ChainerCV Documentation

Release 0.13.1

Preferred Networks, inc.

Jan 07, 2020

Contents

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

CHAPTER 1

Installation Guide

1.1 Pip

You can install ChainerCV using *pip*.

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

1.2 Anaconda

Build instruction using Anaconda is as follows.

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

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

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

# Install ChainerCV
pip install -e .
```

(continues on next page)

(continued from previous page)

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

CHAPTER 2

ChainerCV Tutorial

2.1 Object Detection Tutorial

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

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

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

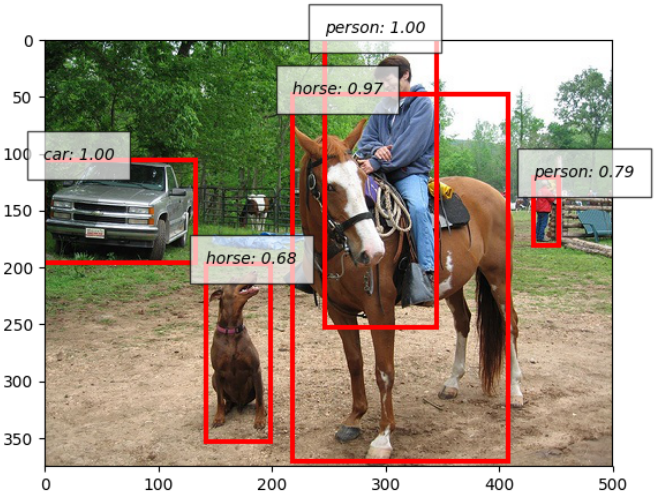
Here is a short example that conducts inference and visualizes output. Please download an image from a link below, and name it as `sample.jpg`. https://cloud.githubusercontent.com/assets/2062128/26187667/9cb236da-3bd5-11e7-8bcf-7dbd4302e2dc.jpg

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

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

# Read an RGB image and return it in CHW format.
img = read_image('sample.jpg')
model = SSD300(pretrained_model='voc0712')
```

(continues on next page)

(continued from previous page)

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



2.1.1 Bounding boxes in ChainerCV

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

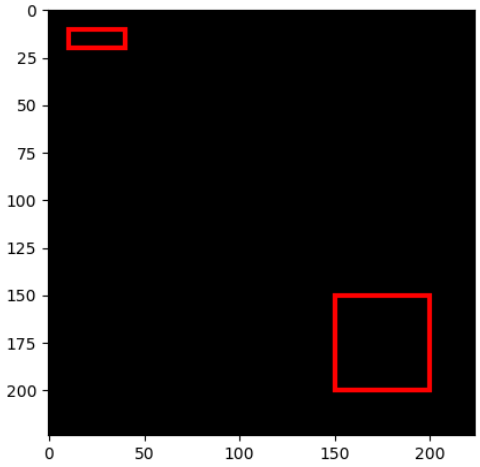
Here is an example with a simple toy data.

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

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

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

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

2.1.2 Bounding Box Dataset

ChainerCV supports dataset loaders, which can be used to easily index examples with list-like interfaces. Dataset classes whose names end with `BboxDataset` contain annotations of where objects locate in an image and which



categories they are assigned to. These datasets can be indexed to return a tuple of an image, bounding boxes and labels. The labels are stored in an `np.int32` array of shape $(R,)$. Each element corresponds to a label of an object in the corresponding bounding box.

