Comp Viz

Release 1.0.0

Lucas Hirt

Nov 27, 2022

CONTENTS:

CHAPTER

ONE

COMP VIZ SUMMARY

Comp Viz is a computer vision oriented package that aims to allow those unfamiliar with the subject to explore and experiment with various computer vision related tasks like object detection, image segmentation, classification, and more.

Comp Viz is built using MXNet's gluoncv API. Comp Viz aims to provide a simple and intuitive abstraction layer over gluoncv to allow even the most novice users to experiment with computer vision.

Comp Viz currently only supports object detection tasks, but in the near future aims to add support for classification and image segmentation.

Comp Viz is a package currently composed of two subpackages- object_detection, which contains the model module, allowing for object detection related tasks, and the utils sub package, which hosts a variety of helper functions to better interact with, but not limited to, other comp viz subpackages.

CHAPTER

TWO

COMP VIZ

2.1 comp_viz package

2.1.1 Subpackages

Object Detection package

object_detection.model module

class comp_viz.object_detection.model.**Model**(*network_name*)

Bases: object

Computer vision object detection model.

Parameters

network_name (*string*) – A string representing the computer vision network that will be used for detection.

Variables

- **net_name** – Holds the string literal for the chosen computer vision network.
- **net** – Holds the crucial mxnet-gluoncv instantiated computer vision model for which our package aims to provides a layer of abstraction over.
- **inference_resolution** – Stores the resolution of images we are to perform inference on.
- **default_object_classes** – Stores the default object classes that come with chosen network:

get_classes()

Get list of the object classes that the computer vision model is detecting for in images.

Return type

List

get_image_prediction(*fname*, *nms=0.0*)

Get image with the bounding box detections and the prediction made by the computer vision model.

Parameters

- **fname** (*string*) – Path to an image file.
- **nms** (*float*) – Stands for non-maximal suppresion. If computer vision model detects and an object in the image with a confidence value less than the nms value, it will not include it in the returned results.

Returns
A pair of values, an image in the form of a numpy array, and the prediction dict.

Return type
(numpy.array, dict)

`get_prediction`(*fname*, *nms=0.0*) → dict

Get prediction made for an image by computer vision model.

Parameters

- **fname** (*string*) – Path to an image file.
- **nms** (*float*) – Stands for non-maximal suppresion. If computer vision model detects and an object in the image with a confidence value less than the nms value, it will not include it in the returned results.

Return type
dict

`list_classes`()

Print the object classes that the computer vision model is detecting for in images.

Return type
void

`reset_classes`()

Change the object classes that the computer vision model is detecting for in images back to defaults.

Return type
void

`set_classes`(*object_classes: list*)

Change the object classes that the computer vision model is detecting for in images. Ensures validity by referencing the original list of available object classes when model was first instantiatied.

Parameters
object_classes (*List*) – List of new object classes to detect for. Ex. “person”, “bicycle”, “banana”.

Return type
void

`show_image_prediction`(*fname*, *nms=0.0*)

Print image with the bounding box detections and the prediction made by the computer vision model.

Parameters

- **fname** (*string*) – Path to an image file.
- **nms** (*float*) – Stands for non-maximal suppresion. If computer vision model detects and an object in the image with a confidence value less than the nms value, it will not include it in the returned results.

Return type
void

Module contents

model.py

```
import mxnet
import gluoncv
import numpy
import time

from .. import utils

class Model:
  """Computer vision object detection model.

  :param network_name: A string representing the computer vision network that will be␣
↪used
                       for detection.
  :type network_name: string
  :ivar net_name: Holds the string literal for the chosen computer vision network.
  :ivar net: Holds the crucial mxnet-gluoncv instantiated computer vision model for which
             our package aims to provides a layer of abstraction over.
  :ivar inference_resolution: Stores the resolution of images we are to perform␣
↪inference on.
  :ivar default_object_classes: Stores the default object classes that come with chosen␣
↪network:
  """

  def __init__(self,network_name):
    """Constructor method
    """
    self.net_name = network_name
    self.net = gluoncv.model_zoo.get_model(network_name, pretrained=True)
    self.inference_resolution = utils.ObjectDetection.get_network_resolution(network_
↪name)
    self.default_object_classes = gluoncv.model_zoo.get_model(network_name,␣
↪pretrained=True).classes

  def list_classes(self):
    """Print the object classes that the computer vision model is detecting for in␣
↪images.

    :rtype: void
    """
    print(self._get_classes())

  def get_classes(self):
    """Get list of the object classes that the computer vision model is detecting for in␣
↪images.

    :rtype: List
    """
    return (self._get_classes())
```

