Package: torchvisionlib (via r-universe)

November 21, 2026

Title Additional Operators for Image Models

Version 0.8.0.9000

Description Implements additional operators for computer vision models, including operators necessary for image segmentation and object detection deep learning models.

License MIT + file LICENSE

URL https://github.com/mlverse/torchvisionlib

BugReports https://github.com/mlverse/torchvisionlib/issues

Depends R (>= 3.6)

Imports glue, Rcpp, rlang, torch (>= 0.17.0), withr

Suggests testthat (>= 3.0.0)

LinkingTo Rcpp, torch

Config/Needs/development Rcpp, torch, torchexport, testthat, roxygen2

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/torchvisionlib/binaries Set TORCHVISIONLIB_URL to a URL, local ZIP, or directory containing pre-built torchvisionlib binaries. If unset, defaults to inst/libs/ (installed) or src/ (development).

Encoding UTF-8

Roxygen list(markdown = TRUE)

SystemRequirements C++17, CMake (>= 3.21), libtorch, libtorchvision

Config/pak/sysreqs cmake

Repository https://cregouby.r-universe.dev

Date/Publication 2026-09-01 11:30:11 UTC

RemoteUrl https://github.com/mlverse/torchvisionlib

RemoteRef HEAD

RemoteSha 2f357d8c0dcec28d02f5df41dd0bfba1babacf0a

Contents

ops_box_iou_rotated 2
ops_deform_conv2d 3
ops_ms_deform_attn 4
ops_nms 5
ops_ps_roi_align 6
ops_roi_align_rotated 7
torchvisionlib_is_installed 9
vision_read_jpeg 9

Index **10**

`ops_box_iou_rotated` *Intersection-over-union between rotated boxes*

Description

Computes the pairwise intersection-over-union (IoU) between two sets of rotated bounding boxes.

Usage

```
ops_box_iou_rotated(boxes1, boxes2, fmt = "cxcywhr")
```

Arguments

`boxes1`	`Tensor[N, K]` first set of rotated boxes.
`boxes2`	`Tensor[M, K]` second set of rotated boxes.
`fmt`	format of the input boxes. One of: • `"cxcywhr"` (`K = 5`): center `(cx, cy)`, width, height and rotation angle `r` in degrees (counter-clockwise positive). • `"xywhr"` (`K = 5`): top-left corner `(x1, y1)`, width, height and angle. • `"xyxyxyxy"` (`K = 8`): the four corners `(x1, y1, x2, y2, x3, y3, x4, y4)`.

Value

`Tensor[N, M]` float32 matrix of pairwise IoU values.

See Also

Other ops: `ops_ms_deform_attn()`, `ops_nms()`, `ops_roi_align_rotated()`

Examples

```
if (torchvisionlib_is_installed()) {
  boxes <- torch::torch_tensor(matrix(c(0, 0, 10, 10, 45), nrow = 1))
  ops_box_iou_rotated(boxes, boxes)
}
```

`ops_deform_conv2d` *Performs Deformable Convolution v2,*

Description

Ddescribed in Deformable ConvNets v2: More Deformable, Better Results if `mask` is not `NULL` and performs Deformable Convolution, described in Deformable Convolutional Networks if `mask` is `NULL`.

Usage

```
ops_deform_conv2d(
  input,
  offset,
  weight,
  bias = NULL,
  stride = c(1, 1),
  padding = c(0, 0),
  dilation = c(1, 1),
  mask = NULL
)
```

Arguments

`input`	(`Tensor[batch_size, in_channels, in_height, in_width]`): input tensor
`offset`	(`Tensor[batch_size, 2 * offset_groups * kernel_height * kernel_width, out_height, out_width]`): offsets to be applied for each position in the convolution kernel.
`weight`	(`Tensor[out_channels, in_channels // groups, kernel_height, kernel_width]`): convolution weights, split into groups of size (in_channels // groups)
`bias`	(`Tensor[out_channels]`): optional bias of shape (out_channels,). Default: `NULL`
`stride`	(int or `Tuple[int, int]`): distance between convolution centers. Default: 1
`padding`	(int or `Tuple[int, int]`): height/width of padding of zeroes around each image. Default: 0
`dilation`	(int or `Tuple[int, int]`): the spacing between kernel elements. Default: 1
`mask`	(`Tensor[batch_size, offset_groups * kernel_height * kernel_width, out_height, out_width]`): masks to be applied for each position in the convolution kernel. Default: `NULL`

Value

`Tensor[batch_sz, out_channels, out_h, out_w]`: result of convolution

Examples

```
if (torchvisionlib_is_installed()) {
  library(torch)
  input <- torch_rand(4, 3, 10, 10)
  kh <- kw <- 3
  weight <- torch_rand(5, 3, kh, kw)
  # offset and mask should have the same spatial size as the output
  # of the convolution. In this case, for an input of 10, stride of 1
  # and kernel size of 3, without padding, the output size is 8
  offset <- torch_rand(4, 2 * kh * kw, 8, 8)
  mask <- torch_rand(4, kh * kw, 8, 8)
  out <- ops_deform_conv2d(input, offset, weight, mask = mask)
  print(out$shape)
}
```

---

ops_ms_deform_attn *Multi-scale deformable attention*

Description

Computes multi-scale deformable attention as used by Deformable-DETR and LW-DETR. Only a CUDA implementation is provided; all input tensors must be on the same CUDA device. On CPU, models should fall back to a pure-R implementation (e.g. based on `torch::nnf_grid_sample()`).

Usage

```
ops_ms_deform_attn(
  value,
  spatial_shapes,
  level_start_index,
  sampling_locations,
  attention_weights,
  im2col_step = 64L
)
```

Arguments

`value`	(`Tensor[batch, Len_in, n_heads, head_dim]`): flattened multi-scale feature values.
`spatial_shapes`	(`Tensor[n_levels, 2]`, integer): the `(H, W)` of each feature level. The sum of `H * W` over levels must equal `Len_in`.
`level_start_index`	(`Tensor[n_levels]`, integer): 0-based start offset of each level within `Len_in`.
`sampling_locations`	(`Tensor[batch, Len_q, n_heads, n_levels, n_points, 2]`): sampling locations in `[0, 1]` (normalized `x, y`).

attention_weights
: (Tensor[batch, Len_q, n_heads, n_levels, n_points]): attention weights, typically normalized over the n_levels * n_points axis.

im2col_step
: (int): batch chunk size used internally by the kernel. Must divide batch. Default: 64.

Details

spatial_shapes and level_start_index use the kernel's 0-based indexing convention: level_start_index[l] is the flat offset (into the Len_in dimension of value) of the first element of level l, and the output is a feature tensor, so no 1-based index adjustment is applied.

Value

Tensor[batch, Len_q, n_heads * head_dim]: the attended output.

See Also

Other ops: ops_box_iou_rotated(), ops_nms(), ops_roi_align_rotated()

---

ops_nms *Performs non-maximum suppression (NMS) on the boxes*

Description

Performs non-maximum suppression (NMS) on the boxes according to their intersection-over-union (IoU).

Usage

```
ops_nms(boxes, scores, iou_threshold)
```

Arguments

boxes
: Tensor[N,4] boxes to perform NMS on. They are expected to be in (x1, y1, x2, y2) format with 0 <= x1 < x2 and 0 <= y1 < y2.

scores
: Tensor[N] scores for each one of the boxes.

iou_threshold
: float discards all overlapping boxes with IoU > iou_threshold.

Details

NMS iteratively removes lower scoring boxes which have an IoU greater than iou_threshold with another (higher scoring) box.

If multiple boxes have the exact same score and satisfy the IoU criterion with respect to a reference box, the selected box is not guaranteed to be the same between CPU and GPU. This is similar to the behavior of argsort in PyTorch when repeated values are present.

Value

int64 tensor with the indices of the elements that have been kept by NMS, sorted in decreasing order of scores

See Also

Other ops: `ops_box_iou_rotated()`, `ops_ms_deform_attn()`, `ops_roi_align_rotated()`

Examples

```
if (torchvisionlib_is_installed()) {
  ops_nms(torch::torch_rand(3, 4), torch::torch_rand(3), 0.5)
}
```

`ops_ps_roi_align` *Performs Position-Sensitive Region of Interest (RoI) Align operator*

Description

The (RoI) Align operator is mentioned in Light-Head R-CNN.

Usage

```
ops_ps_roi_align(
  input,
  boxes,
  output_size,
  spatial_scale = 1,
  sampling_ratio = -1
)

nn_ps_roi_align(output_size, spatial_scale = 1, sampling_ratio = -1)
```

Arguments

`input`	(`Tensor[N, C, H, W]`): The input tensor, i.e. a batch with `N` elements. Each element contains `C` feature maps of dimensions `H x W`.
`boxes`	(`Tensor[K, 5]` or `List[Tensor[L, 4]]`): the box coordinates in (x1, y1, x2, y2) format where the regions will be taken from. The coordinate must satisfy `0 <= x1 < x2` and `0 <= y1 < y2`. If a single Tensor is passed, then the first column should contain the index of the corresponding element in the batch, i.e. a number in `[1, N]`. If a list of Tensors is passed, then each Tensor will correspond to the boxes for an element i in the batch.
`output_size`	(int or `Tuple[int, int]`): the size of the output (in bins or pixels) after the pooling is performed, as (height, width).

`spatial_scale` (float): a scaling factor that maps the box coordinates to the input coordinates. For example, if your boxes are defined on the scale of a 224x224 image and your input is a 112x112 feature map (resulting from a 0.5x scaling of the original image), you'll want to set this to 0.5. Default: 1.0

`sampling_ratio` (int): number of sampling points in the interpolation grid used to compute the output value of each pooled output bin. If > 0, then exactly `sampling_ratio x sampling_ratio` sampling points per bin are used. If <= 0, then an adaptive number of grid points are used (computed as `ceil(roi_width / output_width)`, and likewise for height). Default: -1

Value

`Tensor[K, C / (output_size[1] * output_size[2]), output_size[1], output_size[2]]`: The pooled RoIs

Functions

- `nn_ps_roi_align()`: The `torch::nn_module()` wrapper for `ops_ps_roi_align()`.

