YOLOv3 Documentation

Release 0.5.4

Anthony DeGennaro

Mar 08, 2019

CONTENTS:

Project repository: https://github.com/adegenna/yolov3.

CHAPTER

ONE

INTRODUCTION

The following project is a Python implementation of the YOLOv3 object detection algorithm. This specific software began as a project intended for use with the Xview dataset specifically by Glenn Jocher at Ultralytics (https://github.com/ultralytics/xview-yolov3.git). It has since been modified extensively for the purposes of generality, maintainability, and usability by Anthony DeGennaro at Brookhaven National Laboratory (https://www.bnl.gov/compsci/people/staff.php?q=168 / adegennaro@bnl.gov).

You may access the main project repository at https://github.com/adegenna/xview-yolov3.

CHAPTER

TWO

REQUIREMENTS

This software requires Python 3.6, along with the following packages:

- numpy
- scipy
- scikit-learn
- matplotlib
- pytorch
- opencv
- h5py
- tqdm

CHAPTER

THREE

INSTALLATION

The purpose of this section is to provide detailed, step-by-step instructions on how to install Anaconda for Python virtual environments, the Pytorch framework, Nvidia GPU drivers, and the YOLOv3 project repository.

3.1 Anaconda

We first need to install Anaconda for Python virtual environments.

1. Download the Anaconda installer (shell script) from the Anaconda website:

```
>>> wget https://repo.anaconda.com/archive/Anaconda2-5.3.0-Linux-x86_64.sh
```

   This assumes you have a 64-bit Linux architecture. If you have something else, then visit https://www.anaconda.com/download/ and select your preferred version.

2. Launch the Anaconda installer:

```
>>> bash Anaconda2-5.3.0-Linux-x86_64.sh
```

   Accept the user terms and accept the default filepath for installation, which should be `/home/[user]/anaconda2/`.

3. Open your `/.bashrc` file in a file editor (e.g., `emacs /.bashrc`) and paste the following line to the end:

```
>>> source /home/[user]/anaconda2/etc/profile.d/conda.sh
```

4. Save the `/.bashrc` file, exit, and reload it in your terminal with:

```
>>> source /.bashrc
```

5. Confirm conda was installed:

```
>>> conda --version
```

   This should output the version of the Anaconda install, if successful

6. Create a custom Anaconda virtual environment for this project:

```
>>> conda create -n [envname] python=3.6 anaconda
```

   In the above, replace [envname] with your desired environment name (do not include the brackets)

7. To verify that this was successful, run:

```
>>> conda info --envs
```

If successful, [envname] should appear as one of the choices.

3.2 PyTorch

We will now install PyTorch, a Python deep-learning framework

1. Install PyTorch/Torchvision to your Anaconda environment:

```
>>> conda install -n [envname] pytorch torchvision -c pytorch
```

2. To verify that this was successful, activate your conda environment:

```
>>> conda activate [envname]
```

Then, check the PyTorch version with:

```
>>> python -c "import torch; print(torch.__version__)"
```

Also check the Torchvision version with:

```
>>> python -c "import torchvision; print(torchvision.__version__)"
```

If successful, both commands should output the installed versions.

3.3 GPU Support

Note: This section is only necessary if you have Nvidia GPU hardware, but have not yet installed drivers for it. Please be aware that installing Nvidia drivers can be tricky and should be handled with care. This section is a guide only; a thorough description of how to install GPU drivers is outside of the scope of this project.

Note: These instructions may require sudo priveleges.

Note: These instructions assume a Redhat OS. The equivalent process for another Linux OS (e.g., Ubuntu) is very similar.

1. Prepare your machine by installing necessary prerequisite packages:

```
>>> yum -y update

>>> yum -y groupinstall "Development Tools"

>>> yum -y install kernel-devel epel-release

>>> yum install dkms
```

2. Download desired Nvidia driver version from their archive at https://www.nvidia.com/object/unix.html (e.g., using wget from the terminal)
3. If your machine is currently using open-source drivers (e.g., noveau), you will need to change the configuration `/etc/default/grub` file. Open this file, find the line beginning with `GRUB_CMDLINE_LINUX` and add the following text to it:

```
nouveau.modeset=0
```

4. Reboot your machine
5. Stop all Xorg servers:

```
>>> systemctl isolate multi-user.target
```

6. Run the bash script installer:

```
>>> bash NVIDIA-Linux-x86_64-*
```

7. Reboot your system
8. Confirm that the installation was successful by inspecting the output of this command:

```
>>> nvidia-smi
```

If successful, this should display all Nvidia GPUs currently installed in your machine

3.4 YOLOv3

1. Activate your anaconda environment:

```
>>> conda activate [envname]
```

2. Clone the YOLOv3 git repo:

```
>>> git clone https://github.com/adegenna/yolov3
```

3. All of Python packages listed in the Requirements section of this documentation must be installed to your local conda environment. You may check whether the listed packages are installed with:

```
>>> conda list | grep [package]
```

4. If one of the required packages is missing, then install it; for example, install `opencv` with:

```
>>> conda install -n [envname] -c menpo opencv
```

CHAPTER

FOUR

USER INSTRUCTIONS

The purpose of this section is to provide detailed, step-by-step instructions on how to use the important interfaces for our codebase.