(continues on next page)

(continued from previous page)

```
  def get_prediction(self,fname,nms=0.) -> dict:
    """Get prediction made for an image by computer vision model.

    :param fname: Path to an image file.
    :type fname: string
    :param nms: Stands for non-maximal suppresion. If computer vision model detects and␣
↪an
                object in the image with a confidence value less than the nms value, it
                will not include it in the returned results.
    :type nms: float
    :rtype: dict
    """
    utils.Tools.verify_exists(fname)
    start = time.time()
    cids, scores, bboxes = self._predict(fname,nms)
    end = time.time()
    prediction = {}
    prediction["image"] = str(fname)
    prediction['class_ids'] = cids
    prediction['confidence_scores'] = scores
    prediction['bounding_boxes'] = bboxes
    prediction['nms_thresh'] = nms
    prediction['class_map'] = {cid: self.get_classes()[cid] for cid in set(cids)}
    prediction['time'] = round(float(end - start),4)
    return prediction

  def get_image_prediction(self,fname,nms=0.):
    """Get image with the bounding box detections and the prediction made by the␣
↪computer vision model.

    :param fname: Path to an image file.
    :type fname: string
    :param nms: Stands for non-maximal suppresion. If computer vision model detects and␣
↪an
                object in the image with a confidence value less than the nms value, it
                will not include it in the returned results.
    :type nms: float
    :return: A pair of values, an image in the form of a numpy array, and the prediction␣
↪dict.
    :rtype: (numpy.array, dict)
    """
    pred = self.get_prediction(fname)
    pred_img = utils.ObjectDetection.get_pred_bboxes_image(fname,
                                                           pred["bounding_boxes"],
                                                           pred["class_ids"],
                                                           [val for val in pred["class_
↪map"].values()],
                                                           pred["confidence_scores"])
    return pred_img, pred

  def show_image_prediction(self,fname,nms=0.):
```

(continues on next page)

(continued from previous page)

```
    """Print image with the bounding box detections and the prediction made by the␣
↪computer vision model.

    :param fname: Path to an image file.
    :type fname: string
    :param nms: Stands for non-maximal suppresion. If computer vision model detects and␣
↪an
                object in the image with a confidence value less than the nms value, it
                will not include it in the returned results.
    :type nms: float
    :rtype: void
    """
    image, prediction = self.get_image_prediction(fname)
    utils.Tools.show_image(image)
    print(prediction)

  def set_classes(self,object_classes: list):
    """Change the object classes that the computer vision model is detecting for in␣
↪images. Ensures validity by referencing the original list of available object classes␣
↪when model was first instantiated.

    :param object_classes: List of new object classes to detect for. Ex. "person",
↪"bicycle", "banana".
    :type object_classes: List
    :rtype: void
    """
    unsupported = []
    for obj_class in object_classes:
      if obj_class not in self.default_object_classes:
        unsupported.append(obj_class)
    if unsupported:
      print(f"WARNING: object classes \"{unsupported}\" are not supported by default for␣
↪object detection. Expect no capability for detection.")
    self.net.reset_class(object_classes, reuse_weights=object_classes)
    print(f"Complete. Model set to detect for object classes: {self.get_classes()}.")

  def reset_classes(self):
    """Change the object classes that the computer vision model is detecting for in␣
↪images back to defaults.

    :rtype: void
    """
    self.net.reset_class(self.default_object_classes,reuse_weights=self.default_object_
↪classes)
    print("Object classes for detection restored to defaults.")

  # Get tuple containing class ids, confidence scores and bounding boxes for an image␣
↪prediction,
  # apply NMS if specified and return the tuple.
  def _predict(self,fname,nms) -> tuple:
    base_img = mxnet.image.imread(fname)
    x, img = self.__prepare_image(base_img)
```

(continues on next page)

