# Badblock Documentation

*Release 1.0*

**Liang Li**

**May 09, 2019**

# HERE ALL THE CONTENTS GO:

Deep learning has been fast developed in recent years, which triggers lots of researches in medical image analysis. In this tutorial, I will show you how to improve the quality of image(or say image denoising) by using one of the many networks, UNet.

CHAPTER

ONE

# REASONS FOR THIS TUTORIAL

I started my internship in Siemens, Knoxville at the begining of 2019, where I first touched and learned deep learning. Great thanks to super nice manager(BILL) since I do learn a lot from him!! Please note that I am also a beginner in deep learning, and thus point me out if I will be wrong in the following content.

During my internship, I try to use deep learning(UNet, FrameletNet, GAN) to do image impainting and denoising, while I think this is so great a chance for me to learn about deep learning and explore different kinds of networks, and this makes me feel excited almost every day so I want to share this with you.

In fact, a lot of my coworkers are also very interested in building or learning about deep learning while they think deep learning is very hard to learn. Hence, I also write this tutorial for people who are in the field of medical imaging or other who might want to build their own deep learning networks but they do not know how to start. To be honest, to select the most suitable parameters/models is the most time consuming part of deep learning, but to start deep learning is far more simple than you thought!

CHAPTER

TWO

# GOALS FOR THIS TUTORIAL

Hence, lets begin with this to open your new world to the deep learning(AI)!

> First, we will try to install all the necessary packages for the deep learning, and it will be great if you have a GPU.
>
> And then, we will talk about how to precesss your data (so important)!
>
> Third, a quick start with U-Net will lead you to the heart of deep learning.
>
> Finally, we will train our U-Net and use several visualizations to view our results.

## 2.1 Installation

Python is a programming language that lets you work quickly and integrate systems more effectively. From my point, python is simple to use and learn. The reason why we use python is that currently most deep learning frameworks have already been implemented based on python and plenties of open source packages that are available in the field of data science can be utilitized for our data analysis, such as scipy, scikit-learn, pandas. Pythonic makes life easier.

### 2.1.1 Install Python

#### Direct Python from *Python*

Python2 will not be supported any more by the community, and hence let's work on Python3 and install the latest python. The latest Python can be downloaded and installed from https://www.python.org/downloads/ . I have installed 3.6, please try to install a version above 3.6.

#### Indirect Python from *Conda*

Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies.

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

Conda is recommended when multiple version of python will be installed in the system, and the version of python can be changed easily based on:

```
source activate myenv
```

For example,

```
source activate py36
```

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

```
conda env list
```

other useful methods for install certain packages including

```
conda search scipy
conda install --name myenv scipy
conda install scipy=0.15.0
source deactivate
conda info --envs
conda list -n myenv scipy
```

### 2.1.2 Install Pytorch

Before start with pytorch, make sure CUDA has been installed, where CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia.

#### Install CUDA

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.

#### Install pytorch

Pytorch is an open source deep learning platform that provides a seamless path from research prototyping to production deployment.

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

### 2.1.3 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
- $ pip install scipy

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 needed for the deep learning Unet, and in the next step we will forward to prepare our data.

### 2.1.4 Build in Docker(Optional)

Docker is a computer program that performs operating-system-level virtualization. The advantage of Docker is that we can only install the packages we need and then the extra cost of unnecessary components in the operating system can be reduced.

**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.

Docker is simple to use too!

```
# List Docker images
     docker image ls

# List Docker containers (running, all, all in quiet mode)
     docker container ls
     docker container ls --all
     docker container ls -aq
     docker container stop xxxxxxx
# List Docker containers (running, all, all in quiet mode)
     docker build -t friendlyhello .  # Create image using this directory's Dockerfile
     docker run -p 4000:80 friendlyhello  # Run "friendlyname" mapping port 4000 to 80
     docker run -d -p 4000:80 friendlyhello         # Same thing, but in detached mode
     docker container ls                                # List all running containers
     docker container ls -a             # List all containers, even those not running
     docker container stop <hash>           # Gracefully stop the specified container
     docker container kill <hash>         # Force shutdown of the specified container
     docker container rm <hash>        # Remove specified container from this machine
     docker container rm $(docker container ls -a -q)         # Remove all containers
     docker image ls -a                             # List all images on this machine
     docker image rm <image id>            # Remove specified image from this machine
     docker image rm $(docker image ls -a -q)   # Remove all images from this machine
     docker login             # Log in this CLI session using your Docker credentials
     docker tag <image> username/repository:tag  # Tag <image> for upload to registry
     docker push username/repository:tag            # Upload tagged image to registry
     docker run username/repository:tag                   # Run image from a registry
```

