Module: tf.nn / tf.compat.v1.nn

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn#top_of_page)
* [Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn#modules)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn#functions)
* [Other Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn#other_members)

Wrappers for primitive Neural Net (NN) Operations.

Modules

[rnn\_cell](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/rnn_cell) module: Module for constructing RNN Cells.

Functions

[all\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/all_candidate_sampler): Generate the set of all classes.

[atrous\_conv2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/atrous_conv2d): Atrous convolution (a.k.a. convolution with holes or dilated convolution).

[atrous\_conv2d\_transpose(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/atrous_conv2d_transpose): The transpose of atrous\_conv2d.

[avg\_pool(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/avg_pool): Performs the average pooling on the input.

[avg\_pool1d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool1d): Performs the average pooling on the input.

[avg\_pool2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/avg_pool): Performs the average pooling on the input.

[avg\_pool3d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool3d): Performs the average pooling on the input.

[avg\_pool\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool): Performs the avg pooling on the input.

[batch\_norm\_with\_global\_normalization(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/batch_norm_with_global_normalization): Batch normalization.

[batch\_normalization(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/batch_normalization): Batch normalization.

[bias\_add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/bias_add): Adds bias to value.

[bidirectional\_dynamic\_rnn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/bidirectional_dynamic_rnn): Creates a dynamic version of bidirectional recurrent neural network. (deprecated)

[collapse\_repeated(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/collapse_repeated): Merge repeated labels into single labels.

[compute\_accidental\_hits(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_accidental_hits): Compute the position ids in sampled\_candidates matching true\_classes.

[compute\_average\_loss(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_average_loss): Scales per-example losses with sample\_weights and computes their average.

[conv1d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv1d): Computes a 1-D convolution given 3-D input and filter tensors. (deprecated argument values) (deprecated argument values)

[conv1d\_transpose(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv1d_transpose): The transpose of conv1d.

[conv2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv2d): Computes a 2-D convolution given 4-D input and filter tensors.

[conv2d\_backprop\_filter(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv2d_backprop_filter): Computes the gradients of convolution with respect to the filter.

[conv2d\_backprop\_input(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv2d_backprop_input): Computes the gradients of convolution with respect to the input.

[conv2d\_transpose(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv2d_transpose): The transpose of conv2d.

[conv3d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv3d): Computes a 3-D convolution given 5-D input and filter tensors.

[conv3d\_backprop\_filter(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv3d_backprop_filter): Computes the gradients of 3-D convolution with respect to the filter.

[conv3d\_backprop\_filter\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv3d_backprop_filter): Computes the gradients of 3-D convolution with respect to the filter.

[conv3d\_transpose(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/conv3d_transpose): The transpose of conv3d.

[conv\_transpose(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv_transpose): The transpose of convolution.

[convolution(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/convolution): Computes sums of N-D convolutions (actually cross-correlation).

[crelu(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/crelu): Computes Concatenated ReLU.

[ctc\_beam\_search\_decoder(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/ctc_beam_search_decoder): Performs beam search decoding on the logits given in input.

[ctc\_beam\_search\_decoder\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_beam_search_decoder): Performs beam search decoding on the logits given in input.

[ctc\_greedy\_decoder(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_greedy_decoder): Performs greedy decoding on the logits given in input (best path).

[ctc\_loss(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/ctc_loss): Computes the CTC (Connectionist Temporal Classification) Loss.

[ctc\_loss\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_loss): Computes CTC (Connectionist Temporal Classification) loss.

[ctc\_unique\_labels(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_unique_labels): Get unique labels and indices for batched labels for [tf.nn.ctc\_loss](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_loss).

[depth\_to\_space(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/depth_to_space): DepthToSpace for tensors of type T.

[depthwise\_conv2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/depthwise_conv2d): Depthwise 2-D convolution.

[depthwise\_conv2d\_backprop\_filter(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d_backprop_filter): Computes the gradients of depthwise convolution with respect to the filter.

[depthwise\_conv2d\_backprop\_input(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d_backprop_input): Computes the gradients of depthwise convolution with respect to the input.

[depthwise\_conv2d\_native(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/depthwise_conv2d_native): Computes a 2-D depthwise convolution given 4-D input and filter tensors.