(continued from previous page)

```
    pred = self.net(x)
    cids, scores, bboxes = self._extract_cids_scores_bboxes(pred)
    # resize bounding box from network inference resolution to original image resolution
    bboxes = [utils.ObjectDetection.resize_bbox(bbox,img.shape,base_img.shape) for bbox␣
↪in bboxes]
    if nms == 0:
      return (cids,scores,bboxes)
    nms_cids, nms_scores, nms_bboxes = self._apply_nms(cids, scores, bboxes, nms)
    return (nms_cids,nms_scores,nms_bboxes)

  def _get_classes(self):
    return self.net.classes

  # Return tuple containing only the class ids, confidence scores, and bounding boxes of␣
↪an MXNet
  # inference result.
  def _extract_cids_scores_bboxes(self,pred):
    cids = self._get_class_ids(pred[0])
    scores = self._get_scores(pred[1])
    bboxes = self._get_bboxes(pred[2])
    return (cids, scores, bboxes)

  # Given an tuple containing class ids, confidence scoers and bounding boxes for an␣
↪MXNet prediction
  # filter the results and return them based off the confidence scores of the␣
↪predictions and the
  # specified nms value. (If confidence score < NMS, prune that prediction from results␣
↪to be returned.)
  def _apply_nms(self,cids: list, scores: list, bboxes: list, nms: float):
    prune_indexes = []
    for i, score in enumerate(scores):
      if score < nms:
        prune_indexes.append(i)
    for index in prune_indexes[::-1]:
      cids.pop(index)
      scores.pop(index)
      bboxes.pop(index)
    return (cids, scores, bboxes)

  # Extract the class ids from an MXNet prediction ndarray to a list.
  def _get_class_ids(self,cids: numpy.ndarray) -> list:
    class_ids = []
    for ndarray in cids[0]:
      nparray = ndarray.asnumpy()
      if nparray[0] != -1:
        class_ids.append(int(nparray[0]))
    return class_ids

  # Extract the confidence score float values from an MXNet prediction ndarray to a list
  def _get_scores(self,scores: list):
    confidence_scores = []
    for ndarray in scores[0]:
```

(continues on next page)

(continued from previous page)

```
      nparray = ndarray.asnumpy()
      if nparray[0] != -1:
        confidence_scores.append(round(float(nparray[0]),3))
    return confidence_scores

  # Extract the bounding boxes from an MXNet prediction ndarray to a nested list
  def _get_bboxes(self,bboxes):
    bounding_boxes = []
    for ndarray in bboxes[0]:
      nparray = ndarray.asnumpy()
      if nparray[0] != -1:
        bb = [float(corner) for corner in nparray.tolist()]
        bounding_boxes.append(bb)
    return bounding_boxes

  # Process image such that inference can be performed by the mxnet network.
  def __prepare_image(self,image):
    if "yolo" in self.net_name:
      return gluoncv.data.transforms.presets.yolo.transform_test(image,short=self.
↪inference_resolution)
    elif "rcnn" in self.net_name:
      return gluoncv.data.transforms.presets.rcnn.transform_test(image,short=self.
↪inference_resolution)
    elif "ssd" in self.net_name:
      return gluoncv.data.transforms.presets.ssd.transform_test(image,short=self.
↪inference_resolution)
    elif "center_net" in self.net_name:
      return gluoncv.data.transforms.presets.center_net.transform_test(image,short=self.
↪inference_resolution)
```

__init__.py

```
from .model import *
```

Utils package

utils.toolbox module

class comp_viz.utils.toolbox.**Models**

Bases: object

Utility class centered around conveying available functionality for the comp_viz package.

get_tasks()

Get available tasks for the comp_viz package.

Return type
list

list_tasks()

Show available tasks for the comp_viz package.

Return type
void

class comp_viz.utils.toolbox.**ObjectDetection**

Bases: object

Utility class centered around object detection tasks relevant but not limited to the comp_viz object detection package.

format_object_classes() → list

Given a list of object classes, format all the elements such that they are readable by the network.

Parameters
object_classes (*list*) – List of object classes.

Return type
list

get_network_resolution()

Get image inference resolution of the specified network.

Parameters
net_name (*string*) – A valid network name among the results in get_networks() or list_networks() method.

Return type
int

get_networks()

Get list of the available networks that can be used with the comp_viz object_detection sub-package.

Return type
List

get_pred_bboxes_image(*bboxes: list*, *labels=[]*, *class_names=[]*, *scores=[]*)

Given an image and detection bounding box features, plot the bounding box to the image and return it.

