ChainerCV Documentation

Release 0.2.1

Preferred Networks, inc.

Jun 26, 2017

Contents

1 Install Guide **3**
- 1.1 Pip . . . 3
- 1.2 Anaconda . . . 3

2 Reference Manual **5**
- 2.1 ChainerCV Reference Manual . . . 5
  - 2.1.1 Datasets . . . 5
  - 2.1.2 Evaluations . . . 9
  - 2.1.3 Extensions . . . 13
  - 2.1.4 Links . . . 14
  - 2.1.5 Transforms . . . 32
  - 2.1.6 Visualizations . . . 40
  - 2.1.7 Utils . . . 43

3 Indices and tables **49**

Python Module Index **51**

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

CHAPTER 1

Install Guide

Pip

You can install ChainerCV using *pip*.

```
# If Cython has not been installed yet, install it by a command like
# pip install Cython
pip install chainercv
```

Anaconda

You can setup ChainerCV including its dependencies using anaconda.

```
# 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

Reference Manual

ChainerCV Reference Manual

Datasets

TransformDataset

TransformDataset

chainercv.datasets.**TransformDataset**(*dataset*, *transform*)

Dataset that indexes data of a base dataset and transforms it.

This dataset wraps a base dataset by modifying the behavior of the base dataset's `__getitem__()`. Arrays returned by `__getitem__()` of the base dataset with an integer index are transformed by the given function `transform`.

The function `transform` takes, as an argument, `in_data`, which is output of the base dataset's `__getitem__()`, and returns transformed arrays as output. Please see the following example.