4.1 Importing YOLOv3

1. Make sure you activate the conda environment was described in the Installation section:

```
>>> conda activate [envname]
```

2. In your Python driver script, make sure that the yolov3 filepath is in the Python search path. For example, if the full filepath to the yolov3 repo is `/full/path/to/topdir/yolov3/`, then you would do this as:

```
import sys
sys.path.append('/full/path/to/topdir/')
```

3. The yolov3 project uses the Python `import` system to load all source code in `yolov3/src/` and `yolov3/utils/` in a heirarchical manner. You may therefore import this codebase as a module; for example:

```
import yolov3
darknet = yolov3.src.models.Darknet(<network_cfg_file> , <imgsize>)
```

4.2 Unit Tests

Unit tests are maintained in the `yolov3/tests` subdirectory. To run these, navigate to the directory that contains yolov3, and run the unittests as a Python package from the shell:

```
>>> python3 -m yolov3.tests.unittests
```

4.3 Training Example

An example driver script demonstrating proper interface usage for network training is provided in `scripts/train.py`, together with the `scripts/input_train.dat` input file.

Note: To run this example, certain options must be set in the inputfile. In particular, please ensure that the following pieces of data exist and are specified in the input file:

(1) `targetspath` : filepath to the target metadata file

(2) `traindir` : filepath to the training image dataset

(3) `networkcfg` : YOLOv3 network configuration file

(4) `class_path` : filepath to the class names/labels file

Please consult the Code Documentation on the InputFile class for further details on these and other necessary options.

To run this example, do the following:

1. Edit the `scripts/input_train.dat` input file so that those settings involving filepaths accurately reflect the data/filepaths on your machine.
2. Navigate to the directory that contains yolov3, and issue the following shell command to run the script as a package:

```
>>> python3 -m yolov3.scripts.train yolov3/scripts/input_train.dat
```

Note: Mulitple-GPU support is currently not available for any part of this software. Please run in either CPU or single-GPU mode only. If you have multiple GPUs on your machine, you may use the CUDA_VISIBLE_DEVICES flag to enforce single-GPU mode, e.g.:

```
>>> CUDA_VISIBLE_DEVICES=0 python3 -m yolov3.scripts.train yolov3/scripts/
↪input_train.dat
```

4.4 Detection Example

An example driver script demonstrating proper interface usage for network detection/testing is provided in `scripts/detect.py`, together with the `scripts/input_detect.dat` input file.

Note: To run this example, certain options must be set in the inputfile. For image detection only, set `plot_flag = False` and ensure that the following pieces of data exist and are specified in the input file:

(1) `imagepath` : filepath to image dataset

(2) `networkcfg` : YOLOv3 network configuration file

(3) `networksavefile` : PyTorch saved neural network file (.pt)

To additionally run with scoring, set `plot_flag = True` and ensure that these necessary options are specified in the input file as well:

(1) `targetspath` : filepath to the target metadata file

(2) `class_path` : filepath to the class names/labels file

(3) `class_mean` : filepath to training data class mean statistics

(4) `class_sigma` : filepath to training data class sigma statistics

For more details, consult the Code Documentation on the InputFile class.

To run this example, do the following:

1. Edit the `scripts/input_detect.dat` input file so that those settings involving filepaths accurately reflect the data/filepaths on your machine.
2. Navigate to the directory that contains yolov3, and issue the following shell command to run the script as a package:

```
>>> python3 -m yolov3.scripts.detect yolov3/scripts/input_detect.dat
```

Note: Mulitple-GPU support is currently not available for any part of this software. Please run in either CPU or single-GPU mode only. If you have multiple GPUs on your machine, you may use the CUDA_VISIBLE_DEVICES flag to enforce single-GPU mode, e.g.:

```
>>> CUDA_VISIBLE_DEVICES=0 python3 -m yolov3.scripts.detect yolov3/scripts/
↪input_detect.dat
```

CHAPTER

FIVE

CODE DOCS

5.1 Src/

class yolov3.src.InputFile.**InputFile**(*inputfile*)

Class for packaging all input/config file options together.

Note: There are separate options required by InputFile depending on whether the intended goal is training or testing. The user must declare on the first line of the InputFile either `[TRAIN]` or `[TEST]`, depending on their desired objective.

Inputs

inputfilename [string] String specifying the desired inputfile name.

Train Options

Below are a list of options that must be specified in an inputfile of type `[TRAIN]`:

option	type	description
loaddir	string	• Full path to load directory. • Any pre-trained YOLOv3 PyTorch file (.pt) goes here
outdir	string	• Full path to output directory.
targetspath	string	• Full path to target file. • Supported formats: `.geojson` • **[See notes below]**
targetfiletype	string	• Type of target file. • Supported options: `json`
traindir	string	• Full path to training image dataset • Supported image types: `.tif`, `.bmp` • **[See notes below]**
epochs	int	• Number of training epochs.
epochstart	int	• Starting epoch.
batchsize	int	• Training batch size.
networkcfg	string	• Full path to YOLOv3 network architecture file. • Base template in `yolov3/cfg/yolov3_template.cfg` • **[See notes below]**
imgsize	int	• Base image chip size. • Must be multiple of 32
resume	bool	• Specifies whether training is resuming from previous training.

option	type	description
invalid_class_list	string	• .csv list of classes to be ignored from training data. • **[See notes below]**
boundingboxclusters	int	• Desired number of bounding-box clusters for the YOLO architecture.
computeboundingboxclusters	bool	• Specifies whether to compute bounding box clusters.
class_path	string	• Full path to class names/labels file. • **[See notes below]**
sampling_weight	string	• String specifying type of sampling weight. • Options are `inverse_class_frequency` and `uniform` • **[See notes below]**

Notes on `[TRAIN]` options above:

1. `traindir` : If `.tif` images are used in the training data directory, then a .bmp copy is produced to be used in training.

2. `targetspath` : Currently, the target metadata file must be formatted in .json format, similar to the xView .geojson format. Most important is that the target file must be compatible with the `yolov3.utils.get_labels_geojson( )` function, which provides the implementation for retrieving object coordinates, corresponding filenames, and classes. Please consult the documentation for that function for further details.