Parameters

- **img_fname** (*string*) – Path to image to plot bounding box to.
- **bboxes** (*List[List]*) – Bounding boxes of form [[x_min,y_min,x_max,y_max],...] to plot to the image/
- **labels** (*List[int]*) – Class id values to mape to each bounding box and class name.
- **class_names** (*List[string]*) – List of object classes:
- **scores** (*List[float]*) – List of confidence values for the bounding boxes.

Return type
numpy.ndarray

list_networks()

Show list of the available networks that can be used with the comp_viz object_detection sub-package.

Return type
void

resize_bbox(*orig: tuple*, *dest: tuple*)

Given a bounding box of the format [x,min, y_min, x_max, y_max] and the original image resolution, return a new bounding box resized to the desired image size.

Parameters

- **bbox** (*list*) – Bounding box of the form [x_min, y_min, x_max, y_max].
- **orig** (*Tuple or ndarray.shape*) – Original image resolution of form (height, width, shape). Ex. (500,800,3)
- **dest** (*Tuple or ndarray.shape*) – Image to resize resolution of form (height, width, shape). Ex. (600,900,3)

Return type
list

show_pred_bboxes_image(*bboxes: list*, *labels=[]*, *class_names=[]*, *scores=[]*)

Given an image and detection bounding box features, plot the bounding box to the image and show it.

Parameters

- **img_fname** (*string*) – Path to image to plot bounding box to.
- **bboxes** (*List[List]*) – Bounding boxes of form [[x_min,y_min,x_max,y_max],...] to plot to the image/
- **labels** (*List[int]*) – Class id values to mape to each bounding box and class name.
- **class_names** (*List[string]*) – List of object classes:
- **scores** (*List[float]*) – List of confidence values for the bounding boxes.

Return type
void

class comp_viz.utils.toolbox.**Tools**

Bases: object

Utility class centered around images and filesnames.

exists() → bool

Boolean function to determine if path to filename exists.

Parameters
fname (*string*) – Path to file.

Return type
boolean

filename_show_image()

Given path to an image file, show the said image file to the screen.

Parameters
fname (*string*) – Path to image.

Return type
void

get_cv2_image()

Given path to an image file, return the said image in the form of an numpy ndarray using openCV.

Parameters
fname (*string*) – Path to file.

Return type
numpy.ndarray

get_mxnet_image()

Given path to an image file, return the said image in the form of an mxnet ndarray.

Parameters
fname (*string*) – Path to file.

Return type
mxnet.ndarray.ndarray.NDArray

save_image(*path: str*)

Given an image in the form of an ndarray, save it to the path specified.

Parameters

- **img** (*numpy.ndarray or mxnet.ndarray.ndarray.NDArray*) – Image in the form of an ndarray.
- **path** (*string*) – Path to save image to.

Return type
void

show_image()

Given an image in the form of an numpy ndarray, show the image to the screen.

Parameters
img (*numpy.ndarray or mxnet.ndarray.ndarray.NDArray*) – Image in the form of an ndarray.

Return type
void

verify_exists()

Module contents

toolbox.py

```
import os
import mxnet
import gluoncv
import numpy
import cv2

from ..config import Models as models_config
from ..config import ObjectDetection as obj_det_config

class Models:
  """Utility class centered around conveying available functionality for the comp_viz␣
↪package.
  """

  def list_tasks():
    """Show available tasks for the comp_viz package.

    :rtype: void
```

(continues on next page)

(continued from previous page)

```
    """
    print(Models._get_tasks())

  def get_tasks():
    """Get available tasks for the comp_viz package.

    :rtype: list
    """
    return Models._get_tasks()

  def _get_tasks():
    return models_config.tasks


class ObjectDetection:
  """Utility class centered around object detection tasks relevant but not limited to␣
↪the comp_viz object detection package.
  """

  def list_networks():
    """Show list of the available networks that can be used with the comp_viz object_
↪detection sub-package.

    :rtype: void
    """
    print(ObjectDetection._get_networks())

  def get_networks():
    """Get list of the available networks that can be used with the comp_viz object_
↪detection sub-package.

    :rtype: List
    """
    return ObjectDetection._get_networks()

  def get_network_resolution(net_name):
    """Get image inference resolution of the specified network.

    :param net_name: A valid network name among the results in get_networks() or list_
↪networks() method.
    :type net_name: string
    :rtype: int
    """
    return ObjectDetection._get_resolution(net_name)

  def format_object_classes(object_classes: list) -> list:
    """Given a list of object classes, format all the elements such that they are␣
↪readable by the network.

    :param object_classes: List of object classes.
    :type object_classes: list
    :rtype: list
```