```
>>> from chainer.datasets import get_mnist
>>> from chainercv.datasets import TransformDataset
>>> dataset, _ = get_mnist()
>>> def transform(in_data):
>>>     img, label = in_data
>>>     img -= 0.5  # scale to [-0.5, -0.5]
>>>     return img, label
>>> dataset = TransformDataset(dataset, transform)
```

Note: The index used to access data is either an integer or a slice. If it is a slice, the base dataset is assumed to return a list of outputs each corresponding to the output of the integer indexing.

Parameters

- **dataset** – Underlying dataset. The index of this dataset corresponds to the index of the base dataset.
- **transform** (*callable*) – A function that is called to transform values returned by the underlying dataset's `__getitem__()`.

CamVid

CamVidDataset

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

Dataset class for a semantic segmantion task on CamVid u.

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.

CUB

CUBLabelDataset

`chainercv.datasets.`**`CUBLabelDataset`**(*data_dir='auto'*, *crop_bbox=True*)

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 are between 0 and 199.

There are 200 labels of birds in total.

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`.
- **crop_bbox** (*bool*) – If true, this class returns an image cropped by the bounding box of the bird inside it.

CUBKeypointDataset

`chainercv.datasets.`**`CUBKeypointDataset`**(*data_dir='auto'*, *crop_bbox=True*, *mask_dir='auto'*, *return_mask=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_mask = True`, `mask` will be returned as well,

making the returned tuple to be of length four. `mask` is a `uint8` image which indicates the region of the image where a bird locates.

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 mask image of the bird shows how likely the bird is located at a given pixel. If the value is close to 255, more likely that a bird locates at that pixel. The shape of this array is $(1, H, W)$, where H and W are height and width of the image respectively.

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 `$CHAINER_DATASET_ROOT/pfnet/chainercv/cub`.
- **crop_bbox** (*bool*) – If true, this class returns an image cropped by the bounding box of the bird inside it.
- **mask_dir** (*string*) – Path to the root of the mask data. If this is `auto`, this class will automatically download data for you under `$CHAINER_DATASET_ROOT/pfnet/chainercv/cub`.
- **return_mask** (*bool*) – Decide whether to include mask image of the bird in a tuple served for a query.

OnlineProducts

OnlineProductsDataset

`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 `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.

PASCAL VOC

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

VOCDetectionDataset

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

Dataset class for the detection task of 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 bottom left and the top 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_detection_label_names`.

The array `difficult` is a one dimensional boolean array of shape $(R,)$. R is the number of bounding boxes in the image.

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`.
- **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

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

Dataset class for the semantic segmantion task of 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.

Evaluations

Detection VOC

eval_detection_voc_ap

chainercv.evaluations.**eval_detection_voc_ap**(*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 This function returns 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`.

Return type ndarray

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 does not exist in either `pred_labels` or `gt_labels`, `rec[l]` is set to `None`.

Return type tuple of two lists

PCK

eval_pck

chainercv.evaluations.**eval_pck**(*pred*, *expected*, *alpha*, *L*)

Calculate PCK (Percentage of Correct Keypoints).

This function calculates number of vertices whose positions are correctly pred. A pred keypoint is correctly matched to the ground truth if it lies within Euclidean distance $\alpha \cdot L$ of the ground truth keypoint, where L is the size of the image and $0 < \alpha < 1$ is a variable we control. L is determined differently depending on the context. For example, in evaluation of keypoint matching for CUB dataset, $L = \sqrt{h^2 + w^2}$ is used.

Parameters

- **pred** (*ndarray*) – An array of shape $(K, 2)$ N is the number of keypoints to be evaluated. The two elements of the second axis corresponds to y and x coordinate of the keypoint.
- **expected** (*ndarray*) – Same kind of array as `pred`. This contains ground truth location of the keypoints that the user tries to predict.
- **alpha** (*float*) – A control variable α.
- **L** (*float*) – A size of an image. The definition changes from the tasks to task.

Returns float

Semantic Segmentation IoU

eval_semantic_segmentation_iou

chainercv.evaluations.**eval_semantic_segmentation_iou**(*pred_labels*, *gt_labels*)

Evaluate Intersection over Union from labels.

This function calculates Intersection over Union (IoU) for the task of semantic segmentation.

The definition of IoU and a related metric, mean Intersection over Union (mIoU), are as follow, 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 $= \frac{1}{k} \sum_{i=1}^{k} \frac{N_{ii}}{\sum_{j=1}^{k} N_{ij} + \sum_{j=1}^{k} N_{ji} - N_{ii}}$

mIoU can be computed by taking `numpy.nanmean` of the IoUs returned by this function. The more detailed description of the above metric 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 An array of IoUs for the n_class classes. Its shape is $(n_class,)$.

Return type numpy.ndarray

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.

Return type numpy.ndarray

[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.

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

Extensions

Detection

DetectionVisReport

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'`.

DetectionVOCEvaluator

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_detection_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 which 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]>'`.

Links

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 *chainercv.links.model.faster_rcnn.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 *chainercv.links.model.faster_rcnn.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 used to compute p_y, p_x, p_h, p_w.
- **dst_bbox** (*array*) – An image coordinate array whose shape is $(R, 4)$. These coordinates are used to compute g_y, g_x, g_h, g_w.