3. `networkcfg` : You have two options here: you may provide a complete YOLOv3 architecture file, or provide a network template file and request that the software precompute the architecture for you, prior to training. We strongly recommend the latter option. To do this, provide the full filepath to the generic template file located in `yolov3/cfg/yolov3_template.cfg`, set `computeboundingboxclusters = True` in the inputfile, and provide the desired number of bounding box clusters (i.e. YOLO anchors) in the `boundingboxclusters` argument of the inputfile. This will tell the software to precompute bounding box priors (anchors) for the training dataset, and a custom architecture file will be calculated and outputted to `yolov3/cfg/yolov3_custom.cfg`, which may be used later in detection/testing.

4. `invalid_class_list` : This option was added to give the user the ability to specify a list of classes referred to in the `targetspath` metadata file that either are not present or need to be excluded from the training dataset. For example, the xView dataset makes reference to classes 75 and 82 in the .geojson target file, but these are `None` classes. If there are no "invalid" classes in your target metadata file, then simply leave this option blank.

5. `class_path` : This file is a comma-separated list of all classes and any associated numeric labels. For example, the xView dataset contains 60 classes, with associated labels ranging from 11 to 94. Thus, the `class_path` file for the xView dataset would be a 60-line .csv file that would look as follows:

```
Fixed-wing Aircraft , 11
Small Aircraft , 12
Cargo Plane , 13
...
Tower , 94
```

6. `sampling_weight` : This option sets how images are weighted for random selection at runtime during the training routine. Options are `inverse_class_frequency` and `uniform`. The former weights an image by the sum of the inverse of the class frequencies of all its objects; the latter weights all images uniformly.

Test Options

Below are a list of options that must be specified in an inputfile of type `[TEST]`:

option	data type	meaning
loaddir	string	• Full path to load directory.
outdir	string	• Full path to output directory.
targetspath	string	• Full path to target file. • Only needed for scoring the object detections.
targetfiletype	string	• Type of target file. • Supported options: `json`
imagepath	string	• Full path to directory containing images.
plot_flag	bool	• Flag to indicate whether to plot and output object detections. • **[See notes below]**
networkcfg	string	• Full path to network architecture file. • **[See notes below]**
networksavefile	string	• Full path to trained YOLOv3 network file. • **[See notes below]**
class_path	string	• Full path to .csv file containing list of classes/labels.
conf_thres	float	• Confidence threshold for detection.
cls_thres	float	• Class threshold for detection.
nms_thres	float	• NMS threshold.
batch_size	int	• Batch size.

option	data type	meaning
imgsize	int	• Desired chip size.
rgb_mean	string	• Full path to dataset RGB mean file. • **[See notes below]**
rgb_std	string	• Full path to dataset RGB standard deviation file. • **[See notes below]**
class_mean	string	• Full path to class mean file. • **[See notes below]**
class_sigma	string	• Full path to class standard deviation file. • **[See notes below]**
invalid_class_list	string	• Comma-separated list of classes to be ignored from training data.

Notes on the testing inputfile:

1. `targetspath` , `invalid_class_list` , `imgsize` , `class_path` : Same notes apply as in the training case above.
2. `imagepath` : This option sets the full filepath to the location on your machine where your test dataset resides. There should be nothing in this directory except the test image files. Currently supported image files are `.tif` and `.bmp`.
3. `networkcfg` : This option specifies the full filepath to a trained YOLOv3 configuration file. If you used the recommended input to this option in the training stage, then the code will have produced this file for you, saved as `cfg/yolov3_custom.cfg`. Otherwise, you will have to fill in the YOLO anchors yourself directly into a copy of the template file.
4. `networksavefile` : This option specifies the full filepath to the PyTorch savefile (.pt extension) that contains all weights for the trained network.
5. `rgb_mean` , `rgb_std` : These files contain RGB statistics that were computed on the training dataset by the training routine. Each of them is simply a 3-line file, where each line contains a single numeric value that is the mean (or standard deviation) of the respective RGB channel. These values are used to normalize any data that is fed into the network.
6. `class_mean` , `class_std` : These files contain class statistics that were computed on the training dataset by the training routine. Each of these files contains N-lines, where N is the number of classes, and each line contains a comma-separated list of 4 values, corresponding to the mean (or standard deviation) of the width, height, area, and aspect ratio (in that order) of the respective class objects. These statistics

are used as prior information to reduce false positives in the object detection stage.

7. `plot_flag` : This option specifies whether you would like to score the object detections that are calculated.

Examples

To use this class, follow this interface:

```
input_file_object = InputFile('/full/path/to/input_file.dat')
```

For the `[TRAIN]` case, here is an example of what `input_file.dat` might contain:

```
[TRAIN]
loaddir          = /full/path/to/loaddir/
outdir           = /full/path/to/outdir/
targetspath      = /full/path/to/targetsdir/xView_train.geojson
targetfiletype   = json
traindir         = /full/path/to/traindir/
epochs           = 300
epochstart       = 0
batchsize        = 8
networkcfg       = /full/path/to/networkdir/yolov3_template.cfg
imgsize          = 800
resume           = False
invalid_class_list         = 75,82
boundingboxclusters        = 30
computeboundingboxclusters = False
class_path       = /full/path/to/xview_names_and_labels.csv
sampling_weight  = inverse_class_frequency
```