(continues on next page)

(continued from previous page)

```
    """
    i = 0
    while i < len(object_classes):
      if object_classes[i] == "":
        object_classes.pop(i)
        continue
      object_classes[i] = object_classes[i].lower().strip()
      i += 1
    return object_classes

  def resize_bbox(bbox: list, orig: tuple, dest: tuple):
    """Given a bounding box of the format [x,min, y_min, x_max, y_max] and the original␣
↪image resolution, return a new bounding box resized to the desired image size.

    :param bbox: Bounding box of the form [x_min, y_min, x_max, y_max].
    :type bbox: list
    :param orig: Original image resolution of form (height, width, shape). Ex. (500,800,
↪3)
    :type orig: Tuple or ndarray.shape
    :param dest: Image to resize resolution of form (height, width, shape). Ex. (600,900,
↪3)
    :type dest: Tuple or ndarray.shape
    :rtype: list
    """
    x_min, y_min, x_max, y_max = bbox[0], bbox[1], bbox[2], bbox[3]
    x_scale = dest[1] / orig[1]
    y_scale = dest[0] / orig[0]
    return [float(numpy.round(x_min*x_scale)),
            float(numpy.round(y_min*y_scale)),
            float(numpy.round(x_max*x_scale)),
            float(numpy.round(y_max*y_scale))]

  def show_pred_bboxes_image(img_fname: str, bboxes: list, labels = [], class_names = [],
↪  scores = []):
    """Given an image and detection bounding box features, plot the bounding box to the␣
↪image and show it.

    :param img_fname: Path to image to plot bounding box to.
    :type img_fname: string
    :param bboxes: Bounding boxes of form [[x_min,y_min,x_max,y_max],...] to plot to the␣
↪image/
    :type bboxes: List[List]
    :param labels: Class id values to mape to each bounding box and class name.
    :type labels: List[int]
    :param class_names: List of object classes:
    :type class_names: List[string]
    :param scores: List of confidence values for the bounding boxes.
    :type scores: List[float]
    :rtype: void
    """
    img = ObjectDetection.get_pred_bboxes_image(img_fname,bboxes,labels,class_names,
↪scores)
```

(continues on next page)

(continued from previous page)

```
    Tools.show_image(img)

  def get_pred_bboxes_image(img_fname: str, bboxes: list, labels = [], class_names = [],␣
↪scores = []):
    """Given an image and detection bounding box features, plot the bounding box to the␣
↪image and return it.

    :param img_fname: Path to image to plot bounding box to.
    :type img_fname: string
    :param bboxes: Bounding boxes of form [[x_min,y_min,x_max,y_max],...] to plot to the␣
↪image/
    :type bboxes: List[List]
    :param labels: Class id values to mape to each bounding box and class name.
    :type labels: List[int]
    :param class_names: List of object classes:
    :type class_names: List[string]
    :param scores: List of confidence values for the bounding boxes.
    :type scores: List[float]
    :rtype: numpy.ndarray
    """
    img = Tools.get_mxnet_image(img_fname)
    return gluoncv.utils.viz.cv_plot_bbox(img,numpy.array(bboxes),labels=numpy.
↪array(labels),scores=numpy.array(scores),class_names=class_names,thresh=0.)

  # Get list of networks available for object detection from config.py
  def _get_networks():
    return [network for network in obj_det_config.networks.keys()]

  # Get resolution for a specified network from config.py
  def _get_resolution(net_name: str):
    return obj_det_config.networks[net_name]["resolution"]

class Tools:
  """Utility class centered around images and filesnames.
  """
  def verify_exists(fname: str):
    if not Tools._exists(fname):
      print(f"Error: file {fname} could not be located.")
      return

  def exists(fname: str) -> bool:
    """Boolean function to determine if path to filename exists.

    :param fname: Path to file.
    :type fname: string
    :rtype: boolean
    """
    return Tools._exists(fname)

  def get_mxnet_image(fname: str):
    """Given path to an image file, return the said image in the form of an mxnet␣
↪ndarray.
```