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

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

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

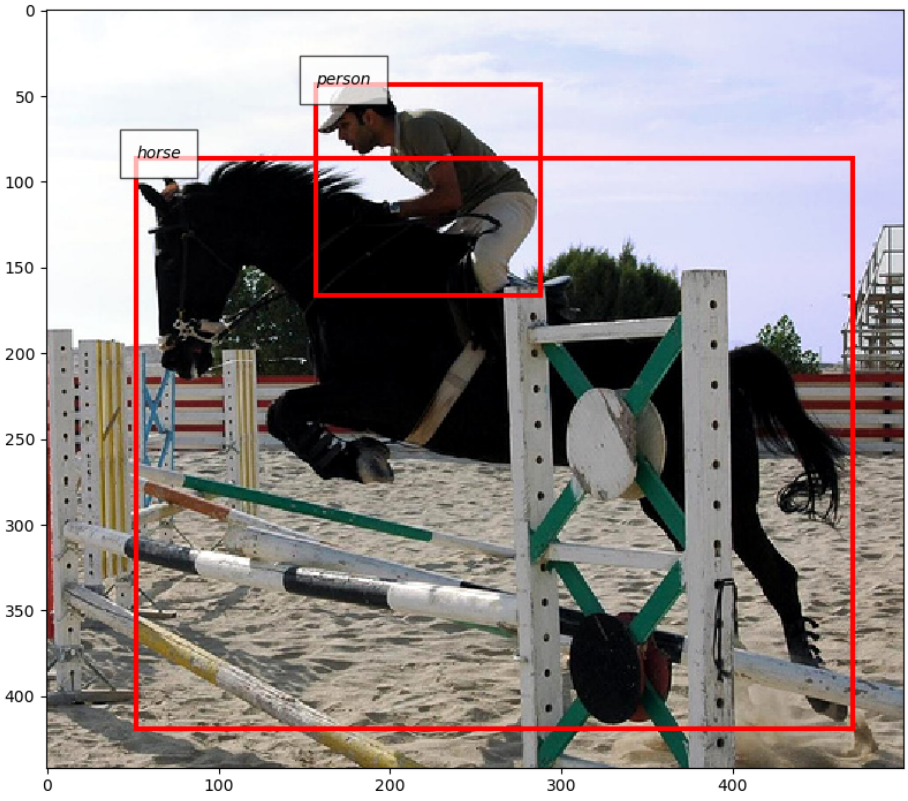
Note that the example downloads VOC 2012 dataset at runtime when it is used for the first time on the machine.

2.1.3 Detection Link

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

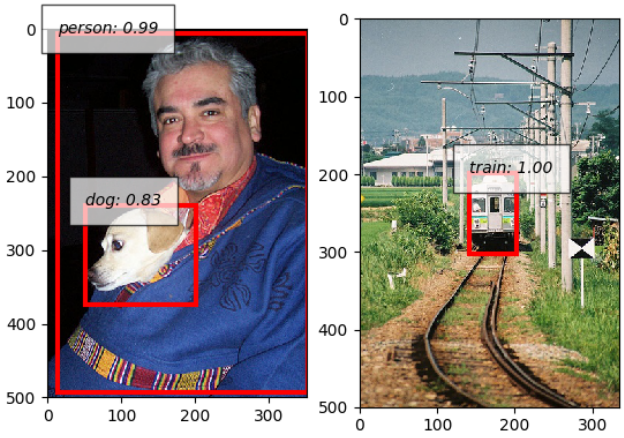
```
from chainercv.datasets import VOCBboxDataset
from chainercv.datasets import voc_bbox_label_names
from chainercv.links import SSD300
from chainercv.visualizations import vis_bbox
```

(continues on next page)


