágenes, con un conjunto de entrenamiento que corresponde al 80 % del total, y usando 100 epochs, se encuentra un *accuracy* $= 72\,\%$ sobre el conjunto de validación. En la imagen 7 se puede evidenciar el resultado de la clasificación, en donde cada clase se distingue por un color diferente.

Uno de los inconvenientes encontrados al entrenar la red, sucedió al transformar las imágenes en tensores para el tratamiento con pytorch. Para esto se uso la función ○ensor(), la cual transforma una imagen en un tensor normalizado. Se pudo dar cuenta que las imagenes de etiquetas no deben ser normalizadas. El conjunto de datos de las imagenes de etiquetas, muestra que cada clase está etiquetada desde el número 0 hasta el 34. Así, considere el caso en donde existe una imagen de etiqueta en la cual la clase máxima es la número 23. La función ○ensor() normalizará el tensor dividiendo el valor del pixel sobre dicho máximo. Ahora, considere otro caso en donde la clase máxima en otra imagen es la número 30. Note que para una misma clase los valores normalizados la clase serán diferentes. Esto es, si en ambos ejemplos existe la clase 10, luego de aplicar ○ensor(), en el primer caso será $10/23$ y en el segundo $10/30$. Por tal motivo, las imagenes etiqueta no deben ser normalizadas y en lugar de ello, se puede usar la función $PILToTensor()$.

4. Conclusiones

Se observa que la arquitectura u-net es un modelo robusto que puede ser entrenado con un conjunto de datos moderado y que no exige una gran cantidad de recursos computacionales, toda vez que en el trabajo desarrollado se pudo obtener una precisión del 72 %.

Referencias

[1] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[2] Katherine (Yi) Li Derrick Mwiti. Image segmentation: Architectures, losses, datasets, and frameworks, 2022.

[3] Priya Dwivedi. Semantic segmentation — popular architectures, 2019.

[4] The click reader. The convolution/pooling operation for rgb images.

[5] Giomuntor. ¿que es un tensor en deep learning?

[6] mlnotebook. Convolutional neural networks - basics, 2017.

[7] Saily Shah. Convolutional neural network: An overview, 2022.

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

Figura 7. Resultados obtenidos para un batch de datos del conjunto de validación. Cada color representa una clase diferente.