For the `[TEST]` case, here is an example of what `input_file.dat` might contain:

```
[TEST]
loaddir              = /full/path/to/loaddir/
outdir               = /full/path/to/outdir/
targetspath          = /full/path/to/targetdir/xView_train.geojson
targetfiletype       = json
imagepath            = /full/path/to/testdir/
plot_flag            = True
networkcfg           = /full/path/to/networksavedir/yolov3_custom.cfg
networksavefile      = /full/path/to/networksavedir/best.pt
class_path           = /full/path/to/classpathdir/xview_names_and_labels.csv
conf_thres           = 0.99
cls_thres            = 0.05
nms_thres            = 0.4
batch_size           = 1
imgsize              = 1632
rgb_mean             = /full/path/to/statdir/training_rgb_mean.out
rgb_std              = /full/path/to/statdir/training_rgb_std.out
class_mean           = /full/path/to/statdir/training_class_mean.out
class_sigma          = /full/path/to/statdir/training_class_sigma.out
invalid_class_list   = 75,82
```

printInputs()
Method to print all config options.

class yolov3.src.models.**Darknet**(*networkcfg*, *imgsize*)
YOLOv3 object detection model. This class, the modules that comprise its architecture, and its interface are inherited from `torch.nn.Module`.

Inputs

networkcfg [string] Absolute path to YOLOv3 network architecture file.

imgsize [int] Desired cropped image size

Member Variables

losses [dict, only created if in training mode] Dictionary containing quantities on training loss

key	meaning
'loss'	total value of training loss.
'x'	bounding box x-position loss.
'y'	bounding box y-position loss.
'w'	bounding box width loss.
'h'	bounding box height loss.
'conf'	objectness confidence loss.
'cls'	object classification loss.
'nGT'	number of ground truths.
'TP'	number of true positives.
'FP'	number of false positives.
'FN'	number of false negatives.
'FPe'	number of false positives in each class.

Examples

Here is an example of how this class may be instantiated:

```
networkcfg = '/full/path/to/network_cfg_file.dat'
imgsize    = 800
darknet    = Darknet(networkcfg, imgsize)
```

`network_cfg_file.dat` is a configuration file that specifies a valid YOLOv3 architecture. Please consult the `cfg/` subdirectory of the main project repo for examples.

The interface to forward-pass an image through the network uses the recipe implemented in the `forward()` routine and takes an image as input and returns the network output as a result:

```
output = darknet(image)
```

If you are training this network, you would also provide corresponding image targets, and possibly other inputs defined in the `forward` method:

```
output = darknet(image,target)
```

Typically, the images/targets are provided by a dataloader like `ListDataset` that produces iterable pairs of images/targets, which would look like this:

```
for i , (image_i , target_i) in enumerate(dataloader):
    output_i = darknet(image_i,target_i)
```

During training, a detection loss would be calculated in the `YOLOLayer` submodules and stored as the `losses` dictionary member, whose members may be accessed directly, e.g.:

```
total_loss = darknet.losses['loss']
```

Other functionality is inherited directly from the `torch.nn.Module` module of PyTorch, so consult those documents for assistance in using it.

forward(*x*, *targets=None*, *requestPrecision=False*, *weight=None*, *epoch=None*)
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 yolov3.src.models.**EmptyLayer**
Placeholder for 'route' and 'shortcut' layers

class yolov3.src.models.**YOLOLayer**(*anchors*, *nC*, *img_dim*, *anchor_idxs*)

forward(*p*, *targets=None*, *requestPrecision=False*, *weight=None*, *epoch=None*)
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.

---

yolov3.src.models.**create_modules**(*module_defs*)
Constructs module list of layer blocks from module configuration in module_defs

yolov3.src.models.**create_yolo_architecture**(*inputs*, *n_classes*, *anchor_coordinates*)
Creates a yolo-v3 layer configuration file from desired options

Inputs

inputs [InputFile object] Specifies some necessary user options.

n_classes [int] Specifies number of classes in the dataset

anchor_coordinates [list<double>] List of doubles of form [x1,y1,x2,y2, . . . , xN,yN] where N = number of anchors and (xi,yi) are the i'th anchor coordinates.

Outputs

output_config_file_path [string] Absolute filepath of the network config file created by this function

yolov3.src.models.**create_yolo_config_file**(*template_file_path*, *output_config_file_path*, *n_anchors*, *n_classes*, *anchor_coordinates*)
Creates a yolo-v3 layer configuration file from desired options

yolov3.src.models.**parse_model_config**(*path*)
Parses the yolo-v3 layer configuration file and returns module definitions

yolov3.src.models.**read_yolo_config_file_anchors**(*cfg_path*)
Reads the anchor coordinates from a specified YOLO configuration file

class yolov3.src.NetworkTester.**NetworkTester**(*model*, *dataloader*, *inputs*)
Class for handling testing and assessing the performance of a trained YOLOv3 model. | **Inputs:** | *model:* trained YOLOv3 network (PyTorch .pt file). | *dataloader:* dataloader object (usually an instantiation of the ImageFolder class) | *inputs:* input file with various user-specified options

detect()
Method to compute object detections over testing dataset

loadClasses()
Method to load class names from specified path in user-input file. Format assumed shall be a csv list of (class_name , class_label)_i

loadSavedModels()
Method to load a saved YOLOv3 model from a PyTorch (.pt) file.

plotDetection()
Method to plot and display all detected objects in the testing dataset

setupCuda()
Basic method to setup GPU/cuda support, if available

class yolov3.src.targets.Target.**Target**(*inputs*)
Class for handling target pre-processing tasks.

apply_mask_to_filtered_data()
Method to apply mask to filtered data variables.

compute_bounding_box_clusters_using_kmeans(*n_clusters*)

Method to compute bounding box clusters using kmeans.

