

# Package ‘torchvisionlib’

April 15, 2026

**Title** Additional Operators for Image Models

**Version** 0.8.0

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

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Depends** R (>= 3.6)

**LinkingTo** Rcpp, torch

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

**Suggests** testthat (>= 3.0.0)

**Config/testthat/edition** 3

**URL** <https://github.com/mlverse/torchvisionlib>

**BugReports** <https://github.com/mlverse/torchvisionlib/issues>

**NeedsCompilation** yes

**Author** Daniel Falbel [aut],  
Tomasz Kalinowski [ctb, cre],  
RStudio [cph]

**Maintainer** Tomasz Kalinowski <tomasz@posit.co>

**Repository** CRAN

**Date/Publication** 2026-04-15 07:00:02 UTC

## Contents

ops_deform_conv2d . . . . .	2
ops_nms . . . . .	3
ops_ps_roi_align . . . . .	4
torchvisionlib_is_installed . . . . .	5
vision_read_jpeg . . . . .	6

**Index**[7](#)


---

ops\_deform\_conv2d      *Performs Deformable Convolution v2,*

---

**Description**

Described 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\_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 $0 \leq x1 < x2$ and $0 \leq y1 < y2$ .
scores	Tensor[N] scores for each one of the boxes.
iou_threshold	float discards all overlapping boxes with $\text{IoU} > \text{iou\_threshold}$ .

**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

**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 $0 \leq x1 < x2$ and $0 \leq y1 < y2$ . 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 <code>sampling_ratio x sampling_ratio</code> sampling points per bin are used. If $\leq 0$ , then an adaptive number of grid points are used (computed as $\text{ceil}(\text{roi\_width} / \text{output\_width})$ , 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)  
}
```

---

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	<i>Read JPEG's directly into torch tensors</i>
------------------	--

---

**Description**

Read JPEG's directly into torch tensors

**Usage**

```
vision_read_jpeg(path)
```

**Arguments**

path	path to JPEG file
------	-------------------

# Index

## \* ops

- `ops_nms`, 3
- `install_torchvisionlib`
  - `(torchvisionlib_is_installed)`, 5
- `nn_ps_roi_align` (`ops_ps_roi_align`), 4
- `ops_deform_conv2d`, 2
- `ops_nms`, 3
- `ops_ps_roi_align`, 4
- `ops_ps_roi_align()`, 5
- `torch::nn_module()`, 5
- `torchvisionlib_is_installed`, 5
- `vision_read_jpeg`, 6