nebulae

A novel and simple framework based on tf and other image processing libs.

Modules Overview

Fuel

Easily manage and read dataset you need anytime

Toolkit

Includes many utilities for better support of nebulae

Toolkit

FuelGenerator

Build a FuelGenerator to spatial efficently store data.

- config: <**dict**> A dictionary containing all parameters.
- file_dir: <**str**> Where your raw data is.
- file_list: <**str**> A csv file in which all the raw datum file name and labels are listed.
- dst_path: <**str**> A hdf5/npz file where you want to save the compressed data.
- dtype: <**list** of **str**> A list of data types of all columns but the first one in *file_list*. Valid data types are 'uint8', 'uint16', 'uint32', 'int8', 'int16', 'int32', 'int64', 'float16', 'float32', 'float64', 'str'.
- height: <**int**, range between **(0, +∞)**> The height of image data. Defaults to 224.
- width: <**int**, range between **(0, +∞)**> The height of image data. Defaults to 224.
- channel: <**int**> The height of image data. Defaults to 1.
- encode: <**str**> The mean by which image data is encoded. 'PNG' is the way without information loss. Defaults to 'JPEG'.

An example of file_list.csv is as follow. 'image' and 'label' are the key names of data and labels respectively.

image	label
img_1.jpg	2
img_2.jpg	0
...	...
img_9.jpg	5

```
import nebulae

fg = nebulae.toolkit.FuelGenerator(file_dir='/file_dir',
                                   file_list='file_list.csv',
                                   dst_path='/file_dir/dst_path.hdf5',
                                   dtype=['uint8', 'int8'],
                                   channel=3,
                                   height=224,
                                   width=224,
                                   encode='jpeg')
```

Call generateFuel() to generate compressed data file.

```
fg.generateFuel()
```

You can edit properties again for generating other file.

```
fg.propertyEdit(height=200, width=200)
fg.generateFuel()
```

Passing a dictionary of changed parameters is equivalent.

```
config = {'height': 200, 'width': 200}
fg.propertyEdit(config=config)
fg.generateFuel()
```

Fuel

FuelDepot

Build a FuelDepot that allows you to deposit datasets.

```
fd = nebulae.fuel.FuelDepot()
```

Call loadFuel() to mount dataset on your FuelDepot.

- name: <**str**> Name of your dataset.
- batch_size: <**int**> The size of mini-batch.
- data_path: <**str**> The full path of your data file. It must be a hdf5/npz file.
- key_data: <**str**> The key name of data.
- if_shuffle: <**bool**> Whether to shuffle data samples every epoch. Defaults to True.
- height: <**int**, range between **(0, +∞)**> Height of image data. Defaults to 0.
- width: <**int**, range between **(0, +∞)**> Width of image data. Defaults to 0.
- resol_ratio: <**float**, range between **(0, 1]**> The coefficient of subsampling for lowering image data resolution. Set it as 0.5 to carry out 1/2 subsampling. Defaults to 1.
- is_seq: <**bool**> Declare whether this dataset is sequential. Defaults to False
- spatial_aug: <comma-separated **str**> Put spatial data augmentations you want in a string with comma as separator. Valid augmentations include 'flip', 'brightness', 'gamma_contrast' and 'log_contrast', e.g. 'flip,brightness'. Defaults to '' which means no augmentation.
- p_sa: <**tuple** of **float**, range between **[0, 1]**> The probabilities of taking spatial data augmentations according to the order in *spatial_aug*. Defaults to (0).
- theta_sa: <**tuple**> The parameters of spatial data augmentations according to the order in *spatial_aug*. Defaults to (0).
- temporal_aug: <comma-separated **str**> Put temporal data augmentations you want in a string with comma as separator. Valid augmentations include 'sample', e.g. 'sample'. Make sure to set *is_seq* as True if you want to enable temporal augmentation. Defaults to '' which means no augmentation.
- p_ta: <**tuple** of **float**, range between **[0, 1]**> The probabilities of taking temporal data augmentations according to the order in *temporal_aug*. Defaults to (0).
- theta_ta=(0): <**tuple**> The parameters of temporal data augmentations according to the order in *temporal_aug*. Defaults to (0).

```
fd.loadFuel(name='test-img',
            batch_size=4,
            key_data='image',
            data_path='/Users/Seria/Desktop/nebulae/test/img/image.hdf5',
            width=200, height=200,
            resol_ratio=0.5,
            spatial_aug='brightness,gamma_contrast',
            p_sa=(0.5, 0.5), theta_sa=(0.2, 1.2))
```

You can edit properties to change the way you fetch batch and process data.

```
fd.propertyEdit('test-img', name='test', batch_size=2)
```

Passing a dictionary of changed parameters is equivalent.

```
config = {'name':'test', 'batch_size':2}
fd.propertyEdit('test-img', config=config)
```

Here are three useful functions:

stepsPerEpoch() returns how many steps you should take to iterate over all data.

currentEpoch() returns which epoch you are currently.

nextBatch() returns a dictionary containing a batch of data, labels and other information.

N.B. Sometimes you have no need to explicitly call the functions above unless you regard FuelDepot as an independent tool for your own use.

```
for s in range(fd.stepsPerEpoch('test')):
    batch = fd.nextBatch('test')
    print(fd.currentEpoch('test'), batch['label'])
```

Call unloadFuel(dataname) to unmount dataset named "dataname" on your FuelDepot.

```
fd.unloadFuel(name='test')
```