Inputs:

n_clusters: number of desired kmeans clusters

compute_class_weights_with_filtered_data()
Method to compute class weights from filtered data. Weight is simply inverse of class frequency.

compute_cropped_data()
Method to crop image data based on the width and height. Filtered variables are then computed based on the updated image coordinates.

compute_filtered_data_mask()
Method to compute filtered data by applying several filtering operations.

compute_filtered_variables_from_filtered_coords()
Method to compute filtered variables from filtered coordinates.

compute_filtered_variables_from_filtered_xy()
Method to compute filtered variables from filtered xy.

compute_image_weights_with_filtered_data()
Method to compute image weights from filtered data. Weight for a given image is the sum of the class weights for each of the objects present in that given image.

count_number_of_nonexistent_chips()
Method to count the number of chips that exist in the target metadata file, but not in the actual database. Returns: number of nonexistent objects, and number of nonexistent files

detect_nonexistent_chip(*chip_i*)
Method to detect all instances in database of a chip that does not exist

edge_requirements(*w_lim*, *h_lim*, *x2_lim*, *y2_lim*)
Method to compute filtering based on edge specifications.

Inputs:

w_lim: limit for image width
h_lim: limit for image height
x2_lim: limit for image x2
y2_lim: limit for image y2

Outputs:

indices where filtered variables satisfy the dimension requirements.

load_target_file()
Method to load a targetfile of type specified in the input file. Supported types: .json.

manual_dimension_requirements(*area_lim*, *w_lim*, *h_lim*, *AR_lim*)
Method to compute filtering based on specified dimension requirements.

Inputs:

area_lim: limit for image area
w_lim: limit for image width
h_lim: limit for image height
AR_lim: limit for image aspect ratio

Outputs:

indices where filtered variables satisfy the dimension requirements.

output_data_for_listdataset()
Method to output data needed for the ListDataset dataloader.

Outputs

object_data [list] list containing three pieces of data on all objects: [filtered_chips,filtered_coords,filtered_classes]

class_weights [array] array containing the weights for each class

image_weights [array] array containing the weights for each image

files [list] list of all files

process_target_data()
Method to perform all target processing.

read_list_of_class_names_and_labels()
Method to read in the user-provided list of class names and associated numeric labels.

remove_nonexistent_chips_from_database(*idx_nonexistent*)
Method to remove all nonexistent chips from database

set_image_w_and_h()
Method to set width and height of images associated with targets.

sigma_rejection_indices(*filtered_data*)
Method to compute a mask based on a sigma rejection criterion.

Inputs:

filtered_data: data to which sigma rejection is applied and from which mask is computed

Outputs:

mask_reject: binary mask computed from sigma rejection

strip_image_number_from_chips_and_files()
Method to strip numbers from image filenames from both chips and files.

yolov3.src.targets.fcn_sigma_rejection.**fcn_sigma_rejection**(*x*, *srl=3*, *ni=3*)
Function to perform sigma rejection on a dataset.

Inputs:

x: dataset
srl: desired cutoff number of standard deviations for rejection
ni: desired number of iterations

Outputs:

x: dataset with outliers removed
inliers: indices of inliers w.r.t. original dataset

yolov3.src.targets.per_class_stats.**per_class_stats**(*classes*, *w*, *h*)
Function to calculate statistics of target data.

Inputs:

classes: target data processed/produced with the Target class
w: image width
h: image height

Outputs:

class_mu: mean of target classes
class_sigma: standard deviation of target classes
class_cov: covariance of target classes

class yolov3.src.datasets.datasets.**ListDataset**(*inputs*, *object_data*, *class_weights*, *image_weights*, *files*)
Image dataset class for training

Inputs

inputs [InputFile] InputFile object containing user-specified config options

object_data [list] list containing three pieces of data on all objects: [chips,coords,classes]

class_weights [array] array containing the weights for each class

image_weights [array] array containing the weights for each image

files [list] list of all files

yolov3.src.datasets.datasetStats.**compute_dataset_rgb_stats**(*dataloader_files*)
Function to compute rgb statistics of a given dataset. Uses a two-pass sequential algorithm.

Inputs:
dataloader_files: list of absolute paths of all dataset files

Outputs:
(mean_rgb,std_rgb): mean and standard deviation of RGB channels of dataset

yolov3.src.datasets.datasetStats.**compute_dataset_rgb_stats_load_into_memory**(*dataloader_files*)
Function to compute rgb statistics of a given dataset. Loads all data into memory and computes statistics in one-shot.

Inputs:
dataloader_files: list of absolute paths of all dataset files

Outputs:
(mean_rgb,std_rgb): mean and standard deviation of RGB channels of dataset

yolov3.src.datasets.datasetTransformations.**augmentHSV**(*img0*)
Function to perform HSV augmentation (by a random factor of +/- 50%)

Inputs:
img0: dataset image loaded by OpenCV

Outputs:
img: transformed image

yolov3.src.datasets.datasetTransformations.**pickRandomPoints**(*pts*, *img0*, *height*, *M*, *img1*)
Function to select random points of a specified chip size from a specified transformed image

Inputs:
pts: number of desired random points
img0: dataset image loaded by OpenCV
height: desired chip size
M: random affine transformation to use (calculated with random_affine)
img1: transformed version of img0 (calculated with random_affine applied to img0)

Outputs:

r: random points from specified image img0, transformed with the same random affine mapping used to take img0 to img1

yolov3.src.datasets.datasetTransformations.**random_affine**(*img*, *targets=None*, *degrees=(-10, 10)*, *translate=(0.1, 0.1)*, *scale=(0.9, 1.1)*, *shear=(-3, 3)*, *borderValue=(0, 0, 0)*)

Function to performs a random affine transformation on a specified image/target combination. See https://medium.com/uruvideo/dataset-augmentation-with-random-homographies-a8f4b44830d4 for a general discussion.