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

(continued from previous page)

```
dataset = VOCBboxDataset(year='2007', split='test')
img_0, _, _ = dataset[0]
img_1, _, _ = dataset[1]
model = SSD300(pretrained_model='voc0712')
# Note that `predict` takes a list of images.
bboxes, labels, scores = model.predict([img_0, img_1])

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



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

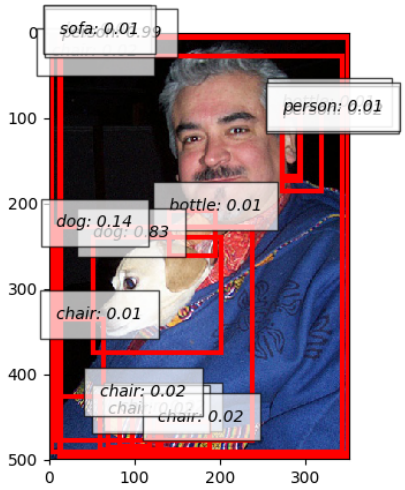
Many detection algorithms post-process bounding box proposals calculated from the output of neural networks by removing unnecessary ones. Faster R-CNN and SSD use non-maximum suppression to remove overlapping bounding boxes. Also, they remove bounding boxes with low confidence scores. These two models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

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

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

(continues on next page)

(continued from previous page)

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



2.1.4 Detection Evaluator

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

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

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

Here is a simple example that uses a detection evaluator.

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

# Only use subset of dataset so that evaluation finishes quickly.
```

(continues on next page)

(continued from previous page)

```
dataset = VOCBboxDataset(year='2007', split='test')
dataset = dataset[:6]
it = SerialIterator(dataset, 2, repeat=False, shuffle=False)
model = SSD300(pretrained_model='voc0712')
evaluator = DetectionVOCEvaluator(it, model,
                                  label_names=voc_bbox_label_names)
# result is a dictionary of evaluation scores. Print it and check it.
result = evaluator()
```

2.1.5 Training Detection Links

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

- Faster R-CNN examples
- SSD examples

2.1.6 References

2.2 Tips using Links

2.2.1 Fine-tuning

Models in ChainerCV support the argument `pretrained_model` to load pretrained weights. This functionality is limited in the case when fine-tuning pretrained weights. In that circumstance, the layers specific to the classes of the original dataset may need to be randomly initialized. In this section, we give a procedure to cope with this problem.

Copying a subset of weights in a chain can be done in few lines of code. Here is a block of code that does this.

```
# src is a model with pretrained weights
# dst is a model randomly initialized
# ignore_names is the name of parameters to skip
# For the case of VGG16, this should be ['/fc7/W', '/fc7/b']
ignore_names = []
src_params = {p[0]: p[1] for p in src.namedparams()}
for dst_named_param in dst.namedparams():
    name = dst_named_param[0]
    if name not in ignore_names:
        dst_named_param[1].array[:] = src_params[name].array[:]
```

Fine-tuning to a dataset with a different number of classes

When the number of classes of the target dataset is different from the source dataset during fine-tuning, the names of the weights to skip can be found automatically with the following method.

```
def get_shape_mismatch_names(src, dst):
    # all parameters are assumed to be initialized
    mismatch_names = []
    src_params = {p[0]: p[1] for p in src.namedparams()}
```

(continues on next page)

(continued from previous page)

```
    for dst_named_param in dst.namedparams():
        name = dst_named_param[0]
        dst_param = dst_named_param[1]
        src_param = src_params[name]
        if src_param.shape != dst_param.shape:
            mismatch_names.append(name)
    return mismatch_names
```

Finally, this is a complete example using SSD300.

```
from chainercv.links import SSD300
import numpy as np

src = SSD300(pretrained_model='voc0712')
# the number of classes in VOC is different from 50
dst = SSD300(n_fg_class=50)
# initialized weights
dst(np.zeros((1, 3, dst.insize, dst.insize), dtype=np.float32))

# the method described above
ignore_names = get_shape_mismatch_names(src, dst)
src_params = {p[0]: p[1] for p in src.namedparams()}
for dst_named_param in dst.namedparams():
    name = dst_named_param[0]
    if name not in ignore_names:
        dst_named_param[1].array[:] = src_params[name].array[:]

# check that weights are transfered
np.testing.assert_equal(dst.extractor.conv1_1.W.data,
                        src.extractor.conv1_1.W.data)
# the names of the weights that are skipped
print(ignore_names)
```

2.3 Sliceable Dataset

This tutorial will walk you through the features related to sliceable dataset. We assume that readers have a basic understanding of Chainer dataset (e.g. understand `chainer.dataset.DatasetMixin`).