#### Using Docker with DGX

[Special Requirement for DGX user] In order to connect to his Docker daemon a user has to commit the parameter "-H unix:///mnt/docker_socks/<user_name>/docker.sock" with every Docker command.

- e.g. "docker -H unix:///mnt/docker_socks/<user_name>/docker.sock run –rm -ti <image_name> [optional_command]"
- e.g. "docker -H unix:///mnt/docker_socks/<user_name>/docker.sock image ls" alternatively use the script "run-docker.sh" in /usr/local/bin:
- e.g. "run-docker.sh –rm -ti [further_options] <image_name> [optional_command]"

I was using a line like below in a bash file to let the docker run within DGX by using slum.

```
sbatch --gres=gpu:1 --mail-user=liang.li.uestc@gmail.com --mail-type=ALL --output=/
↪badblock/li_dataset/log.txt --job-name=trainbd --error=/badblock/li_dataset/error.
↪txt run-nvidia-docker.sh --rm --name train_unet -v /badblock/li_dataset/:/badblock/␣
↪-w /badblock/ nvcr.io/nvidia/pytorch:19.01-py3 python /badblock/code/badblock-
↪fillgaps/train_unet.py --training-file="/badblock/data/DataTOF_Train/" --test-file=
↪"/badblock/data/DataTOF_test/" --mask-file="/badblock/data/mask.pkl" --output-path=
↪"/badblock/output/"
```

## 2.2 Prepare your data

### 2.2.1 Process the raw data

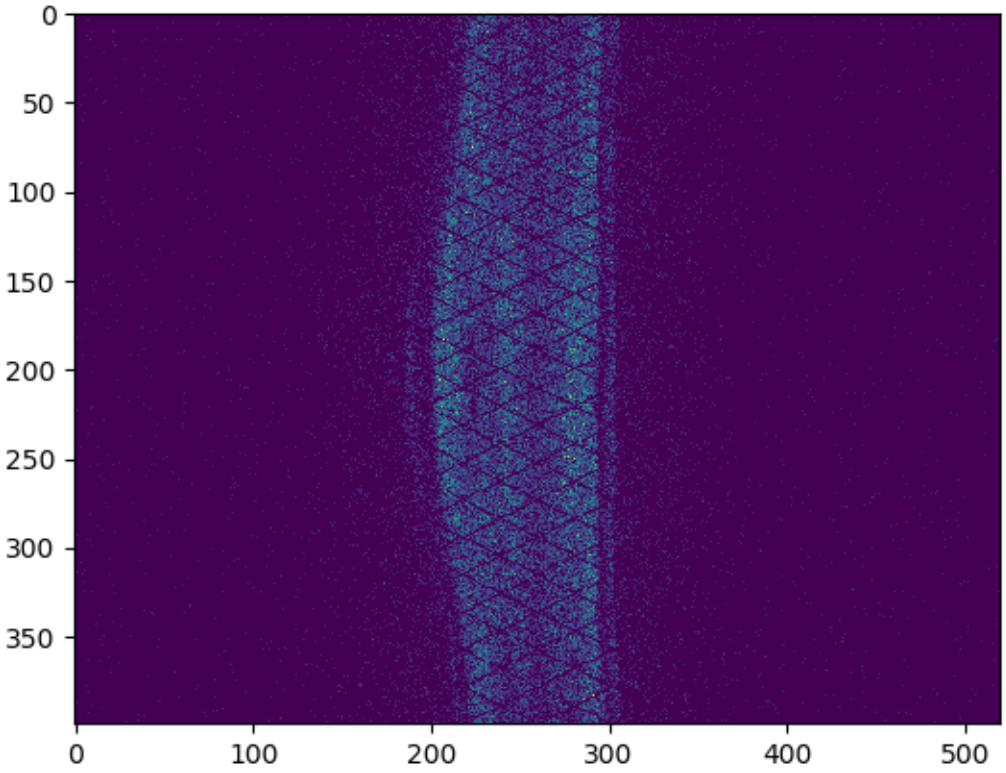
#### Read and reshape

The size(shape) and the distribution of the data would affect both the performance and the learning speed of the network, and hence reshaping or preprocessing the raw data to the shape/distribution we want and post-processing it back to the origin format are usually common in machine learning[The reasons behind this are a lot, say we want the learning converge similar to all directions in our training data]. In most the cases, the methods include but not limited to normalization, reshaping, etc.