(continues on next page)

(continued from previous page)

```
    :param fname: Path to file.
    :type fname: string
    :rtype: mxnet.ndarray.ndarray.NDArray
    """
    Tools.verify_exists(fname)
    return mxnet.image.imread(fname)

  def get_cv2_image(fname: str):
    """Given path to an image file, return the said image in the form of an numpy␣
↪ndarray using openCV.

    :param fname: Path to file.
    :type fname: string
    :rtype: numpy.ndarray
    """
    Tools.verify_exists(fname)
    return cv2.cvtColor(cv2.imread(fname), cv2.COLOR_BGR2RGB)

  def show_image(img: numpy.ndarray):
    """Given an image in the form of an numpy ndarray, show the image to the screen.

    :param img: Image in the form of an ndarray.
    :type img: numpy.ndarray or mxnet.ndarray.ndarray.NDArray
    :rtype: void
    """
    gluoncv.utils.viz.plot_image(img)

  def filename_show_image(fname: str):
    """Given path to an image file, show the said image file to the screen.

    :param fname: Path to image.
    :type fname: string
    :rtype: void
    """
    Tools.verify_exists(fname)
    img = mxnet.image.imread(fname)
    gluoncv.utils.viz.plot_image(img)

  def save_image(img: numpy.ndarray, path: str):
    """Given an image in the form of an ndarray, save it to the path specified.

    :param img: Image in the form of an ndarray.
    :type img: numpy.ndarray or mxnet.ndarray.ndarray.NDArray
    :param path: Path to save image to.
    :type path: string
    :rtype: void
    """
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    cv2.imwrite(path, img)

  def _exists(fname: str) -> bool:
```

(continues on next page)

(continued from previous page)

```
    if os.path.exists(fname):
      return True
    return False
```

__init__.py

```
from .toolbox import *
```

2.1.2 Comp Viz package configuration file

2.1.3 comp_viz.config

class comp_viz.config.**CompViz**

Bases: object

Most parental configuration class for the comp_viz package.

Variables
version – Version number for the comp_viz package.

version = '1.0.0'

class comp_viz.config.**Models**

Bases: *CompViz*

Configuration class for available functionality for the comp_viz package.

Variables
tasks – List of supported tasks provided by comp_viz package.

tasks = ['Object Detection']

class comp_viz.config.**ObjectDetection**

Bases: *CompViz*

Configuration class for the object detection task for the comp_viz package.

Variables
networks – Dictionary of supported networks for object detection for the comp viz package. Each network has an associated inference resolution.

networks = {'center_net_resnet101_v1b_dcnv2_coco': {'resolution': 416},
'faster_rcnn_fpn_resnet50_v1b_coco': {'resolution': 416},
'faster_rcnn_fpn_syncbn_resnest269_coco': {'resolution': 416},
'ssd_512_resnet50_v1_coco': {'resolution': 416}, 'yolo3_darknet53_coco':
{'resolution': 416}, 'yolo3_mobilenet1.0_coco': {'resolution': 416}}

2.1.4 config.py source code

```
class CompViz:
  """Most parental configuration class for the comp_viz package.

  :ivar version: Version number for the comp_viz package.
  """
  version = "1.0.0"

class Models(CompViz):
  """Configuration class for available functionality for the comp_viz package.

  :ivar tasks: List of supported tasks provided by comp_viz package.
  """
  tasks = ["Object Detection"]

class ObjectDetection(CompViz):
  """Configuration class for the object detection task for the comp_viz package.

  :ivar networks: Dictionary of supported networks for object detection for the comp viz␣
↪package. Each network has an associated inference resolution.
  """
  networks = {
      "yolo3_mobilenet1.0_coco": { "resolution": 416 },
      "yolo3_darknet53_coco": { "resolution": 416 },
      "ssd_512_resnet50_v1_coco": { "resolution": 416 },
      "center_net_resnet101_v1b_dcnv2_coco": { "resolution": 416 },
      "faster_rcnn_fpn_resnet50_v1b_coco": { "resolution": 416 },
      "faster_rcnn_fpn_syncbn_resnest269_coco": { "resolution": 416 }
  }
```

2.1.5 __init__.py source code

```
from . import utils
from . import object_detection
```

PYTHON MODULE INDEX

C

INDEX

T

V