In ChainerCV, we introduce *sliceable* feature to datasets. Sliceable datasets support `slice()` that returns a view of the dataset.

This example that shows the basic usage.

```
# VOCBboxDataset supports sliceable feature
from chainercv.datasets import VOCBboxDataset
dataset = VOCBboxDataset()

# keys returns the names of data
print(dataset.keys)  # ('img', 'bbox', 'label')
# we can get an example by []
img, bbox, label = dataset[0]

# get a view of the first 100 examples
view = dataset.slice[:100]
print(len(view))  # 100
```

(continues on next page)

(continued from previous page)

```
# get a view of image and label
view = dataset.slice[:, ('img', 'label')]
# the view also supports sliceable, so that we can call keys
print(view.keys)  # ('img', 'label')
# we can get an example by []
img, label = view[0]
```

2.3.1 Motivation

`slice()` returns a view of the dataset without conducting data loading, where `DatasetMixin.__getitem__()` conducts `get_example()` for all required examples. Users can write efficient code by this view.

This example counts the number of images that contain dogs. With the sliceable feature, we can access the label information without loading images from disk.. Therefore, the first case becomes faster.

```
import time

from chainercv.datasets import VOCBboxDataset
from chainercv.datasets import voc_bbox_label_names

dataset = VOCBboxDataset()
dog_lb = voc_bbox_label_names.index('dog')

# with slice
t = time.time()
count = 0
# get a view of label
view = dataset.slice[:, 'label']
for i in range(len(view)):
    # we can focus on label
    label = view[i]
    if dog_lb in label:
        count += 1
print('w/ slice: {} secs'.format(time.time() - t))
print('{} images contain dogs'.format(count))
print()

# without slice
t = time.time()
count = 0
for i in range(len(dataset)):
    # img and bbox are loaded but not needed
    img, bbox, label = dataset[i]
    if dog_lb in label:
        count += 1
print('w/o slice: {} secs'.format(time.time() - t))
print('{} images contain dogs'.format(count))
print()
```

2.3.2 Usage: slice along with the axis of examples

`slice()` takes indices of examples as its first argument.

```
from chainercv.datasets import VOCBboxDataset
dataset = VOCBboxDataset()

# the view of the first 100 examples
view = dataset.slice[:100]

# the view of the last 100 examples
view = dataset.slice[-100:]

# the view of the 3rd, 5th, and 7th examples
view = dataset.slice[3:8:2]

# the view of the 3rd, 1st, and 4th examples
view = dataset.slice[[3, 1, 4]]
```

Also, it can take a list of booleans as its first argument. Note that the length of the list should be the same as `len(dataset)`.

```
from chainercv.datasets import VOCBboxDataset
dataset = VOCBboxDataset()

# make booleans
bboxes = dataset.slice[:, 'bbox']
booleans = [len(bbox) >= 3 for bbox in bboxes]

# a collection of samples that contain at least three bounding boxes
view = dataset.slice[booleans]
```

2.3.3 Usage: slice along with the axis of data

`slice()` takes names or indices of data as its second argument. `keys` returns all available names.

```
from chainercv.datasets import VOCBboxDataset
dataset = VOCBboxDataset()

# the view of image
# note that : of the first argument means all examples
view = dataset.slice[:, 'img']
print(view.keys)  # 'img'
img = view[0]

# the view of image and label
view = dataset.slice[:, ('img', 'label')]
print(view.keys)  # ('img', 'label')
img, label = view[0]

# the view of image (returns a tuple)
view = dataset.slice[:, ('img',)]
print(view.keys)  # ('img',)
img, = view[0]

# use an index instead of a name
view = dataset.slice[:, 1]
print(view.keys)  # 'bbox'
bbox = view[0]
```

(continues on next page)