[depthwise\_conv2d\_native\_backprop\_filter(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d_backprop_filter): Computes the gradients of depthwise convolution with respect to the filter.

[depthwise\_conv2d\_native\_backprop\_input(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d_backprop_input): Computes the gradients of depthwise convolution with respect to the input.

[dilation2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/dilation2d): Computes the grayscale dilation of 4-D input and 3-D filter tensors.

[dropout(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/dropout): Computes dropout. (deprecated arguments)

[dynamic\_rnn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/dynamic_rnn): Creates a recurrent neural network specified by RNNCell cell. (deprecated)

[elu(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/elu): Computes exponential linear: exp(features) - 1 if < 0, features otherwise.

[embedding\_lookup(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/embedding_lookup): Looks up ids in a list of embedding tensors.

[embedding\_lookup\_sparse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/embedding_lookup_sparse): Computes embeddings for the given ids and weights.

[erosion2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/erosion2d): Computes the grayscale erosion of 4-D value and 3-D kernel tensors.

[fixed\_unigram\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/fixed_unigram_candidate_sampler): Samples a set of classes using the provided (fixed) base distribution.

[fractional\_avg\_pool(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/fractional_avg_pool): Performs fractional average pooling on the input. (deprecated)

[fractional\_max\_pool(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/fractional_max_pool): Performs fractional max pooling on the input. (deprecated)

[fused\_batch\_norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/fused_batch_norm): Batch normalization.

[in\_top\_k(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/math/in_top_k): Says whether the targets are in the top K predictions.

[l2\_loss(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/l2_loss): L2 Loss.

[l2\_normalize(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/linalg/l2_normalize): Normalizes along dimension axis using an L2 norm. (deprecated arguments)

[leaky\_relu(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/leaky_relu): Compute the Leaky ReLU activation function.

[learned\_unigram\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/learned_unigram_candidate_sampler): Samples a set of classes from a distribution learned during training.

[local\_response\_normalization(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/local_response_normalization): Local Response Normalization.

[log\_poisson\_loss(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/log_poisson_loss): Computes log Poisson loss given log\_input.

[log\_softmax(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/math/log_softmax): Computes log softmax activations. (deprecated arguments)

[log\_uniform\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/log_uniform_candidate_sampler): Samples a set of classes using a log-uniform (Zipfian) base distribution.

[lrn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/local_response_normalization): Local Response Normalization.

[max\_pool(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/max_pool): Performs the max pooling on the input.

[max\_pool1d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool1d): Performs the max pooling on the input.

[max\_pool2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool2d): Performs the max pooling on the input.

[max\_pool3d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool3d): Performs the max pooling on the input.

[max\_pool\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool): Performs the max pooling on the input.

[max\_pool\_with\_argmax(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/max_pool_with_argmax): Performs max pooling on the input and outputs both max values and indices.

[moments(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/moments): Calculate the mean and variance of x.

[nce\_loss(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/nce_loss): Computes and returns the noise-contrastive estimation training loss.

[normalize\_moments(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/normalize_moments): Calculate the mean and variance of based on the sufficient statistics.

[pool(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/pool): Performs an N-D pooling operation.

[quantized\_avg\_pool(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/quantized_avg_pool): Produces the average pool of the input tensor for quantized types.

[quantized\_conv2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/quantized_conv2d): Computes a 2D convolution given quantized 4D input and filter tensors.

[quantized\_max\_pool(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/quantized_max_pool): Produces the max pool of the input tensor for quantized types.

[quantized\_relu\_x(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/quantized_relu_x): Computes Quantized Rectified Linear X: min(max(features, 0), max\_value)

[raw\_rnn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/raw_rnn): Creates an RNN specified by RNNCell cell and loop function loop\_fn.

[relu(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/relu): Computes rectified linear: max(features, 0).

[relu6(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/relu6): Computes Rectified Linear 6: min(max(features, 0), 6).

[relu\_layer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/relu_layer): Computes Relu(x \* weight + biases).

[safe\_embedding\_lookup\_sparse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/safe_embedding_lookup_sparse): Lookup embedding results, accounting for invalid IDs and empty features.

[sampled\_softmax\_loss(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/sampled_softmax_loss): Computes and returns the sampled softmax training loss.

