**INSTITUTO TECNÓLOGICO DE LA PAZ**

**DIVISIÓN DE ESTUDIOS PROFESIONALES**

**Reporte semanal de residencia semana 03**

**“Reconocimiento y seguimiento de objetos en entornos controlados”**

**CARRERA**

**Ingeniería en Sistemas Computacionales**

**PRESENTA**

**17310793 – Eloy Antonio Clemente Rosas**

**La Paz, Baja California Sur, México, 12 de Febrero del 2021**

# Índice

Contenido

[Índice 2](#_Toc95588138)

[Introducción 3](#_Toc95588139)

[Desarrollo 4](#_Toc95588140)

[Explicación de código. 9](#_Toc95588141)

[Conclusiones 17](#_Toc95588142)

# Introducción

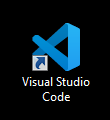
El presente documento contiene las actividades, y aprendizajes realizados durante la segunda semana en mi residencia profesional. Esta semana fue un poco más teórica que la anterior, enfocando principalmente la atención en lo que son las redes neuronales convolucionales, de las cuales se explicara más adelante dentro del documento.

Para programar en el entorno previamente preparado para entrenar redes neuronales en la GPU, fue necesario primeramente preparar un IDE que sea igual de potente. Para esto Visual Studio Code fue ideal por la gran flexibilidad que nos ofrece.

También fue necesario dar un paso más adelante con la detección de objetos que nos otorgaba una red neuronal convolucional, en donde fácilmente lograba identificar un objeto, y ahora poder lograr identificar dos o más objetos en una sola imagen, para esto la segmentación semántica sale a relucir. A continuación explicare un poco más en código y en definición.

# Desarrollo

**Preparación del entorno**

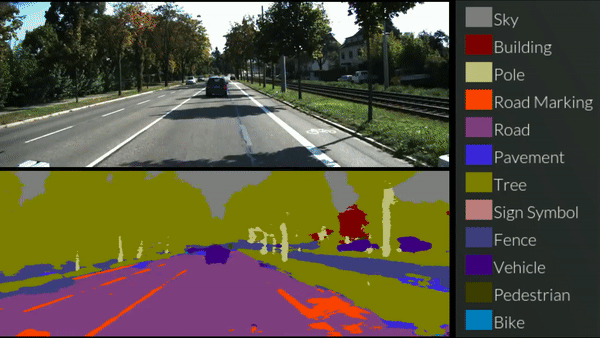
Empezando por la instalación de Visual Studio Code. Para instalar Visual Studio Code en Ubuntu ejecutamos el siguiente comando en Bash, es decir, la ventana de comandos de Linux.

sudo snap install --classic code # or code-insiders

Una vez instalado es posible correr los programas en Python que mostrare y explicare más adelante.

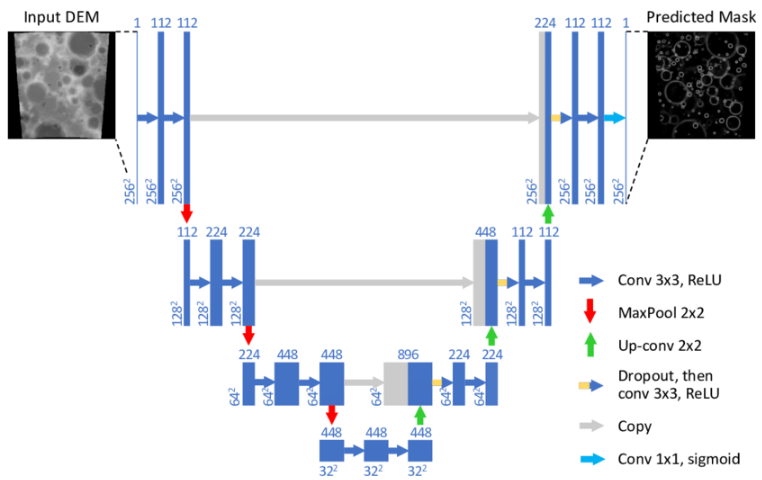
**Segmentación semántica**

La Segmentación semántica consistente en clasificar todos y cada uno de los píxeles en una imagen. Como anteriormente he trabajado en la tarea de clasificación que consiste en asignar una etiqueta a una imagen en particular, en la tarea de segmentación se tiene que asignar una etiqueta a cada pixel produciendo *mapas de segmentación*, imágenes con la misma resolución que la imagen utilizada a la entrada de nuestro modelo en la que cada pixel es sustituido por una etiqueta.

