Badblock Documentation

Release 1.0

Liang Li

Mar 07, 2019

CONTENTS:

CHAPTER

ONE

INSTALLATION!

1.1 Install Python

1.1.1 Install Python from *Python*

The latest Python can be downloaded and installed from https://www.python.org/downloads/ , a version above 3.6 is recommended.

1.1.2 Install Python from *Conda*

Conda can be found via https://docs.conda.io/projects/conda/en/latest/user-guide/install/.

Conda is also recommended since many different versions of python could be changed based on:

```
source activate myenv
```

with the whole list of pythons with different versions can be shown as:

```
conda env list
```

1.2 Install Pytorch

1.2.1 Before start with pytorch, make sure CUDA has been installed.

Install your cuda from https://developer.nvidia.com/cuda-downloads?target_os=Linux.

Note: Check here to update with the latest driver. A good match of driver with the GPU will largerly increase the speed, thus please make sure you have the latest driver with your GPU.

1.2.2 Install pytorch

pytorch can be found via https://pytorch.org/get-started/locally/.

1.3 Other packages

There are a list of other packages that are optional to install, while most could be install by using **pip**. [if you are under conda, make sure you are using the correct **pip** under the correct version of python]

- $ pip install visdom
- $ pip install numpy
- $ pip install matplotlib
- $ pip install Pillow

Plus, please include 'pytorch_msssim' folder if you want to use msssim as a loss, and if there are other package needed, try **pip**

```
pip install packagename
```

NOW, you are all set with all the packages need for the deep learning Unet, and next we will forward to prepare our data.

See also:

If a python environment with pytorch is hard to obtain locally, Docker is always a good choice to make your network run in cloud. Note: a most recent pytorch with NVIDIA can be pulled from https://docs.nvidia.com/deeplearning/dgx/pytorch-release-notes/running.html.

CHAPTER

TWO

PREPARE YOUR DATA!

2.1 Process the raw data.

2.1.1 Read and reshape

The size and the shape of the data would affect not only the output but also the learning speed of the network, and hence making the raw data the shape we want and postprocessing it back to the origin shape are usually common in machine learning.

In our case, we have .s file as both the good data and badblock data, which are too large for our network, thus we need to reshape the data based on [TOF, Slice, W, L]. In simple cases without down/up sampling(we will cover this later), W and L do not matter, but there we need try to make W and L as a power of 2.

Numpy provides the padding function:

```
data = np.pad(data, ((x1, y1), (x2, y2), (x3, y3), (x4, y4)), 'wrap')
```

In our case, if we have [TOF=34, Slice=815, W=50, L=520], we can do:

```
data = np.pad(data, ((0, 0), (0, 0), (7, 7), (0, 0)), 'wrap')
```

to make it [TOF=34, Slice=815, W=64, L=520], which would be good enough for 3 times downsampling since W and L can be divided by 2^3.

Note: Here I only list one way to change the shape, and actually there might be tons of other methods which are good to try.

2.1.2 Save the data

Here is something I want to talk about:

```
def my_fn(foo, bar=True):
    """A really useful function.

    Returns None
    """
```

This module is used to generate intermediate pickles before the training Created on Thu Jun 14 16:19:07 2018 @author: bill @update: liang

class sino_process_tof.**Sinogram_Processor**(*data_file*, *for_test=False*, *slices=-1*, *min_blocks=0*, *max_blocks=0*)

process_data(*file*, *tof*, *slices*, *theta*, *dist*)
This function is used to process the sinogram file

The function will first read the file with the extension .s, and find its corresponding good and badblock file, output a stack with both good and bad sinograms: Note, here we padding the axis to make it more suitable for the network to train, e.g. we make axis 3 from 50 to 64 by padding wrapped pixels from up and down

Args: file

Returns: output_sino

Raises: IOError: An error occurred accessing the file .

process_sino_file()
This function is used to process the sinogram file

The function will first read the file with the extension .s.hdr, and find its corresponding good and badblock file, output a stack with both good and bad sinograms

Args: self.file

Returns: output_sino_xy

Raises: IOError: An error occurred accessing the file .

remove_block(*orig_img*, *block_num*)
This function is omitted.

sino_process_tof.**main**()
The main function to call for processing all the data files.

sino_process_tof.**process_sinogram**(*file*)
This function is used to process one patient.

2.2 Loading all the data into the GPU is not only time consuming and not efficient.

2.2.1 Load

CHAPTER

THREE

BUILD YOUR NETWORK!

Created on Tue Mar 20 21:17:05 2018

@author: bill @updated: Liang

class unet.**Sino_repair_net**(*opts*, *device_ids*, *load_model=False*)

optimize(*output*, *target_img*, *input_img*)

save_network()

set_optimizer(*optimizer*)

test(*x*, *y*)

train_batch(*input_img*, *target_img*)

class unet.**UNet**(*opts*)

forward(*x*, *test=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class unet.**UNet_down_block**(*input_channel*, *output_channel*, *down_sample*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class unet.**UNet_up_block**(*prev_channel*, *input_channel*, *output_channel*, *ID*)

forward(*prev_feature_map*, *x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

CHAPTER

FOUR

TRAIN YOUR NETWORK!

Created on Thu Jan 15 18:45:25 2019

@author: bill @updated: Liang

class train_unet.**Network_Trainer**(*opts*, *device_ids*)

test(*network*, *batch_size*, *perform_recon_loss=False*, *output_all_sinos=False*)

train()

visualize_progress(*opts*, *images*, *istest=False*)

write_log(*value*, *log_type='test'*)

train_unet.**fill_image**(*input_sino*, *out_img*)

train_unet.**main**()

train_unet.**preprocess**(*input_img*, *target_img*)

train_unet.**random**() → x in the interval [0, 1).

CHAPTER

FIVE

VISUALIZE YOUR RESULTS!

CHAPTER

SIX

ADJUST YOUR PARAMETERS!

To start to view the result by using visdom:

```
python -m visdom.server
```

To watch the using of GPU by using:

```
watch -n 1 nvidia-smi
```

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX