YOLOv3 Documentation

Release 0.5

Anthony DeGennaro

Feb 09, 2019

CONTENTS:

Project repository: https://github.com/adegenna/xview-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
- sklearn
- matplotlib
- torch
- opencv-python
- h5py
- tqdm

CHAPTER

THREE

INSTALLATION

The purpose of this document is to provide detailed, step-by-step instructions on how to install Pytorch, YOLOv3, and all associated dependencies.

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
   ```

   Note: 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

If you have Nvidia GPU hardware but do not have the drivers installed, you may do so as follows. If you already have Nvidia drivers installed, skip this. Note: this may require sudo priveleges. Also, the following 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

Note: For now, we are simply using a version of YOLOv3 freely available on Github. We plan to fork this and modify it as needed. For now, we only describe the installation directions for the community-available version of YOLOv3.

1. Activate your anaconda environment:

```
conda activate [envname]
```

2. Clone the YOLOv3 git repo:

```
git clone https://github.com/adegenna/xview-yolov3
```

3. Navigate to the project directory (xview-yolov3) and open the file requirements.txt. All of Python packages listed there must be installed to your local conda environment. 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-python with:

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

CHAPTER

FOUR

CODE DOCS

4.1 Src/

src.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

src.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

class src.InputFile.**InputFile**(*args=[]*)
Class for packaging all input/config file options together.

Inputs

args [command line arguments] (passed to constructor at runtime) command line arguments used in shell call for main driver script. args must have a inputfilename member that specifies the desired inputfile name.

Options

inputtype [string] Options are *train* or *detect*

projdir [string] Absolute path to project directory

datadir [string] Absolute path to data directory

loaddir [string] Absolute path to load directory

outdir [string] Absolute path to output directory

targetspath [string] Absolute path to target file

targetfiletype [string] Type of target file

traindir [string] Type of target file

Options (Train-Specific)

traindir [string] Type of target file

epochs [int] Number of training epochs

epochstart [int] Starting epoch

batchsize [int] Training batch size

networkcfg [string] Network architecture file

imgsize [int] Base image crop size

resume [bool] Boolean value specifying whether training is resuming from previous iteration

invalid_class_list [string (csv format)] Comma-separated list of classes to be ignored from training data

boundingboxclusters [int] Desired number of bounding-box clusters for the YOLO architecture

computeboundingboxclusters [bool] Boolean value specifying whether to compute bounding box clusters

Options (Detect-Specific)

imagepath [string] Image path