El objetivo que se logra como resultado es una imagen como la siguiente:

El fin es obtener las categorías que clasifica cada conjunto de pixeles.

Es aquí donde entra la arquitectura UNet, en la que tenemos el encoder que es la parte de la izquierda y el decoder que es la parte de la derecha con las convoluciones traspuestas que se encargan de ir aumentando los mapas intermedios. En cada etapa del decoder no solo entra la salida de la capa anterior sino también la salida de la capa correspondiente del encoder. De esta manera la red es capaz de aprovechar mucho mejor la información a las diferentes escalas.



**Detección de objetos**

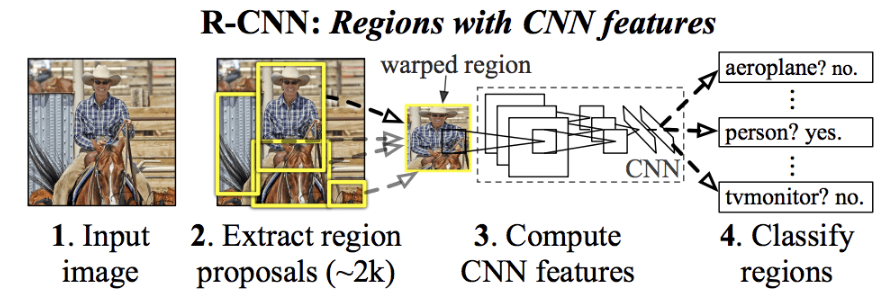
Ahora bien, es necesario explicar un poco sobre la detección de objetos y como se logra esta.

Existen dos grandes grupos de arquitecturas utilizadas para la detección de objetos:

* Detectores de dos etapas: En una primera etapa, la red neuronal propone las cajas en las que cree que se encuentran los objetos. En una segunda, el modelo clasifica los objetos dentro de las cajas propuestas. Estos modelos son más precisos pero computacionalmente más caros, lo que limita su aplicabilidad en aplicaciones en tiempo real.
* Detectores de una etapa: Estos modelos dan las cajas y clasificaciones a la vez, en una sola etapa. Para ello, utilizan un conjunto de cajas pre-definido. Estos modelos son menos precisos pero muy eficientes, por lo que se utilizan en aplicaciones en tiempo real.

Detectores de dos etapas:

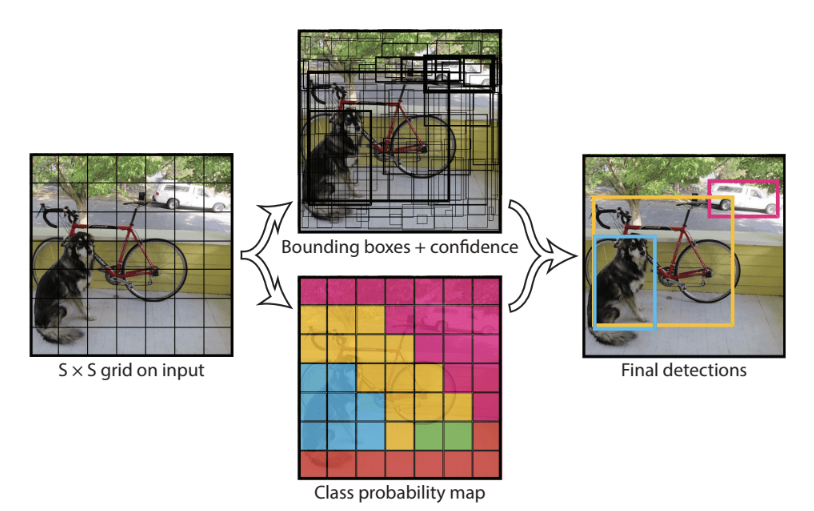
Entre estos detectores destacan los conocidos como la familia R-CNN. Fueron los primeros detectores en incorporar redes neuronales convolucionales, mejorando considerablemente los resultados obtenidos hasta la fecha con otros algoritmos. El primer modelo de la familia, R-CNN, utilizaba una primera etapa en la que se proponen las cajas (originalmente no usaba una red neuronal) y luego una CNN como las que ya conocemos para clasificar los objetos en cada caja.



