Tarea 6 EL7008 – Primavera 2022
Segmentación semántica en dataset Kitti

Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla

Fecha enunciado: 17 de noviembre de 2022
Fecha entrega: 30 de noviembre de 2022

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

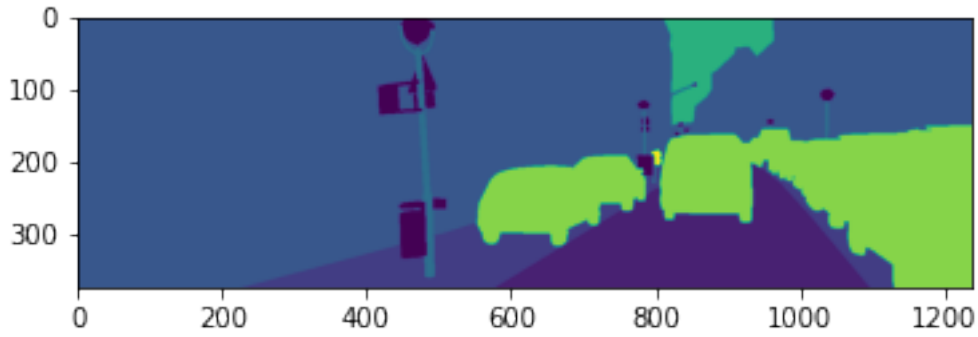
El código base se puede bajar desde el repositorio con el siguiente comando en colaboratory:

```
!git clone https://github.com/milesial/Pytorch-UNet
```

Además se debe instalar wandb:

```
!pip install wandb
```

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 un ejemplo de 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
```

Una vez bajado el código y el dataset, se debe entrar en la carpeta del repositorio, ejecutando:

```
cd /content/Pytorch-UNet/
```

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.

Objeto dataset para pytorch

Se entrega el archivo data_loading.py, el cual debe pegarse en una celda del notebook.

Modelo de la red

Se debe copiar el archivo unet_model.py al notebook. Se debe reemplazar el import por el siguiente:

```
from unet.unet_parts import *
```

Dataloader

Dado que colaboratory puede entregar sólo 1 gpu en algunas instancias, se debe reemplazar la línea de código del dataloader por la siguiente:

```
loader_args = dict(batch_size=batch_size, num_workers=1, pin_memory=True)
```

Funciones de pérdida

Se debe almacenar el valor de la función de pérdida, tanto de entrenamiento como de validación, cada vez que se llame a la función `evaluate()`. Para lograr esto, se entregará un archivo modificado evaluate.py, el cual se debe copiar a una celda del notebook. Además de esto, se debe implementar el cálculo del valor promedio de la función de pérdida de entrenamiento promedio, sobre todos los batches de entrenamiento procesados hasta el momento en cada época.

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 deben reemplazar las líneas que crean el objeto dataset por la siguiente:

```
dataset = KittiDataset(dir_img, dir_mask, 'gray', img_scale)
```

Se debe comentar (deshabilitar) las siguientes dos líneas:

```
from evaluate import evaluate
from unet import UNet
```

Además se debe reemplazar la función `get_args ()` por la función `get_args_train()` entregada en el archivo get_args_train.py

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. Se debe usar 2 épocas para el entrenamiento, con un batch de tamaño 1. Se debe guardar los valores de la función de pérdida de entrenamiento y validación para poder graficarlos posteriormente. El código base guarda un checkpoint después de completar cada época.

Visualización de máscaras de segmentación

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)
5) Aplicarle .astype(np.ubyte)

En los pasos anteriores, es posible juntar los pasos 2, 3 y 4 si el alumno lo desea. Además, la imagen de entrada a la red se debe multiplicar por 255 antes de visualizarla. Se entrega un código base para realizar esto, en predict.py, el cual debe pegarse en una celda del notebook.

Evaluación de la red

En esta tarea, la evaluación del desempeño de la red se realizará analizando visualmente el resultado de la segmentación. Esto se debe a que las imágenes del conjunto de prueba de Kitti no están etiquetadas.

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. Copiar tanto el modelo de la red como el evaluate.py entregado al notebook
4. 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 entrenamiento y validación.
5. Entrenar la red considerando 2 épocas
6. Graficar la evolución de la función de pérdida de entrenamiento y validación
7. 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.
8. 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.
9. Repetir los pasos 5 – 8, usando una red u-net con sólo 3 capas Up y 3 Down
10. Repetir los pasos 5 – 8, usando una red u-net con sólo 2 capas Up y 2 Down
11. Analizar el desempeño de los tres modelos de segmentador a partir de las máscaras obtenidas en los puntos anteriores, tanto individual como comparativamente.
12. Analizar el nivel de sobreajuste de los tres modelos, tanto individual como comparativamente.
13. 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 miércoles 30 de noviembre.

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. El link se publicará en el foro.