Parameterized transformations

This section includes functions for performing transformations on images requiring parameter Tensors like a warp `field` or a convolution `kernel`.

[res] image.warp([dst,]src,field, [mode,offset,clamp_mode,pad_val])

Warps image `src` (of size `KxHxW`)
according to flow field `field`. The latter has size `2xHxW` where the first dimension is for the `(y,x)` flow field. String `mode` can take on values lanczos,
bicubic,
bilinear (the default),
or *simple*. When `offset` is true (the default), `(x,y)` is added to the flow field.
The `clamp_mode` variable specifies how to handle the interpolation of samples off the input image.
Permitted values are strings *clamp* (the default) or *pad*.
When `clamp_mode` equals `pad`, the user can specify the padding value with `pad_val` (default = 0). Note: setting this value when `clamp_mode` equals `clamp` will result in an error.
If `dst` is specified, it is used to store the result of the warp.
Otherwise, returns a new `res` Tensor.

[res] image.affinetransform([dst,]src,matrix, [mode,translation,clamp_mode,pad_val])

Warps image `src` (of size `KxHxW`)
according to `(y,x)` affine transformation defined by `matrix`.
The latter has size `2x2`. String `mode` can take on values lanczos,
bicubic,
bilinear (the default),
or *simple*.
Additional translation can be added to the image before affine transformation with `translation`.(Default is `torch.Tensor{0, 0}`.)

The `clamp_mode` variable specifies how to handle the interpolation of samples off the input image.
Permitted values are strings *clamp* (the default) or *pad*.
When `clamp_mode` equals `pad`, the user can specify the padding value with `pad_val` (default = 0). Note: setting this value when `clamp_mode` equals `clamp` will result in an error.
If `dst` is specified, it is used to store the result of the warp.
Otherwise, returns a new `res` Tensor.

[res] image.convolve([dst,] src, kernel, [mode])

Convolves Tensor `kernel` over image `src`. Valid string values for argument `mode` are :
* *full* : the `src` image is effectively zero-padded such that the `res` of the convolution has the same size as `src`;
* *valid* (the default) : the `res` image will have `math.ceil(kernel/2)` less columns and rows on each side;
* *same* : performs a *full* convolution, but crops out the portion fitting the output size of *valid*;
Note that this function internally uses
torch.conv2.
If `dst` is provided, it is used to store the output image.
Otherwise, returns a new `res` Tensor.

[res] image.lcn(src, [kernel])

Local contrast normalization (LCN) on a given `src` image using kernel `kernel`.
If `kernel` is not given, then a default `9x9` Gaussian is used
(see image.gaussian).

To prevent border effects, the image is first global contrast normalized (GCN) by substracting the global mean and dividing by the global standard deviation.

Then the image is locally contrast normalized using the following equation:

```
res = (src - lm(src)) / sqrt( lm(src) - lm(src*src) )
```

where `lm(x)` is the local mean of each pixel in the image (i.e. `image.convolve(x,kernel)`) and `sqrt(x)` is the element-wise square root of `x`. In other words, LCN performs local substractive and divisive normalization.

Note that this implementation is different than the LCN Layer defined on page 3 of What is the Best Multi-Stage Architecture for Object Recognition?.

[res] image.erode(src, [kernel, pad])

Performs a morphological erosion
on binary (zeros and ones) image `src` using odd
dimensioned morphological binary kernel `kernel`.
The default is a kernel consisting of ones of size `3x3`. Number
`pad` is the value to assume outside the image boundary when performing
the convolution. The default is 1.

[res] image.dilate(src, [kernel, pad])

Performs a morphological dilation
on binary (zeros and ones) image `src` using odd
dimensioned morphological binary kernel `kernel`.
The default is a kernel consisting of ones of size `3x3`. Number
`pad` is the value to assume outside the image boundary when performing
the convolution. The default is 0.