(continued from previous page)

```
# mixture of names and indices
view = dataset.slice[:, (1, 'label')]
print(view.keys)  # ('bbox', 'label')
bbox, label = view[0]

# use booleans
# note that the number of booleans should be the same as len(dataset.keys)
view = dataset.slice[:, (True, True, False)]
print(view.keys)  # ('img', 'bbox')
img, bbox = view[0]
```

2.3.4 Usage: slice along with both axes

```
from chainercv.datasets import VOCBboxDataset
dataset = VOCBboxDataset()

# the view of the labels of the first 100 examples
view = dataset.slice[:100, 'label']
```

2.3.5 Concatenate and transform

ChainerCV provides `ConcatenatedDataset` and `TransformDataset`. The difference from `chainer.datasets.ConcatenatedDataset` and `chainer.datasets.TransformDataset` is that they take sliceable dataset(s) and return a sliceable dataset.

```
from chainercv.chainer_experimental.datasets.sliceable import ConcatenatedDataset
from chainercv.chainer_experimental.datasets.sliceable import TransformDataset
from chainercv.datasets import VOCBboxDataset
from chainercv.datasets import voc_bbox_label_names

dataset_07 = VOCBboxDataset(year='2007')
print('07:', dataset_07.keys, len(dataset_07))  # 07: ('img', 'bbox', 'label') 2501

dataset_12 = VOCBboxDataset(year='2012')
print('12:', dataset_12.keys, len(dataset_12))  # 12: ('img', 'bbox', 'label') 5717

# concatenate
dataset_0712 = ConcatenatedDataset(dataset_07, dataset_12)
print('0712:', dataset_0712.keys, len(dataset_0712))  # 0712: ('img', 'bbox', 'label
↪') 8218

# transform
def transform(in_data):
    img, bbox, label = in_data

    dog_lb = voc_bbox_label_names.index('dog')
    bbox_dog = bbox[label == dog_lb]

    return img, bbox_dog

# we need to specify the names of data that the transform function returns
dataset_0712_dog = TransformDataset(dataset_0712, ('img', 'bbox_dog'), transform)
print('0712_dog:', dataset_0712_dog.keys, len(dataset_0712_dog))  # 0712_dog: ('img',
↪'bbox_dog') 8218
```

(continues on next page)

(continued from previous page)

2.3.6 Make your own dataset

ChainerCV provides `GetterDataset` to construct a new sliceable dataset.

This example implements a sliceable bounding box dataset.

```
import numpy as np

from chainercv.chainer_experimental.datasets.sliceable import GetterDataset
from chainercv.utils import generate_random_bbox

class SampleBboxDataset(GetterDataset):
    def __init__(self):
        super(SampleBboxDataset, self).__init__()

        # register getter method for image
        self.add_getter('img', self.get_image)
        # register getter method for bbox and label
        self.add_getter(('bbox', 'label'), self.get_annotation)

    def __len__(self):
        return 20

    def get_image(self, i):
        print('get_image({})'.format(i))
        # generate dummy image
        img = np.random.uniform(0, 255, size=(3, 224, 224)).astype(np.float32)
        return img

    def get_annotation(self, i):
        print('get_annotation({})'.format(i))
        # generate dummy annotations
        bbox = generate_random_bbox(10, (224, 224), 10, 224)
        label = np.random.randint(0, 9, size=10).astype(np.int32)
        return bbox, label

dataset = SampleBboxDataset()
img, bbox, label = dataset[0]  # get_image(0) and get_annotation(0)

view = dataset.slice[:, 'label']
label = view[1]  # get_annotation(1)
```

If you have arrays of data, you can use `TupleDataset`.

```
import numpy as np

from chainercv.chainer_experimental.datasets.sliceable import TupleDataset
from chainercv.utils import generate_random_bbox

n = 20
imgs = np.random.uniform(0, 255, size=(n, 3, 224, 224)).astype(np.float32)
bboxes = [generate_random_bbox(10, (224, 224), 10, 224) for _ in range(n)]
labels = np.random.randint(0, 9, size=(n, 10)).astype(np.int32)
```

(continues on next page)

(continued from previous page)

```
dataset = TupleDataset(('img', imgs), ('bbox', bboxes), ('label', labels))

print(dataset.keys)  # ('img', 'bbox', 'label')
view = dataset.slice[:, 'label']
label = view[1]
```

CHAPTER 3

ChainerCV Reference Manual

3.1 Chainer Experimental

This module contains WIP modules of Chainer. After they are merged into chainer, these modules will be removed from ChainerCV.