Inputs:

img: dataset image loaded by OpenCV

targets: a target from a ListDataset object

degrees: min/max range of possible degrees of rotation

translate: max possible values for scaling, specified as a percentage of the vertical and horizontal dimensions of img

scale: min/max range of possible values for scaling (specified such that no scaling = 1)

shear: min/max range of possible values of degrees for shearing

borderValue: triplet specifying fill values for image borders

Outputs:

imw: transformed image

targets: transformed targets (if targets is not None)

M: affine transformation used (if targets is not None)

yolov3.src.datasets.datasetTransformations.**resize_square**(*img*, *height=416*, *color=(0, 0, 0)*)

Function to resize a rectangular image to a padded square

Inputs:

img: dataset image loaded by OpenCV

height: desired image height

color: triplet specifying fill values for image borders

Outputs:

img: transformed image

Utility methods for computing the performance metrics.

yolov3.src.scoring.evaluation.**compute_average_precision_recall**(*groundtruth_coordinates*, *coordinates*, *iou_threshold*)

Computes the average precision (AP) and average recall (AR).

Args:

groundtruth_info_dict: the groundtruth_info_dict holds all the groundtruth information for an evaluation dataset. The format of this groundtruth_info_dict is as follows: {'image_id_0':

[xmin_0,ymin_0,xmax_0,ymax_0,...,xmin_N0,ymin_N0,xmax_N0,ymax_N0], ..., 'image_id_M': [xmin_0,ymin_0,xmax_0,ymax_0,...,xmin_NM,ymin_NM,xmax_NM,ymax_NM]},

where image_id_* is an image_id that has the groundtruth rectangles labeled. xmin_*,ymin_*,xmax_*,ymax_* is the top-left and bottom-right corners

of one groundtruth rectangle.

test_info_dict: the test_info_dict holds all the test information for an

evaluation dataset. The format of this test_info_dict is the same

as the above groundtruth_info_dict.

iou_threshold_range: the IOU threshold range to compute the average precision (AP) and average recall (AR). For example: iou_threshold_range = [0.50:0.05:0.95]

Returns: average_precision, average_recall, as well as the precision_recall_dict, where precision_recall_dict holds the full precision/recall information for each of the iou_threshold in the iou_threshold_range.

Raises: ValueError: if the input groundtruth_info_dict and test_info_dict show inconsistent information.

yolov3.src.scoring.evaluation.**compute_average_precision_recall_given_precision_recall_dict**

Computes the average precision (AP) and average recall (AR).

Args: precision_recall_dict: the precision_recall_dict holds the dictionary of precision and recall information returned by the compute_precision_recall_given_image_statistics_list method, which is calcualted under a range of iou_thresholds, where the iou_threshold is the key.

Returns: average_precision, average_recall.

yolov3.src.scoring.evaluation.**compute_precision_recall_given_image_statistics_list**(*iou_thresh*
im-
age_statist

Computes the precision recall numbers given iou_threshold and statistics.

Args: iou_threshold: the iou_threshold under which the statistics are computed. image_statistics_list: a list of the statistics computed and returned by the compute_statistics_given_rectangle_matches method for a list of images.

Returns: A dictionary holding the precision, recall as well as the inputs.

yolov3.src.scoring.evaluation.**compute_statistics_given_rectangle_matches**(*groundtruth_rects_matche*
rects_matched)

Computes the staticstics given the groundtruth_rects and rects matches.

Args: image_id: the image_id referring to the image to be evaluated. groundtruth_rects_matched: the groundtruth_rects_matched represents

a list of integers returned from the Matching class instance to indicate the matched rectangle indices from rects for each of the groundtruth_rects.

rects_matched: the rects_matched represents a list of integers returned from the Matching class instance to indicate the matched rectangle indices from groundtruth_rects for each of the rects.

Returns: A dictionary holding the computed statistics as well as the inputs.

yolov3.src.scoring.evaluation.**convert_to_rectangle_list**(*coordinates*)
Converts the coordinates in a list to the Rectangle list.

yolov3.src.scoring.evaluation.**safe_divide**(*numerator*, *denominator*)
Computes the safe division to avoid the divide by zero problem.

class yolov3.src.scoring.matching.**Matching**(*groundtruth_rects*, *rects*)
Matching class.

yolov3.src.scoring.matching.**cartesian**(*arrays*, *out=None*)
Generate a cartesian product of input arrays.

arrays [list of array-like] 1-D arrays to form the cartesian product of.

out [ndarray] Array to place the cartesian product in.

out [ndarray] 2-D array of shape (M, len(arrays)) containing cartesian products formed of input arrays.

class yolov3.src.scoring.rectangle.**Rectangle**(*xmin*, *ymin*, *xmax*, *ymax*)
Rectangle class.

area()
Returns the area of the Rectangle instance.

contains(*x*, *y*)
Tests if a point is inside or on any of the edges of the rectangle.

height()
Returns the height of the Rectangle instance.

intersect(*other*)
Returns the intersection of this rectangle with the other rectangle.

intersect_over_union(*other*)
Returns the intersection over union ratio of this and other rectangle.

intersects(*other*)
Tests if this rectangle has an intersection with another rectangle.

is_empty()
Determines if the Rectangle instance is valid or not.

width()
Returns the width of the Rectangle instance.

yolov3.src.scoring.score.**score**(*opt*, *iou_threshold=0.5*)
Compute metrics on a number of prediction files, given a folder of prediction files and a ground truth. Primary metric is mean average precision (mAP).

Inputs

opt [InputFile] InputFile member specifying all user options.

Note: Prediction files in opt.outdir will have filename format 'XYZ.tif.txt', where 'XYZ.tif' is the .tif file used for prediction. Prediction files should be in space-delimited csv format; each line appears as: (xmin ymin xmax ymax class_prediction score_prediction).