plotflag [bool] Flag for plotting

secondary_classifier [bool] Boolean value specifying whether to use a secondary classifier

networkcfg [string] Network architecture file

networksavefile [string] Absolute path to trained YOLOv3 network file, saved by PyTorch (.pt)

class_path [string] Absolute path to class

conf_thres [float] Confidence threshold for detection

nms_thres [float] NMS threshold

batch_size [int] Desired batchsize

img_size [int] Desired cropped image size

rgb_mean [string] Absolute path to dataset RGB mean file

rgb_std [string] Absolute path to dataset RGB standard deviation file

class_mean [string] Absolute path to class mean file

class_sigma [string] Absolute path to class standard deviation file

invalid_class_list [string (csv format)] Comma-separated list of classes to be ignored from training data

printInputs()
Method to print all config options.

readDetectInputfile(*inputfilestream*)
Method to read config options from a detection inputfile

Inputs:
inputfilestream: specified inputfilestream.

readTrainingInputfile(*inputfilestream*)
Method to read config options from a training inputfile.

Inputs:
inputfilestream: specified inputfilestream.

class src.models.**ConvNetb**(*num_classes=60*)

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 src.models.**Darknet**(*inputs*)
YOLOv3 object detection model

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 src.models.**EmptyLayer**
Placeholder for 'route' and 'shortcut' layers

class 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.

---

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

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

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

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

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

class 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.

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 src.targets.**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.

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.

process_target_data()
Method to perform all target processing.

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.

src.targets.**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

src.targets.**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 src.datasets.**ListDataset**(*inputs*)
Image dataset class for training

src.datasets.**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

src.datasets.**augmentHSV**(*img0*)

Function to perform HSV augmentation (by a random factor of +/- 50%)

Inputs:

img0: dataset image loaded by OpenCV

Outputs:

img: transformed image

src.datasets.**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

src.datasets.**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)

src.scoring.**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

src.scoring.**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