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 *FasterRCNN.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 `chainercv.links.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 used to compute p_y, p_x, p_h, p_w.

[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 $\hat{g}_y, \hat{g}_x, \hat{g}_h, \hat{g}_w$.

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 *chainercv.links.model.faster_rcnn.ProposalCreator*.

See also:

chainercv.links.model.faster_rcnn.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)

VGG16FeatureExtractor

class chainercv.links.model.faster_rcnn.**VGG16FeatureExtractor**(*initialW=None*)
Truncated VGG-16 that extracts a conv5_3 feature map.

Parameters **initialW** (*callable*) – Initializer for the weights.

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 *chainercv.links.model.faster_rcnn.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.

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

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)

FasterRCNNTrainChain

class chainercv.links.model.faster_rcnn.**FasterRCNNTrainChain**(*faster_rcnn*, *rpn_sigma=3.0*, *roi_sigma=1.0*, *anchor_target_creator=<chainercv.links.model.fa object>*, *proposal_target_creator=<chainercv.links.model.f 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 *chainercv.links.model.faster_rcnn.AnchorTargetCreator*.
- **proposal_target_creator_params** – An instantiation of *chainercv.links.model.faster_rcnn.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.

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

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.

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.

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

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.
- **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*.
  - *filepath*: A path of npz file. In this case, `n_fg_class` must be specified properly.
  - `None`: Do not load weights.

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

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.

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*.
  - *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 loc and conf from feature maps. loc contains information of the coordinates of bounding boxes and conf contains that of 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.GlorotUniform.
- **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 loc and conf 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.

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

Returns This method returns two `chainer.Variable`, `loc` and `conf`. `loc` is an array whose shape is $(B, K, 4)$, where B is the number of samples in the batch and K is the number of default bounding boxes. `conf` is an array whose shape is (B, K, n_class)

Return type tuple of chainer.Variable

Normalize

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

Learnable L2 normalization[4].

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

Parameters

- **`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[5].

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.
  - `__call__()`: A method which computes feature maps. It must take a batched images and return batched feature maps.
- **`multibox`** – A link which computes loc and conf 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.

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

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

- `aspect_ratios`: An iterable of tuple of integer. Each tuple indicates the aspect ratios of default bounding boxes at each feature maps.
- *`__call__()`*: A method which computes `loc` and `conf`. It must take a batched feature maps and return `loc` and `conf`.

- **steps** (*iterable of float*) – The step size for each feature map.
- **sizes** (*iterable of float*) – The base size of default bounding boxes for each feature map.
- **variance** (*tuple of float*) – Two coefficients for encoding the locations of bounding boxe. The first value is used to encode coordinates of the centers. The second value is used to encode the sizes of bounding boxes. The default value is `(0.1, 0.2)`.
- **nms_thresh** (*float*) – The threshold value for `chainercv.transfroms.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, `loc` and `conf`. `_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, `loc` and `conf`.

- **loc**: 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.
- **conf**: 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[6]. The differences from original VGG-16[7] 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.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV 2016.
[7] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR 2015.

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

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

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

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

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

Transforms

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 $[0, 255]$).

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

- **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.

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.

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

- **fill** (*float, tuple or ndarray*) – The value of padded pixels. In the original paper, this value is the mean of ImageNet.
- **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.
- **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.
- **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*)

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`.

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 $(10, C, H, W)$.

Bounding Box

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 bottom left and the top 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 bottom left and the top 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 bottom left and the top 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

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

Visualizations

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_detection_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_detection_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

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

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

vis_label

chainercv.visualizations.**vis_label**(*label*, *label_names=None*, *label_colors=None*, *ignore_label_color=(0, 0, 0)*, *alpha=1*, *ax=None*)

Visualize a label for 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_label
>>> import matplotlib.pyplot as plot