3.1.1 Datasets

Sliceable

Sliceable

This module support sliceable feature. Please note that this module will be removed after Chainer implements sliceable feature.

See also:

https://github.com/chainer/chainercv/pull/454

ConcatenatedDataset

GetterDataset

TupleDataset

TransformDataset

3.1.2 Training

Extensions

Extensions

make_shift

3.2 Datasets

3.2.1 General datasets

DirectoryParsingLabelDataset

directory_parsing_label_names

MixUpSoftLabelDataset

SiameseDataset

3.2.2 ADE20K

ADE20KSemanticSegmentationDataset

ADE20KTestImageDataset

3.2.3 CamVid

CamVidDataset

3.2.4 Cityscapes

CityscapesSemanticSegmentationDataset

CityscapesTestImageDataset

3.2.5 CUB

CUBLabelDataset

CUBKeypointDataset

3.2.6 MS COCO

COCOBboxDataset

COCOInstanceSegmentationDataset

COCOSemanticSegmentationDataset

3.2.7 OnlineProducts

OnlineProductsDataset

3.2.8 PASCAL VOC

VOCBboxDataset

VOCInstanceSegmentationDataset

VOCSemanticSegmentationDataset

3.2.9 Semantic Boundaries Dataset

SBDInstanceSegmentationDataset

3.3 Evaluations

3.3.1 Detection COCO

eval_detection_coco

3.3.2 Detection VOC

eval_detection_voc

calc_detection_voc_ap

calc_detection_voc_prec_rec

3.3.3 Instance Segmentation COCO

eval_instance_segmentation_coco

3.3.4 Instance Segmentation VOC

eval_instance_segmentation_voc

calc_instance_segmentation_voc_prec_rec

3.3.5 Semantic Segmentation IoU

eval_semantic_segmentation

calc_semantic_segmentation_confusion

calc_semantic_segmentation_iou

3.4 Experimental

3.4.1 Links

Detection

Detection links share a common method `predict()` to detect objects in images.

YOLO

Object Detection Link

YOLOv2Tiny

Utility

DarknetExtractor

Semantic Segmentation

Semantic segmentation links share a common method `predict()` to conduct semantic segmentation of images.

PSPNet

Semantic Segmentation Link

PSPNetResNet101

PSPNetResNet50

Utility

convolution_crop

PSPNet

Instance Segmentation

Instance segmentation share a common method `predict()` to detect objects in images. For more details, please read `FCIS.predict()`.

FCIS

Instance Segmentation Link

FCISResNet101

Utility

FCIS

FCISResNet101Head

mask_voting

ResNet101Extractor

Train-only Utility

FCISTrainChain

ProposalTargetCreator

3.5 Extensions

3.5.1 Evaluator

DetectionCOCOEvaluator

DetectionVOCEvaluator

InstanceSegmentationCOCOEvaluator

InstanceSegmentationVOCEvaluator

SemanticSegmentationEvaluator

3.5.2 Visualization Report

DetectionVisReport

3.6 Functions

3.6.1 Spatial Pooling

ps_roi_average_align_2d

ps_roi_average_pooling_2d

ps_roi_max_align_2d

ps_roi_max_pooling_2d

3.7 Links

3.7.1 Model

General Chain

General Chain

FeaturePredictor

PickableSequentialChain

Feature Extraction

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

ResNet

Feature Extraction Link

ResNet

ResNet50

ResNet101

ResNet152

Utility

Bottleneck

ResBlock

SEResNet

Feature Extraction Link

SEResNet

SEResNet50

SEResNet101

SEResNet152

SEResNeXt

SEResNeXt50

SEResNeXt101

VGG

VGG16

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

Utility

bbox2loc

FasterRCNN

generate_anchor_base

loc2bbox

ProposalCreator

RegionProposalNetwork

VGG16RoIHead

Train-only Utility

AnchorTargetCreator

FasterRCNNTrainChain

ProposalTargetCreator

SSD (Single Shot Multibox Detector)