In the following, I will give two examples about the data we deal with for medical imageing, sinograms and images.

#### eg. Sinograms

PET-CT and PET-MR scanners store the raw data in proprietary formats which can be processed only by the software provided by the scanner manufacturer, where one of the raw data is sinograms, which looks like below.




In our case, we have .s file as both the input file and target file with both files around 1.4G(with flatten data stored as uint16), and our goal is to using U-Net to learn the data in the input file and to make it close to the data in the target file. In fact, these files are too large as one input for neural network, because I do not want to load all the datas into the memory if we do not have a lot memory. (It is so important! Hence, I will take this three times: We do not want to feed all into the memory!We do not want to feed all into the memory!We do not want to feed all into the memory!).

### eg. Images

usage/images.png

To advoid feed all the data into the memory, we do reshape the data to more informative matrix and partition the data into smaller pieces, and then we could let the network learn smaller pieces with the pain that we will lose the relations among the smaller pieces. In my cases, I am working on sinograms and the sinograms have their own informative structure, and then I used [TOF, Slice, W, L] as my shape of the data, where I feed the network based on slices.

Here, for each slice, the image has shape [50, 520]. For most cases in machine learning without down/up sampling in the images, the number of W and L does not matter. But since we are working on UNet (as autoencoder, we need to encoder and then decode the data) we would better make W and L as a power of 2 (since we will consider the network structure as UNet which will include both downsampling and upsampling).

**Note:** An example of reshaping to 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 $2^3$.

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

### Save the data in pickle

Pickle module implement binary protocols for serializing and de-serializing a Python object structure. Here we could just dump our matrix into pickle by using:

```
pickle.dump(result, f, pickle.HIGHEST_PROTOCOL)
```

**Note:** Be careful with you pickle version, since it might not match between python2 and python3.

The details of the data process can be refered from sino_process_tof.py

### Precess and Save your data!

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.

**Warning:** Loading all the data into the GPU is not only time consuming and not efficient.

### 2.2.2 Load your data

#### Dataset Class

The abstract class in pytorch **torch.utils.data.Dataset** is the main class to call for loading data, and there are mainly two methods would be called.

- __len__: which returns the size/length of the dataset
- __getitem__: which returns one sample of the dataset based on the index, such that dataset[i] for index i

In our case, we define the class `class Sino_Dataset(dataset)` inherits from `torch.utils.data.Dataset`

Here, the length of the dataset is:

```
def __len__(self):

    return self.epoch_size
```

and the item of the dataset is:

```
def __getitem__(self, idx):

     return self.data[idx]
```

the `self.data.shape=[tof*slice, W, L]`

The details of getting item also include shuffling and file updating, which can be viewed as different methods to improve the randomness of the data set.

#### Dataset!

This module is used to create the dataset.

Created on Sun Mar 18 23:07:28 2018 @author: bill @updated: Liang

**class** datasets.**Sino_Dataset**(*datafile*, *epoch_size*, *testing=False*, *loading_multiplefiles=False*, *input_depth=1*, *output_depth=1*, *is_test_in_train=False*)
Sinogram dataset

**checkfeature**(*bad*)

**loadFile**(*data_file='training_data.npy'*)

**setMaxMissing**(*max_missing*)

## 2.3 Build your network!

### 2.3.1 Unet!

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*)

**save_network**()

**set_optimizer**(*optimizer*)

**test**(*x*, *y*, *valid=None*)

**train_batch**(*input_img*, *target_img*, *valid=None*)

## 2.4 Train your network!

### 2.4.1 Train_Unet!

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=False*)

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

## 2.5 Visualize your results!

## 2.6 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

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX