ChainerCV Documentation

Release 0.8.0

Preferred Networks, inc.

Jan 11, 2018

Contents

1 Installation Guide 3

2 ChainerCV Tutorial 5

3 ChainerCV Reference Manual 13

4 Indices and tables 73

Bibliography 75

Python Module Index 77

ChainerCV is a **deep learning based computer vision library** built on top of Chainer.

CHAPTER 1

Installation Guide

1.1 Pip

You can install ChainerCV using *pip*.

```
pip install -U numpy
pip install chainercv
```

1.2 Anaconda

Build instruction using Anaconda is as follows.

```
# For python 3
# wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh -O␣
↪miniconda.sh
wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh -O␣
↪miniconda.sh

bash miniconda.sh -b -p $HOME/miniconda
export PATH="$HOME/miniconda/bin:$PATH"
conda config --set always_yes yes --set changeps1 no
conda update -q conda

# Download ChainerCV and go to the root directory of ChainerCV
git clone https://github.com/chainer/chainercv
cd chainercv
conda env create -f environment.yml
source activate chainercv

# Install ChainerCV
pip install -e .
```

```
# Try our demos at examples/* !
```

CHAPTER 2

ChainerCV Tutorial

2.1 Object Detection Tutorial

This tutorial will walk you through the features related to object detection that ChainerCV supports. We assume that readers have a basic understanding of Chainer framework (e.g. understand `chainer.Link`). For users new to Chainer, please first read Introduction to Chainer.

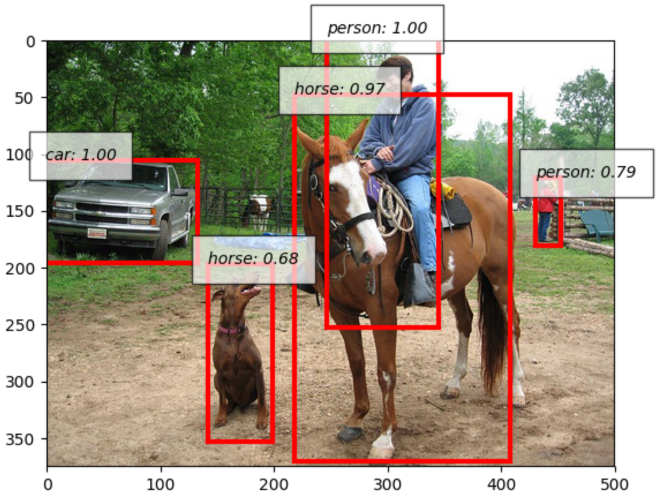
In ChainerCV, we define the object detection task as a problem of, given an image, bounding box based localization and categorization of objects. ChainerCV supports the task by providing the following features:

- Visualization
- BboxDataset
- Detection Link
- DetectionEvaluator
- Training script for various detection models

Here is a short example that conducts inference and visualizes output. Please download an image from a link below, and name it as `sample.jpg`. https://cloud.githubusercontent.com/assets/2062128/26187667/9cb236da-3bd5-11e7-8bcf-7dbd4302e2dc.jpg

```
# In the rest of the tutorial, we assume that the `plt`
# is imported before every code snippet.
import matplotlib.pyplot as plt

from chainercv.datasets import voc_bbox_label_names
from chainercv.links import SSD300
from chainercv.utils import read_image
from chainercv.visualizations import vis_bbox

# Read an RGB image and return it in CHW format.
img = read_image('sample.jpg')
model = SSD300(pretrained_model='voc0712')
bboxes, labels, scores = model.predict([img])
```

```
vis_bbox(img, bboxes[0], labels[0], scores[0],
         label_names=voc_bbox_label_names)
plt.show()
```



2.1.1 Bounding boxes in ChainerCV

Bounding boxes in an image are represented as a two-dimensional array of shape $(R, 4)$, where R is the number of bounding boxes and the second axis corresponds to the coordinates of bounding boxes. The coordinates are ordered in the array by `(y_min, x_min, y_max, x_max)`, where `(y_min, x_min)` and `(y_max, x_max)` are the `(y, x)` coordinates of the top left and the bottom right vertices. Notice that ChainerCV orders coordinates in `yx` order, which is the opposite of the convention used by other libraries such as OpenCV. This convention is adopted because it is more consistent with the memory order of an image that follows row-column order. Also, the `dtype` of bounding box array is `numpy.float32`.

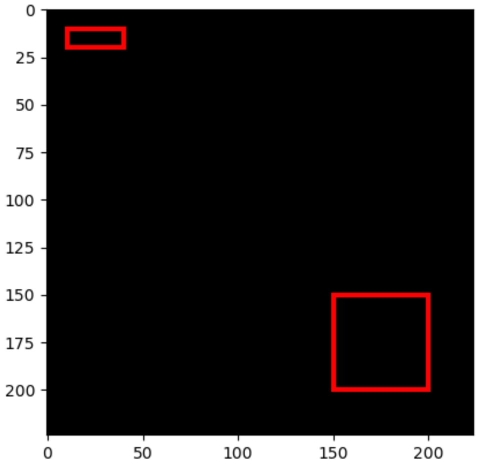
Here is an example with a simple toy data.

```
from chainercv.visualizations import vis_bbox
import numpy as np

img = np.zeros((3, 224, 224), dtype=np.float32)
# We call a variable/array of bounding boxes as `bbox` throughout the library
bbox = np.array([[10, 10, 20, 40], [150, 150, 200, 200]], dtype=np.float32)

chainercv.visualizations.vis_bbox(img, bbox)
plt.show()
```

In this example, two bounding boxes are displayed on top of a black image. `vis_bbox()` is a utility function that visualizes bounding boxes and an image together.

2.1.2 Bounding Box Dataset

ChainerCV supports dataset loaders, which can be used to easily index examples with list-like interface. Dataset classes whose names end with `BboxDataset` contain annotations of where objects locate in an image and which categories they are assigned to. These datasets can be indexed to return a tuple of an image, bounding boxes and labels.



The labels are stored in an `np.int32` array of shape $(R,)$. Each element corresponds to a label of an object in the corresponding bounding box.

A mapping between an integer label and a category differs between datasets. This mapping can be obtained from objects whose names end with `label_names`, such as `voc_bbox_label_names`. These mappings become helpful when bounding boxes need to be visualized with label names. In the next example, the interface of `BboxDataset` and the functionality of *`vis_bbox()`* to visualize label names are illustrated.

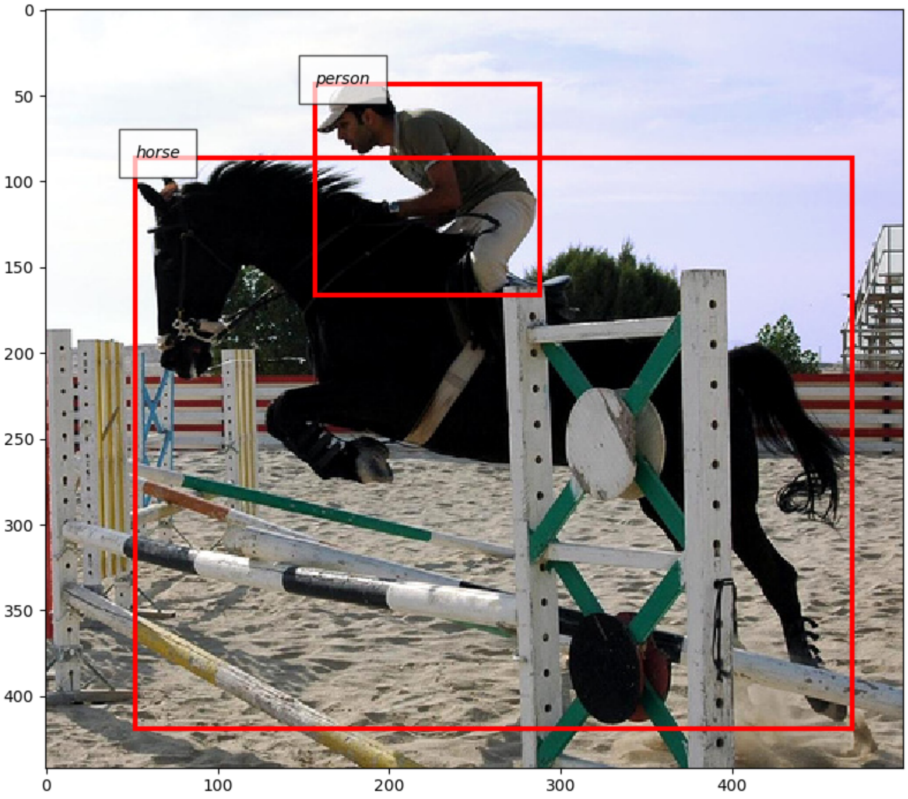
```
from chainercv.datasets import VOCBboxDataset
from chainercv.datasets import voc_bbox_label_names
from chainercv.visualizations import vis_bbox

dataset = VOCBboxDataset(year='2012')
img, bbox, label = dataset[0]
print(bbox.shape)  # (2, 4)
print(label.shape)  # (2,)
vis_bbox(img, bbox, label, label_names=voc_bbox_label_names)
plt.show()
```

Note that the example downloads VOC 2012 dataset at runtime when it is used for the first time on the machine.

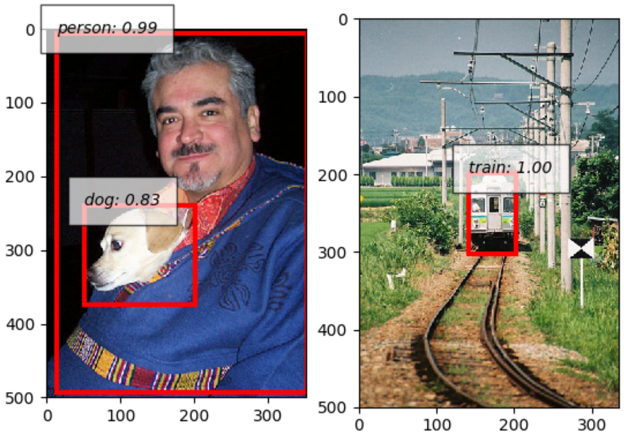
2.1.3 Detection Link

ChainerCV provides several network implementations that carry out object detection. For example, Single Shot MultiBox Detector (SSD) *[Liu16]* and Faster R-CNN *[Ren15]* are supported. Despite the difference between the models in how prediction is carried out internally, they support the common method for prediction called *`predict()`*. This method takes a list of images and returns prediction result, which is a tuple of lists `bboxes, labels, scores`. The more description can be found here (*`predict()`*). Inference on these models runs smoothly by downloading necessary pre-trained weights from the internet automatically.

```
from chainercv.datasets import VOCBboxDataset
from chainercv.datasets import voc_bbox_label_names
from chainercv.links import SSD300
from chainercv.visualizations import vis_bbox

dataset = VOCBboxDataset(year='2007', split='test')
```


person
horse
0
50
100
150
200
250
300
350
400
0
100
200
300
400

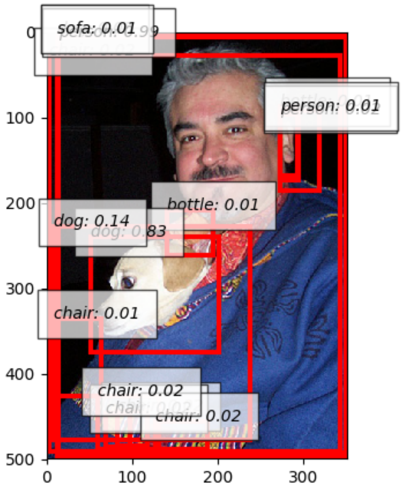
```
img_0, _, _ = dataset[0]
img_1, _, _ = dataset[1]
model = SSD300(pretrained_model='voc0712')
# Note that `predict` takes a list of images.
bboxes, labels, scores = model.predict([img_0, img_1])

# Visualize output of the first image on the left and
# the second image on the right.
fig = plt.figure()
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
vis_bbox(img_0, bboxes[0], labels[0], scores[0],
         label_names=voc_bbox_label_names, ax=ax1)
vis_bbox(img_1, bboxes[1], labels[1], scores[1],
         label_names=voc_bbox_label_names, ax=ax2)
plt.show()
```



The above example puts together functinality of detection link. It instantiates SSD300 model with weights trained on VOC 2007 and VOC 2012 datasets. The model runs prediction using *predict()*, and the outputs are visualized using *vis_bbox()*. Note that in this case, confidence scores are visualized together with other data.

Many detection algorithms post-process bounding box proposals calculated from the output of neural networks by removing unnecessary ones. In the case of Faster R-CNN and SSD, bounding boxes with low confidence are suppressed and overlapping boxes are removed using non-maximum suppression. These two models have attributes `nms_thresh` and `score_thresh`, which configure the post-processing. In the following example, the algorithm runs with a very low `score_thresh` so that bounding boxes with low scores are kept. It is known that lower `score_thresh` produces higher mAP.

```
from chainercv.datasets import VOCBboxDataset
from chainercv.datasets import voc_bbox_label_names
from chainercv.links import SSD300
from chainercv.visualizations import vis_bbox

dataset = VOCBboxDataset(year='2007', split='test')
img, _, _ = dataset[0]
model = SSD300(pretrained_model='voc0712')
# Alternatively, you can use predefined parameters by
# model.use_preset('evaluate')
model.score_thresh = 0.01
bboxes, labels, scores = model.predict([img])
vis_bbox(img, bboxes[0], labels[0], scores[0],
         label_names=voc_bbox_label_names)
plt.show()
```



2.1.4 Detection Evaluator

ChainerCV provides functionalities that make evaluating detection links easy. They are provided at two levels: evaluator extensions and evaluation functions.

Evaluator extensions such as *`DetectionVOCEvaluator`* inherit from `Evaluator`, and have similar interface. They are initialized by taking an iterator and a network that carries out prediction with method *`predict()`*. When this class is called (i.e. `__call__()` of *`DetectionVOCEvaluator`*), several actions are taken. First, it iterates over a dataset based on an iterator. Second, the network makes prediction using the images collected from the dataset. Last, an evaluation function is called with the ground truth annotations and the prediction results.

In contrast to evaluators that hide details, evaluation functions such as *`eval_detection_voc()`* are provided for those who need a finer level of control. These functions take the ground truth annotations and prediction results as arguments and return measured performance.

Here is a simple example that uses a detection evaluator.

```
from chainer.iterators import SerialIterator
from chainer.datasets import SubDataset
from chainercv.datasets import VOCBboxDataset
from chainercv.datasets import voc_bbox_label_names
from chainercv.extensions import DetectionVOCEvaluator
from chainercv.links import SSD300

# Only use subset of dataset so that evaluation finishes quickly.
dataset = VOCBboxDataset(year='2007', split='test')
dataset = dataset[:6]
it = SerialIterator(dataset, 2, repeat=False, shuffle=False)
model = SSD300(pretrained_model='voc0712')
```

```
evaluator = DetectionVOCEvaluator(it, model,
                                  label_names=voc_bbox_label_names)
# result is a dictionary of evaluation scores. Print it and check it.
result = evaluator()
```

2.1.5 Training Detection Links

By putting together all the functions and utilities, training scripts can be easily written. Please check training scripts contained in the examples. Also, ChainerCV posts the performance achieved through running the training script in README.

- Faster R-CNN examples
- SSD examples

2.1.6 References

CHAPTER 3

ChainerCV Reference Manual

3.1 Datasets

3.1.1 General datasets

DirectoryParsingLabelDataset

class chainercv.datasets.**DirectoryParsingLabelDataset**(*root*, *check_img_file=None*, *color=True*, *numerical_sort=False*)

A label dataset whose label names are the names of the subdirectories.

The label names are the names of the directories that locate a layer below the root directory. All images locating under the subdirectoies will be categorized to classes with subdirectory names. An image is parsed only when the function `check_img_file` returns `True` by taking the path to the image as an argument. If `check_img_file` is `None`, the path with any image extensions will be parsed.

Example

A directory structure should be one like below.

```
root
|-- class_0
|    |-- img_0.png
|    |-- img_1.png
|
--- class_1
     |-- img_0.png
```

```
>>> from chainercv.datasets import DirectoryParsingLabelDataset
>>> dataset = DirectoryParsingLabelDataset('root')
>>> dataset.paths
```

```
['root/class_0/img_0.png', 'root/class_0/img_1.png',
'root_class_1/img_0.png']
>>> dataset.labels
array([0, 0, 1])
```

Parameters

- **`root`** (*`str`*) – The root directory.
- **`check_img_file`** (*`callable`*) – A function to determine if a file should be included in the dataset.
- **`color`** (*`bool`*) – If `True`, this dataset read images as color images.
- **`numerical_sort`** (*`bool`*) – Label names are sorted numerically. This means that label `2` is before label `10`, which is not the case when string sort is used. Regardless of this option, string sort is used for the order of files with the same label. The default value is `False`.

directory_parsing_label_names

`chainercv.datasets.`**`directory_parsing_label_names`**(*root*, *numerical_sort=False*)

Get label names from the directories that are named by them.

The label names are the names of the directories that locate a layer below the root directory.

The label names can be used together with *`DirectoryParsingLabelDataset`*. The index of a label name corresponds to the label id that is used by the dataset to refer the label.

Parameters

- **`root`** (*`str`*) – The root directory.
- **`numerical_sort`** (*`bool`*) – Label names are sorted numerically. This means that label `2` is before label `10`, which is not the case when string sort is used. The default value is `False`.

Returns Sorted names of classes.

Return type list of strings

3.1.2 ADE20K

ADE20KSemanticSegmentationDataset

`class` `chainercv.datasets.`**`ADE20KSemanticSegmentationDataset`**(*data_dir='auto'*, *split='train'*)

Semantic segmentation dataset for *ADE20K*.

This is ADE20K dataset distributed in MIT Scene Parsing Benchmark website. It has 20,210 training images and 2,000 validation images.

Parameters

- **`data_dir`** (*`string`*) – Path to the dataset directory. The directory should contain the `ADEChallengeData2016` directory. And that directory should contain at least `images` and `annotations` directries. If `auto` is given, the dataset is automatically downloaded into `$CHAINER_DATASET_ROOT/pfnet/chainercv/ade20k`.

- **split** (*{'train', 'val'}*) – Select from dataset splits used in MIT Scene Parsing Benchmark dataset (ADE20K).

ADE20KTestImageDataset

class `chainercv.datasets.`**`ADE20KTestImageDataset`**(*data_dir='auto'*)

Image dataset for test split of *ADE20K*.

This is an image dataset of test split in ADE20K dataset distributed at MIT Scene Parsing Benchmark website. It has 3,352 test images.

Parameters **data_dir** (*string*) – Path to the dataset directory. The directory should contain the `release_test` dir. If `auto` is given, the dataset is automatically downloaded into `$CHAINER_DATASET_ROOT/pfnet/chainercv/ade20k`.

3.1.3 CamVid

CamVidDataset

class `chainercv.datasets.`**`CamVidDataset`**(*data_dir='auto'*, *split='train'*)

Semantic segmentation dataset for CamVid.

Parameters

- **data_dir** (*string*) – Path to the root of the training data. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/camvid`.
- **split** (*{'train', 'val', 'test'}*) – Select from dataset splits used in CamVid Dataset.

3.1.4 Cityscapes

CityscapesSemanticSegmentationDataset

class `chainercv.datasets.`**`CityscapesSemanticSegmentationDataset`**(*data_dir='auto'*, *label_resolution=None*, *split='train'*, *ignore_labels=True*)

Semantic segmentation dataset for Cityscapes dataset.

Note: Please manually download the data because it is not allowed to re-distribute Cityscapes dataset.

Parameters

- **data_dir** (*string*) – Path to the dataset directory. The directory should contain at least two directories, `leftImg8bit` and either `gtFine` or `gtCoarse`. If `auto` is given, it uses `$CHAINER_DATSET_ROOT/pfnet/chainercv/cityscapes` by default.
- **label_resolution** (*{'fine', 'coarse'}*) – The resolution of the labels. It should be either `fine` or `coarse`.
- **split** (*{'train', 'val'}*) – Select from dataset splits used in Cityscapes dataset.

- **ignore_labels** (*bool*) – If `True`, the labels marked `ignoreInEval` defined in the original *cityscapesScripts<https://github.com/mcordts/cityscapesScripts>_* will be replaced with `-1` in the `get_example()` method. The default value is `True`.

CityscapesTestImageDataset

class `chainercv.datasets.`**`CityscapesTestImageDataset`**(*data_dir='auto'*)

Image dataset for test split of Cityscapes dataset.

Note: Please manually download the data because it is not allowed to re-distribute Cityscapes dataset.

Parameters **data_dir** (*string*) – Path to the dataset directory. The directory should contain the `leftImg8bit` directory. If `auto` is given, it uses `$CHAINER_DATSET_ROOT/pfnet/chainercv/cityscapes` by default.

3.1.5 CUB

CUBLabelDataset

class `chainercv.datasets.`**`CUBLabelDataset`**(*data_dir='auto'*, *return_bb=False*, *prob_map_dir='auto'*, *return_prob_map=False*)

Caltech-UCSD Birds-200-2011 dataset with annotated class labels.

When queried by an index, this dataset returns a corresponding `img, label`, a tuple of an image and class id. The image is in RGB and CHW format. The class id is between 0 and 199. If `return_bb = True`, a bounding box `bb` is appended to the tuple. If `return_prob_map = True`, a probability map `prob_map` is appended.

A bounding box is a one-dimensional array of shape $(4,)$. The elements of the bounding box corresponds to $(y_{min}, x_{min}, y_{max}, x_{max})$, where the four attributes are coordinates of the top left and the bottom right vertices. This information can optionally be retrieved from the dataset by setting `return_bb = True`.

The probability map of a bird shows how likely the bird is located at each pixel. If the value is close to 1, it is likely that the bird locates at that pixel. The shape of this array is (H, W), where H and W are height and width of the image respectively. This information can optionally be retrieved from the dataset by setting `return_prob_map = True`.

Parameters

- **data_dir** (*string*) – Path to the root of the training data. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/cub`.
- **return_bb** (*bool*) – If `True`, this returns a bounding box around a bird. The default value is `False`.
- **prob_map_dir** (*string*) – Path to the root of the probability maps. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/cub`.
- **return_prob_map** (*bool*) – Decide whether to include a probability map of the bird in a tuple served for a query. The default value is `False`.

CUBKeypointDataset

class chainercv.datasets.**CUBKeypointDataset**(*data_dir='auto'*, *return_bb=False*, *prob_map_dir='auto'*, *return_prob_map=False*)

Caltech-UCSD Birds-200-2011 dataset with annotated keypoints.

An index corresponds to each image.

When queried by an index, this dataset returns the corresponding `img, keypoint, kp_mask`, a tuple of an image, keypoints and a keypoint mask that indicates visible keypoints in the image. The data type of the three elements are `float32, float32, bool`. If `return_bb = True`, a bounding box `bb` is appended to the tuple. If `return_prob_map = True`, a probability map `prob_map` is appended.

keypoints are packed into a two dimensional array of shape $(K, 2)$, where K is the number of keypoints. Note that $K = 15$ in CUB dataset. Also note that not all fifteen keypoints are visible in an image. When a keypoint is not visible, the values stored for that keypoint are undefined. The second axis corresponds to the y and x coordinates of the keypoints in the image.

A keypoint mask array indicates whether a keypoint is visible in the image or not. This is a boolean array of shape $(K,)$.

A bounding box is a one-dimensional array of shape $(4,)$. The elements of the bounding box corresponds to $(y_{min}, x_{min}, y_{max}, x_{max})$, where the four attributes are coordinates of the top left and the bottom right vertices. This information can optionally be retrieved from the dataset by setting `return_bb = True`.

The probability map of a bird shows how likely the bird is located at each pixel. If the value is close to 1, it is likely that the bird locates at that pixel. The shape of this array is (H, W), where H and W are height and width of the image respectively. This information can optionally be retrieved from the dataset by setting `return_prob_map = True`.

Parameters

- **data_dir** (*string*) – Path to the root of the training data. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/cub`.
- **return_bb** (*bool*) – If `True`, this returns a bounding box around a bird. The default value is `False`.
- **prob_map_dir** (*string*) – Path to the root of the probability maps. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/cub`.
- **return_prob_map** (*bool*) – Decide whether to include a probability map of the bird in a tuple served for a query. The default value is `False`.

3.1.6 OnlineProducts

OnlineProductsDataset

class chainercv.datasets.**OnlineProductsDataset**(*data_dir='auto'*, *split='train'*)

Dataset class for Stanford Online Products Dataset.

When queried by an index, this dataset returns a corresponding `img, class_id, super_class_id`, a tuple of an image, a class id and a coarse level class id. Images are in RGB and CHW format. Class ids start from 0. The name of the l th coarse level class is l th element of `chainercv.datasets.online_products_super_label_names`.

The `split` selects train and test split of the dataset as done in[1]. The train split contains the first 11318 classes and the test split contains the remaining 11316 classes.

Parameters

- **`data_dir`** (*`string`*) – Path to the root of the training data. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/online_products`.
- **`split`** (*`{'train', 'test'}`*) – Select a split of the dataset.

3.1.7 PASCAL VOC

VOCBboxDataset

class `chainercv.datasets.`**`VOCBboxDataset`**(*data_dir='auto', split='train', year='2012', use_difficult=False, return_difficult=False*)

Bounding box dataset for PASCAL VOC.

The index corresponds to each image.

When queried by an index, if `return_difficult == False`, this dataset returns a corresponding `img, bbox, label`, a tuple of an image, bounding boxes and labels. This is the default behaviour. If `return_difficult == True`, this dataset returns corresponding `img, bbox, label, difficult`. `difficult` is a boolean array that indicates whether bounding boxes are labeled as difficult or not.

The bounding boxes are packed into a two dimensional tensor of shape $(R, 4)$, where R is the number of bounding boxes in the image. The second axis represents attributes of the bounding box. They are $(y_{min}, x_{min}, y_{max}, x_{max})$, where the four attributes are coordinates of the top left and the bottom right vertices.

The labels are packed into a one dimensional tensor of shape $(R,)$. R is the number of bounding boxes in the image. The class name of the label l is l th element of `chainercv.datasets.voc_bbox_label_names`.

The array `difficult` is a one dimensional boolean array of shape $(R,)$. R is the number of bounding boxes in the image. If `use_difficult` is `False`, this array is a boolean array with all `False`.

The type of the image, the bounding boxes and the labels are as follows.

- `img.dtype == numpy.float32`
- `bbox.dtype == numpy.float32`
- `label.dtype == numpy.int32`
- `difficult.dtype == numpy.bool`

Parameters

- **`data_dir`** (*`string`*) – Path to the root of the training data. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/voc`.
- **`split`** (*`{'train', 'val', 'trainval', 'test'}`*) – Select a split of the dataset. `test` split is only available for 2007 dataset.
- **`year`** (*`{'2007', '2012'}`*) – Use a dataset prepared for a challenge held in `year`.

---

[1] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, Silvio Savarese. Deep Metric Learning via Lifted Structured Feature Embedding. arXiv 2015.

- **use_difficult** (*bool*) – If `True`, use images that are labeled as difficult in the original annotation.
- **return_difficult** (*bool*) – If `True`, this dataset returns a boolean array that indicates whether bounding boxes are labeled as difficult or not. The default value is `False`.

VOCSemanticSegmentationDataset

class `chainercv.datasets.`**`VOCSemanticSegmentationDataset`**(*data_dir='auto'*, *split='train'*)

Semantic segmentation dataset for PASCAL VOC2012.

The class name of the label l is l th element of `chainercv.datasets.voc_semantic_segmentation_label_names`.

Parameters

- **data_dir** (*string*) – Path to the root of the training data. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/voc`.
- **split** (*{'train', 'val', 'trainval'}*) – Select a split of the dataset.

3.2 Evaluations

3.2.1 Detection VOC

eval_detection_voc

`chainercv.evaluations.`**`eval_detection_voc`**(*pred_bboxes*, *pred_labels*, *pred_scores*, *gt_bboxes*, *gt_labels*, *gt_difficults=None*, *iou_thresh=0.5*, *use_07_metric=False*)

Calculate average precisions based on evaluation code of PASCAL VOC.

This function evaluates predicted bounding boxes obtained from a dataset which has N images by using average precision for each class. The code is based on the evaluation code used in PASCAL VOC Challenge.

Parameters

- **pred_bboxes** (*iterable of numpy.ndarray*) – An iterable of N sets of bounding boxes. Its index corresponds to an index for the base dataset. Each element of `pred_bboxes` is a set of coordinates of bounding boxes. This is an array whose shape is $(R, 4)$, where R corresponds to the number of bounding boxes, which may vary among boxes. The second axis corresponds to $y_{min}, x_{min}, y_{max}, x_{max}$ of a bounding box.
- **pred_labels** (*iterable of numpy.ndarray*) – An iterable of labels. Similar to `pred_bboxes`, its index corresponds to an index for the base dataset. Its length is N.
- **pred_scores** (*iterable of numpy.ndarray*) – An iterable of confidence scores for predicted bounding boxes. Similar to `pred_bboxes`, its index corresponds to an index for the base dataset. Its length is N.
- **gt_bboxes** (*iterable of numpy.ndarray*) – An iterable of ground truth bounding boxes whose length is N. An element of `gt_bboxes` is a bounding box whose shape is $(R, 4)$. Note that the number of bounding boxes in each image does not need to be same as the number of corresponding predicted boxes.

- **gt_labels** (*iterable of numpy.ndarray*) – An iterable of ground truth labels which are organized similarly to `gt_bboxes`.
- **gt_difficults** (*iterable of numpy.ndarray*) – An iterable of boolean arrays which is organized similarly to `gt_bboxes`. This tells whether the corresponding ground truth bounding box is difficult or not. By default, this is `None`. In that case, this function considers all bounding boxes to be not difficult.
- **iou_thresh** (*float*) – A prediction is correct if its Intersection over Union with the ground truth is above this value.
- **use_07_metric** (*bool*) – Whether to use PASCAL VOC 2007 evaluation metric for calculating average precision. The default value is `False`.

Returns

The keys, value-types and the description of the values are listed below.

- **ap** (*numpy.ndarray*): An array of average precisions. The l-th value corresponds to the average precision for class l. If class l does not exist in either `pred_labels` or `gt_labels`, the corresponding value is set to `numpy.nan`.
- **map** (*float*): The average of Average Precisions over classes.

Return type dict

calc_detection_voc_ap

`chainercv.evaluations.`**`calc_detection_voc_ap`**(*prec*, *rec*, *use_07_metric=False*)

Calculate average precisions based on evaluation code of PASCAL VOC.

This function calculates average precisions from given precisions and recalls. The code is based on the evaluation code used in PASCAL VOC Challenge.

Parameters

- **prec** (*list of numpy.array*) – A list of arrays. `prec[l]` indicates precision for class l. If `prec[l]` is `None`, this function returns `numpy.nan` for class l.
- **rec** (*list of numpy.array*) – A list of arrays. `rec[l]` indicates recall for class l. If `rec[l]` is `None`, this function returns `numpy.nan` for class l.
- **use_07_metric** (*bool*) – Whether to use PASCAL VOC 2007 evaluation metric for calculating average precision. The default value is `False`.

Returns This function returns an array of average precisions. The l-th value corresponds to the average precision for class l. If `prec[l]` or `rec[l]` is `None`, the corresponding value is set to `numpy.nan`.

Return type ndarray

calc_detection_voc_prec_rec

`chainercv.evaluations.`**`calc_detection_voc_prec_rec`**(*pred_bboxes*, *pred_labels*, *pred_scores*, *gt_bboxes*, *gt_labels*, *gt_difficults=None*, *iou_thresh=0.5*)

Calculate precision and recall based on evaluation code of PASCAL VOC.

This function calculates precision and recall of predicted bounding boxes obtained from a dataset which has N images. The code is based on the evaluation code used in PASCAL VOC Challenge.

Parameters

- **pred_bboxes** (*iterable of numpy.ndarray*) – An iterable of N sets of bounding boxes. Its index corresponds to an index for the base dataset. Each element of `pred_bboxes` is a set of coordinates of bounding boxes. This is an array whose shape is $(R, 4)$, where R corresponds to the number of bounding boxes, which may vary among boxes. The second axis corresponds to $y_{min}, x_{min}, y_{max}, x_{max}$ of a bounding box.
- **pred_labels** (*iterable of numpy.ndarray*) – An iterable of labels. Similar to `pred_bboxes`, its index corresponds to an index for the base dataset. Its length is N.
- **pred_scores** (*iterable of numpy.ndarray*) – An iterable of confidence scores for predicted bounding boxes. Similar to `pred_bboxes`, its index corresponds to an index for the base dataset. Its length is N.
- **gt_bboxes** (*iterable of numpy.ndarray*) – An iterable of ground truth bounding boxes whose length is N. An element of `gt_bboxes` is a bounding box whose shape is $(R, 4)$. Note that the number of bounding boxes in each image does not need to be same as the number of corresponding predicted boxes.
- **gt_labels** (*iterable of numpy.ndarray*) – An iterable of ground truth labels which are organized similarly to `gt_bboxes`.
- **gt_difficults** (*iterable of numpy.ndarray*) – An iterable of boolean arrays which is organized similarly to `gt_bboxes`. This tells whether the corresponding ground truth bounding box is difficult or not. By default, this is `None`. In that case, this function considers all bounding boxes to be not difficult.
- **iou_thresh** (*float*) – A prediction is correct if its Intersection over Union with the ground truth is above this value..

Returns

This function returns two lists: `prec` and `rec`.

- `prec`: A list of arrays. `prec[l]` is precision for class l. If class l does not exist in either `pred_labels` or `gt_labels`, `prec[l]` is set to `None`.
- `rec`: A list of arrays. `rec[l]` is recall for class l. If class l that is not marked as difficult does not exist in `gt_labels`, `rec[l]` is set to `None`.

Return type tuple of two lists

3.2.2 Semantic Segmentation IoU

eval_semantic_segmentation

`chainercv.evaluations.`**`eval_semantic_segmentation`**(*pred_labels*, *gt_labels*)

Evaluate metrics used in Semantic Segmentation.

This function calculates Intersection over Union (IoU), Pixel Accuracy and Class Accuracy for the task of semantic segmentation.

The definition of metrics calculated by this function is as follows, where N_{ij} is the number of pixels that are labeled as class i by the ground truth and class j by the prediction.

- IoU of the i-th class $= \frac{N_{ii}}{\sum_{j=1}^k N_{ij} + \sum_{j=1}^k N_{ji} - N_{ii}}$
- mIoU $= \frac{1}{k} \sum_{i=1}^k \frac{N_{ii}}{\sum_{j=1}^k N_{ij} + \sum_{j=1}^k N_{ji} - N_{ii}}$

- Pixel Accuracy $= \frac{\sum_{i=1}^k N_{ii}}{\sum_{i=1}^k \sum_{j=1}^k N_{ij}}$
- Class Accuracy $= \frac{N_{ii}}{\sum_{j=1}^k N_{ij}}$
- Mean Class Accuracy $= \frac{1}{k} \sum_{i=1}^k \frac{N_{ii}}{\sum_{j=1}^k N_{ij}}$

The more detailed description of the above metrics can be found in a review on semantic segmentation[1].

The number of classes n_class is $max(pred\_labels, gt\_labels) + 1$, which is the maximum class id of the inputs added by one.

Parameters

- **`pred_labels`** (*`iterable of numpy.ndarray`*) – A collection of predicted labels. The shape of a label array is (H, W). H and W are height and width of the label. For example, this is a list of labels `[label_0, label_1, ...]`, where `label_i.shape = (H_i, W_i)`.
- **`gt_labels`** (*`iterable of numpy.ndarray`*) – A collection of ground truth labels. The shape of a ground truth label array is (H, W), and its corresponding prediction label should have the same shape. A pixel with value `-1` will be ignored during evaluation.

Returns

The keys, value-types and the description of the values are listed below.

- **iou** (*numpy.ndarray*): An array of IoUs for the n_class classes. Its shape is $(n\_class,)$.
- **miou** (*float*): The average of IoUs over classes.
- **pixel_accuracy** (*float*): The computed pixel accuracy.
- **class_accuracy** (*numpy.ndarray*): An array of class accuracies for the n_class classes. Its shape is $(n\_class,)$.
- **mean_class_accuracy** (*float*): The average of class accuracies.

Return type dict

calc_semantic_segmentation_confusion

`chainercv.evaluations.`**`calc_semantic_segmentation_confusion`**(*pred_labels*, *gt_labels*)

Collect a confusion matrix.

The number of classes n_class is $max(pred\_labels, gt\_labels) + 1$, which is the maximum class id of the inputs added by one.

Parameters

- **`pred_labels`** (*`iterable of numpy.ndarray`*) – A collection of predicted labels. The shape of a label array is (H, W). H and W are height and width of the label.
- **`gt_labels`** (*`iterable of numpy.ndarray`*) – A collection of ground truth labels. The shape of a ground truth label array is (H, W), and its corresponding prediction label should have the same shape. A pixel with value `-1` will be ignored during evaluation.

Returns A confusion matrix. Its shape is (n_class, n_class). The (i, j) th element corresponds to the number of pixels that are labeled as class i by the ground truth and class j by the prediction.

---

[1] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, Jose Garcia-Rodriguez. A Review on Deep Learning Techniques Applied to Semantic Segmentation. arXiv 2017.

Return type numpy.ndarray

calc_semantic_segmentation_iou

`chainercv.evaluations.`**`calc_semantic_segmentation_iou`**(*confusion*)

Calculate Intersection over Union with a given confusion matrix.

The definition of Intersection over Union (IoU) is as follows, where N_{ij} is the number of pixels that are labeled as class i by the ground truth and class j by the prediction.

- IoU of the i-th class $= \frac{N_{ii}}{\sum_{j=1}^k N_{ij} + \sum_{j=1}^k N_{ji} - N_{ii}}$

Parameters **`confusion`** (*`numpy.ndarray`*) – A confusion matrix. Its shape is (n_class, n_class). The (i, j) th element corresponds to the number of pixels that are labeled as class i by the ground truth and class j by the prediction.

Returns An array of IoUs for the n_class classes. Its shape is $(n\_class,)$.

Return type numpy.ndarray

3.3 Extensions

3.3.1 Evaluator

DetectionVOCEvaluator

`class` `chainercv.extensions.`**`DetectionVOCEvaluator`**(*iterator*, *target*, *use_07_metric=False*, *label_names=None*)

An extension that evaluates a detection model by PASCAL VOC metric.

This extension iterates over an iterator and evaluates the prediction results by average precisions (APs) and mean of them (mean Average Precision, mAP). This extension reports the following values with keys. Please note that `'ap/<label_names[l]>'` is reported only if `label_names` is specified.

- `'map'`: Mean of average precisions (mAP).
- `'ap/<label_names[l]>'`: Average precision for class `label_names[l]`, where l is the index of the class. For example, this evaluator reports `'ap/aeroplane'`, `'ap/bicycle'`, etc. if `label_names` is `voc_bbox_label_names`. If there is no bounding box assigned to class `label_names[l]` in either ground truth or prediction, it reports `numpy.nan` as its average precision. In this case, mAP is computed without this class.

Parameters

- **`iterator`** (*`chainer.Iterator`*) – An iterator. Each sample should be following tuple `img, bbox, label` or `img, bbox, label, difficult`. `img` is an image, `bbox` is coordinates of bounding boxes, `label` is labels of the bounding boxes and `difficult` is whether the bounding boxes are difficult or not. If `difficult` is returned, difficult ground truth will be ignored from evaluation.
- **`target`** (*`chainer.Link`*) – A detection link. This link must have `predict()` method that takes a list of images and returns `bboxes`, `labels` and `scores`.
- **`use_07_metric`** (*`bool`*) – Whether to use PASCAL VOC 2007 evaluation metric for calculating average precision. The default value is `False`.

- **label_names** (*iterable of strings*) – An iterable of names of classes. If this value is specified, average precision for each class is also reported with the key `'ap/<label_names[l]>'`.

SemanticSegmentationEvaluator

class `chainercv.extensions.`**`SemanticSegmentationEvaluator`**(*iterator*, *target*, *label_names=None*)

An extension that evaluates a semantic segmentation model.

This extension iterates over an iterator and evaluates the prediction results of the model by common evaluation metrics for semantic segmentation. This extension reports values with keys below. Please note that `'iou/<label_names[l]>'` and `'class_accuracy/<label_names[l]>'` are reported only if `label_names` is specified.

- `'miou'`: Mean of IoUs (mIoU).
- `'iou/<label_names[l]>'`: IoU for class `label_names[l]`, where l is the index of the class. For example, if `label_names` is `camvid_label_names`, this evaluator reports `'iou/Sky'`, `'ap/Building'`, etc.
- `'mean_class_accuracy'`: Mean of class accuracies.
- `'class_accuracy/<label_names[l]>'`: Class accuracy for class `label_names[l]`, where l is the index of the class.
- `'pixel_accuracy'`: Pixel accuracy.

If there is no label assigned to class `label_names[l]` in the ground truth, values corresponding to keys `'iou/<label_names[l]>'` and `'class_accuracy/<label_names[l]>'` are `numpy.nan`. In that case, the means of them are calculated by excluding them from calculation.

For details on the evaluation metrics, please see the documentation for *`chainercv.evaluations.eval_semantic_segmentation()`*.

See also:

`chainercv.evaluations.eval_semantic_segmentation()`.

Parameters

- **iterator** (*chainer.Iterator*) – An iterator. Each sample should be following tuple `img, label`. `img` is an image, `label` is pixel-wise label.
- **target** (*chainer.Link*) – A semantic segmentation link. This link should have `predict()` method that takes a list of images and returns `labels`.
- **label_names** (*iterable of strings*) – An iterable of names of classes. If this value is specified, IoU and class accuracy for each class are also reported with the keys `'iou/<label_names[l]>'` and `'class_accuracy/<label_names[l]>'`.

3.3.2 Visualization Report

DetectionVisReport

class `chainercv.extensions.`**`DetectionVisReport`**(*iterator*, *target*, *label_names=None*, *filename='detection_iter={iteration}_idx={index}.jpg'*)

An extension that visualizes output of a detection model.

This extension visualizes the predicted bounding boxes together with the ground truth bounding boxes.

Internally, this extension takes examples from an iterator, predict bounding boxes from the images in the examples, and visualizes them using `chainercv.visualizations.vis_bbox()`. The process can be illustrated in the following code.

```
batch = next(iterator)
# Convert batch -> imgs, gt_bboxes, gt_labels
pred_bboxes, pred_labels, pred_scores = target.predict(imgs)
# Visualization code
for img, gt_bbox, gt_label, pred_bbox, pred_label, pred_score \
        in zip(imgs, gt_boxes, gt_labels,
               pred_bboxes, pred_labels, pred_scores):
    # the ground truth
    vis_bbox(img, gt_bbox, gt_label)
    # the prediction
    vis_bbox(img, pred_bbox, pred_label, pred_score)
```

Note: `gt_bbox` and `pred_bbox` are float arrays of shape $(R, 4)$, where R is the number of bounding boxes in the image. Each bounding box is organized by $(y_{min}, x_{min}, y_{max}, x_{max})$ in the second axis.

`gt_label` and `pred_label` are intenger arrays of shape $(R,)$. Each label indicates the class of the bounding box.

`pred_score` is a float array of shape $(R,)$. Each score indicates how confident the prediction is.

Parameters

- **`iterator`** – Iterator object that produces images and ground truth.
- **`target`** – Link object used for detection.
- **`label_names`** (*`iterable of str`*) – Name of labels ordered according to label ids. If this is `None`, labels will be skipped.
- **`filename`** (*`str`*) – Basename for the saved image. It can contain two keywords, `'{iteration}'` and `'{index}'`. They are replaced with the iteration of the trainer and the index of the sample when this extension save an image. The default value is `'detection_iter={iteration}_idx={index}.jpg'`.

3.4 Links

3.4.1 Model

General Chain

General Chain

FeaturePredictor

class `chainercv.links.`**`FeaturePredictor`**(*extractor*, *crop_size*, *scale_size=None*, *crop='center'*, *mean=None*)

Wrapper that adds a prediction method to a feature extraction link.

The `predict()` takes three steps to make a prediction.

1. Preprocess input images
2. Forward the preprocessed images to the network
3. Average features in the case when more than one crops are extracted.

Example

```
>>> from chainercv.links import VGG16
>>> from chainercv.links import FeaturePredictor
>>> base_model = VGG16()
>>> model = FeaturePredictor(base_model, 224, 256)
>>> prob = model.predict([img])
# Predicting multiple features
>>> model.extractor.pick = ['conv5_3', 'fc7']
>>> conv5_3, fc7 = model.predict([img])
```

When `self.crop == 'center'`, *`predict()`* extracts features from the center crop of the input images. When `self.crop == '10'`, *`predict()`* extracts features from patches that are ten-cropped from the input images.

When extracting more than one crops from an image, the output of *`predict()`* returns the average of the features computed from the crops.

Parameters

- **extractor** – A feature extraction link. This is a callable chain that takes a batch of images and returns a variable or a tuple of variables.
- **crop_size** (*int or tuple*) – The height and the width of an image after cropping in preprocessing. If this is an integer, the image is cropped to $(crop_size, crop_size)$.
- **scale_size** (*int or tuple*) – If `scale_size` is `None`, neither scaling nor resizing is conducted during preprocessing. This is the default behavior. If this is an integer, an image is resized so that the length of the shorter edge is equal to `scale_size`. If this is a tuple `(height, width)`, the image is resized to $(height, width)$.
- **crop** (*{'center', '10'}*) – Determines the style of cropping.
- **mean** (*numpy.ndarray*) – A mean value. If this is `None`, `extractor.mean` is used as the mean value.

predict (*imgs*)

Predict features from images.

Given N input images, this method outputs a batched array with batchsize N.

Parameters **imgs** (*iterable of numpy.ndarray*) – Array-images. All images are in CHW format and the range of their value is $[0, 255]$.

Returns A batch of features or a tuple of them.

Return type numpy.ndarray or tuple of numpy.ndarray

PickableSequentialChain

class chainercv.links.**PickableSequentialChain**

A sequential chain that can pick intermediate layers.

Callable objects, such as `chainer.Link` and `chainer.Function`, can be registered to this chain with `init_scope()`. This chain keeps the order of registrations and `__call__()` executes callables in that order. A `chainer.Link` object in the sequence will be added as a child link of this link.

`__call__()` returns single or multiple layers that are picked up through a stream of computation. These layers can be specified by `pick`, which contains the names of the layers that are collected. When `pick` is a string, single layer is returned. When `pick` is an iterable of strings, a tuple of layers is returned. The order of the layers is the same as the order of the strings in `pick`. When `pick` is `None`, the last layer is returned.

Examples

```
>>> import chainer.functions as F
>>> import chainer.links as L
>>> model = PickableSequentialChain()
>>> with model.init_scope():
>>>     model.l1 = L.Linear(None, 1000)
>>>     model.l1_relu = F.relu
>>>     model.l2 = L.Linear(None, 1000)
>>>     model.l2_relu = F.relu
>>>     model.l3 = L.Linear(None, 10)
>>> # This is layer l3
>>> layer3 = model(x)
>>> # The layers to be collected can be changed.
>>> model.pick = ('l2_relu', 'l1_relu')
>>> # These are layers l2_relu and l1_relu.
>>> layer2, layer1 = model(x)
```

Parameters

- **pick** (*string or iterable of strings*) – Names of layers that are collected during the forward pass.
- **layer_names** (*iterable of strings*) – Names of layers that can be collected from this chain. The names are ordered in the order of computation.

remove_unused()

Delete all layers that are not needed for the forward pass.

Feature Extraction

Feature extraction links extract feature(s) from given images.

VGG

VGG16

class `chainercv.links.model.vgg.`**VGG16**(*n_class=None*, *pretrained_model=None*, *mean=None*, *initialW=None*, *initial_bias=None*)

VGG-16 Network.

This is a feature extraction link. The network can choose output layers from set of all intermediate layers. The attribute `pick` is the names of the layers that are going to be picked by `__call__()`. The attribute `layer_names` is the names of all layers that can be picked.

Examples

```
>>> model = VGG16()
# By default, __call__ returns a probability score (after Softmax).
>>> prob = model(imgs)
>>> model.pick = 'conv5_3'
# This is layer conv5_3 (after ReLU).
>>> conv5_3 = model(imgs)
>>> model.pick = ['conv5_3', 'fc6']
>>> # These are layers conv5_3 (after ReLU) and fc6 (before ReLU).
>>> conv5_3, fc6 = model(imgs)
```

See also:

`chainercv.links.model.PickableSequentialChain`

When `pretrained_model` is the path of a pre-trained chainer model serialized as a `npz` file in the constructor, this chain model automatically initializes all the parameters with it. When a string in the prespecified set is provided, a pretrained model is loaded from weights distributed on the Internet. The list of pretrained models supported are as follows:

- `imagenet`: Loads weights trained with ImageNet and distributed at Model Zoo.

Parameters

- **n_class** (*int*) – The number of classes. If `None`, the default values are used. If a supported pretrained model is used, the number of classes used to train the pretrained model is used. Otherwise, the number of classes in ILSVRC 2012 dataset is used.
- **pretrained_model** (*str*) – The destination of the pre-trained chainer model serialized as a `npz` file. If this is one of the strings described above, it automatically loads weights stored under a directory `$CHAINER_DATASET_ROOT/pfnet/chainercv/models/`, where `$CHAINER_DATASET_ROOT` is set as `$HOME/.chainer/dataset` unless you specify another value by modifying the environment variable.
- **mean** (*numpy.ndarray*) – A mean value. If `None`, the default values are used. If a supported pretrained model is used, the mean value used to train the pretrained model is used. Otherwise, the mean value calculated from ILSVRC 2012 dataset is used.
- **initialW** (*callable*) – Initializer for the weights.
- **initial_bias** (*callable*) – Initializer for the biases.

Detection

Detection links share a common method `predict()` to detect objects in images. For more details, please read `FasterRCNN.predict()`.

Faster R-CNN

Detection Link

FasterRCNNVGG16

class `chainercv.links.model.faster_rcnn.`**`FasterRCNNVGG16`**(*n_fg_class=None, pretrained_model=None, min_size=600, max_size=1000, ratios=[0.5, 1, 2], anchor_scales=[8, 16, 32], vgg_initialW=None, rpn_initialW=None, loc_initialW=None, score_initialW=None, proposal_creator_params={}*)

Faster R-CNN based on VGG-16.

When you specify the path of a pre-trained chainer model serialized as a `npz` file in the constructor, this chain model automatically initializes all the parameters with it. When a string in prespecified set is provided, a pretrained model is loaded from weights distributed on the Internet. The list of pretrained models supported are as follows:

- `voc07`: Loads weights trained with the trainval split of PASCAL VOC2007 Detection Dataset.
- `imagenet`: Loads weights trained with ImageNet Classfication task for the feature extractor and the head modules. Weights that do not have a corresponding layer in VGG-16 will be randomly initialized.

For descriptions on the interface of this model, please refer to `FasterRCNN`.

`FasterRCNNVGG16` supports finer control on random initializations of weights by arguments `vgg_initialW`, `rpn_initialW`, `loc_initialW` and `score_initialW`. It accepts a callable that takes an array and edits its values. If `None` is passed as an initializer, the default initializer is used.

Parameters

- **`n_fg_class`** (*`int`*) – The number of classes excluding the background.
- **`pretrained_model`** (*`str`*) – The destination of the pre-trained chainer model serialized as a `npz` file. If this is one of the strings described above, it automatically loads weights stored under a directory `$CHAINER_DATASET_ROOT/pfnet/chainercv/models/`, where `$CHAINER_DATASET_ROOT` is set as `$HOME/.chainer/dataset` unless you specify another value by modifying the environment variable.
- **`min_size`** (*`int`*) – A preprocessing paramter for `prepare()`.
- **`max_size`** (*`int`*) – A preprocessing paramter for `prepare()`.
- **`ratios`** (*`list of floats`*) – This is ratios of width to height of the anchors.
- **`anchor_scales`** (*`list of numbers`*) – This is areas of anchors. Those areas will be the product of the square of an element in `anchor_scales` and the original area of the reference window.
- **`vgg_initialW`** (*`callable`*) – Initializer for the layers corresponding to the VGG-16 layers.
- **`rpn_initialW`** (*`callable`*) – Initializer for Region Proposal Network layers.
- **`loc_initialW`** (*`callable`*) – Initializer for the localization head.

- **score_initialW** (*callable*) – Initializer for the score head.
- **proposal_creator_params** (*dict*) – Key valued paramters for *ProposalCreator*.

Utility

bbox2loc

`chainercv.links.model.faster_rcnn.`**bbox2loc**(*src_bbox*, *dst_bbox*)

Encodes the source and the destination bounding boxes to "loc".

Given bounding boxes, this function computes offsets and scales to match the source bounding boxes to the target bounding boxes. Mathematcially, given a bounding box whose center is $(y, x) = p_y, p_x$ and size p_h, p_w and the target bounding box whose center is g_y, g_x and size g_h, g_w, the offsets and scales t_y, t_x, t_h, t_w can be computed by the following formulas.

- $t_y = \frac{(g_y - p_y)}{p_h}$
- $t_x = \frac{(g_x - p_x)}{p_w}$
- $t_h = \log(\frac{g_h}{p_h})$
- $t_w = \log(\frac{g_w}{p_w})$

The output is same type as the type of the inputs. The encoding formulas are used in works such as R-CNN[1].

Parameters

- **src_bbox** (*array*) – An image coordinate array whose shape is $(R, 4)$. R is the number of bounding boxes. These coordinates are $p_{ymin}, p_{xmin}, p_{ymax}, p_{xmax}$.
- **dst_bbox** (*array*) – An image coordinate array whose shape is $(R, 4)$. These coordinates are $g_{ymin}, g_{xmin}, g_{ymax}, g_{xmax}$.

Returns Bounding box offsets and scales from `src_bbox` to `dst_bbox`. This has shape $(R, 4)$. The second axis contains four values t_y, t_x, t_h, t_w.

Return type array

FasterRCNN

class `chainercv.links.model.faster_rcnn.`**FasterRCNN**(*extractor*, *rpn*, *head*, *mean*, *min_size=600*, *max_size=1000*, *loc_normalize_mean=(0.0, 0.0, 0.0, 0.0)*, *loc_normalize_std=(0.1, 0.1, 0.2, 0.2)*)

Base class for Faster R-CNN.

This is a base class for Faster R-CNN links supporting object detection API[2]. The following three stages constitute Faster R-CNN.

1. **Feature extraction**: Images are taken and their feature maps are calculated.

---

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR 2014.

[2] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS 2015.

2. **Region Proposal Networks**: Given the feature maps calculated in the previous stage, produce set of RoIs around objects.
3. **Localization and Classification Heads**: Using feature maps that belong to the proposed RoIs, classify the categories of the objects in the RoIs and improve localizations.

Each stage is carried out by one of the callable `chainer.Chain` objects `feature`, `rpn` and `head`.

There are two functions `predict()` and `__call__()` to conduct object detection. `predict()` takes images and returns bounding boxes that are converted to image coordinates. This will be useful for a scenario when Faster R-CNN is treated as a black box function, for instance. `__call__()` is provided for a scnerario when intermediate outputs are needed, for instance, for training and debugging.

Links that support obejct detection API have method `predict()` with the same interface. Please refer to `predict()` for further details.

Parameters

- **extractor** (*callable Chain*) – A callable that takes a BCHW image array and returns feature maps.
- **rpn** (*callable Chain*) – A callable that has the same interface as `RegionProposalNetwork`. Please refer to the documentation found there.
- **head** (*callable Chain*) – A callable that takes a BCHW array, RoIs and batch indices for RoIs. This returns class dependent localization paramters and class scores.
- **mean** (*numpy.ndarray*) – A value to be subtracted from an image in `prepare()`.
- **min_size** (*int*) – A preprocessing paramter for `prepare()`. Please refer to a docstring found for `prepare()`.
- **max_size** (*int*) – A preprocessing paramter for `prepare()`.
- **loc_normalize_mean** (*tuple of four floats*) – Mean values of localization estimates.
- **loc_normalize_std** (*tupler of four floats*) – Standard deviation of localization estimates.

__call__(*x*, *scale=1.0*)

Forward Faster R-CNN.

Scaling paramter `scale` is used by RPN to determine the threshold to select small objects, which are going to be rejected irrespective of their confidence scores.

Here are notations used.

- N is the number of batch size
- R' is the total number of RoIs produced across batches. Given R_i proposed RoIs from the i th image, $R' = \sum_{i=1}^{N} R_i$.
- L is the number of classes excluding the background.

Classes are ordered by the background, the first class, . . . , and the L th class.

Parameters

- **x** (*Variable*) – 4D image variable.
- **scale** (*float*) – Amount of scaling applied to the raw image during preprocessing.

Returns

Returns tuple of four values listed below.

- **roi_cls_locs**: Offsets and scalings for the proposed RoIs. Its shape is $(R', (L+1) \times 4)$.
- **roi_scores**: Class predictions for the proposed RoIs. Its shape is $(R', L+1)$.
- **rois**: RoIs proposed by RPN. Its shape is $(R', 4)$.
- **roi_indices**: Batch indices of RoIs. Its shape is $(R',)$.

Return type Variable, Variable, array, array

`predict` (*imgs*)

Detect objects from images.

This method predicts objects for each image.

Parameters **`imgs`** (*`iterable of numpy.ndarray`*) – Arrays holding images. All images are in CHW and RGB format and the range of their value is $[0, 255]$.

Returns

This method returns a tuple of three lists, `(bboxes, labels, scores)`.

- **bboxes**: A list of float arrays of shape $(R, 4)$, where R is the number of bounding boxes in a image. Each bouding box is organized by $(y_{min}, x_{min}, y_{max}, x_{max})$ in the second axis.
- **labels** : A list of integer arrays of shape $(R,)$. Each value indicates the class of the bounding box. Values are in range $[0, L-1]$, where L is the number of the foreground classes.
- **scores** : A list of float arrays of shape $(R,)$. Each value indicates how confident the prediction is.

Return type tuple of lists

`prepare` (*img*)

Preprocess an image for feature extraction.

The length of the shorter edge is scaled to `self.min_size`. After the scaling, if the length of the longer edge is longer than `self.max_size`, the image is scaled to fit the longer edge to `self.max_size`.

After resizing the image, the image is subtracted by a mean image value `self.mean`.

Parameters **`img`** (*`ndarray`*) – An image. This is in CHW and RGB format. The range of its value is $[0, 255]$.

Returns A preprocessed image.

Return type ndarray

`use_preset` (*preset*)

Use the given preset during prediction.

This method changes values of `self.nms_thresh` and `self.score_thresh`. These values are a threshold value used for non maximum suppression and a threshold value to discard low confidence proposals in `predict()`, respectively.

If the attributes need to be changed to something other than the values provided in the presets, please modify them by directly accessing the public attributes.

Parameters **`preset`** (*`{'visualize', 'evaluate'}`*) – A string to determine the preset to use.

generate_anchor_base

`chainercv.links.model.faster_rcnn.`**`generate_anchor_base`**(*base_size=16, ratios=[0.5, 1, 2], anchor_scales=[8, 16, 32]*)

Generate anchor base windows by enumerating aspect ratio and scales.

Generate anchors that are scaled and modified to the given aspect ratios. Area of a scaled anchor is preserved when modifying to the given aspect ratio.

`R = len(ratios) * len(anchor_scales)` anchors are generated by this function. The `i * len(anchor_scales) + j` th anchor corresponds to an anchor generated by `ratios[i]` and `anchor_scales[j]`.

For example, if the scale is 8 and the ratio is 0.25, the width and the height of the base window will be stretched by 8. For modifying the anchor to the given aspect ratio, the height is halved and the width is doubled.

Parameters

- **base_size** (*number*) – The width and the height of the reference window.
- **ratios** (*list of floats*) – This is ratios of width to height of the anchors.
- **anchor_scales** (*list of numbers*) – This is areas of anchors. Those areas will be the product of the square of an element in `anchor_scales` and the original area of the reference window.

Returns An array of shape $(R, 4)$. Each element is a set of coordinates of a bounding box. The second axis corresponds to $(y_{min}, x_{min}, y_{max}, x_{max})$ of a bounding box.

Return type ndarray

loc2bbox

`chainercv.links.model.faster_rcnn.`**`loc2bbox`**(*src_bbox, loc*)

Decode bounding boxes from bounding box offsets and scales.

Given bounding box offsets and scales computed by *`bbox2loc()`*, this function decodes the representation to coordinates in 2D image coordinates.

Given scales and offsets t_y, t_x, t_h, t_w and a bounding box whose center is $(y, x) = p_y, p_x$ and size p_h, p_w, the decoded bounding box's center $\hat{g}_y$, $\hat{g}_x$ and size $\hat{g}_h$, $\hat{g}_w$ are calculated by the following formulas.

- $\hat{g}_y = p_h t_y + p_y$
- $\hat{g}_x = p_w t_x + p_x$
- $\hat{g}_h = p_h \exp(t_h)$
- $\hat{g}_w = p_w \exp(t_w)$

The decoding formulas are used in works such as R-CNN[3].

The output is same type as the type of the inputs.

Parameters

- **src_bbox** (*array*) – A coordinates of bounding boxes. Its shape is $(R, 4)$. These coordinates are $p_{ymin}, p_{xmin}, p_{ymax}, p_{xmax}$.

---

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR 2014.

- **`loc`** (*array*) – An array with offsets and scales. The shapes of `src_bbox` and `loc` should be same. This contains values t_y, t_x, t_h, t_w.

Returns Decoded bounding box coordinates. Its shape is $(R, 4)$. The second axis contains four values $\hat{g}_{ymin}, \hat{g}_{xmin}, \hat{g}_{ymax}, \hat{g}_{xmax}$.

Return type array

ProposalCreator

class `chainercv.links.model.faster_rcnn.`**`ProposalCreator`**(*nms_thresh=0.7*, *n_train_pre_nms=12000*, *n_train_post_nms=2000*, *n_test_pre_nms=6000*, *n_test_post_nms=300*, *force_cpu_nms=False*, *min_size=16*)

Proposal regions are generated by calling this object.

The *`__call__()`* of this object outputs object detection proposals by applying estimated bounding box offsets to a set of anchors.

This class takes parameters to control number of bounding boxes to pass to NMS and keep after NMS. If the paramters are negative, it uses all the bounding boxes supplied or keep all the bounding boxes returned by NMS.

This class is used for Region Proposal Networks introduced in Faster R-CNN[4].

Parameters

- **`nms_thresh`** (*float*) – Threshold value used when calling NMS.
- **`n_train_pre_nms`** (*int*) – Number of top scored bounding boxes to keep before passing to NMS in train mode.
- **`n_train_post_nms`** (*int*) – Number of top scored bounding boxes to keep after passing to NMS in train mode.
- **`n_test_pre_nms`** (*int*) – Number of top scored bounding boxes to keep before passing to NMS in test mode.
- **`n_test_post_nms`** (*int*) – Number of top scored bounding boxes to keep after passing to NMS in test mode.
- **`force_cpu_nms`** (*bool*) – If this is `True`, always use NMS in CPU mode. If `False`, the NMS mode is selected based on the type of inputs.
- **`min_size`** (*int*) – A paramter to determine the threshold on discarding bounding boxes based on their sizes.

`__call__`(*loc*, *score*, *anchor*, *img_size*, *scale=1.0*)

Propose RoIs.

Inputs `loc, score, anchor` refer to the same anchor when indexed by the same index.

On notations, R is the total number of anchors. This is equal to product of the height and the width of an image and the number of anchor bases per pixel.

Type of the output is same as the inputs.

Parameters

---

[4] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS 2015.

- **loc** (*array*) – Predicted offsets and scaling to anchors. Its shape is $(R, 4)$.
- **score** (*array*) – Predicted foreground probability for anchors. Its shape is $(R,)$.
- **anchor** (*array*) – Coordinates of anchors. Its shape is $(R, 4)$.
- **img_size** (*tuple of ints*) – A tuple `height, width`, which contains image size after scaling.
- **scale** (*float*) – The scaling factor used to scale an image after reading it from a file.

Returns An array of coordinates of proposal boxes. Its shape is $(S, 4)$. S is less than `self.n_test_post_nms` in test time and less than `self.n_train_post_nms` in train time. S depends on the size of the predicted bounding boxes and the number of bounding boxes discarded by NMS.

Return type array

RegionProposalNetwork

class `chainercv.links.model.faster_rcnn.`**`RegionProposalNetwork`**(*in_channels=512, mid_channels=512, ratios=[0.5, 1, 2], anchor_scales=[8, 16, 32], feat_stride=16, initialW=None, proposal_creator_params={}*)

Region Proposal Network introduced in Faster R-CNN.

This is Region Proposal Network introduced in Faster R-CNN[5]. This takes features extracted from images and propose class agnostic bounding boxes around "objects".

Parameters

- **in_channels** (*int*) – The channel size of input.
- **mid_channels** (*int*) – The channel size of the intermediate tensor.
- **ratios** (*list of floats*) – This is ratios of width to height of the anchors.
- **anchor_scales** (*list of numbers*) – This is areas of anchors. Those areas will be the product of the square of an element in `anchor_scales` and the original area of the reference window.
- **feat_stride** (*int*) – Stride size after extracting features from an image.
- **initialW** (*callable*) – Initial weight value. If `None` then this function uses Gaussian distribution scaled by 0.1 to initialize weight. May also be a callable that takes an array and edits its values.
- **proposal_creator_params** (*dict*) – Key valued paramters for `ProposalCreator`.

See also:

`ProposalCreator`

---

[5] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS 2015.

`__call__`(*x*, *img_size*, *scale=1.0*)

Forward Region Proposal Network.

Here are notations.

- N is batch size.
- C channel size of the input.
- H and W are height and witdh of the input feature.
- A is number of anchors assigned to each pixel.

Parameters

- **x** (*`Variable`*) – The Features extracted from images. Its shape is (N, C, H, W).
- **img_size** (*`tuple of ints`*) – A tuple `height, width`, which contains image size after scaling.
- **scale** (*`float`*) – The amount of scaling done to the input images after reading them from files.

Returns

This is a tuple of five following values.

- **rpn_locs**: Predicted bounding box offsets and scales for anchors. Its shape is $(N, HWA, 4)$.
- **rpn_scores**: Predicted foreground scores for anchors. Its shape is $(N, HWA, 2)$.
- **rois**: A bounding box array containing coordinates of proposal boxes. This is a concatenation of bounding box arrays from multiple images in the batch. Its shape is $(R', 4)$. Given R_i predicted bounding boxes from the i th image, $R' = \sum_{i=1}^N R_i$.
- **roi_indices**: An array containing indices of images to which RoIs correspond to. Its shape is $(R',)$.
- **anchor**: Coordinates of enumerated shifted anchors. Its shape is $(HWA, 4)$.

Return type (Variable, Variable, array, array, array)

VGG16RoIHead

class `chainercv.links.model.faster_rcnn.`**`VGG16RoIHead`**(*n_class*, *roi_size*, *spatial_scale*, *vgg_initialW=None*, *loc_initialW=None*, *score_initialW=None*)

Faster R-CNN Head for VGG-16 based implementation.

This class is used as a head for Faster R-CNN. This outputs class-wise localizations and classification based on feature maps in the given RoIs.

Parameters

- **n_class** (*`int`*) – The number of classes possibly including the background.
- **roi_size** (*`int`*) – Height and width of the feature maps after RoI-pooling.
- **spatial_scale** (*`float`*) – Scale of the roi is resized.
- **vgg_initialW** (*`callable`*) – Initializer for the layers corresponding to the VGG-16 layers.

- **loc_initialW** (*callable*) – Initializer for the localization head.
- **score_initialW** (*callable*) – Initializer for the score head.

Train-only Utility

AnchorTargetCreator

class `chainercv.links.model.faster_rcnn.`**`AnchorTargetCreator`**(*n_sample=256*, *pos_iou_thresh=0.7*, *neg_iou_thresh=0.3*, *pos_ratio=0.5*)

Assign the ground truth bounding boxes to anchors.

Assigns the ground truth bounding boxes to anchors for training Region Proposal Networks introduced in Faster R-CNN[6].

Offsets and scales to match anchors to the ground truth are calculated using the encoding scheme of `bbox2loc()`.

Parameters

- **n_sample** (*int*) – The number of regions to produce.
- **pos_iou_thresh** (*float*) – Anchors with IoU above this threshold will be assigned as positive.
- **neg_iou_thresh** (*float*) – Anchors with IoU below this threshold will be assigned as negative.
- **pos_ratio** (*float*) – Ratio of positive regions in the sampled regions.

`__call__`(*bbox*, *anchor*, *img_size*)

Assign ground truth supervision to sampled subset of anchors.

Types of input arrays and output arrays are same.

Here are notations.

- S is the number of anchors.
- R is the number of bounding boxes.

Parameters

- **bbox** (*array*) – Coordinates of bounding boxes. Its shape is $(R, 4)$.
- **anchor** (*array*) – Coordinates of anchors. Its shape is $(S, 4)$.
- **img_size** (*tuple of ints*) – A tuple `H, W`, which is a tuple of height and width of an image.

Returns

- **loc**: Offsets and scales to match the anchors to the ground truth bounding boxes. Its shape is $(S, 4)$.
- **label**: Labels of anchors with values `(1=positive, 0=negative, -1=ignore)`. Its shape is $(S,)$.

Return type (array, array)

---

[6] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS 2015.

FasterRCNNTrainChain

class `chainercv.links.model.faster_rcnn.`**`FasterRCNNTrainChain`**(*faster_rcnn*, *rpn_sigma=3.0*, *roi_sigma=1.0*, *anchor_target_creator=<chainercv.links.mode object>*, *proposal_target_creator=<chainercv.links.mod object>*)

Calculate losses for Faster R-CNN and report them.

This is used to train Faster R-CNN in the joint training scheme[7].

The losses include:

- `rpn_loc_loss`: The localization loss for Region Proposal Network (RPN).
- `rpn_cls_loss`: The classification loss for RPN.
- `roi_loc_loss`: The localization loss for the head module.
- `roi_cls_loss`: The classification loss for the head module.

Parameters

- **`faster_rcnn`** (`FasterRCNN`) – A Faster R-CNN model that is going to be trained.
- **`rpn_sigma`** (*`float`*) – Sigma parameter for the localization loss of Region Proposal Network (RPN). The default value is 3, which is the value used in[7].
- **`roi_sigma`** (*`float`*) – Sigma paramter for the localization loss of the head. The default value is 1, which is the value used in[7].
- **`anchor_target_creator`** – An instantiation of *`AnchorTargetCreator`*.
- **`proposal_target_creator_params`** – An instantiation of *`ProposalTargetCreator`*.

`__call__`(*imgs*, *bboxes*, *labels*, *scale*)

Forward Faster R-CNN and calculate losses.

Here are notations used.

- N is the batch size.
- R is the number of bounding boxes per image.

Currently, only $N = 1$ is supported.

Parameters

- **`imgs`** (*`Variable`*) – A variable with a batch of images.
- **`bboxes`** (*`Variable`*) – A batch of bounding boxes. Its shape is $(N, R, 4)$.
- **`labels`** (*`Variable`*) – A batch of labels. Its shape is (N, R). The background is excluded from the definition, which means that the range of the value is $[0, L-1]$. L is the number of foreground classes.
- **`scale`** (*`float or Variable`*) – Amount of scaling applied to the raw image during preprocessing.

---

[7] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS 2015.

Returns Scalar loss variable. This is the sum of losses for Region Proposal Network and the head module.

Return type chainer.Variable

ProposalTargetCreator

class `chainercv.links.model.faster_rcnn.`**`ProposalTargetCreator`**(*n_sample=128*, *pos_ratio=0.25*, *pos_iou_thresh=0.5*, *neg_iou_thresh_hi=0.5*, *neg_iou_thresh_lo=0.0*)

Assign ground truth bounding boxes to given RoIs.

The `__call__()` of this class generates training targets for each object proposal. This is used to train Faster RCNN[8].

Parameters

- **n_sample** (*int*) – The number of sampled regions.
- **pos_ratio** (*float*) – Fraction of regions that is labeled as a foreground.
- **pos_iou_thresh** (*float*) – IoU threshold for a RoI to be considered as a foreground.
- **neg_iou_thresh_hi** (*float*) – RoI is considered to be the background if IoU is in [`neg_iou_thresh_hi`, `neg_iou_thresh_hi`).
- **neg_iou_thresh_lo** (*float*) – See above.

`__call__`(*roi*, *bbox*, *label*, *loc_normalize_mean=(0.0, 0.0, 0.0, 0.0)*, *loc_normalize_std=(0.1, 0.1, 0.2, 0.2)*)

Assigns ground truth to sampled proposals.

This function samples total of `self.n_sample` RoIs from the combination of `roi` and `bbox`. The RoIs are assigned with the ground truth class labels as well as bounding box offsets and scales to match the ground truth bounding boxes. As many as `pos_ratio * self.n_sample` RoIs are sampled as foregrounds.

Offsets and scales of bounding boxes are calculated using `chainercv.links.model.faster_rcnn.bbox2loc()`. Also, types of input arrays and output arrays are same.

Here are notations.

- S is the total number of sampled RoIs, which equals `self.n_sample`.
- L is number of object classes possibly including the background.

Parameters

- **roi** (*array*) – Region of Interests (RoIs) from which we sample. Its shape is $(R, 4)$
- **bbox** (*array*) – The coordinates of ground truth bounding boxes. Its shape is $(R', 4)$.
- **label** (*array*) – Ground truth bounding box labels. Its shape is $(R',)$. Its range is $[0, L-1]$, where L is the number of foreground classes.
- **loc_normalize_mean** (*tuple of four floats*) – Mean values to normalize coordinates of bouding boxes.

---

[8] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS 2015.

- **`loc_normalize_std`** (*`tupler of four floats`*) – Standard deviation of the coordinates of bounding boxes.

Returns

- **sample_roi**: Regions of interests that are sampled. Its shape is $(S, 4)$.
- **gt_roi_loc**: Offsets and scales to match the sampled RoIs to the ground truth bounding boxes. Its shape is $(S, 4)$.
- **gt_roi_label**: Labels assigned to sampled RoIs. Its shape is $(S,)$. Its range is $[0, L]$. The label with value 0 is the background.

Return type (array, array, array)

SSD (Single Shot Multibox Detector)

Detection Links

SSD300

class `chainercv.links.model.ssd.`**`SSD300`**(*n_fg_class=None*, *pretrained_model=None*)

Single Shot Multibox Detector with 300x300 inputs.

This is a model of Single Shot Multibox Detector[1]. This model uses *`VGG16Extractor300`* as its feature extractor.

Parameters

- **`n_fg_class`** (*`int`*) – The number of classes excluding the background.
- **`pretrained_model`** (*`str`*) – The weight file to be loaded. This can take `'voc0712'`, *filepath* or `None`. The default value is `None`.
  - `'voc0712'`: Load weights trained on trainval split of PASCAL VOC 2007 and 2012. The weight file is downloaded and cached automatically. `n_fg_class` must be `20` or `None`. These weights were converted from the Caffe model provided by the original implementation. The conversion code is *chainercv/examples/ssd/caffe2npz.py*.
  - `'imagenet'`: Load weights of VGG-16 trained on ImageNet. The weight file is downloaded and cached automatically. This option initializes weights partially and the rests are initialized randomly. In this case, `n_fg_class` can be set to any number.
  - *filepath*: A path of npz file. In this case, `n_fg_class` must be specified properly.
  - `None`: Do not load weights.

SSD512

class `chainercv.links.model.ssd.`**`SSD512`**(*n_fg_class=None*, *pretrained_model=None*)

Single Shot Multibox Detector with 512x512 inputs.

This is a model of Single Shot Multibox Detector[2]. This model uses *`VGG16Extractor512`* as its feature extractor.

---

[1] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

Parameters

- **n_fg_class** (*int*) – The number of classes excluding the background.
- **pretrained_model** (*str*) – The weight file to be loaded. This can take `'voc0712'`, *filepath* or `None`. The default value is `None`.
  - `'voc0712'`: Load weights trained on trainval split of PASCAL VOC 2007 and 2012. The weight file is downloaded and cached automatically. `n_fg_class` must be `20` or `None`. These weights were converted from the Caffe model provided by the original implementation. The conversion code is *chainercv/examples/ssd/caffe2npz.py*.
  - `'imagenet'`: Load weights of VGG-16 trained on ImageNet. The weight file is downloaded and cached automatically. This option initializes weights partially and the rests are initialized randomly. In this case, `n_fg_class` can be set to any number.
  - *filepath*: A path of npz file. In this case, `n_fg_class` must be specified properly.
  - `None`: Do not load weights.

Utility

Multibox

class `chainercv.links.model.ssd.`**`Multibox`**(*n_class, aspect_ratios, initialW=None, initial_bias=None*)

Multibox head of Single Shot Multibox Detector.

This is a head part of Single Shot Multibox Detector[3]. This link computes `mb_locs` and `mb_confs` from feature maps. `mb_locs` contains information of the coordinates of bounding boxes and `mb_confs` contains confidence scores of each classes.

Parameters

- **n_class** (*int*) – The number of classes possibly including the background.
- **aspect_ratios** (*iterable of tuple or int*) – The aspect ratios of default bounding boxes for each feature map.
- **initialW** – An initializer used in `chainer.links.Convolution2d.__init__()`. The default value is `chainer.initializers.LeCunUniform`.
- **initial_bias** – An initializer used in `chainer.links.Convolution2d.__init__()`. The default value is `chainer.initializers.Zero`.

`__call__`(*xs*)

Compute loc and conf from feature maps

This method computes `mb_locs` and `mb_confs` from given feature maps.

Parameters xs (*iterable of chainer.Variable*) – An iterable of feature maps. The number of feature maps must be same as the number of `aspect_ratios`.

Returns

This method returns two `chainer.Variable`: `mb_locs` and `mb_confs`.

- **mb_locs**: A variable of float arrays of shape $(B, K, 4)$, where B is the number of samples in the batch and K is the number of default bounding boxes.

---

[3] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

- **mb_confs**: A variable of float arrays of shape $(B, K, n\_fg\_class + 1)$.

Return type tuple of chainer.Variable

MultiboxCoder

class `chainercv.links.model.ssd.`**`MultiboxCoder`**(*grids*, *aspect_ratios*, *steps*, *sizes*, *variance*)

A helper class to encode/decode bounding boxes.

This class encodes `(bbox, label)` to `(mb_loc, mb_label)` and decodes `(mb_loc, mb_conf)` to `(bbox, label, score)`. These encoding/decoding are used in Single Shot Multibox Detector[4].

- `mb_loc`: An array representing offsets and scales from the default bounding boxes. Its shape is $(K, 4)$, where K is the number of the default bounding boxes. The second axis is composed by $(\Delta y, \Delta x, \Delta h, \Delta w)$. These values are computed by the following formulas.
  - $\Delta y = (b_y - m_y)/(m_h * v_0)$
  - $\Delta x = (b_x - m_x)/(m_w * v_0)$
  - $\Delta h = log(b_h/m_h)/v_1$
  - $\Delta w = log(b_w/m_w)/v_1$

  (m_y, m_x) and (m_h, m_w) are center coodinates and size of a default bounding box. (b_y, b_x) and (b_h, b_w) are center coodinates and size of a given bounding boxes that is assined to the default bounding box. (v_0, v_1) are coefficients that can be set by argument `variance`.
- `mb_label`: An array representing classes of ground truth bounding boxes. Its shape is $(K,)$.
- `mb_conf`: An array representing classes of predicted bounding boxes. Its shape is $(K, n\_fg\_class + 1)$.

Parameters

- **grids** (*iterable of ints*) – An iterable of integers. Each integer indicates the size of a feature map.
- **aspect_ratios** (*iterable of tuples of ints*) – An iterable of tuples of integers used to compute the default bouding boxes. Each tuple indicates the aspect ratios of the default bounding boxes at each feature maps. The length of this iterable should be `len(grids)`.
- **steps** (*iterable of floats*) – The step size for each feature map. The length of this iterable should be `len(grids)`.
- **sizes** (*iterable of floats*) – The base size of default bounding boxes for each feature map. The length of this iterable should be `len(grids) + 1`.
- **variance** (*tuple of floats*) – Two coefficients for encoding/decoding the locations of bounding boxes. The first value is used to encode/decode coordinates of the centers. The second value is used to encode/decode the sizes of bounding boxes.

decode(*mb_loc*, *mb_conf*, *nms_thresh=0.45*, *score_thresh=0.6*)

Decodes back to coordinates and classes of bounding boxes.

This method decodes `mb_loc` and `mb_conf` returned by a SSD network back to `bbox`, `label` and `score`.

---

[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

Parameters

- **`mb_loc`** (*`array`*) – A float array whose shape is $(K, 4)$, K is the number of default bounding boxes.
- **`mb_conf`** (*`array`*) – A float array whose shape is $(K, n\_fg\_class + 1)$.
- **`nms_thresh`** (*`float`*) – The threshold value for *`non_maximum_suppression()`*. The default value is `0.45`.
- **`score_thresh`** (*`float`*) – The threshold value for confidence score. If a bounding box whose confidence score is lower than this value, the bounding box will be suppressed. The default value is `0.6`.

Returns

This method returns a tuple of three arrays, `(bbox, label, score)`.

- **bbox**: A float array of shape $(R, 4)$, where R is the number of bounding boxes in a image. Each bouding box is organized by $(y_{min}, x_{min}, y_{max}, x_{max})$ in the second axis.
- **label** : An integer array of shape $(R,)$. Each value indicates the class of the bounding box.
- **score** : A float array of shape $(R,)$. Each value indicates how confident the prediction is.

Return type tuple of three arrays

`encode`(*bbox*, *label*, *iou_thresh=0.5*)

Encodes coordinates and classes of bounding boxes.

This method encodes `bbox` and `label` to `mb_loc` and `mb_label`, which are used to compute multibox loss.

Parameters

- **`bbox`** (*`array`*) – A float array of shape $(R, 4)$, where R is the number of bounding boxes in an image. Each bouding box is organized by $(y_{min}, x_{min}, y_{max}, x_{max})$ in the second axis.
- **`label`** (*`array`*) – An integer array of shape $(R,)$. Each value indicates the class of the bounding box.
- **`iou_thresh`** (*`float`*) – The threshold value to determine a default bounding box is assigned to a ground truth or not. The default value is `0.5`.

Returns

This method returns a tuple of two arrays, `(mb_loc, mb_label)`.

- **mb_loc**: A float array of shape $(K, 4)$, where K is the number of default bounding boxes.
- **mb_label**: An integer array of shape $(K,)$.

Return type tuple of two arrays

Normalize

class `chainercv.links.model.ssd.`**`Normalize`**(*n_channel*, *initial=0*, *eps=1e-05*)

Learnable L2 normalization[5].

This link normalizes input along the channel axis and scales it. The scale factors are trained channel-wise.

Parameters

---

[5] Wei Liu, Andrew Rabinovich, Alexander C. Berg. ParseNet: Looking Wider to See Better. ICLR 2016.

- **n_channel** (*int*) – The number of channels.
- **initial** – A value to initialize the scale factors. It is pased to `chainer.initializers._get_initializer()`. The default value is 0.
- **eps** (*float*) – A small value to avoid zero-division. The default value is $1e-5$.

__call__(*x*)

Normalize input and scale it.

Parameters **x** (*chainer.Variable*) – A variable holding 4-dimensional array. Its `dtype` is `numpy.float32`.

Returns The shape and `dtype` are same as those of input.

Return type chainer.Variable

SSD

class `chainercv.links.model.ssd.`**SSD**(*extractor, multibox, steps, sizes, variance=(0.1, 0.2), mean=0*)

Base class of Single Shot Multibox Detector.

This is a base class of Single Shot Multibox Detector[6].

Parameters

- **extractor** – A link which extracts feature maps. This link must have `insize`, `grids` and `__call__()`.
  - `insize`: An integer which indicates the size of input images. Images are resized to this size before feature extraction.
  - `grids`: An iterable of integer. Each integer indicates the size of feature map. This value is used by `MultiBboxCoder`.
  - `__call_()`: A method which computes feature maps. It must take a batched images and return batched feature maps.
- **multibox** – A link which computes `mb_locs` and `mb_confs` from feature maps. This link must have `n_class`, `aspect_ratios` and `__call__()`.
  - `n_class`: An integer which indicates the number of classes. This value should include the background class.
  - `aspect_ratios`: An iterable of tuple of integer. Each tuple indicates the aspect ratios of default bounding boxes at each feature maps. This value is used by `MultiboxCoder`.
  - `__call__()`: A method which computes `mb_locs` and `mb_confs`. It must take a batched feature maps and return `mb_locs` and `mb_confs`.
- **steps** (*iterable of float*) – The step size for each feature map. This value is used by `MultiboxCoder`.
- **sizes** (*iterable of float*) – The base size of default bounding boxes for each feature map. This value is used by `MultiboxCoder`.
- **variance** (*tuple of floats*) – Two coefficients for decoding the locations of bounding boxe. This value is used by `MultiboxCoder`. The default value is `(0.1, 0.2)`.

---

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

- **nms_thresh** (*float*) – The threshold value for `non_maximum_suppression()`. The default value is `0.45`. This value can be changed directly or by using `use_preset()`.
- **score_thresh** (*float*) – The threshold value for confidence score. If a bounding box whose confidence score is lower than this value, the bounding box will be suppressed. The default value is `0.6`. This value can be changed directly or by using `use_preset()`.

`__call__`(*x*)

Compute localization and classification from a batch of images.

This method computes two variables, `mb_locs` and `mb_confs`. `self.coder.decode()` converts these variables to bounding box coordinates and confidence scores. These variables are also used in training SSD.

Parameters **x** (*chainer.Variable*) – A variable holding a batch of images. The images are preprocessed by `_prepare()`.

Returns

This method returns two variables, `mb_locs` and `mb_confs`.

- **mb_locs**: A variable of float arrays of shape $(B, K, 4)$, where B is the number of samples in the batch and K is the number of default bounding boxes.
- **mb_confs**: A variable of float arrays of shape $(B, K, n\_fg\_class + 1)$.

Return type tuple of chainer.Variable

`predict`(*imgs*)

Detect objects from images.

This method predicts objects for each image.

Parameters **imgs** (*iterable of numpy.ndarray*) – Arrays holding images. All images are in CHW and RGB format and the range of their value is $[0, 255]$.

Returns

This method returns a tuple of three lists, `(bboxes, labels, scores)`.

- **bboxes**: A list of float arrays of shape $(R, 4)$, where R is the number of bounding boxes in a image. Each bouding box is organized by $(y_{min}, x_{min}, y_{max}, x_{max})$ in the second axis.
- **labels** : A list of integer arrays of shape $(R,)$. Each value indicates the class of the bounding box. Values are in range $[0, L - 1]$, where L is the number of the foreground classes.
- **scores** : A list of float arrays of shape $(R,)$. Each value indicates how confident the prediction is.

Return type tuple of lists

`use_preset`(*preset*)

Use the given preset during prediction.

This method changes values of `nms_thresh` and `score_thresh`. These values are a threshold value used for non maximum suppression and a threshold value to discard low confidence proposals in `predict()`, respectively.

If the attributes need to be changed to something other than the values provided in the presets, please modify them by directly accessing the public attributes.

Parameters **preset** (*{'visualize', 'evaluate'}*) – A string to determine the preset to use.

VGG16

class `chainercv.links.model.ssd.`**`VGG16`**

An extended VGG-16 model for SSD300 and SSD512.

This is an extended VGG-16 model proposed in[7]. The differences from original VGG-16[8] are shown below.

- `conv5_1`, `conv5_2` and `conv5_3` are changed from `Convolution2d` to `DilatedConvolution2d`.
- *`Normalize`* is inserted after `conv4_3`.
- The parameters of max pooling after `conv5_3` are changed.
- `fc6` and `fc7` are converted to `conv6` and `conv7`.

VGG16Extractor300

class `chainercv.links.model.ssd.`**`VGG16Extractor300`**

A VGG-16 based feature extractor for SSD300.

This is a feature extractor for *`SSD300`*. This extractor is based on *`VGG16`*.

`__call__`(*x*)

Compute feature maps from a batch of images.

This method extracts feature maps from `conv4_3`, `conv7`, `conv8_2`, `conv9_2`, `conv10_2`, and `conv11_2`.

Parameters **x** (*ndarray*) – An array holding a batch of images. The images should be resized to 300×300.

Returns Each variable contains a feature map.

Return type list of Variable

VGG16Extractor512

class `chainercv.links.model.ssd.`**`VGG16Extractor512`**

A VGG-16 based feature extractor for SSD512.

This is a feature extractor for *`SSD512`*. This extractor is based on *`VGG16`*.

`__call__`(*x*)

Compute feature maps from a batch of images.

This method extracts feature maps from `conv4_3`, `conv7`, `conv8_2`, `conv9_2`, `conv10_2`, `conv11_2`, and `conv12_2`.

Parameters **x** (*ndarray*) – An array holding a batch of images. The images should be resized to 512×512.

Returns Each variable contains a feature map.

Return type list of Variable

---

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

[8] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR 2015.

Train-only Utility

GradientScaling

class `chainercv.links.model.ssd.`**`GradientScaling`**(*rate*)

Optimizer/UpdateRule hook function for scaling gradient.

This hook function scales gradient by a constant value.

Parameters **`rate`** (*`float`*) – Coefficient for scaling.

Variables **`rate`** (*`float`*) – Coefficient for scaling.

multibox_loss

`chainercv.links.model.ssd.`**`multibox_loss`**(*mb_locs*, *mb_confs*, *gt_mb_locs*, *gt_mb_labels*, *k*)

Computes multibox losses.

This is a loss function used in[9]. This function returns `loc_loss` and `conf_loss`. `loc_loss` is a loss for localization and `conf_loss` is a loss for classification. The formulas of these losses can be found in the equation (2) and (3) in the original paper.

Parameters

- **`mb_locs`** (*`chainer.Variable or array`*) – The offsets and scales for predicted bounding boxes. Its shape is $(B, K, 4)$, where B is the number of samples in the batch and K is the number of default bounding boxes.
- **`mb_confs`** (*`chainer.Variable or array`*) – The classes of predicted bounding boxes. Its shape is (B, K, n_class). This function assumes the first class is background (negative).
- **`gt_mb_locs`** (*`chainer.Variable or array`*) – The offsets and scales for ground truth bounding boxes. Its shape is $(B, K, 4)$.
- **`gt_mb_labels`** (*`chainer.Variable or array`*) – The classes of ground truth bounding boxes. Its shape is (B, K).
- **`k`** (*`float`*) – A coefficient which is used for hard negative mining. This value determines the ratio between the number of positives and that of mined negatives. The value used in the original paper is `3`.

Returns This function returns two `chainer.Variable`: `loc_loss` and `conf_loss`.

Return type tuple of chainer.Variable

---

[9] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

random_crop_with_bbox_constraints

`chainercv.links.model.ssd.`**`random_crop_with_bbox_constraints`**(*img*, *bbox*, *min_scale=0.3*, *max_scale=1*, *max_aspect_ratio=2*, *constraints=None*, *max_trial=50*, *return_param=False*)

Crop an image randomly with bounding box constraints.

This data augmentation is used in training of Single Shot Multibox Detector[10]. More details can be found in data augmentation section of the original paper.

Parameters

- **`img`** (*ndarray*) – An image array to be cropped. This is in CHW format.
- **`bbox`** (*ndarray*) – Bounding boxes used for constraints. The shape is $(R, 4)$. R is the number of bounding boxes.
- **`min_scale`** (*float*) – The minimum ratio between a cropped region and the original image. The default value is `0.3`.
- **`max_scale`** (*float*) – The maximum ratio between a cropped region and the original image. The default value is `1`.
- **`max_aspect_ratio`** (*float*) – The maximum aspect ratio of cropped region. The default value is `2`.
- **`constaraints`** (*iterable of tuples*) – An iterable of constraints. Each constraint should be `(min_iou, max_iou)` format. If you set `min_iou` or `max_iou` to `None`, it means not limited. If this argument is not specified, `((0.1, None), (0.3, None), (0.5, None), (0.7, None), (0.9, None), (None, 1))` will be used.
- **`max_trial`** (*int*) – The maximum number of trials to be conducted for each constraint. If this function can not find any region that satisfies the constraint in max_trial trials, this function skips the constraint. The default value is `50`.
- **`return_param`** (*bool*) – If `True`, this function returns information of intermediate values.

Returns

If `return_param = False`, returns an array `img` that is cropped from the input array.

If `return_param = True`, returns a tuple whose elements are `img, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **constraint** (*tuple*): The chosen constraint.
- **y_slice** (*slice*): A slice in vertical direction used to crop the input image.
- **x_slice** (*slice*): A slice in horizontal direction used to crop the input image.

Return type ndarray or (ndarray, dict)

---

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

random_distort

`chainercv.links.model.ssd.`**`random_distort`**(*img*, *brightness_delta=32*, *contrast_low=0.5*, *contrast_high=1.5*, *saturation_low=0.5*, *saturation_high=1.5*, *hue_delta=18*)

A color related data augmentation used in SSD.

This function is a combination of four augmentation methods: brightness, contrast, saturation and hue.

- brightness: Adding a random offset to the intensity of the image.
- contrast: Multiplying the intensity of the image by a random scale.
- saturation: Multiplying the saturation of the image by a random scale.
- hue: Adding a random offset to the hue of the image randomly.

This data augmentation is used in training of Single Shot Multibox Detector[11].

Note that this function requires `cv2`.

Parameters

- **img** (*ndarray*) – An image array to be augmented. This is in CHW and RGB format.
- **brightness_delta** (*float*) – The offset for saturation will be drawn from $[-brightness\_delta, brightness\_delta]$. The default value is `32`.
- **contrast_low** (*float*) – The scale for contrast will be drawn from $[contrast\_low, contrast\_high]$. The default value is `0.5`.
- **contrast_high** (*float*) – See `contrast_low`. The default value is `1.5`.
- **saturation_low** (*float*) – The scale for saturation will be drawn from $[saturation\_low, saturation\_high]$. The default value is `0.5`.
- **saturation_high** (*float*) – See `saturation_low`. The default value is `1.5`.
- **hue_delta** (*float*) – The offset for hue will be drawn from $[-hue\_delta, hue\_delta]$. The default value is `18`.

Returns An image in CHW and RGB format.

resize_with_random_interpolation

`chainercv.links.model.ssd.`**`resize_with_random_interpolation`**(*img*, *size*, *return_param=False*)

Resize an image with a randomly selected interpolation method.

This function is similar to `chainercv.transforms.resize()`, but this chooses the interpolation method randomly.

This data augmentation is used in training of Single Shot Multibox Detector[12].

Note that this function requires `cv2`.

Parameters

- **img** (*ndarray*) – An array to be transformed. This is in CHW format and the type should be `numpy.float32`.

---

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

- **`size`** (*`tuple`*) – This is a tuple of length 2. Its elements are ordered as (height, width).
- **`return_param`** (*`bool`*) – Returns information of interpolation.

Returns

If `return_param = False`, returns an array `img` that is the result of rotation.

If `return_param = True`, returns a tuple whose elements are `img, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **interpolatation**: The chosen interpolation method.

Return type ndarray or (ndarray, dict)

Semantic Segmentation

Semantic segmentation links share a common method `predict()` to conduct semantic segmentation of images. For more details, please read *`SegNetBasic.predict()`*.

SegNet

Semantic Segmentation Link

SegNetBasic

class `chainercv.links.model.segnet.`**`SegNetBasic`**(*n_class=None*, *pretrained_model=None*, *initialW=None*)

SegNet Basic for semantic segmentation.

This is a SegNet[1] model for semantic segmenation. This is based on SegNetBasic model that is found here.

When you specify the path of a pretrained chainer model serialized as a `npz` file in the constructor, this chain model automatically initializes all the parameters with it. When a string in prespecified set is provided, a pretrained model is loaded from weights distributed on the Internet. The list of pretrained models supported are as follows:

- `camvid`: Loads weights trained with the train split of CamVid dataset.

Parameters

- **`n_class`** (*`int`*) – The number of classes. If `None`, it can be infered if `pretrained_model` is given.
- **`pretrained_model`** (*`str`*) – The destination of the pretrained chainer model serialized as a `npz` file. If this is one of the strings described above, it automatically loads weights stored under a directory `$CHAINER_DATASET_ROOT/pfnet/chainercv/models/`, where `$CHAINER_DATASET_ROOT` is set as `$HOME/.chainer/dataset` unless you specify another value by modifying the environment variable.
- **`initialW`** (*`callable`*) – Initializer for convolution layers.

`__call__`(*x*)

Compute an image-wise score from a batch of images

---

[1] Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation.” PAMI, 2017

Parameters **x** (*chainer.Variable*) – A variable with 4D image array.

Returns An image-wise score. Its channel size is `self.n_class`.

Return type chainer.Variable

predict (*imgs*)

Conduct semantic segmentations from images.

Parameters **imgs** (*iterable of numpy.ndarray*) – Arrays holding images. All images are in CHW and RGB format and the range of their values are $[0, 255]$.

Returns List of integer labels predicted from each image in the input list.

Return type list of numpy.ndarray

Classifiers

Classifier

PixelwiseSoftmaxClassifier

class chainercv.links.**PixelwiseSoftmaxClassifier** (*predictor*, *ignore_label=-1*, *class_weight=None*)

A pixel-wise classifier.

It computes the loss based on a given input/label pair for semantic segmentation.

Parameters

- **predictor** (*Link*) – Predictor network.
- **ignore_label** (*int*) – A class id that is going to be ignored in evaluation. The default value is -1.
- **class_weight** (*array*) – An array that contains constant weights that will be multiplied with the loss values along with the channel dimension. This will be used in `chainer.functions.softmax_cross_entropy()`.

__call__ (*x*, *t*)

Computes the loss value for an image and label pair.

Parameters

- **x** (*Variable*) – A variable with a batch of images.
- **t** (*Variable*) – A variable with the ground truth image-wise label.

Returns Loss value.

Return type Variable

3.4.2 Connection

Connection

Conv2DActiv

class `chainercv.links.connection.`**`Conv2DActiv`**(*in_channels, out_channels, ksize=None, stride=1, pad=0, nobias=False, initialW=None, initial_bias=None, activ=<function relu>*)

Convolution2D –> Activation

This is a chain that does two-dimensional convolution and applies an activation.

The arguments are the same as those of `chainer.links.Convolution2D` except for `activ`.

Example

There are sevaral ways to initialize a `Conv2DActiv`.

1. Give the first three arguments explicitly:

```
>>> l = Conv2DActiv(5, 10, 3)
```

2. Omit `in_channels` or fill it with `None`:

   In these ways, attributes are initialized at runtime based on the channel size of the input.

```
>>> l = Conv2DActiv(10, 3)
>>> l = Conv2DActiv(None, 10, 3)
```

Parameters

- **`in_channels`** (*`int or None`*) – Number of channels of input arrays. If `None`, parameter initialization will be deferred until the first forward data pass at which time the size will be determined.
- **`out_channels`** (*`int`*) – Number of channels of output arrays.
- **`ksize`** (*`int or pair of ints`*) – Size of filters (a.k.a. kernels). `ksize=k` and `ksize=(k, k)` are equivalent.
- **`stride`** (*`int or pair of ints`*) – Stride of filter applications. `stride=s` and `stride=(s, s)` are equivalent.
- **`pad`** (*`int or pair of ints`*) – Spatial padding width for input arrays. `pad=p` and `pad=(p, p)` are equivalent.
- **`nobias`** (*`bool`*) – If `True`, then this link does not use the bias term.
- **`initialW`** (*`4-D array`*) – Initial weight value. If `None`, the default initializer is used. May also be a callable that takes `numpy.ndarray` or `cupy.ndarray` and edits its value.
- **`initial_bias`** (*`1-D array`*) – Initial bias value. If `None`, the bias is set to 0. May also be a callable that takes `numpy.ndarray` or `cupy.ndarray` and edits its value.
- **`activ`** (*`callable`*) – An activation function. The default value is `chainer.functions.relu()`.

Conv2DBNActiv

class `chainercv.links.connection.`**`Conv2DBNActiv`**(*in_channels*, *out_channels*, *ksize=None*, *stride=1*, *pad=0*, *nobias=True*, *initialW=None*, *initial_bias=None*, *activ=<function relu>*, *bn_kwargs={}*)

Convolution2D –> Batch Normalization –> Activation

This is a chain that sequentially apllies a two-dimensional convolution, a batch normalization and an activation.

The arguments are the same as that of `chainer.links.Convolution2D` except for `activ` and `bn_kwargs`. Note that the default value for the `nobias` is changed to `True`.

Example

There are sevaral ways to initialize a *`Conv2DBNActiv`*.

1. Give the first three arguments explicitly:

```
>>> l = Conv2DBNActiv(5, 10, 3)
```

2. Omit `in_channels` or fill it with `None`:

   In these ways, attributes are initialized at runtime based on the channel size of the input.

```
>>> l = Conv2DBNActiv(10, 3)
>>> l = Conv2DBNActiv(None, 10, 3)
```

Parameters

- **`in_channels`** (*int or None*) – Number of channels of input arrays. If `None`, parameter initialization will be deferred until the first forward data pass at which time the size will be determined.
- **`out_channels`** (*int*) – Number of channels of output arrays.
- **`ksize`** (*int or pair of ints*) – Size of filters (a.k.a. kernels). `ksize=k` and `ksize=(k, k)` are equivalent.
- **`stride`** (*int or pair of ints*) – Stride of filter applications. `stride=s` and `stride=(s, s)` are equivalent.
- **`pad`** (*int or pair of ints*) – Spatial padding width for input arrays. `pad=p` and `pad=(p, p)` are equivalent.
- **`nobias`** (*bool*) – If `True`, then this link does not use the bias term.
- **`initialW`** (*4-D array*) – Initial weight value. If `None`, the default initializer is used. May also be a callable that takes `numpy.ndarray` or `cupy.ndarray` and edits its value.
- **`initial_bias`** (*1-D array*) – Initial bias value. If `None`, the bias is set to 0. May also be a callable that takes `numpy.ndarray` or `cupy.ndarray` and edits its value.
- **`activ`** (*callable*) – An activation function. The default value is `chainer.functions.relu()`.
- **`bn_kwargs`** (*dict*) – Keyword arguments passed to initialize `chainer.links.BatchNormalization`.

3.5 Transforms

3.5.1 Image

center_crop

`chainercv.transforms.`**`center_crop`**(*img*, *size*, *return_param=False*, *copy=False*)

Center crop an image by *size*.

An image is cropped to `size`. The center of the output image and the center of the input image are same.

Parameters

- **`img`** (*`ndarray`*) – An image array to be cropped. This is in CHW format.
- **`size`** (*`tuple`*) – The size of output image after cropping. This value is $(height, width)$.
- **`return_param`** (*`bool`*) – If `True`, this function returns information of slices.
- **`copy`** (*`bool`*) – If `False`, a view of `img` is returned.

Returns

If `return_param = False`, returns an array `out_img` that is cropped from the input array.

If `return_param = True`, returns a tuple whose elements are `out_img, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **y_slice** (*slice*): A slice used to crop the input image. The relation below holds together with `x_slice`.
- **x_slice** (*slice*): Similar to `y_slice`.

```
out_img = img[:, y_slice, x_slice]
```

Return type ndarray or (ndarray, dict)

flip

`chainercv.transforms.`**`flip`**(*img*, *y_flip=False*, *x_flip=False*, *copy=False*)

Flip an image in vertical or horizontal direction as specified.

Parameters

- **`img`** (*`ndarray`*) – An array that gets flipped. This is in CHW format.
- **`y_flip`** (*`bool`*) – Flip in vertical direction.
- **`x_flip`** (*`bool`*) – Flip in horizontal direction.
- **`copy`** (*`bool`*) – If False, a view of `img` will be returned.

Returns Transformed `img` in CHW format.

pca_lighting

`chainercv.transforms.`**`pca_lighting`**(*img*, *sigma*, *eigen_value=None*, *eigen_vector=None*)

AlexNet style color augmentation

This method adds a noise vector drawn from a Gaussian. The direction of the Gaussian is same as that of the principal components of the dataset.

This method is used in training of AlexNet[1].

Parameters

- **`img`** (*ndarray*) – An image array to be augmented. This is in CHW and RGB format.
- **`sigma`** (*float*) – Standard deviation of the Gaussian. In the original paper, this value is 10% of the range of intensity (25.5 if the range is $[0, 255]$).
- **`eigen_value`** (*ndarray*) – An array of eigen values. The shape has to be $(3,)$. If it is not specified, the values computed from ImageNet are used.
- **`eigen_vector`** (*ndarray*) – An array of eigen vectors. The shape has to be $(3, 3)$. If it is not specified, the vectors computed from ImageNet are used.

Returns An image in CHW format.

random_crop

`chainercv.transforms.`**`random_crop`**(*img*, *size*, *return_param=False*, *copy=False*)

Crop array randomly into *size*.

The input image is cropped by a randomly selected region whose shape is `size`.

Parameters

- **`img`** (*ndarray*) – An image array to be cropped. This is in CHW format.
- **`size`** (*tuple*) – The size of output image after cropping. This value is $(height, width)$.
- **`return_param`** (*bool*) – If `True`, this function returns information of slices.
- **`copy`** (*bool*) – If `False`, a view of `img` is returned.

Returns

If `return_param = False`, returns an array `out_img` that is cropped from the input array.

If `return_param = True`, returns a tuple whose elements are `out_img, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **y_slice** (*slice*): A slice used to crop the input image. The relation below holds together with `x_slice`.
- **x_slice** (*slice*): Similar to `x_slice`.

```
out_img = img[:, y_slice, x_slice]
```

Return type ndarray or (ndarray, dict)

random_expand

`chainercv.transforms.`**`random_expand`**(*img*, *max_ratio=4*, *fill=0*, *return_param=False*)

Expand an image randomly.

---

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012.

This method randomly place the input image on a larger canvas. The size of the canvas is (rH, rW), where (H, W) is the size of the input image and r is a random ratio drawn from $[1, max\_ratio]$. The canvas is filled by a value `fill` except for the region where the original image is placed.

This data augmentation trick is used to create "zoom out" effect[2].

Parameters

- **img** (*ndarray*) – An image array to be augmented. This is in CHW format.
- **max_ratio** (*float*) – The maximum ratio of expansion. In the original paper, this value is 4.
- **fill** (*float, tuple or ndarray*) – The value of padded pixels. In the original paper, this value is the mean of ImageNet. If it is `numpy.ndarray`, its shape should be $(C, 1, 1)$, where C is the number of channels of `img`.
- **return_param** (*bool*) – Returns random parameters.

Returns

If `return_param = False`, returns an array `out_img` that is the result of expansion.

If `return_param = True`, returns a tuple whose elements are `out_img, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **ratio** (*float*): The sampled value used to make the canvas.
- **y_offset** (*int*): The y coodinate of the top left corner of the image after placing on the canvas.
- **x_offset** (*int*): The x coordinate of the top left corner of the image after placing on the canvas.

Return type ndarray or (ndarray, dict)

random_flip

`chainercv.transforms.`**`random_flip`**(*img*, *y_random=False*, *x_random=False*, *return_param=False*, *copy=False*)

Randomly flip an image in vertical or horizontal direction.

Parameters

- **img** (*ndarray*) – An array that gets flipped. This is in CHW format.
- **y_random** (*bool*) – Randomly flip in vertical direction.
- **x_random** (*bool*) – Randomly flip in horizontal direction.
- **return_param** (*bool*) – Returns information of flip.
- **copy** (*bool*) – If False, a view of `img` will be returned.

Returns

If `return_param = False`, returns an array `out_img` that is the result of flipping.

If `return_param = True`, returns a tuple whose elements are `out_img, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **y_flip** (*bool*): Whether the image was flipped in the vertical direction or not.

---

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

- **x_flip** (*bool*): Whether the image was flipped in the horizontal direction or not.

Return type ndarray or (ndarray, dict)

random_rotate

`chainercv.transforms.`**`random_rotate`**(*img*, *return_param=False*)

Randomly rotate images by 90, 180, 270 or 360 degrees.

Parameters

- **`img`** (*`ndarray`*) – An arrays that get flipped. This is in CHW format.
- **`return_param`** (*`bool`*) – Returns information of rotation.

Returns

If `return_param = False`, returns an array `out_img` that is the result of rotation.

If `return_param = True`, returns a tuple whose elements are `out_img, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **k** (*int*): The integer that represents the number of times the image is rotated by 90 degrees.

Return type ndarray or (ndarray, dict)

resize

`chainercv.transforms.`**`resize`**(*img*, *size*, *interpolation=2*)

Resize image to match the given shape.

This method uses `cv2` or `PIL` for the backend. If `cv2` is installed, this function uses the implementation in `cv2`. This implementation is faster than the implementation in `PIL`. Under Anaconda environment, `cv2` can be installed by the following command.

```
$ conda install -c menpo opencv3=3.2.0
```

Parameters

- **`img`** (*`ndarray`*) – An array to be transformed. This is in CHW format and the type should be `numpy.float32`.
- **`size`** (*`tuple`*) – This is a tuple of length 2. Its elements are ordered as (height, width).
- **`interpolation`** (*`int`*) – Determines sampling strategy. This is one of `PIL.Image.NEAREST`, `PIL.Image.BILINEAR`, `PIL.Image.BICUBIC`, `PIL.Image.LANCZOS`. Bilinear interpolation is the default strategy.

Returns A resize array in CHW format.

Return type ndarray

resize_contain

`chainercv.transforms.`**`resize_contain`**(*img*, *size*, *fill=0*, *return_param=False*)

Resize the image to fit in the given area while keeping aspect ratio.

If both the height and the width in `size` are larger than the height and the width of the `img`, the `img` is placed on the center with an appropriate padding to match `size`.

Otherwise, the input image is scaled to fit in a canvas whose size is `size` while preserving aspect ratio.

Parameters

- **`img`** (*`ndarray`*) – An array to be transformed. This is in CHW format.
- **`size`** (*`tuple of two ints`*) – A tuple of two elements: `height, width`. The size of the image after resizing.
- **`fill`** (*`float, tuple or ndarray`*) – The value of padded pixels. If it is `numpy.ndarray`, its shape should be $(C, 1, 1)$, where C is the number of channels of `img`.
- **`return_param`** (*`bool`*) – Returns information of resizing and offsetting.

Returns

If `return_param = False`, returns an array `out_img` that is the result of resizing.

If `return_param = True`, returns a tuple whose elements are `out_img, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **y_offset** (*int*): The y coodinate of the top left corner of the image after placing on the canvas.
- **x_offset** (*int*): The x coordinate of the top left corner of the image after placing on the canvas.
- **scaled_size** (*tuple*): The size to which the image is scaled to before placing it on a canvas. This is a tuple of two elements: `height, width`.

Return type ndarray or (ndarray, dict)

scale

`chainercv.transforms.`**`scale`**`(`*img*, *size*, *fit_short=True*, *interpolation=2*`)`

Rescales the input image to the given "size".

When `fit_short == True`, the input image will be resized so that the shorter edge will be scaled to length `size` after resizing. For example, if the height of the image is larger than its width, image will be resized to (size * height / width, size).

Otherwise, the input image will be resized so that the longer edge will be scaled to length `size` after resizing.

Parameters

- **`img`** (*`ndarray`*) – An image array to be scaled. This is in CHW format.
- **`size`** (*`int`*) – The length of the smaller edge.
- **`fit_short`** (*`bool`*) – Determines whether to match the length of the shorter edge or the longer edge to `size`.
- **`interpolation`** (*`int`*) – Determines sampling strategy. This is one of `PIL.Image.NEAREST`, `PIL.Image.BILINEAR`, `PIL.Image.BICUBIC`, `PIL.Image.LANCZOS`. Bilinear interpolation is the default strategy.

Returns A scaled image in CHW format.

Return type ndarray

ten_crop

`chainercv.transforms.`**`ten_crop`**(*img*, *size*)

Crop 10 regions from an array.

This method crops 10 regions. All regions will be in shape `size`. These regions consist of 1 center crop and 4 corner crops and horizontal flips of them.

The crops are ordered in this order.

- center crop
- top-left crop
- bottom-left crop
- top-right crop
- bottom-right crop
- center crop (flipped horizontally)
- top-left crop (flipped horizontally)
- bottom-left crop (flipped horizontally)
- top-right crop (flipped horizontally)
- bottom-right crop (flipped horizontally)

Parameters

- **`img`** (*`ndarray`*) – An image array to be cropped. This is in CHW format.
- **`size`** (*`tuple`*) – The size of output images after cropping. This value is $(height, width)$.

Returns The cropped arrays. The shape of tensor is $(10, C, H, W)$.

3.5.2 Bounding Box

crop_bbox

`chainercv.transforms.`**`crop_bbox`**(*bbox*, *y_slice=None*, *x_slice=None*, *allow_outside_center=True*, *return_param=False*)

Translate bounding boxes to fit within the cropped area of an image.

This method is mainly used together with image cropping. This method translates the coordinates of bounding boxes like *`translate_bbox()`*. In addition, this function truncates the bounding boxes to fit within the cropped area. If a bounding box does not overlap with the cropped area, this bounding box will be removed.

The bounding boxes are expected to be packed into a two dimensional tensor of shape $(R, 4)$, where R is the number of bounding boxes in the image. The second axis represents attributes of the bounding box. They are $(y_{min}, x_{min}, y_{max}, x_{max})$, where the four attributes are coordinates of the top left and the bottom right vertices.

Parameters

- **`bbox`** (*`ndarray`*) – Bounding boxes to be transformed. The shape is $(R, 4)$. R is the number of bounding boxes.
- **`y_slice`** (*`slice`*) – The slice of y axis.
- **`x_slice`** (*`slice`*) – The slice of x axis.

- **`allow_outside_center`** (*`bool`*) – If this argument is `False`, bounding boxes whose centers are outside of the cropped area are removed. The default value is `True`.
- **`return_param`** (*`bool`*) – If `True`, this function returns indices of kept bounding boxes.

Returns

If `return_param = False`, returns an array `bbox`.

If `return_param = True`, returns a tuple whose elements are `bbox, param`. `param` is a dictionary of intermediate parameters whose contents are listed below with key, value-type and the description of the value.

- **index** (*numpy.ndarray*): An array holding indices of used bounding boxes.

Return type ndarray or (ndarray, dict)

flip_bbox

`chainercv.transforms.`**`flip_bbox`**(*bbox*, *size*, *y_flip=False*, *x_flip=False*)

Flip bounding boxes accordingly.

The bounding boxes are expected to be packed into a two dimensional tensor of shape $(R, 4)$, where R is the number of bounding boxes in the image. The second axis represents attributes of the bounding box. They are $(y_{min}, x_{min}, y_{max}, x_{max})$, where the four attributes are coordinates of the top left and the bottom right vertices.

Parameters

- **`bbox`** (*`ndarray`*) – An array whose shape is $(R, 4)$. R is the number of bounding boxes.
- **`size`** (*`tuple`*) – A tuple of length 2. The height and the width of the image before resized.
- **`y_flip`** (*`bool`*) – Flip bounding box according to a vertical flip of an image.
- **`x_flip`** (*`bool`*) – Flip bounding box according to a horizontal flip of an image.

Returns Bounding boxes flipped according to the given flips.

Return type ndarray

resize_bbox

`chainercv.transforms.`**`resize_bbox`**(*bbox*, *in_size*, *out_size*)

Resize bounding boxes according to image resize.

The bounding boxes are expected to be packed into a two dimensional tensor of shape $(R, 4)$, where R is the number of bounding boxes in the image. The second axis represents attributes of the bounding box. They are $(y_{min}, x_{min}, y_{max}, x_{max})$, where the four attributes are coordinates of the top left and the bottom right vertices.

Parameters

- **`bbox`** (*`ndarray`*) – An array whose shape is $(R, 4)$. R is the number of bounding boxes.
- **`in_size`** (*`tuple`*) – A tuple of length 2. The height and the width of the image before resized.
- **`out_size`** (*`tuple`*) – A tuple of length 2. The height and the width of the image after resized.

Returns Bounding boxes rescaled according to the given image shapes.

Return type ndarray

translate_bbox

`chainercv.transforms.`**`translate_bbox`**(*bbox*, *y_offset=0*, *x_offset=0*)

Translate bounding boxes.

This method is mainly used together with image transforms, such as padding and cropping, which translates the left top point of the image from coordinate $(0,0)$ to coordinate $(y,x) = (y_{offset}, x_{offset})$.

The bounding boxes are expected to be packed into a two dimensional tensor of shape $(R,4)$, where R is the number of bounding boxes in the image. The second axis represents attributes of the bounding box. They are $(y_{min}, x_{min}, y_{max}, x_{max})$, where the four attributes are coordinates of the top left and the bottom right vertices.

Parameters

- **`bbox`** (*`ndarray`*) – Bounding boxes to be transformed. The shape is $(R,4)$. R is the number of bounding boxes.
- **`y_offset`** (*`int or float`*) – The offset along y axis.
- **`x_offset`** (*`int or float`*) – The offset along x axis.

Returns Bounding boxes translated according to the given offsets.

Return type ndarray

3.5.3 Keypoint

flip_keypoint

`chainercv.transforms.`**`flip_keypoint`**(*keypoint*, *size*, *y_flip=False*, *x_flip=False*)

Modify keypoints according to image flips.

Parameters

- **`keypoint`** (*`ndarray`*) – Keypoints in the image. The shape of this array is $(K,2)$. K is the number of keypoints in the image. The last dimension is composed of y and x coordinates of the keypoints.
- **`size`** (*`tuple`*) – A tuple of length 2. The height and the width of the image which is associated with the keypoints.
- **`y_flip`** (*`bool`*) – Modify keypoints according to a vertical flip of an image.
- **`x_flip`** (*`bool`*) – Modify keypoints according to a horizontal flip of an image.

Returns Keypoints modified according to image flips.

Return type ndarray

resize_keypoint

`chainercv.transforms.`**`resize_keypoint`**(*keypoint*, *in_size*, *out_size*)

Change values of keypoint according to paramters for resizing an image.

Parameters

- **`keypoint`** (*`ndarray`*) – Keypoints in the image. The shape of this array is $(K,2)$. K is the number of keypoint in the image. The last dimension is composed of y and x coordinates of the keypoints.

- **in_size** (*tuple*) – A tuple of length 2. The height and the width of the image before resized.
- **out_size** (*tuple*) – A tuple of length 2. The height and the width of the image after resized.

Returns Keypoint rescaled according to the given image shapes.

Return type ndarray

translate_keypoint

`chainercv.transforms.`**`translate_keypoint`**(*keypoint*, *y_offset=0*, *x_offset=0*)

Translate keypoints.

This method is mainly used together with image transforms, such as padding and cropping, which translates the top left point of the image to the coordinate $(y, x) = (y_{offset}, x_{offset})$.

Parameters

- **keypoint** (*ndarray*) – Keypoints in the image. The shape of this array is $(K, 2)$. K is the number of keypoints in the image. The last dimension is composed of y and x coordinates of the keypoints.
- **y_offset** (*int or float*) – The offset along y axis.
- **x_offset** (*int or float*) – The offset along x axis.

Returns Keypoints modified translation of an image.

Return type ndarray

3.6 Visualizations

3.6.1 vis_bbox

`chainercv.visualizations.`**`vis_bbox`**(*img*, *bbox*, *label=None*, *score=None*, *label_names=None*, *ax=None*)

Visualize bounding boxes inside image.

Example

```
>>> from chainercv.datasets import VOCDetectionDataset
>>> from chainercv.datasets import voc_bbox_label_names
>>> from chainercv.visualizations import vis_bbox
>>> import matplotlib.pyplot as plot
>>> dataset = VOCDetectionDataset()
>>> img, bbox, label = dataset[60]
>>> vis_bbox(img, bbox, label,
...          label_names=voc_bbox_label_names)
>>> plot.show()
```

Parameters

- **img** (*ndarray*) – An array of shape $(3, height, width)$. This is in RGB format and the range of its value is $[0, 255]$.

- **bbox** (*ndarray*) – An array of shape $(R, 4)$, where R is the number of bounding boxes in the image. Each element is organized by $(y_{min}, x_{min}, y_{max}, x_{max})$ in the second axis.
- **label** (*ndarray*) – An integer array of shape $(R,)$. The values correspond to id for label names stored in `label_names`. This is optional.
- **score** (*ndarray*) – A float array of shape $(R,)$. Each value indicates how confident the prediction is. This is optional.
- **label_names** (*iterable of strings*) – Name of labels ordered according to label ids. If this is `None`, labels will be skipped.
- **ax** (*matplotlib.axes.Axis*) – The visualization is displayed on this axis. If this is `None` (default), a new axis is created.

Returns Returns the Axes object with the plot for further tweaking.

Return type Axes

3.6.2 vis_image

`chainercv.visualizations.`**`vis_image`**(*img*, *ax=None*)

Visualize a color image.

Parameters

- **img** (*ndarray*) – An array of shape $(3, height, width)$. This is in RGB format and the range of its value is $[0, 255]$.
- **ax** (*matplotlib.axes.Axis*) – The visualization is displayed on this axis. If this is `None` (default), a new axis is created.

Returns Returns the Axes object with the plot for further tweaking.

Return type Axes

3.6.3 vis_keypoint

`chainercv.visualizations.`**`vis_keypoint`**(*img*, *keypoint*, *kp_mask=None*, *ax=None*)

Visualize keypoints in an image.

Example

```
>>> import chainercv
>>> import matplotlib.pyplot as plot
>>> dataset = chainercv.datasets.CUBKeypointDataset()
>>> img, keypoint, kp_mask = dataset[0]
>>> chainercv.visualizations.vis_keypoint(img, keypoint, kp_mask)
>>> plot.show()
```

Parameters

- **img** (*ndarray*) – An image of shape $(3, height, width)$. This is in RGB format and the range of its value is $[0, 255]$. This should be visualizable using `matplotlib.pyplot.imshow(img)`

- **keypoint** (*ndarray*) – An array with keypoint pairs whose shape is $(K, 2)$, where K is the number of keypoints in the array. The second axis corresponds to y and x coordinates of the keypoint.
- **kp_mask** (*ndarray, optional*) – A boolean array whose shape is $(K,)$. If i th index is `True`, the i th keypoint is not displayed. If not specified, all keypoints in `keypoint` will be displayed.
- **ax** (*matplotlib.axes.Axes, optional*) – If provided, plot on this axis.

Returns Returns the Axes object with the plot for further tweaking.

Return type Axes

3.6.4 vis_semantic_segmentation

`chainercv.visualizations.`**`vis_semantic_segmentation`**(*label, label_names=None, label_colors=None, ignore_label_color=(0, 0, 0), alpha=1, all_label_names_in_legend=False, ax=None*)

Visualize a semantic segmentation.

Example

```
>>> from chainercv.datasets import VOCSemanticSegmentationDataset
>>> from chainercv.datasets         ...     import voc_semantic_segmentation_
↪label_colors
>>> from chainercv.datasets         ...     import voc_semantic_segmentation_
↪label_names
>>> from chainercv.visualizations import vis_image
>>> from chainercv.visualizations import vis_semantic_segmentation
>>> import matplotlib.pyplot as plot
>>> dataset = VOCSemanticSegmentationDataset()
>>> img, label = dataset[60]
>>> ax = vis_image(img)
>>> _, legend_handles = vis_semantic_segmentation(
...     label,
...     label_names=voc_semantic_segmentation_label_names,
...     label_colors=voc_semantic_segmentation_label_colors,
...     alpha=0.9, ax=ax)
>>> ax.legend(handles=legend_handles, bbox_to_anchor=(1, 1), loc=2)
>>> plot.show()
```

Parameters

- **label** (*ndarray*) – An integer array of shape $(height, width)$. The values correspond to id for label names stored in `label_names`.
- **label_names** (*iterable of strings*) – Name of labels ordered according to label ids.
- **label_colors** – (iterable of tuple): An iterable of colors for regular labels. Each color is RGB format and the range of its values is $[0, 255]$. If `colors` is `None`, the default color map used.

- **ignore_label_color** (*tuple*) – Color for ignored label. This is RGB format and the range of its values is $[0, 255]$. The default value is `(0, 0, 0)`.
- **alpha** (*float*) – The value which determines transparency of the figure. The range of this value is $[0, 1]$. If this value is `0`, the figure will be completely transparent. The default value is `1`. This option is useful for overlaying the label on the source image.
- **all_label_names_in_legend** (*bool*) – Determines whether to include all label names in a legend. If this is `False`, the legend does not contain the names of unused labels. An unused label is defined as a label that does not appear in `label`. The default value is `False`.
- **ax** (*matplotlib.axes.Axis*) – The visualization is displayed on this axis. If this is `None` (default), a new axis is created.

Returns Returns `ax` and `legend_handles`. `ax` is an `matploblib.axes.Axes` with the plot. It can be used for further tweaking. `legend_handles` is a list of legends. It can be passed `matploblib.pyplot.legend()` to show a legend.

Return type matploblib.axes.Axes and list of matplotlib.patches.Patch

3.7 Utils

3.7.1 Bounding Box Utilities

bbox_iou

`chainercv.utils.`**`bbox_iou`**(*bbox_a*, *bbox_b*)

Calculate the Intersection of Unions (IoUs) between bounding boxes.

IoU is calculated as a ratio of area of the intersection and area of the union.

This function accepts both `numpy.ndarray` and `cupy.ndarray` as inputs. Please note that both `bbox_a` and `bbox_b` need to be same type. The output is same type as the type of the inputs.

Parameters

- **bbox_a** (*array*) – An array whose shape is $(N, 4)$. N is the number of bounding boxes. The dtype should be `numpy.float32`.
- **bbox_b** (*array*) – An array similar to `bbox_a`, whose shape is $(K, 4)$. The dtype should be `numpy.float32`.

Returns An array whose shape is (N, K). An element at index (n, k) contains IoUs between n th bounding box in `bbox_a` and k th bounding box in `bbox_b`.

Return type array

non_maximum_suppression

`chainercv.utils.`**`non_maximum_suppression`**(*bbox*, *thresh*, *score=None*, *limit=None*)

Suppress bounding boxes according to their IoUs.

This method checks each bounding box sequentially and selects the bounding box if the Intersection over Unions (IoUs) between the bounding box and the previously selected bounding boxes is less than `thresh`. This method is mainly used as postprocessing of object detection. The bounding boxes are selected from ones with higher scores. If `score` is not provided as an argument, the bounding box is ordered by its index in ascending order.

The bounding boxes are expected to be packed into a two dimensional tensor of shape $(R, 4)$, where R is the number of bounding boxes in the image. The second axis represents attributes of the bounding box. They are $(y_{min}, x_{min}, y_{max}, x_{max})$, where the four attributes are coordinates of the top left and the bottom right vertices.

`score` is a float array of shape $(R,)$. Each score indicates confidence of prediction.

This function accepts both `numpy.ndarray` and `cupy.ndarray` as an input. Please note that both `bbox` and `score` need to be the same type. The type of the output is the same as the input.

Parameters

- **`bbox`** (*`array`*) – Bounding boxes to be transformed. The shape is $(R, 4)$. R is the number of bounding boxes.
- **`thresh`** (*`float`*) – Threshold of IoUs.
- **`score`** (*`array`*) – An array of confidences whose shape is $(R,)$.
- **`limit`** (*`int`*) – The upper bound of the number of the output bounding boxes. If it is not specified, this method selects as many bounding boxes as possible.

Returns An array with indices of bounding boxes that are selected. They are sorted by the scores of bounding boxes in descending order. The shape of this array is $(K,)$ and its dtype is `numpy.int32`. Note that $K \leq R$.

Return type array

3.7.2 Download Utilities

cached_download

`chainercv.utils.`**`cached_download`**`(`*url*`)`

Downloads a file and caches it.

This is different from the original `cached_download()` in that the download progress is reported.

It downloads a file from the URL if there is no corresponding cache. After the download, this function stores a cache to the directory under the dataset root (see `set_dataset_root()`). If there is already a cache for the given URL, it just returns the path to the cache without downloading the same file.

Parameters **`url`** (*`str`*) – URL to download from.

Returns Path to the downloaded file.

Return type str

download_model

`chainercv.utils.`**`download_model`**`(`*url*`)`

Downloads a model file and puts it under model directory.

It downloads a file from the URL and puts it under model directory. For exmaple, if `url` is *http://example.com/subdir/model.npz*, the pretrained weights file will be saved to *$CHAINER_DATASET_ROOT/pfnet/chainercv/models/model.npz*. If there is already a file at the destination path, it just returns the path without downloading the same file.

Parameters **`url`** (*`str`*) – URL to download from.

Returns Path to the downloaded file.

Return type str

extractall

`chainercv.utils.`**`extractall`**(*file_path*, *destination*, *ext*)

Extracts an archive file.

This function extracts an archive file to a destination.

Parameters

- **`file_path`** (*`str`*) – The path of a file to be extracted.
- **`destination`** (*`str`*) – A directory path. The archive file will be extracted under this directory.
- **`ext`** (*`str`*) – An extension suffix of the archive file. This function supports `'.zip'`, `'.tar'`, `'.gz'` and `'.tgz'`.

3.7.3 Image Utilities

read_image

`chainercv.utils.`**`read_image`**(*path*, *dtype=<type 'numpy.float32'>*, *color=True*)

Read an image from a file.

This function reads an image from given file. The image is CHW format and the range of its value is $[0, 255]$. If `color = True`, the order of the channels is RGB.

Parameters

- **`path`** (*`str`*) – A path of image file.
- **`dtype`** – The type of array. The default value is `float32`.
- **`color`** (*`bool`*) – This option determines the number of channels. If `True`, the number of channels is three. In this case, the order of the channels is RGB. This is the default behaviour. If `False`, this function returns a grayscale image.

Returns An image.

Return type ndarray

tile_images

`chainercv.utils.`**`tile_images`**(*imgs*, *n_col*, *pad=2*, *fill=0*)

Make a tile of images

Parameters

- **`imgs`** (*`numpy.ndarray`*) – A batch of images whose shape is BCHW.
- **`n_col`** (*`int`*) – The number of columns in a tile.
- **`pad`** (*`int`*) – Amount of pad. The default value is 2.
- **`fill`** (*`float, tuple or ndarray`*) – The value of padded pixels. If it is `numpy.ndarray`, its shape should be $(C, 1, 1)$, where C is the number of channels of `img`.

Returns An image array in CHW format. The size of this image is $((H + pad) \times \lceil B / n_{n_{col}} \rceil, (W + pad) \times n_{col})$.

Return type ndarray

write_image

`chainercv.utils.`**`write_image`**(*img*, *path*)

Save an image to a file.