---

iou_threshold [float] iou threshold (between 0 and 1) indicating the percentage iou required to count a prediction as a true positive

Outputs

opt.outdir/metrics.txt [text file] contains the scoring metrics in per-line format (metric/class_num: score_float)

Raises

ValueError Raised if there are files in the prediction folder that are not in the ground truth file. For example, a prediction file is titled '15.tif.txt', but the file '15.tif' is not in the ground truth.

class yolov3.src.scoring.scoringfunctions.**ScoringData**
Structure to package various intermediate data/calculations together for scoring purposes.

yolov3.src.scoring.scoringfunctions.**ap_from_pr**(*p*, *r*)
Calculates AP from precision and recall values as specified in the PASCAL VOC devkit.

Args: p: an array of precision values r: an array of recall values

Outputs: An average precision value

yolov3.src.scoring.scoringfunctions.**convert_to_rectangle_list**(*coordinates*)
Converts a list of coordinates to a list of rectangles

Args:

coordinates: a flattened list of bounding box coordinates in format (xmin,ymin,xmax,ymax)

Outputs: A list of rectangles

5.2 Utils/

yolov3.utils.datasetProcessing.**determine_common_and_rare_classes**(*opt*)
Function to determine the common and rare classes in a dataset using 2-means.

yolov3.utils.datasetProcessing.**determine_number_of_class_members**(*opt*)
Function to determine the number of elements in each class of a dataset.

yolov3.utils.datasetProcessing.**determine_small_medium_large_classes**(*opt*)
Function to determine the small/medium/large size classes in a dataset using 3-means.

yolov3.utils.datasetProcessing.**get_labels_geojson**(*fname='xView_train.geojson'*)
Processes a WorldView3 GEOJSON file

Args: fname: filepath to the GeoJson file.

Outputs: Bounding box coordinate array, Chip-name array, and Classes array

yolov3.utils.utils.**assert_single_gpu_support**()
Function to check that only a single GPU is being used. Currently, all software must be run with a single GPU only, so this routine does a simple assert check on the environment variable that ensures this.

yolov3.utils.utils.**bbox_iou**(*box1*, *box2*, *x1y1x2y2=True*)
Returns the IoU of two bounding boxes

yolov3.utils.utils.**build_targets**(*pred_boxes*, *pred_conf*, *pred_cls*, *target*, *anchor_wh*, *nA*, *nC*, *nG*, *requestPrecision*)
returns nGT, nCorrect, tx, ty, tw, th, tconf, tcls

yolov3.utils.utils.**compute_ap**(*recall*, *precision*)
Compute the average precision, given the recall and precision curves. Code originally from https://github.com/rbgirshick/py-faster-rcnn. # Arguments

recall: The recall curve (list). precision: The precision curve (list).

 # Returns The average precision as computed in py-faster-rcnn.

yolov3.utils.utils.**convert_class_labels_to_indices**(*class_labels*, *unique_class_labels*)
Function that takes a list of N class labels and the list of all M<N unique class labels and returns a list of size N, where each entry is the index of the corresponding label in the list of unique class labels. For example, given class_labels = [34,89,34,34,11] and unique_class_labels = [11,34,89], the output = [1,2,1,1,0].

yolov3.utils.utils.**convert_tif2bmp**(*p*)
Function to convert .tif –> .bmp

Inputs:

p: Absolute path to the dataset directory

yolov3.utils.utils.**load_classes**(*xview_names_and_labels_filepath*)
Loads class labels at 'xview_names_and_labels_filepath' Format shall be assumed to be csv where one line is (name , label)_i

yolov3.utils.utils.**readBmpDataset**(*path*)
Function to read a .bmp dataset. If the provided directory does not contain .bmp files, a conversion is attempted.

Inputs:

path: Absolute path to the dataset directory

yolov3.utils.utils.**zerocenter_class_indices**(*classes*)
This function takes a list of N elements with M<N unique labels, and relabels them such that the labels are 0,1,. . . ,M-1. Note that this function assumes that all class labels of interest appear at least once in classes.

Inputs:

classes: N-list of original class indices.

Outputs:

classes_zeroed: N-list of classes relabeled such that the labels are 0...M-1

e.g., [5,9,7,12,7,9] –> [0,2,1,3,1,2]

5.3 Tests/

class yolov3.tests.unittests.**DataProcessingTests**(*methodName='runTest'*)

Class for all data processing unit tests.

setUp()

Basic setup method. Note that ResourceWarnings and DeprecationWarnings are ignored.

test_get_dataset_filenames()

Test loading of dataset filenames.

test_get_dataset_height_width_channels()

Test loading sizes of dataset images.

test_get_labels_geojson()

Test loading of geojson formatted data.

test_strip_image_number_from_filename()

Test functionality to strip image number from image filename.

class yolov3.tests.unittests.**GPUtests**(*methodName='runTest'*)

Class for all GPU/cuda unit tests.

setUp()

Basic setup method. Note that ResourceWarnings and DeprecationWarnings are ignored.

test_cuda_available()

Test whether cuda is available.

test_cuda_version()

Test that cuda version is >= 9.

test_gpu_avail()

Test that GPU hardware is available.

class yolov3.tests.unittests.**ModelsTests**(*methodName='runTest'*)

Class for all models-involved unit tests.

setUp()

Basic setup method. Note that ResourceWarnings and DeprecationWarnings are ignored.

test_create_yolo_config_file()

Test functionality to create custom YOLOv3 config file from a template.

class yolov3.tests.unittests.**TargetTests**(*methodName='runTest'*)

Class for all target-involved unit tests.

setUp()

Basic setup method. Note that ResourceWarnings and DeprecationWarnings are ignored.

