Segmed_Class_Example

February 3, 2020

Universidad de Guanajuato

División de Ciencias e Ingenierías

Grupo de investigación : DCI-NET

Ejemplo de uso de la clase Segmed

Investigador Responsable : Dr. Luis Carlos Padierna García

Alumno : Gustavo Magaña López

En este documento se muestra la utilidad del sistema segnet/Segmed, la cual fue desarrollada dentro de mi *fork* del repositorio principal del grupo de investigación DCI-NET. Mi trabajo se desarrolló principalmente en las ramas mru_tests y callback_test, aunque se puede observar que existen otras ramas que fueron utilizadas de forma transitoria. Las pruebas se llevaron a cabo utilizando Google Colaboratory.

Se desarrolló un sistema que automáticamente genera archivos de configuración, mide tiempos de ejecución y guarda los parámetros utilizados cada vez que se llama un método de la clase. La clase **segnet.utils.Segmed.Segmed** (vea el código fuente) tiene como objetivo facilitar el entrenamiento de arquitecturas de **Deep Learning**, utilizando Keras con el *backend* TensorFlow. Esta libreta de jupyter se tomó de una creada con Google Colaboratory para realizar experimentos con la MultiResUNet.

1. Gracias al escape **!**, se pueden invocar comandos **shell**, que para el caso de las máquinas virtuales de Google Colab que tienen el sistema operativo Ubuntu (distribución de GNU/Linux) es **bash**, por defecto. A continuación se enlistan los directorios disponibles en el entorno de trabajo dentro de la máquina virtual asignada por Google para la sesión :

```
In [0]: !ls
```

```
drive  sample_data
```

2. Se tiene acceso por ejemplo a al gestor de paquetes **apt** con el cual se pueden instalar programas y utilidades que podrían ser necesitados al realizar los experimentos. Aquí se instala por ejemplo jq un parser de JSON, formato utilizado para generar diversos registros de los experimentos. En realidad la clase **Segmed** usa otro estándar (JSON-Lines), pero **jq** trabaja fácilmente con cualquiera de los dos.

```
In [ ]: !apt install jq
```

3. Una herramienta muy importante para el trabajo con el lenguaje de programación Python es el uso del gestor de paquetes **pip** el cual permite instalar paquetes y módulos desde el **índice de paquetes de Python** (PyPI por sus siglas en inglés) o algún repositorio de GitHub, como se muestra a continuación.

```
In [2]: #!pip uninstall segnet
        !pip install git+https://github.com/gmagannaDevelop/segnet.git@callback_test
```

Aquí se instaló el paquete segnet, desde el fork mencionado al inicio del documento. **Importante**: En caso de estar trabajando con un paquete que usted esté desarrollando, después de cada cambio guardado con `git add` y `git commit` los cambios deberán enviarse la copia que se encuentra en GitHub con `git push`. Finalmente será necesario reinstalar el paquete y reiniciar el entorno de ejecución de Colab para poder utilizar la última versión que probablemente se desea poner a prueba.

4. A contnuación se importan todos los módulos necesarios para realizar entrenamientos con diferentes versiones de las arquitecturas U-Net y MultiResUNet.

```
In [0]: import os
        import collections

        ##################################################################
        import tensorflow as tf
        import segnet.metrics as mts
        from segnet.models import unet
        from segnet.models import multiresunet as mru
        from segnet.models import multiresunet2 as mru2
        from segnet.models import multiresunet3 as mru3
        from segnet.utils.Segmed import Segmed
        ##################################################################
```

```
<IPython.core.display.HTML object>
```

5. Se monta una cuenta de Google Drive a la sesión de Google Colab, para poder acceder a los *datasets* y llevar a cabo los entrenamientos. Esto requiere copiar y pegar una dirección generada automáticamente e ingresar las credenciales de la cuenta a cuyo Drive se desea acceder.

```
In [0]: ### Data-related
        from google.colab import drive, files
        drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/con
```

6. Nótese que la raíz del árbol no quedará montada en `drive` sino en `drive/My Drive/`, es por esto que a continuación se define esta dirección y se asigna a la variable `root_dir`. La otra variable `log_dir` es muy importante ya que cada instancia de la clase **Segmed** requiere de un directorio para generar la bitácora, la cual será un directorio que contiene diversos archivos relacionados al modelo y entrenamientos que puedan llevarse a cabo.

```
In [0]: # Where can we find everything :
        root_dir = "drive/My Drive/DCI-Net"
        # Where will we log everything :
        log_dir  = os.path.join(root_dir, "Last Tests/Logs/ISBI_ADAM_CPU")
```