[scale\_regularization\_loss(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/scale_regularization_loss): Scales the sum of the given regularization losses by number of replicas.

[selu(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/selu): Computes scaled exponential linear: scale \* alpha \* (exp(features) - 1)

[separable\_conv2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/separable_conv2d): 2-D convolution with separable filters.

[sigmoid(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sigmoid): Computes sigmoid of x element-wise.

[sigmoid\_cross\_entropy\_with\_logits(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/sigmoid_cross_entropy_with_logits): Computes sigmoid cross entropy given logits.

[softmax(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/math/softmax): Computes softmax activations. (deprecated arguments)

[softmax\_cross\_entropy\_with\_logits(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/softmax_cross_entropy_with_logits): Computes softmax cross entropy between logitsand labels. (deprecated)

[softmax\_cross\_entropy\_with\_logits\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/softmax_cross_entropy_with_logits_v2): Computes softmax cross entropy between logits and labels. (deprecated arguments)

[softplus(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/softplus): Computes softplus: log(exp(features) + 1).

[softsign(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softsign): Computes softsign: features / (abs(features) + 1).

[space\_to\_batch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/space_to_batch): SpaceToBatch for 4-D tensors of type T.

[space\_to\_depth(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/space_to_depth): SpaceToDepth for tensors of type T.

[sparse\_softmax\_cross\_entropy\_with\_logits(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/sparse_softmax_cross_entropy_with_logits): Computes sparse softmax cross entropy between logits and labels.

[static\_bidirectional\_rnn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/static_bidirectional_rnn): Creates a bidirectional recurrent neural network. (deprecated)

[static\_rnn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/static_rnn): Creates a recurrent neural network specified by RNNCell cell. (deprecated)

[static\_state\_saving\_rnn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/static_state_saving_rnn): RNN that accepts a state saver for time-truncated RNN calculation. (deprecated)

[sufficient\_statistics(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/sufficient_statistics): Calculate the sufficient statistics for the mean and variance of x.

[tanh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/tanh): Computes hyperbolic tangent of x element-wise.

[top\_k(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/top_k): Finds values and indices of the k largest entries for the last dimension.

[uniform\_candidate\_sampler(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform_candidate_sampler): Samples a set of classes using a uniform base distribution.

[weighted\_cross\_entropy\_with\_logits(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/weighted_cross_entropy_with_logits): Computes a weighted cross entropy. (deprecated arguments)

[weighted\_moments(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/weighted_moments): Returns the frequency-weighted mean and variance of x.

[with\_space\_to\_batch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/with_space_to_batch): Performs op on the space-to-batch representation of input.

[xw\_plus\_b(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn/xw_plus_b): Computes matmul(x, weights) + biases.

[zero\_fraction(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/zero_fraction): Returns the fraction of zeros in value.

Other Members

* swish

# tf.nn.atrous\_conv2d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/atrous_conv2d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/atrous_conv2d#aliases)

Atrous convolution (a.k.a. convolution with holes or dilated convolution).

### Aliases:

* tf.compat.v1.nn.atrous\_conv2d
* tf.compat.v2.nn.atrous\_conv2d
* tf.nn.atrous\_conv2d

tf.nn.atrous\_conv2d(  
    value,  
    filters,  
    rate,  
    padding,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This function is a simpler wrapper around the more general [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution), and exists only for backwards compatibility. You can use [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) to perform 1-D, 2-D, or 3-D atrous convolution.

Computes a 2-D atrous convolution, also known as convolution with holes or dilated convolution, given 4-D value and filters tensors. If the rate parameter is equal to one, it performs regular 2-D convolution. If the rate parameter is greater than one, it performs convolution with holes, sampling the input values every rate pixels in the height and width dimensions. This is equivalent to convolving the input with a set of upsampled filters, produced by inserting rate - 1 zeros between two consecutive values of the filters along the height and width dimensions, hence the name atrous convolution or convolution with holes (the French word trous means holes in English).

