Tarea 6 EL7008 – Primavera 2020
Segmentación semántica en dataset Kitti

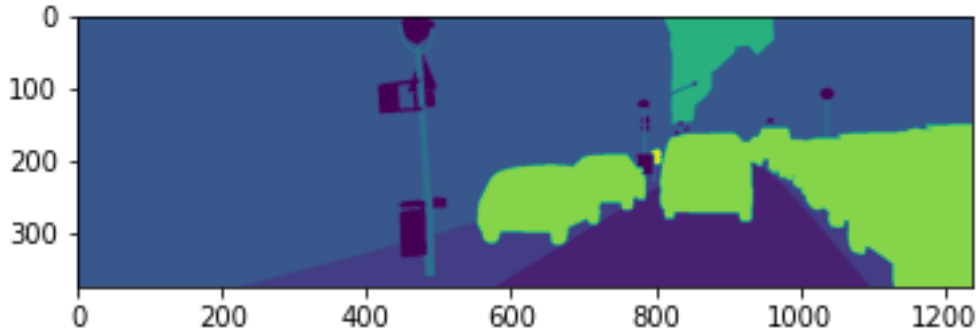
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla
Ayudantes: Juan Pablo Cáceres, Hans Starke, Javier Smith, José Villagrán

Fecha enunciado: 16 de diciembre de 2020
Fecha entrega: 5 de enero de 2021

El objetivo de esta tarea es entrenar y probar un sistema de segmentación semántica basado en U-net (red vista en clases). El código base a usar es el siguiente:
https://github.com/milesial/Pytorch-UNet

El código base se puede bajar desde el repositorio con el siguiente comando en colaboratory:
```
!git clone https://github.com/milesial/Pytorch-UNet
```

El dataset a usar es Kitti, el cual contiene 200 imágenes de entrenamiento etiquetadas píxel a píxel, y 200 imágenes de test sin etiquetar. Las etiquetas corresponden a 11+1 clases posibles (11 clases de objetos, y una clase extra para píxeles no válidos). El dataset contiene imágenes capturadas por un vehículo en movimiento. A continuación, se muestra una máscara de segmentación del dataset:



El dataset se puede bajar usando el siguiente comando en colaboratory:
```
!wget https://s3.eu-central-1.amazonaws.com/avg-kitti/data_semantics.zip
```

La red, al entrenarse, recibe dos imágenes: una imagen RGB de tamaño 1x3xHxW, donde W y H es el ancho y alto de la imagen de entrada, y una máscara de tamaño 1xHxW conteniendo las etiquetas por cada píxel (al usar batch de tamaño 1). El objeto `KittiDataset` debe entregar imágenes de tamaño 3xHxW y el DataLoader se encarga de redimensionar la imagen a 1x3xHxW.

Las máscaras de Kitti contienen 31 valores posibles, los cuales deben ser reducidos a 11+1 etiquetas dentro de la clase `KittiDataset`. La función `kitty_inverse_map_1channel()` que realiza la conversión será entregada junto con el enunciado de la tarea.

Una vez entrenada la red de segmentación semántica, se puede usar para segmentar imágenes. La entrada de la red es un tensor de tamaño 1x3xHxW. La salida de la red es un tensor de dimensiones 1x12xHxW, la cual considera scores para cada una de las 12 clases.

Para obtener las etiquetas de cada píxel (la máscara) de la salida de la red y poder mostrarla con plt.imshow(), se debe:

1) Transformar la imagen al formato de numpy, la imagen es de tamaño (1x12xHxW)
2) Transformar el tensor resultante al tamaño (HxWx12x1) usando transpose( )
3) Transformar el tensor resultante al tamaño (HxWx12)
4) Aplicar np.argmax( ) al tensor resultante, tras lo cual el tensor resultante es una máscara de (HxWx1)

En los pasos anteriores, es posible juntar los pasos 2 y 3 si el alumno lo desea.

Implementación del dataset para pytorch

Se debe copiar el archivo utils/dataset.py al notebook (la clase BasicDataset) y modificarlo para poder leer el dataset Kitti.