Examples

```
if (torchvisionlib_is_installed()) {
library(torch)
library(torchvisionlib)
input <- torch_randn(1, 3, 28, 28)
boxes <- list(torch_tensor(matrix(c(1,1,5,5), ncol = 4)))
roi <- nn_ps_roi_align(output_size = c(1, 1))
roi(input, boxes)
}
```

---

ops_roi_align_rotated *RoI align pooling for rotated proposals*

Description

Performs RoI (Region of Interest) align pooling for rotated proposals, as implemented by the MMCV `roi_align_rotated` operator. Only a CPU implementation is provided.

Usage

```
ops_roi_align_rotated(
  input,
  rois,
  output_size,
  spatial_scale,
  sampling_ratio = 0,
  aligned = TRUE,
```

```
  clockwise = FALSE
)

nn_roi_align_rotated(
  output_size,
  spatial_scale,
  sampling_ratio = 0,
  aligned = TRUE,
  clockwise = FALSE
)
```

Arguments

input	(Tensor[N, C, H, W]): input feature map.
rois	(Tensor[K, 6]): rotated boxes with columns (batch_index, cx, cy, w, h, angle), where batch_index is an index into the first dimension of input, (cx, cy) is the box center, (w, h) the box size and angle the rotation angle in radians (counterclockwise unless clockwise = TRUE).
output_size	(integer or integer vector): the output size (height, width) after pooling.
spatial_scale	(float): scaling factor mapping box coordinates to input coordinates. For example, if boxes are defined on a 224x224 image and input is a 112x112 feature map, set this to 0.5.
sampling_ratio	(int): number of sampling points per output bin. If <= 0, an adaptive number of grid points is used (computed as ceil(roi_size / output_size)). Default: 0
aligned	(bool): if TRUE (default), the aligned implementation is used (results are shifted by -0.5 before interpolation, matching detectron2). If FALSE, the legacy MMDetection implementation is used and boxes are clamped to a minimum size of 1.
clockwise	(bool): if TRUE, the rotation angle is interpreted in a clockwise fashion in image space, otherwise it is counterclockwise. Default: FALSE

Value

Tensor[K, C, output_height, output_width]: the pooled features, where the r-th element corresponds to the r-th RoI in rois.

Functions

- nn_roi_align_rotated(): The torch::nn_module() wrapper for ops_roi_align_rotated().

See Also

Other ops: ops_box_iou_rotated(), ops_ms_deform_attn(), ops_nms()

Examples

```
if (torchvisionlib_is_installed()) {
  library(torch)
  input <- torch_randn(1, 3, 28, 28)
```

```
  # (batch_index, cx, cy, w, h, angle) with 0-based batch index
  rois <- torch_tensor(matrix(c(0, 14, 14, 10, 10, 0.5), ncol = 6))
  ops_roi_align_rotated(input, rois, output_size = c(5, 5),
                        spatial_scale = 1, sampling_ratio = 2)
}
```

`torchvisionlib_is_installed` *Checks if an installation of torchvisionlib was found.*

Description

Checks if an installation of torchvisionlib was found.

Install additional libraries

Usage

```
torchvisionlib_is_installed()

install_torchvisionlib(url = Sys.getenv("TORCHVISIONLIB_URL", unset = NA))
```

Arguments

`url`	Url for the binaries. Can also be the file path to the binaries.

`vision_read_jpeg` *Read JPEG's directly into torch tensors*

Description

Read JPEG's directly into torch tensors

Usage

```
vision_read_jpeg(path)
```

Arguments

`path`	path to JPEG file

Index

∗ **ops**
ops_box_iou_rotated, 2
ops_ms_deform_attn, 4
ops_nms, 5
ops_roi_align_rotated, 7

install_torchvisionlib
*(*torchvisionlib_is_installed*)*, 9

nn_ps_roi_align *(*ops_ps_roi_align*)*, 6
nn_roi_align_rotated
*(*ops_roi_align_rotated*)*, 7

ops_box_iou_rotated, 2
ops_box_iou_rotated(), *5, 6*, *8*
ops_deform_conv2d, 3
ops_ms_deform_attn, 4
ops_ms_deform_attn(), *2*, *6*, *8*
ops_nms, 5
ops_nms(), *2*, *5*, *8*
ops_ps_roi_align, 6
ops_ps_roi_align(), *7*
ops_roi_align_rotated, 7
ops_roi_align_rotated(), *2*, *5, 6*, *8*

torch::nn_module(), *7, 8*
torch::nnf_grid_sample(), *4*
torchvisionlib_is_installed, 9

vision_read_jpeg, 9