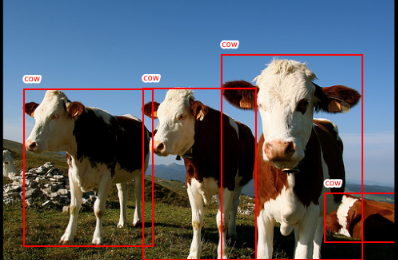
Mejoras a este algoritmo se propusieron en los años consecuentes, por ejemplo el modelo *Fast R-CNN* utiliza un conjunto de cajas predefinido siendo más eficiente.

**Detectores de una etapa**

Entre los detectores de una etapa, destacan el modelo YOLO (you only look once) que divide la imagen de entrada en una malla pre-definida. Cada celda es entonces responsable de predecir las cajas y las categorías a la vez. Mejores a este modelo se implementaron en YLOv2 y YOLOv3, mejorando tanto su eficiencia como precisión.

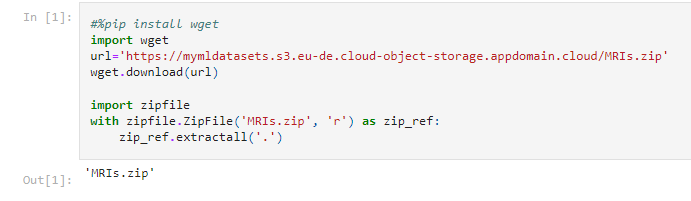


El modelo SSD es uno de los más utilizados, ya que representa un buen compromiso entre prestaciones y eficiencia (utilizado sobretodo en entornos con bajos recursos computacionales y aplicaciones en tiempo real como teléfonos móviles y IoT). Utiliza un conjunto de cajas pre-definidas, y durante el entrenamiento el modelo refina y clasifica cada una de estas cajas a diferentes escalas.



# Explicación de código.

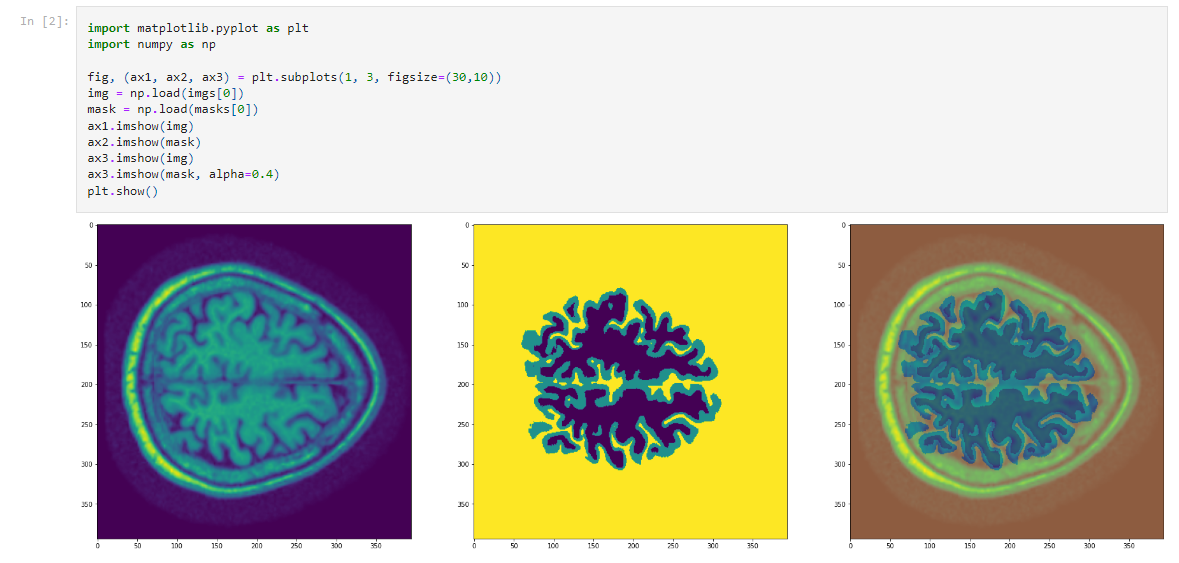
Podemos descargar un conjunto de imágenes de MRIs usando el siguiente enlace. En esta parte del código se descarga un compilado de MRI (Imagen por resonancia magnética) cerebral con sus correspondientes máscaras de segmentación, para detectar la materia gris y blanca. Determinar la cantidad de ambas así como su evolución en el tiempo para un mismo paciente es clave para la detección temprana y tratamiento de enfermedades como el Alzheimer. Con este código entonces se descarga y se descomprime.