test_apply_mask_to_filtered_data()

Test mask application to filtered data method.

test_area_requirements()

Test area requirements method.

test_compute_bounding_box_clusters_using_kmeans()
Test bounding box cluster computation method.

test_compute_cropped_data()
Test functionality for cropping targets.

test_compute_image_weights_with_filtered_data()
Test class weight computation method.

test_compute_width_height_area()
Test functionality for computing target coordinate area.

test_edge_requirements()
Test functionality for computing edge requirements on target data.

test_fcn_sigma_rejection()
Test functionality for computing sigma rejection.

test_invalid_class_requirement()
Test invalid class requirement method.

test_load_target_file()
Test functionality for loading target data (.json file).

test_manual_dimension_requirements()
Test functionality for imposing manual dimensions requirements on target data.

test_nan_inf_size_requirements()
Test nan/inf/size requirements method.

test_per_class_stats()
Test per_class_stats function.

test_sigma_rejection_indices()
Test functionality for computing sigma rejection indices.

test_xy_coords()
Test functionality for target coordinate parsing.

5.4 Scripts/

yolov3.scripts.train.**main**()
Main driver script for training the YOLOv3 network.

Inputs

args [command line arguments] Command line arguments used in shell call for this main driver script. args must have a inputfilename member that specifies the desired inputfile name.

Outputs

inputs.outdir/results.txt [text file] output metrics for each training epoch

inputs.loaddir/latest.pt [YOLOv3 network PyTorch save file] checkpoint file for latest network configuration

inputs.loaddir/best.pt [YOLOv3 network PyTorch save file] checkpoint file for best current network configuration

inputs.loaddir/backup.pt [YOLOv3 network PyTorch save file] checkpoint file for backup purposes

yolov3.scripts.detect.**detect**()

Main driver script for testing the YOLOv3 network.

Inputs

args [command line arguments] command line arguments used in shell call for this main driver script. args must have a inputfilename member that specifies the desired inputfile name.

Outputs

inputs.outdir/metrics.txt [text file] output metrics for specified test image given by inputs.imagepath

inputs.loaddir/<inputs.imagepath>.jpg [jpeg image] test image with detected bounding boxes, classes and confidence scores

inputs.loaddir/<inputs.imagepath>.tif.txt [text file] text file with bounding boxes, classes and confidence scores for all detections

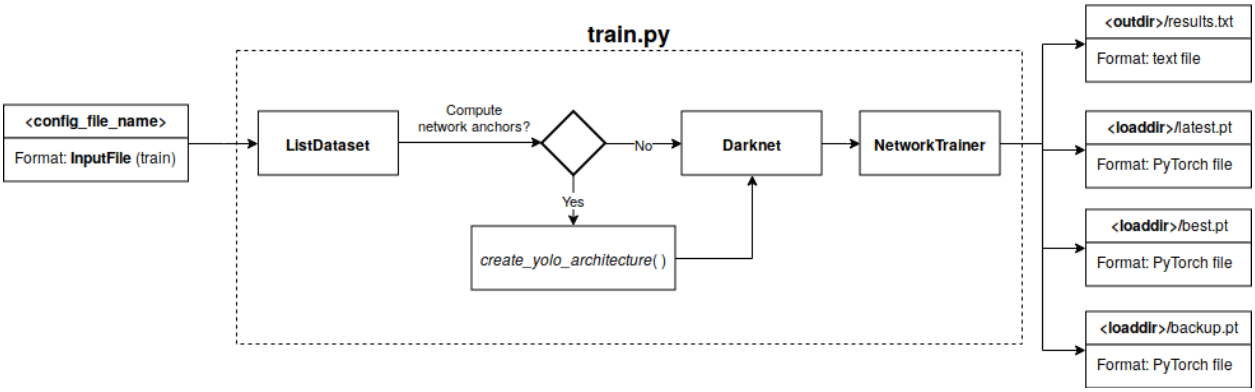
CHAPTER

SIX

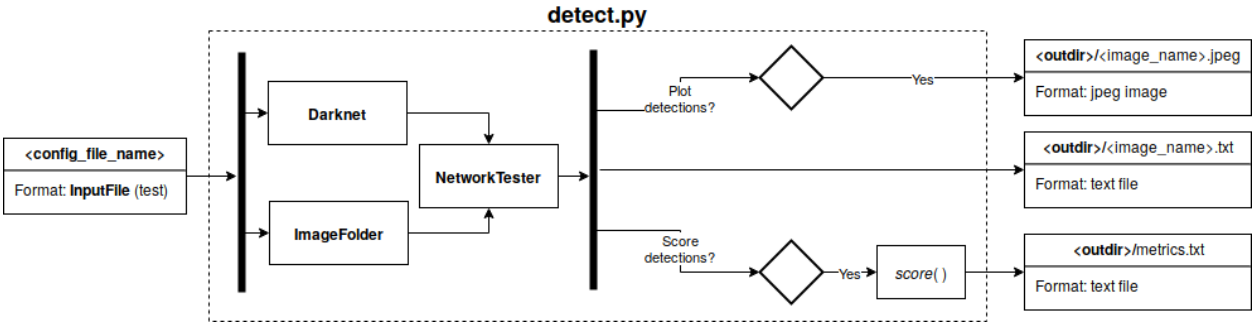
FLOW DIAGRAMS

Below are a collection of flow diagrams for all code in Src/.

6.1 train



6.2 detect



CHAPTER

SEVEN

CONTACT

Any questions/comments may be directed to the main BNL project developer, Anthony DeGennaro (https://www.bnl.gov/compsci/people/staff.php?q=168 / adegennaro@bnl.gov).

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

y

INDEX

T

W

Y

Z