7. Aquí se define un diccionario para facilitar el acceso a los **datasets**.

```
In [0]: dataset_paths = {
          "ISBI": os.path.join(root_dir, "Colab_data/ISBI_neural/structured"),
          "colonoscopy":   os.path.join(root_dir, "Colab_data/colonoscopy"),    # Full original
          "dermoscopy80":  os.path.join(root_dir, "Colab_data/dermoscopy80"),   # reduced to 80 i
          "dermoscopy150": os.path.join(root_dir, "Colab_data/dermoscopy150"), # reduced to 150,
          "chinese1":      os.path.join(root_dir, "Colab_data/Dataset 2")      # Chinese dataset
        }
```

8. Para generar un objeto de la clase **Segmed** es necesario que los directorios pasados como parámetros sean válidos, es decir existan. Eso se puede verificar gracias al método estático de la clase `Segmed.assert_isdir(camino_hasta_directorio)`. Dicho método devuelve el parámetro dado si éste es un directorio dentro del sistema de archivos y genera una excepción si no.

```
In [0]: Segmed.assert_isdir(dataset_paths["ISBI"])

Out[0]: 'drive/My Drive/DCI-Net/Colab_data/ISBI_neural/structured'
```

9. Aquí se muestra de nuevo la ventaja de trabajar con diccionarios, manejando un alto nivel de abstracción y disminuyendo la redundancia de código (boilerplate). El diccionario `my_compiling_kw` será utilizado posteriormente para correr los entrenamientos.

```
In [ ]: optimizers = {
            "chinese": tf.keras.optimizers.SGD(
                learning_rate=0.06, momentum=0.2, nesterov=False
            ),
            "chinese nesterov": tf.keras.optimizers.SGD(
                learning_rate=0.06, momentum=0.2, nesterov=True
            ),
            "Original Adam": tf.keras.optimizers.Adam(
                beta_1=0.9, beta_2=0.999, epsilon=10e-8
            ),
            "Original Adamax": tf.keras.optimizers.Adamax(
                learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7
            ),
            "Aggresive Adamax": tf.keras.optimizers.Adamax(
```

```
            learning_rate=0.06, beta_1=0.9, beta_2=0.999, epsilon=1e-7
        )
    }


    my_compiling_kw = {
        'optimizer': optimizers["Original Adam"],
        'loss': 'binary_crossentropy',
        'metrics': [
          mts.jaccard_index, mts.dice_coef,
          mts.O_Rate, mts.U_Rate, mts.Err_rate
        ]
    }
```

10. En esta celda se cuenta el número de imágenes y máscaras de segmentación para posteriormente estimar el tamaño de lote y demás híper parámetros.

```
In [0]: dataset = dataset_paths["ISBI"]
        n_masks = len(os.listdir(os.path.join(dataset, "msks/masks")))
        n_images = len(os.listdir(os.path.join(dataset, "imgs/images")))
        print(f" Number \n masks:\t{n_masks}\nimages:\t{n_images}")
```

```
 Number
 masks:         30
images:         30
```

11. Se generan híper parámetros para los entrenamientos.

```
In [0]: my_hyper_params = {
            'batch_size': 8,
            'epochs': 40,
            'steps_per_epoch': 4
        }
```