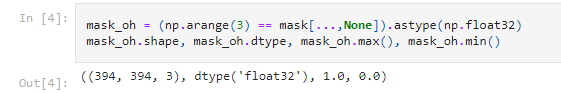


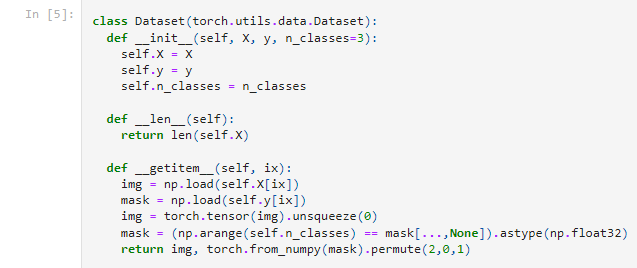
En el siguiente código se extraen el conjunto de imágenes con sus respectivas máscaras y se almacenan dentro las variables.



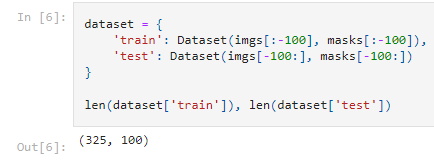
Ahora para mostrar una imagen se debe convertir desde arreglo Numpy a una forma gráfica, ploteando el conjunto de datos de que se almacena, aquí solo se está mostrando una imagen cerebral y su respectiva mascara.

 Nuestras imágenes tienen 394 x 394 píxeles, almacenadas como arrays de NumPy (que podemos cargar con la función *np.load*). Ya están normalizadas y en formato float32.