src.scoring.**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 should have filename format 'XYZ.tif.txt', where 'XYZ.tif' is the xView TIFF file being predicted on. Prediction files should be in space-delimited csv format, with each line like (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 geojson. EG a prediction file is titled '15.tif.txt', but the file '15.tif' is not in the ground truth.

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

src.scoring.**compute_statistics_given_rectangle_matches**(*groundtruth_rects_matched*, *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.

src.scoring.**compute_precision_recall_given_image_statistics_list**(*iou_threshold*, *image_statistics_list*)

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.

src.scoring.**compute_average_precision_recall_given_precision_recall_dict**(*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.

src.scoring.**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

src.scoring.**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.

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

src.scoring.**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 src.scoring.**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.

4.2 Utils/

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

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

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

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

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

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

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.

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

Inputs:

p: Absolute path to the dataset directory

utils.utils.**load_classes**(*path*)
Loads class labels at 'path'

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

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]

4.3 Tests/

class 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 tests.unittests.**DatasetTests**(*methodName='runTest'*)
Class for all dataset-involved unit tests.

setUp()
Basic setup method. Note that ResourceWarnings and DeprecationWarnings are ignored.

test_load_targets()
Test functionality to load training data.

test_show_targets()
Test functionality to label training data.

class 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 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 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.

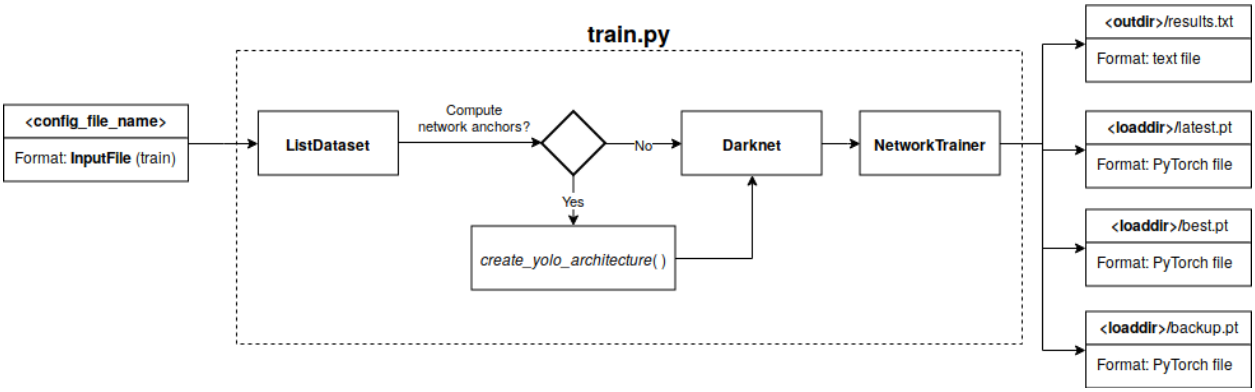
CHAPTER

FIVE

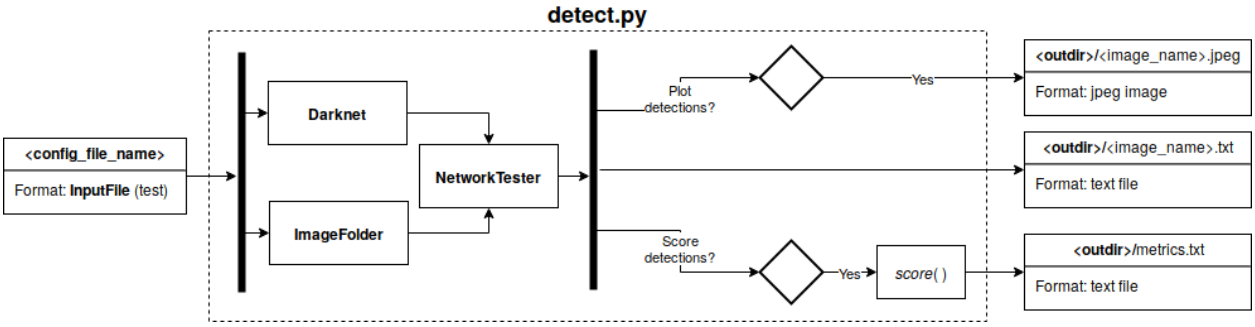
FLOW DIAGRAMS

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

5.1 train



5.2 detect



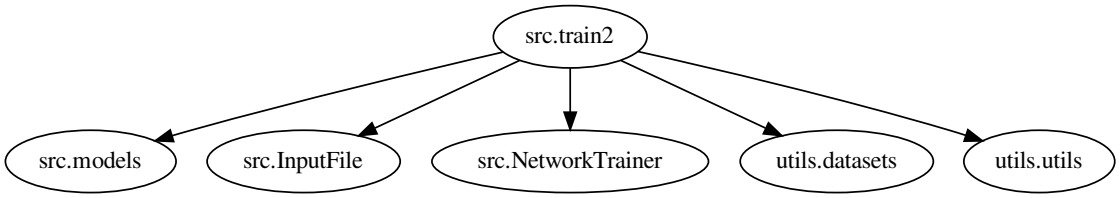
CHAPTER

SIX

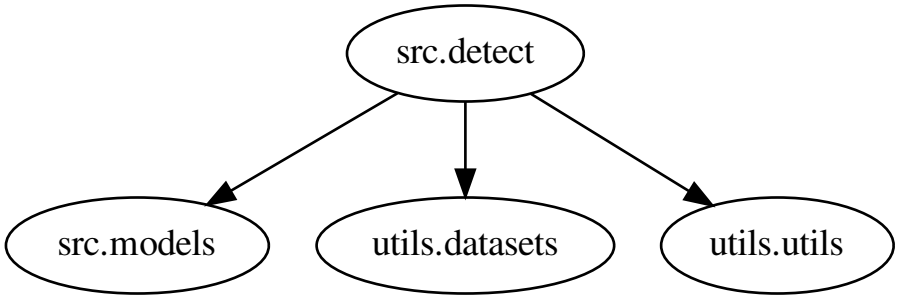
DEPENDENCY GRAPHS

Below are a collection of dependency graphs for all code in Src/.

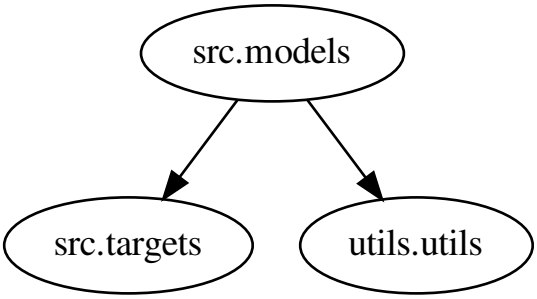
6.1 train2



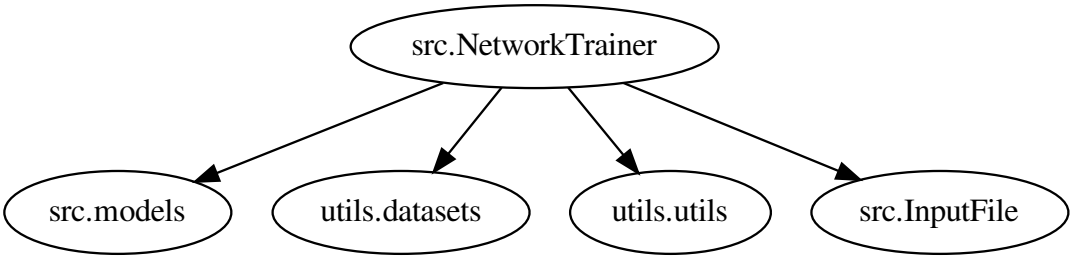
6.2 detect



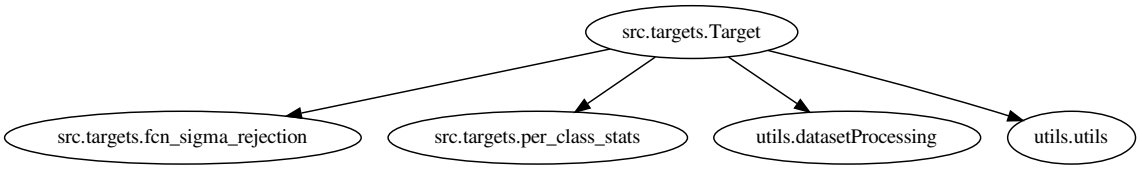
6.3 models



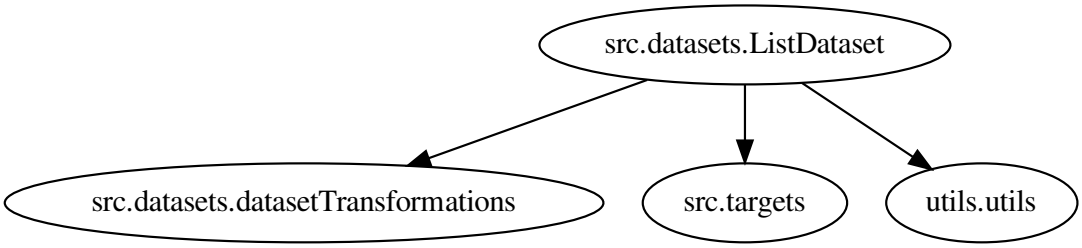
6.4 NetworkTrainer



6.5 targets.Target



6.6 datasets.ListDataset



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

s

t

u

INDEX

I

L

M

N

P

R

S

T

U

W

Y

Z