>>> dataset = VOCSemanticSegmentationDataset()
>>> img, label = dataset[60]
>>> ax = vis_image(img)
>>> _, legend_handles = vis_label(
...     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.
- **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

Utils

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 inputs. Please note that both `bbox` and `score` need to be same type. The output is same type as the type of the inputs.

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

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'`.

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

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

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

Testing Utilities

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 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

chainercv, 5
chainercv.datasets, 5
chainercv.evaluations, 9
chainercv.extensions, 13
chainercv.links, 32
chainercv.links.model.faster_rcnn, 15
chainercv.links.model.segnet, 31
chainercv.links.model.ssd, 26
chainercv.transforms, 32
chainercv.utils, 43
chainercv.visualizations, 40

Index

Symbols

__call__() (chainercv.links.PixelwiseSoftmaxClassifier method), 32
__call__() (chainercv.links.model.faster_rcnn.AnchorTargetCreator method), 23
__call__() (chainercv.links.model.faster_rcnn.FasterRCNN method), 17
__call__() (chainercv.links.model.faster_rcnn.FasterRCNNTrainChain method), 24
__call__() (chainercv.links.model.faster_rcnn.ProposalCreator method), 20
__call__() (chainercv.links.model.faster_rcnn.ProposalTargetCreator method), 25
__call__() (chainercv.links.model.faster_rcnn.RegionProposalNetwork method), 21
__call__() (chainercv.links.model.segnet.SegNetBasic method), 31
__call__() (chainercv.links.model.ssd.Multibox method), 27
__call__() (chainercv.links.model.ssd.Normalize method), 28
__call__() (chainercv.links.model.ssd.SSD method), 29
__call__() (chainercv.links.model.ssd.VGG16Extractor300 method), 30
__call__() (chainercv.links.model.ssd.VGG16Extractor512 method), 31

A

AnchorTargetCreator (class in chainercv.links.model.faster_rcnn), 23
apply_prediction_to_iterator() (in module chainercv.utils), 45

B

bbox2loc() (in module chainercv.links.model.faster_rcnn), 16
bbox_iou() (in module chainercv.utils), 43

C

cached_download() (in module chainercv.utils), 44
calc_detection_voc_ap() (in module chainercv.evaluations), 10
calc_detection_voc_prec_rec() (in module chainercv.evaluations), 10
calc_semantic_segmentation_confusion() (in module chainercv.evaluations), 12
calc_semantic_segmentation_iou() (in module chainercv.evaluations), 13
CamVidDataset() (in module chainercv.datasets), 6
center_crop() (in module chainercv.transforms), 32
chainercv (module), 5
chainercv.datasets (module), 5
chainercv.evaluations (module), 9
chainercv.extensions (module), 13
chainercv.links (module), 32
chainercv.links.model.faster_rcnn (module), 14, 15
chainercv.links.model.segnet (module), 31
chainercv.links.model.ssd (module), 26
chainercv.transforms (module), 32
chainercv.utils (module), 43
chainercv.visualizations (module), 40
ConstantStubLink (class in chainercv.utils), 47
CUBKeypointDataset() (in module chainercv.datasets), 6
CUBLabelDataset() (in module chainercv.datasets), 6

D

DetectionVisReport() (in module chainercv.extensions), 13
DetectionVOCEvaluator() (in module chainercv.extensions), 14
download_model() (in module chainercv.utils), 44

E

eval_detection_voc_ap() (in module chainercv.evaluations), 9
eval_pck() (in module chainercv.evaluations), 11
eval_semantic_segmentation_iou() (in module chainercv.evaluations), 12
extractall() (in module chainercv.utils), 45

F

FasterRCNN (class in chainercv.links.model.faster_rcnn), 16
FasterRCNNTrainChain (class in chainercv.links.model.faster_rcnn), 24
FasterRCNNVGG16 (class in chainercv.links.model.faster_rcnn), 15
flip() (in module chainercv.transforms), 33
flip_bbox() (in module chainercv.transforms), 38
flip_keypoint() (in module chainercv.transforms), 39

G

generate_anchor_base() (in module chainercv.links.model.faster_rcnn), 19
generate_random_bbox() (in module chainercv.utils), 47

L

loc2bbox() (in module chainercv.links.model.faster_rcnn), 19

M

Multibox (class in chainercv.links.model.ssd), 27

N

non_maximum_suppression() (in module chainercv.utils), 43
Normalize (class in chainercv.links.model.ssd), 28

O

OnlineProductsDataset() (in module chainercv.datasets), 7

P

pca_lighting() (in module chainercv.transforms), 33
PixelwiseSoftmaxClassifier (class in chainercv.links), 32
predict() (chainercv.links.model.faster_rcnn.FasterRCNN method), 18
predict() (chainercv.links.model.segnet.SegNetBasic method), 32
predict() (chainercv.links.model.ssd.SSD method), 29
prepare() (chainercv.links.model.faster_rcnn.FasterRCNN method), 18
ProposalCreator (class in chainercv.links.model.faster_rcnn), 20
ProposalTargetCreator (class in chainercv.links.model.faster_rcnn), 25

R

random_crop() (in module chainercv.transforms), 34
random_expand() (in module chainercv.transforms), 34
random_flip() (in module chainercv.transforms), 35
random_rotate() (in module chainercv.transforms), 35
read_image() (in module chainercv.utils), 45
RegionProposalNetwork (class in chainercv.links.model.faster_rcnn), 21
resize() (in module chainercv.transforms), 36
resize_bbox() (in module chainercv.transforms), 38
resize_contain() (in module chainercv.transforms), 36
resize_keypoint() (in module chainercv.transforms), 39

S

scale() (in module chainercv.transforms), 37
SegNetBasic (class in chainercv.links.model.segnet), 31
SSD (class in chainercv.links.model.ssd), 28
SSD300 (class in chainercv.links.model.ssd), 26
SSD512 (class in chainercv.links.model.ssd), 27

T

ten_crop() (in module chainercv.transforms), 37
TransformDataset() (in module chainercv.datasets), 5
translate_bbox() (in module chainercv.transforms), 39
translate_keypoint() (in module chainercv.transforms), 40

U

unzip() (in module chainercv.utils), 47
use_preset() (chainercv.links.model.faster_rcnn.FasterRCNN method), 18
use_preset() (chainercv.links.model.ssd.SSD method), 30

V

VGG16 (class in chainercv.links.model.ssd), 30
VGG16Extractor300 (class in chainercv.links.model.ssd), 30
VGG16Extractor512 (class in chainercv.links.model.ssd), 30
VGG16FeatureExtractor (class in chainercv.links.model.faster_rcnn), 22
VGG16RoIHead (class in chainercv.links.model.faster_rcnn), 22
vis_bbox() (in module chainercv.visualizations), 40
vis_image() (in module chainercv.visualizations), 41
vis_keypoint() (in module chainercv.visualizations), 41
vis_label() (in module chainercv.visualizations), 42
VOCDetectionDataset() (in module chainercv.datasets), 8
VOCSemanticSegmentationDataset() (in module chainercv.datasets), 8