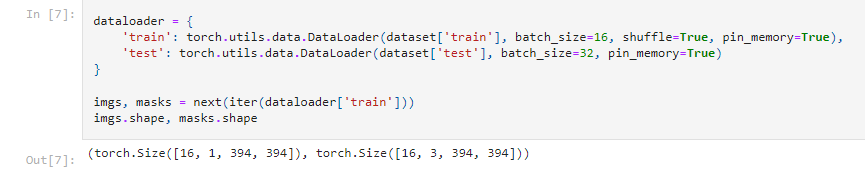


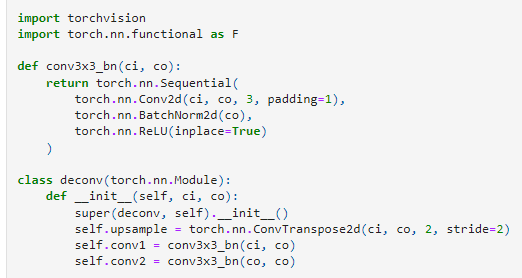


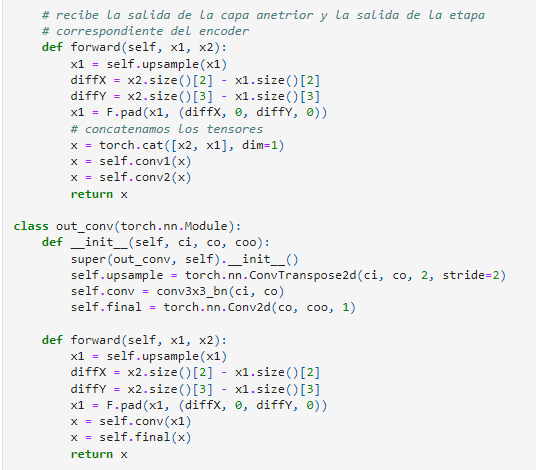
Creamos un data set en donde almacenamos las primeras 325 imágenes para entrenamiento y 100 imágenes para pruebas.

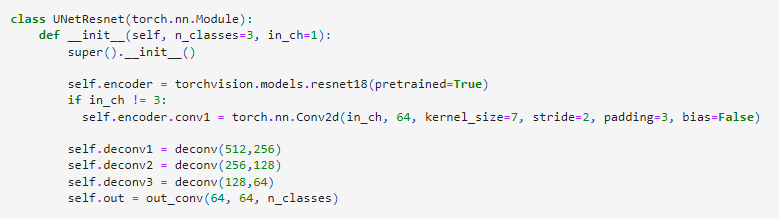


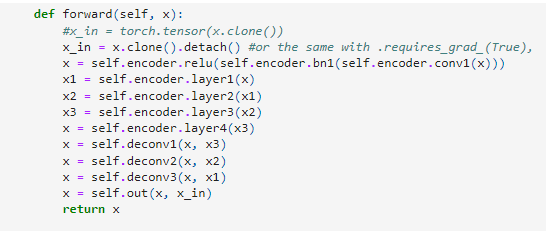
Creamos los batches para que se haga un entrenamiento mas eficiente, creamos batches de 16 en 16, y en los datos de prueba batches de 32 en 32.

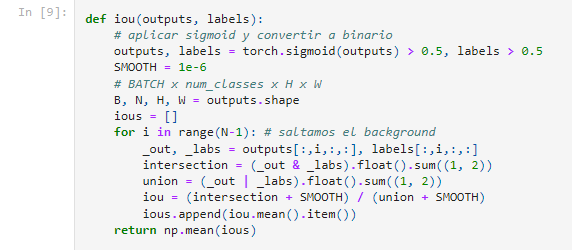
 Aquí está el código del modelo de red neuronal convolucional