Detection Links

SSD300

SSD512

Utility

Multibox

MultiboxCoder

Normalize

SSD

VGG16

VGG16Extractor300

VGG16Extractor512

Train-only Utility

GradientScaling

multibox_loss

random_crop_with_bbox_constraints

random_distort

resize_with_random_interpolation

YOLO

Detection Links

YOLOv2

YOLOv3

Utility

ResidualBlock

Darknet19Extractor

Darknet53Extractor

YOLOBase

YOLOv2Base

Light Head R-CNN

Detection Link

LightHeadRCNNResNet101

Utility

LightHeadRCNN

Train-only Utility

LightHeadRCNNTrainChain

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

DeepLab

Semantic Segmentation Link

Train-only Utility

bbox_head_loss_pre

bbox_head_loss_post

rpn_loss

mask_head_loss_pre

mask_head_loss_post

mask_to_segm

Classifiers

Classifier

PixelwiseSoftmaxClassifier

3.7.2 Connection

Connection

Conv2DActiv

Conv2DBNActiv

SEBlock

SeparableConv2DBNActiv

3.8 Transforms

3.8.1 Image

center_crop

flip

pca_lighting

random_crop

random_expand

random_flip

random_rotate

random_sized_crop

resize

resize_contain

rotate

scale

ten_crop

3.8.2 Bounding Box

crop_bbox

flip_bbox

resize_bbox

rotate_bbox

translate_bbox

3.8.3 Point

flip_point

resize_point

translate_point

3.9 Visualizations

3.9.1 vis_bbox

3.9.2 vis_image

3.9.3 vis_instance_segmentation

3.9.4 vis_point

3.9.5 vis_semantic_segmentation

3.10 Utils

3.10.1 Bounding Box Utilities

bbox_iou

non_maximum_suppression

3.10.2 Download Utilities

cached_download

download_model

extractall

3.10.3 Image Utilities

read_image

read_label

tile_images

write_image

3.10.4 Iterator Utilities

apply_to_iterator

ProgressHook

unzip

3.10.5 Link Utilities

prepare_pretrained_model

3.10.6 Mask Utilities

mask_iou

mask_to_bbox

scale_mask

3.10.7 Testing Utilities

assert_is_bbox

assert_is_bbox_dataset

assert_is_detection_link

assert_is_image

assert_is_instance_segmentation_dataset

assert_is_label_dataset

assert_is_point

assert_is_point_dataset

assert_is_semantic_segmentation_dataset

assert_is_semantic_segmentation_link

ConstantStubLink

generate_random_bbox

CHAPTER 4

Naming Conventions

Here are the notations used.

- B is the size of a batch.
- H is the height of an image.
- W is the width of an image.
- C is the number of channels.
- R is the total number of instances in an image.
- L is the number of classes.

4.1 Data objects

4.1.1 Images

- `imgs`: (B, C, H, W) or $[(C, H, W)]$
- `img`: (C, H, W)

Note: `image` is used for a name of a function or a class (e.g., `chainercv.utils.write_image()`).