12. Se crea un diccionario ordenado con diversas arquitecturas convolucionales. Cada uno de los objetos que corres ponden al conjunto de valores de este diccionario son modelos de **tensorflow.keras**.

```
In [0]: architectures = collections.OrderedDict({
          "MultiResUNet Edwin": mru.MultiResUnet(),
          "MultiResUNet Gustavo": mru2.MultiResUNet(),
          "Unet": unet(),
          "MultiResUNet Original": mru3.MultiResUnet()
        })
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resour
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

13. Se crean objetos de la clase Segmed, a partir de este momento, en el directorio **log_dir** se encontrará un directorio bitácora cuyo nombre será una combinación de:
    1. El nombre de la arquitectura (especificado con el parámetro **name**)
    2. El autor (quien está realizando el experimento, parámetro **author**)
    3. La hora y fecha UTC, generada automáticamente gracias al paquete de Python **datetime**.

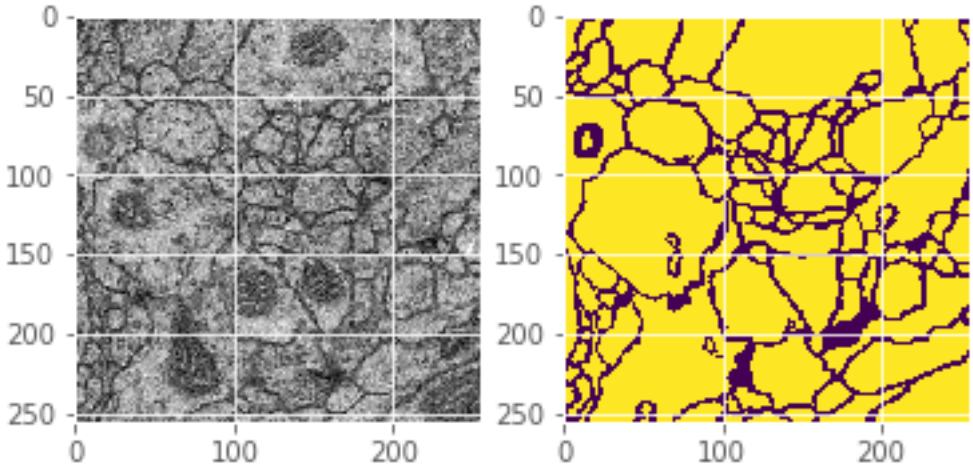
```
In [0]: models = collections.OrderedDict({
            key: Segmed(
                model = architectures[key],
                name = key,
                base_dir = log_dir,
                data_path = dataset,
                author = "Gustavo Magaña"
            )
            for key in architectures.keys()
        })
```

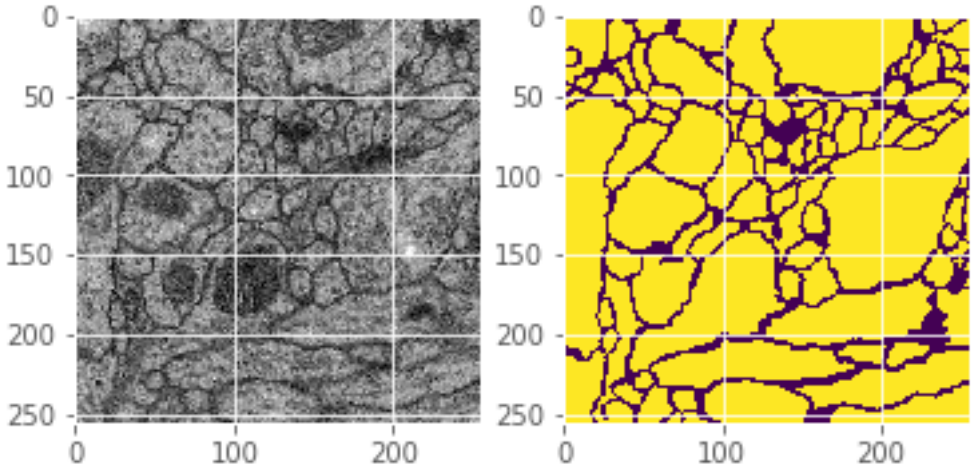
14. Antes de realizar los entrenamientos, se visualizan los pares de imágenes y máscaras de segmentación gracias al método `Segmed.show_img_and_mask`

```
In [0]: # Create the generators to enable visualisation :
        for model in models.values():
          model.create_train_test_generators()

        for model in models.keys():
          print(f"Image/mask pairs for achitecture `{model}`")
          models[model].show_img_and_mask(n=2)
```

```
Found 24 images belonging to 1 classes.
Found 24 images belonging to 1 classes.
Found 6 images belonging to 1 classes.
Found 6 images belonging to 1 classes.
Image/mask pairs for achitecture `MultiResUNet Original`
```





15. Gracias al método `Segmed.comment`, se pueden guardar comentarios durante la sesión interactiva, los cuales se guardarán en un archivo de texto dentro del directorio bitácora creado por el constructor al generar el objeto correspondiente. Aquí se itera sobre todos los modelos pero bien podría haberse llamado simplemente en un objeto y el comentario sería exclusivo a éste.

```
In [0]: for model in models.values():
            model.comment(" Ejemplo de comentario ")
```

16. Iterando sobre los objetos de la clase **Segmed** se entrenan todas y cada una de las arquitecturas que fueron definidas en el punto 12.

```
In [ ]: for model in models.values():
            model.train(
                compiling_kw = my_compiling_kw,
                hyper_params = my_hyper_params
            )
```