This function saves an image to given file. The image is in CHW format and the range of its value is $[0, 255]$.

Parameters

- **`image`** (*`ndarray`*) – An image to be saved.
- **`path`** (*`str`*) – The path of an image file.

3.7.4 Iterator Utilities

apply_prediction_to_iterator

`chainercv.utils.`**`apply_prediction_to_iterator`**(*predict*, *iterator*, *hook=None*)

Apply a prediction function/method to an iterator.

This function applies a prediction function/method to an iterator. It assumes that the iterator returns a batch of images or a batch of tuples whose first element is an image. In the case that it returns a batch of tuples, the rests are treated as ground truth values.

```
>>> imgs = next(iterator)
>>> # imgs: [img]
or
>>> batch = next(iterator)
>>> # batch: [(img, gt_val0, gt_val1)]
```

This function applys `predict()` to a batch of images and gets predicted value(s). `predict()` should take a batch of images and return a batch of prediction values or a tuple of batches of prediction values.

```
>>> pred_vals0 = predict(imgs)
>>> # pred_vals0: [pred_val0]
or
>>> pred_vals0, pred_vals1 = predict(imgs)
>>> # pred_vals0: [pred_val0]
>>> # pred_vals1: [pred_val1]
```

Here is an exmple, which applies a pretrained Faster R-CNN to PASCAL VOC dataset.

```
>>> from chainer import iterators
>>>
>>> from chainercv.datasets import VOCDetectionDataset
>>> from chainercv.links import FasterRCNNVGG16
>>> from chainercv.utils import apply_prediction_to_iterator
>>>
>>> dataset = VOCDetectionDataset(year='2007', split='test')
>>> # next(iterator) -> [(img, gt_bbox, gt_label)]
>>> iterator = iterators.SerialIterator(
...     dataset, 2, repeat=False, shuffle=False)
>>>
>>> # model.predict([img]) -> ([pred_bbox], [pred_label], [pred_score])
>>> model = FasterRCNNVGG16(pretrained_model='voc07')
>>>
>>> imgs, pred_values, gt_values = apply_prediction_to_iterator(
...     model.predict, iterator)
```

```
>>>
>>> # pred_values contains three iterators
>>> pred_bboxes, pred_labels, pred_scores = pred_values
>>> # gt_values contains two iterators
>>> gt_bboxes, gt_labels = gt_values
```

Parameters

- **`predict`** – A callable that takes a batch of images and returns prediction.
- **`iterator`** (*`chainer.Iterator`*) – An iterator. Each sample should have an image as its first element. This image is passed to `predict()` as an argument. The rests are treated as ground truth values.
- **`hook`** – A callable that is called after each iteration. `imgs`, `pred_values` and `gt_values` are passed as arguments. Note that these values do not contain data from the previous iterations.

Returns

This function returns an iterator and two tuples of iterators: `imgs`, `pred_values` and `gt_values`.

- `imgs`: An iterator that returns an image.
- `pred_values`: A tuple of iterators. Each iterator returns a corresponding predicted value. For example, if `predict()` returns `([pred_val0], [pred_val1])`, `next(pred_values[0])` and `next(pred_values[1])` will be `pred_val0` and `pred_val1`.
- `gt_values`: A tuple of iterators. Each iterator returns a corresponding ground truth value. For example, if the `iterator` returns `[(img, gt_val0, gt_val1)]`, `next(gt_values[0])` and `next(gt_values[1])` will be `gt_val0` and `gt_val1`. If the input iterator does not give any ground truth values, this tuple will be empty.

Return type An iterator and two tuples of iterators

ProgressHook

class `chainercv.utils.`**`ProgressHook`**`(`*n_total=None*`)`

A hook class reporting the progress of iteration.

This is a hook class designed for *`apply_prediction_to_iterator()`*.

Parameters **`n_total`** (*`int`*) – The number of images. This argument is optional.

unzip

`chainercv.utils.`**`unzip`**`(`*iterable*`)`

Converts an iterable of tuples into a tuple of iterators.

This function converts an iterable of tuples into a tuple of iterators. This is an inverse function of `six.moves.zip()`.

```
>>> from chainercv.utils import unzip
>>> data = [(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
>>> int_iter, str_iter = unzip(data)
```

```
>>>
>>> next(int_iter)   # 0
>>> next(int_iter)   # 1
>>> next(int_iter)   # 2
>>>
>>> next(str_iter)   # 'a'
>>> next(str_iter)   # 'b'
>>> next(str_iter)   # 'c'
```

Parameters **`iterable`** (*`iterable`*) – An iterable of tuples. All tuples should have the same length.

Returns Each iterator corresponds to each element of input tuple. Note that each iterator stores values until they are popped. To reduce memory usage, it is recommended to delete unused iterators.

Return type tuple of iterators

3.7.5 Testing Utilities

assert_is_bbox

`chainercv.utils.`**`assert_is_bbox`**(*bbox*, *size=None*)

Checks if bounding boxes satisfy bounding box format.

This function checks if given bounding boxes satisfy bounding boxes format or not. If the bounding boxes do not satifiy the format, this function raises an `AssertionError`.

Parameters

- **`bbox`** (*`ndarray`*) – Bounding boxes to be checked.
- **`size`** (*`tuple of ints`*) – The size of an image. If this argument is specified, Each bounding box should be within the image.

assert_is_bbox_dataset

`chainercv.utils.`**`assert_is_bbox_dataset`**(*dataset*, *n_fg_class*, *n_example=None*)

Checks if a dataset satisfies bounding box dataset APIs.

This function checks if a given dataset satisfies bounding box dataset APIs or not. If the dataset does not satifiy the APIs, this function raises an `AssertionError`.

Parameters

- **`dataset`** – A dataset to be checked.
- **`n_fg_class`** (*`int`*) – The number of foreground classes.
- **`n_example`** (*`int`*) – The number of examples to be checked. If this argument is specified, this function picks examples ramdomly and checks them. Otherwise, this function checks all examples.

assert_is_detection_link

`chainercv.utils.`**`assert_is_detection_link`**(*link*, *n_fg_class*)

Checks if a link satisfies detection link APIs.

This function checks if a given link satisfies detection link APIs or not. If the link does not satifiy the APIs, this function raises an `AssertionError`.

Parameters

- **`link`** – A link to be checked.
- **`n_fg_class`** (*`int`*) – The number of foreground classes.

assert_is_image

`chainercv.utils.`**`assert_is_image`**(*img*, *color=True*, *check_range=True*)

Checks if an image satisfies image format.

This function checks if a given image satisfies image format or not. If the image does not satifiy the format, this function raises an `AssertionError`.

Parameters

- **`img`** (*`ndarray`*) – An image to be checked.
- **`color`** (*`bool`*) – A boolean that determines the expected channel size. If it is `True`, the number of channels should be `3`. Otherwise, it should be `1`. The default value is `True`.
- **`check_range`** (*`bool`*) – A boolean that determines whether the range of values are checked or not. If it is `True`, The values of image must be in $[0, 255]$. Otherwise, this function does not check the range. The default value is `True`.

assert_is_label_dataset

`chainercv.utils.`**`assert_is_label_dataset`**(*dataset*, *n_class*, *n_example=None*, *color=True*)

Checks if a dataset satisfies label dataset APIs.

This function checks if a given dataset satisfies label dataset APIs or not. If the dataset does not satifiy the APIs, this function raises an `AssertionError`.

Parameters

- **`dataset`** – A dataset to be checked.
- **`n_class`** (*`int`*) – The number of classes.
- **`n_example`** (*`int`*) – The number of examples to be checked. If this argument is specified, this function picks examples ramdomly and checks them. Otherwise, this function checks all examples.
- **`color`** (*`bool`*) – A boolean that determines the expected channel size. If it is `True`, the number of channels should be `3`. Otherwise, it should be `1`. The default value is `True`.

assert_is_semantic_segmentation_dataset

`chainercv.utils.`**`assert_is_semantic_segmentation_dataset`**(*dataset*, *n_class*, *n_example=None*)

Checks if a dataset satisfies semantic segmentation dataset APIs.

This function checks if a given dataset satisfies semantic segmentation dataset APIs or not. If the dataset does not satifiy the APIs, this function raises an `AssertionError`.

Parameters

- **`dataset`** – A dataset to be checked.
- **`n_class`** (*`int`*) – The number of classes including background.
- **`n_example`** (*`int`*) – The number of examples to be checked. If this argument is specified, this function picks examples ramdomly and checks them. Otherwise, this function checks all examples.

assert_is_semantic_segmentation_link

`chainercv.utils.`**`assert_is_semantic_segmentation_link`**(*link*, *n_class*)

Checks if a link satisfies semantic segmentation link APIs.

This function checks if a given link satisfies semantic segmentation link APIs or not. If the link does not satifiy the APIs, this function raises an `AssertionError`.

Parameters

- **`link`** – A link to be checked.
- **`n_class`** (*`int`*) – The number of classes including background.

ConstantStubLink

`class` `chainercv.utils.`**`ConstantStubLink`**(*outputs*)

A chainer.Link that returns constant value(s).

This is a `chainer.Link` that returns constant `chainer.Variable` (s) when `__call__()` method is called.

Parameters **`outputs`** (*`ndarray or tuple or ndarray`*) – The value(s) of variable(s) returned by `__call__()`. If an array is specified, `__call__()` returns a `chainer.Variable`. Otherwise, it returns a tuple of `chainer.Variable`.

generate_random_bbox

`chainercv.utils.`**`generate_random_bbox`**(*n*, *img_size*, *min_length*, *max_length*)

Generate valid bounding boxes with random position and shape.

Parameters

- **`n`** (*`int`*) – The number of bounding boxes.
- **`img_size`** (*`tuple`*) – A tuple of length 2. The height and the width of the image on which bounding boxes locate.
- **`min_length`** (*`float`*) – The minimum length of edges of bounding boxes.
- **`max_length`** (*`float`*) – The maximum length of edges of bounding boxes.

Returns Coordinates of bounding boxes. Its shape is $(R, 4)$. Here, R equals `n`. The second axis contains $y_{min}, x_{min}, y_{max}, x_{max}$, where $min\_length \leq y_{max} - y_{min} < max\_length$. and $min\_length \leq x_{max} - x_{min} < max\_length$

Return type numpy.ndarray

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Bibliography

[Ren15] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS 2015.

[Liu16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.

Python Module Index

c

chainercv, 13
chainercv.datasets, 13
chainercv.evaluations, 19
chainercv.extensions, 23
chainercv.links, 25
chainercv.links.connection, 52
chainercv.links.model.faster_rcnn, 29
chainercv.links.model.segnet, 50
chainercv.links.model.ssd, 40
chainercv.links.model.vgg, 27
chainercv.transforms, 54
chainercv.utils, 65
chainercv.visualizations, 62

Index

Symbols

__call__() (chainercv.links.PixelwiseSoftmaxClassifier method), 51
__call__() (chainercv.links.model.faster_rcnn.AnchorTargetCreator method), 37
__call__() (chainercv.links.model.faster_rcnn.FasterRCNN method), 31
__call__() (chainercv.links.model.faster_rcnn.FasterRCNNTrainChain method), 38
__call__() (chainercv.links.model.faster_rcnn.ProposalCreator method), 34
__call__() (chainercv.links.model.faster_rcnn.ProposalTargetCreator method), 39
__call__() (chainercv.links.model.faster_rcnn.RegionProposalNetwork method), 35
__call__() (chainercv.links.model.segnet.SegNetBasic method), 50
__call__() (chainercv.links.model.ssd.Multibox method), 41
__call__() (chainercv.links.model.ssd.Normalize method), 44
__call__() (chainercv.links.model.ssd.SSD method), 45
__call__() (chainercv.links.model.ssd.VGG16Extractor300 method), 46
__call__() (chainercv.links.model.ssd.VGG16Extractor512 method), 46

A

ADE20KSemanticSegmentationDataset (class in chainercv.datasets), 14
ADE20KTestImageDataset (class in chainercv.datasets), 15
AnchorTargetCreator (class in chainercv.links.model.faster_rcnn), 37
apply_prediction_to_iterator() (in module chainercv.utils), 68
assert_is_bbox() (in module chainercv.utils), 70
assert_is_bbox_dataset() (in module chainercv.utils), 70
assert_is_detection_link() (in module chainercv.utils), 71
assert_is_image() (in module chainercv.utils), 71
assert_is_label_dataset() (in module chainercv.utils), 71
assert_is_semantic_segmentation_dataset() (in module chainercv.utils), 71
assert_is_semantic_segmentation_link() (in module chainercv.utils), 72

B

bbox2loc() (in module chainercv.links.model.faster_rcnn), 30
bbox_iou() (in module chainercv.utils), 65

C

cached_download() (in module chainercv.utils), 66
calc_detection_voc_ap() (in module chainercv.evaluations), 20
calc_detection_voc_prec_rec() (in module chainercv.evaluations), 20
calc_semantic_segmentation_confusion() (in module chainercv.evaluations), 22
calc_semantic_segmentation_iou() (in module chainercv.evaluations), 23
CamVidDataset (class in chainercv.datasets), 15
center_crop() (in module chainercv.transforms), 54
chainercv (module), 13
chainercv.datasets (module), 13
chainercv.evaluations (module), 19
chainercv.extensions (module), 23
chainercv.links (module), 25, 51
chainercv.links.connection (module), 52
chainercv.links.model.faster_rcnn (module), 29
chainercv.links.model.segnet (module), 50
chainercv.links.model.ssd (module), 40
chainercv.links.model.vgg (module), 27
chainercv.transforms (module), 54
chainercv.utils (module), 65
chainercv.visualizations (module), 62
CityscapesSemanticSegmentationDataset (class in chainercv.datasets), 15

CityscapesTestImageDataset (class in chainercv.datasets), 16
ConstantStubLink (class in chainercv.utils), 72
Conv2DActiv (class in chainercv.links.connection), 52
Conv2DBNActiv (class in chainercv.links.connection), 53
crop_bbox() (in module chainercv.transforms), 59
CUBKeypointDataset (class in chainercv.datasets), 17
CUBLabelDataset (class in chainercv.datasets), 16

D

decode() (chainercv.links.model.ssd.MultiboxCoder method), 42
DetectionVisReport (class in chainercv.extensions), 24
DetectionVOCEvaluator (class in chainercv.extensions), 23
directory_parsing_label_names() (in module chainercv.datasets), 14
DirectoryParsingLabelDataset (class in chainercv.datasets), 13
download_model() (in module chainercv.utils), 66

E

encode() (chainercv.links.model.ssd.MultiboxCoder method), 43
eval_detection_voc() (in module chainercv.evaluations), 19
eval_semantic_segmentation() (in module chainercv.evaluations), 21
extractall() (in module chainercv.utils), 67

F

FasterRCNN (class in chainercv.links.model.faster_rcnn), 30
FasterRCNNTrainChain (class in chainercv.links.model.faster_rcnn), 38
FasterRCNNVGG16 (class in chainercv.links.model.faster_rcnn), 29
FeaturePredictor (class in chainercv.links), 25
flip() (in module chainercv.transforms), 54
flip_bbox() (in module chainercv.transforms), 60
flip_keypoint() (in module chainercv.transforms), 61

G

generate_anchor_base() (in module chainercv.links.model.faster_rcnn), 33
generate_random_bbox() (in module chainercv.utils), 72
GradientScaling (class in chainercv.links.model.ssd), 47

L

loc2bbox() (in module chainercv.links.model.faster_rcnn), 33

M

Multibox (class in chainercv.links.model.ssd), 41
multibox_loss() (in module chainercv.links.model.ssd), 47
MultiboxCoder (class in chainercv.links.model.ssd), 42

N

non_maximum_suppression() (in module chainercv.utils), 65
Normalize (class in chainercv.links.model.ssd), 43

O

OnlineProductsDataset (class in chainercv.datasets), 17

P

pca_lighting() (in module chainercv.transforms), 54
PickableSequentialChain (class in chainercv.links), 26
PixelwiseSoftmaxClassifier (class in chainercv.links), 51
predict() (chainercv.links.FeaturePredictor method), 26
predict() (chainercv.links.model.faster_rcnn.FasterRCNN method), 32
predict() (chainercv.links.model.segnet.SegNetBasic method), 51
predict() (chainercv.links.model.ssd.SSD method), 45
prepare() (chainercv.links.model.faster_rcnn.FasterRCNN method), 32
ProgressHook (class in chainercv.utils), 69
ProposalCreator (class in chainercv.links.model.faster_rcnn), 34
ProposalTargetCreator (class in chainercv.links.model.faster_rcnn), 39

R

random_crop() (in module chainercv.transforms), 55
random_crop_with_bbox_constraints() (in module chainercv.links.model.ssd), 48
random_distort() (in module chainercv.links.model.ssd), 49
random_expand() (in module chainercv.transforms), 55
random_flip() (in module chainercv.transforms), 56
random_rotate() (in module chainercv.transforms), 57
read_image() (in module chainercv.utils), 67
RegionProposalNetwork (class in chainercv.links.model.faster_rcnn), 35
remove_unused() (chainercv.links.PickableSequentialChain method), 27
resize() (in module chainercv.transforms), 57
resize_bbox() (in module chainercv.transforms), 60
resize_contain() (in module chainercv.transforms), 57
resize_keypoint() (in module chainercv.transforms), 61
resize_with_random_interpolation() (in module chainercv.links.model.ssd), 49

S

scale() (in module chainercv.transforms), 58

SegNetBasic (class in chainercv.links.model.segnet), 50
SemanticSegmentationEvaluator (class in chainercv.extensions), 24
SSD (class in chainercv.links.model.ssd), 44
SSD300 (class in chainercv.links.model.ssd), 40
SSD512 (class in chainercv.links.model.ssd), 40

T

ten_crop() (in module chainercv.transforms), 59
tile_images() (in module chainercv.utils), 67
translate_bbox() (in module chainercv.transforms), 61
translate_keypoint() (in module chainercv.transforms), 62

U

unzip() (in module chainercv.utils), 69
use_preset() (chainercv.links.model.faster_rcnn.FasterRCNN method), 32
use_preset() (chainercv.links.model.ssd.SSD method), 45

V

VGG16 (class in chainercv.links.model.ssd), 46
VGG16 (class in chainercv.links.model.vgg), 27
VGG16Extractor300 (class in chainercv.links.model.ssd), 46
VGG16Extractor512 (class in chainercv.links.model.ssd), 46
VGG16RoIHead (class in chainercv.links.model.faster_rcnn), 36
vis_bbox() (in module chainercv.visualizations), 62
vis_image() (in module chainercv.visualizations), 63
vis_keypoint() (in module chainercv.visualizations), 63
vis_semantic_segmentation() (in module chainercv.visualizations), 64
VOCBboxDataset (class in chainercv.datasets), 18
VOCSemanticSegmentationDataset (class in chainercv.datasets), 19

W

write_image() (in module chainercv.utils), 68