El alumno debe analizar el dataset y crear una clase KittiDataset, la cual tiene la siguiente estructura mínima:

```
class KittiDataset(Dataset):
    def __init__(self, imgs_dir, masks_dir, read_mask, scale=1, mask_suffix=''):
        (código)
    def __len__(self):
        (código)
    def __getitem__(self, i):
        (código)
```

Las variables son las siguientes:

`imgs_dir` es la carpeta conteniendo el dataset

`masks_dir` es la carpeta conteniendo las máscaras

La variable `read_mask` debe indicar si se devuelve una imagen RGB junto con una máscara (para imágenes de training), o sólo la imagen RGB junto con un objeto None (para imágenes de testing, las cuales no traen máscaras).

En la función `__getitem__()` usada como base se deben efectuar las siguientes modificaciones:

1) Se debe eliminar las llamadas a los assert cuando `read_mask=False`
2) Se debe eliminar la llamada a `mask = self.preprocess(mask, self.scale)`
3) Se debe llamar la función entregada junto al enunciado:
   `mask = kitty_inverse_map_1channel(np.array(mask, dtype=np.int32))`
4) La máscara se debe devolver como un objeto de tipo torch.IntTensor:
   `mask_torch = torch.from_numpy(mask).type(torch.IntTensor)`
5) Se recuerda devolver `mask_torch=None` cuando `read_mask=False`. Los pasos 2, 3 y 4 no deben realizarse cuando `read_mask=False`.

Entrenamiento de la red

Se recomienda copiar el archivo train.py al notebook, y modificarlo para poder usar el dataset Kitti, considerando 12 clases posibles. El código en train.py ya contiene todo el código necesario para efectuar el entrenamiento. Se debe reemplazar `BasicDataset` por `KittiDataset`.

Se debe usar 7 épocas para el entrenamiento, con un batch de tamaño 1. Se debe guardar los valores de la función de pérdida de validación para poder graficarlos posteriormente. Además, se recomienda evaluar sobre el conjunto de validación cada vez que una época avance en un 50% (dos veces por época).
El código base guarda un checkpoint después de completar cada época.

Evaluación de la red

En esta tarea, la evaluación del desempeño de la red se realizará analizando visualmente el resultado de la segmentación.

Se pide realizar los siguientes pasos y detallarlos en el informe:

1. Analizar el dataset Kitti, describir su estructura
2. Implementar la clase KittiDataset
3. Adaptar el código del entrenamiento de la red para usar un objeto de tipo KittiDataset y almacenar la evolución de la función de pérdida de validación (la cual ya se calcula).
4. Entrenar la red considerando 7 épocas
5. Graficar la evolución de la función de pérdida
6. Graficar la imagen RGB, la máscara ground truth y la máscara predicha para 5 imágenes del conjunto de entrenamiento, usando el checkpoint de la segunda época. Analizar visualmente el desempeño de la red.
7. Graficar la imagen RGB y la máscara predicha para 5 imágenes del conjunto de test, usando el checkpoint de la segunda época. Analizar visualmente el desempeño de la red.
8. Graficar la imagen RGB, la máscara ground truth y la máscara predicha para 5 imágenes del conjunto de entrenamiento, usando el checkpoint de la séptima época. Analizar visualmente el desempeño de la red.
9. Graficar la imagen RGB y la máscara predicha para 5 imágenes del conjunto de test, usando el checkpoint de la séptima época. Analizar visualmente el desempeño de la red.
10. Analizar el desempeño del segmentador a partir de las máscaras obtenidas en los puntos anteriores, comparando el desempeño obtenido usando el checkpoint de la segunda época versus el checkpoint de la séptima época.
11. Documentar cada uno de los pasos anteriores en el informe.

El código debe ser desarrollado en colaboratory, eligiendo un entorno de ejecución con gpu.
Los informes, los códigos y el archivo README.txt deben ser subidos a U-Cursos hasta las 23:59 horas del día 5 de enero de 2021.

Importante: La evaluación de esta tarea considerará el correcto funcionamiento del código, la calidad de los experimentos realizados y de su análisis, las conclusiones, así como la prolijidad y calidad del informe entregado.

Nota: El informe de la tarea debe ser subido a turnitin.