4.1.2 Bounding boxes

- `bboxes`: $(B, R, 4)$ or $[(R, 4)]$
- `bbox`: $(R, 4)$
- `bb`: $(4,)$

4.1.3 Labels

name	classification	detection and instance segmentation	semantic segmentation	
`labels`	$(B,)$	(B, R) or $[(R,)]$	(B, H, W)	
`label`	$()$	$(R,)$	(H, W)	
`l`	r `lb`	–	$()$	–

4.1.4 Scores and probabilities

score represents an unbounded confidence value. On the other hand, probability is bounded in `[0, 1]` and sums to 1.

name	classification	detection and instance segmentation	semantic segmentation
`scores` or `probs`	(B, L)	(B, R, L) or $[(R, L)]$	(B, L, H, W)
`score` or `prob`	$(L,)$	(R, L)	(L, H, W)
`sc` or `pb`	–	$(L,)$	–

Note: Even for objects that satisfy the definition of probability, they can be named as `score`.

4.1.5 Instance segmentations

- `masks`: (B, R, H, W) or $[(R, H, W)]$
- `mask`: (R, H, W)
- `msk`: (H, W)

4.2 Attributing an additonal meaning to a basic data object

4.2.1 RoIs

- `rois`: $(R', 4)$, which consists of bounding boxes for multiple images. Assuming that there are B images each containing R_i bounding boxes, the formula $R' = \sum R_i$ is true.
- `roi_indices`: An array of shape $(R',)$ that contains batch indices of images to which bounding boxes correspond.
- `roi`: $(R, 4)$. This is RoIs for single image.

4.2.2 Attributes associated to RoIs

RoIs may have additional attributes, such as class scores and masks. These attributes are named by appending `roi_` (e.g., `scores`-like object is named as `roi_scores`).

- `roi_xs`: $(R',) + x_{shape}$
- `roi_x`: $(R,) + x_{shape}$

In the case of `scores` with shape $(L,)$, `roi_xs` would have shape (R', L).

Note: `roi_nouns = roi_noun = noun` when `batchsize=1`. Changing names interchangeably is fine.

4.2.3 Class-wise vs class-independent

`cls_nouns` is a multi-class version of `nouns`. For instance, `cls_locs` is $(B, R, L, 4)$ and `locs` is $(B, R, 4)$.

Note: `cls_probs` and `probs` can be used interchangeably in the case when there is no confusion.

4.2.4 Arbitrary input

`x` is a variable whose shape can be inferred from the context. It can be used only when there is no confusion on its shape. This is usually the case when naming an input to a neural network.

CHAPTER 5

License

5.1 Source Code

The source code of ChainerCV is licensed under MIT-License.

5.2 Pretrained Models

Pretrained models provided by ChainerCV are benefited from the following resources. See the following resources for the terms of use of a model with weights pretrained by any of such resources.

model	resource
ResNet50/101/152 (`imagenet`)	• ResNet50/101/152 (trained on ImageNet)
SEResNet50/101/152 (`imagenet`)	• SEResNet50/101/152 (trained on ImageNet)
SEResNeXt50/101 (`imagenet`)	• SEResNeXt50/101 (trained on ImageNet)
VGG16 (`imagenet`)	• VGG-16 (trained on ImageNet)
FasterRCNNVGG16 (`imagenet`)	• VGG-16 (trained on ImageNet)
FasterRCNNVGG16 (`voc07`/`voc0712`)	• VGG-16 (trained on ImageNet) • PASCAL VOC
SSD300/SSD512 (`imagenet`)	• VGG-16 (trained on ImageNet, FC reduced)
SSD300/SSD512 (`voc0712`)	• SSD300/SSD512 (trained on PASCAL VOC 2007 and 2012)
YOLOv2 (`voc0712`)	• Darknet19 (trained on ImageNet) • PASCAL VOC
YOLOv3 (`voc0712`)	• Darknet53 (trained on ImageNet) • PASCAL VOC
PSPNetResNet101 (`cityscapes`)	• PSPNet101 (trained on Cityscapes)
SegNetBasic (`camvid`)	• CamVid
FCISResNet101 (`sbd`)	• ResNet101 (trained on ImageNet) • SBD

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Bibliography

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

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

Python Module Index

C

Index

C