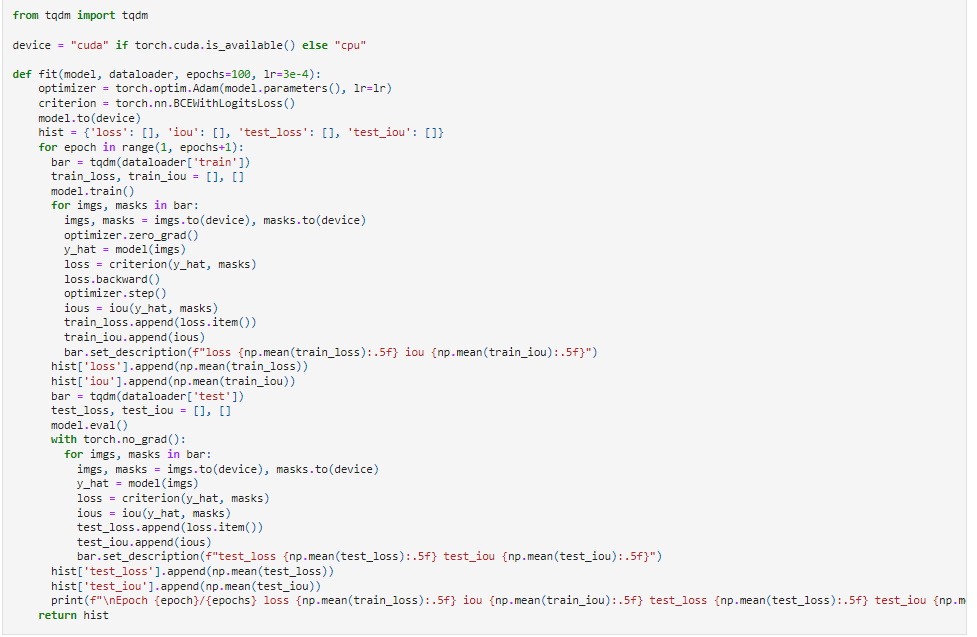




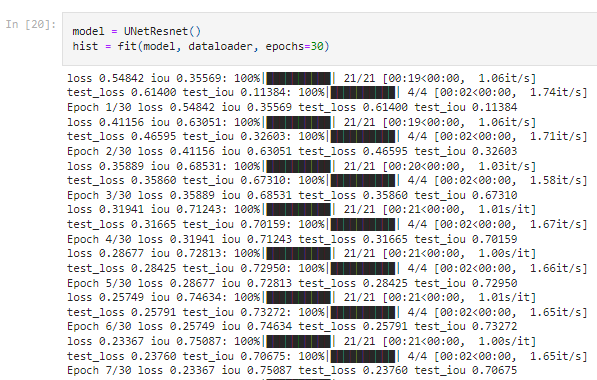


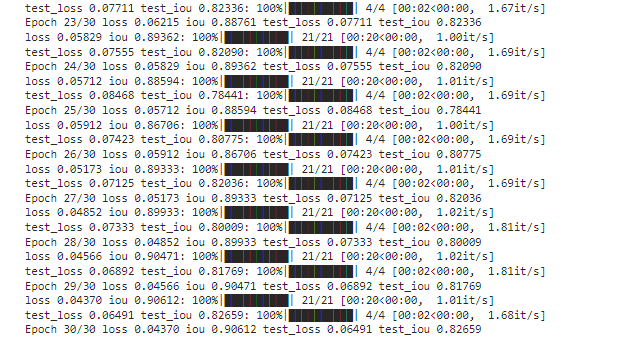




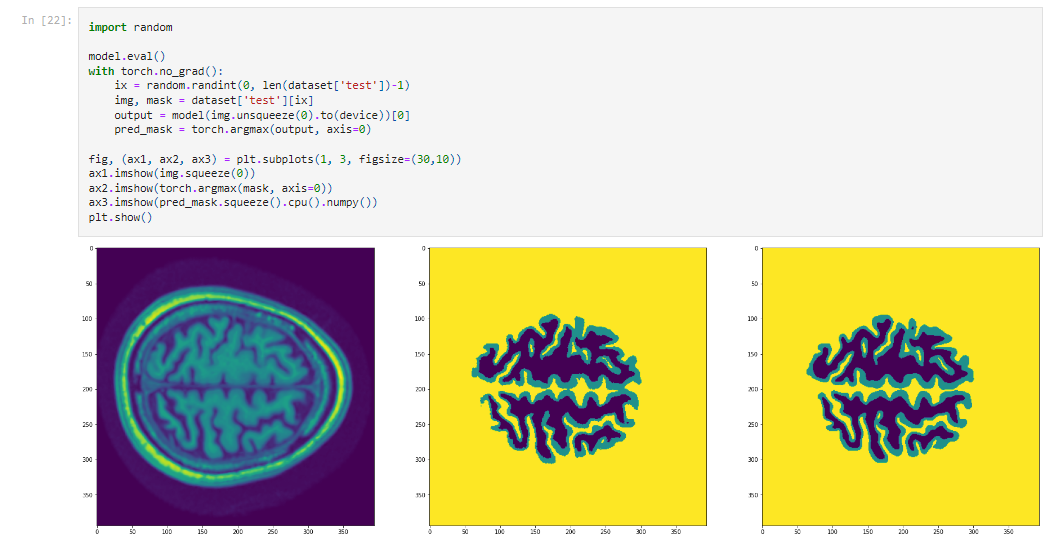
Metodos para Entrenamiento de la red neuronal

Ejecución del entrenamiento a 30 épocas





El resultado de la predicción, donde se puede ver que es casi exacta a la máscara real.

* Es la imagen de prueba
* Es el resultado esperado
* Resultado obtenido