#### More specifically:

output[batch, height, width, out\_channel] =  
    sum\_{dheight, dwidth, in\_channel} (  
        filters[dheight, dwidth, in\_channel, out\_channel] \*  
        value[batch, height + rate\*dheight, width + rate\*dwidth, in\_channel]  
    )

Atrous convolution allows us to explicitly control how densely to compute feature responses in fully convolutional networks. Used in conjunction with bilinear interpolation, it offers an alternative to conv2d\_transpose in dense prediction tasks such as semantic image segmentation, optical flow computation, or depth estimation. It also allows us to effectively enlarge the field of view of filters without increasing the number of parameters or the amount of computation.

For a description of atrous convolution and how it can be used for dense feature extraction, please see: [Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs](http://arxiv.org/abs/1412.7062). The same operation is investigated further in [Multi-Scale Context Aggregation by Dilated Convolutions](http://arxiv.org/abs/1511.07122). Previous works that effectively use atrous convolution in different ways are, among others, [OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks](http://arxiv.org/abs/1312.6229) and [Fast Image Scanning with Deep Max-Pooling Convolutional Neural Networks](http://arxiv.org/abs/1302.1700). Atrous convolution is also closely related to the so-called noble identities in multi-rate signal processing.

There are many different ways to implement atrous convolution (see the refs above). The implementation here reduces

    atrous\_conv2d(value, filters, rate, padding=padding)

to the following three operations:

    paddings = ...  
    net = space\_to\_batch(value, paddings, block\_size=rate)  
    net = conv2d(net, filters, strides=[1, 1, 1, 1], padding="VALID")  
    crops = ...  
    net = batch\_to\_space(net, crops, block\_size=rate)

Advanced usage. Note the following optimization: A sequence of atrous\_conv2d operations with identical rate parameters, 'SAME' padding, and filters with odd heights/ widths:

    net = atrous\_conv2d(net, filters1, rate, padding="SAME")  
    net = atrous\_conv2d(net, filters2, rate, padding="SAME")  
    ...  
    net = atrous\_conv2d(net, filtersK, rate, padding="SAME")

can be equivalently performed cheaper in terms of computation and memory as:

    pad = ...  # padding so that the input dims are multiples of rate  
    net = space\_to\_batch(net, paddings=pad, block\_size=rate)  
    net = conv2d(net, filters1, strides=[1, 1, 1, 1], padding="SAME")  
    net = conv2d(net, filters2, strides=[1, 1, 1, 1], padding="SAME")  
    ...  
    net = conv2d(net, filtersK, strides=[1, 1, 1, 1], padding="SAME")  
    net = batch\_to\_space(net, crops=pad, block\_size=rate)

because a pair of consecutive space\_to\_batch and batch\_to\_space ops with the same block\_size cancel out when their respective paddings and crops inputs are identical.

#### Args:

* **value**: A 4-D Tensor of type float. It needs to be in the default "NHWC" format. Its shape is [batch, in\_height, in\_width, in\_channels].
* **filters**: A 4-D Tensor with the same type as value and shape [filter\_height, filter\_width, in\_channels, out\_channels]. filters' in\_channels dimension must match that of value. Atrous convolution is equivalent to standard convolution with upsampled filters with effective height filter\_height + (filter\_height - 1) \* (rate - 1) and effective width filter\_width + (filter\_width - 1) \* (rate - 1), produced by inserting rate - 1 zeros along consecutive elements across the filters' spatial dimensions.
* **rate**: A positive int32. The stride with which we sample input values across the height and width dimensions. Equivalently, the rate by which we upsample the filter values by inserting zeros across the height and width dimensions. In the literature, the same parameter is sometimes called input stride or dilation.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm.
* **name**: Optional name for the returned tensor.

#### Returns:

A Tensor with the same type as value. Output shape with 'VALID' padding is:

[batch, height - 2 \* (filter\_width - 1),  
 width - 2 \* (filter\_height - 1), out\_channels].

Output shape with 'SAME' padding is:

[batch, height, width, out\_channels].

#### Raises:

# ValueError: If input/output depth does not match filters' shape, or if padding is other than 'VALID' or 'SAME'. tf.nn.atrous\_conv2d\_transpose

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/atrous_conv2d_transpose#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/atrous_conv2d_transpose#aliases)

The transpose of atrous\_conv2d.

### Aliases:

* tf.compat.v1.nn.atrous\_conv2d\_transpose
* tf.compat.v2.nn.atrous\_conv2d\_transpose
* tf.nn.atrous\_conv2d\_transpose

tf.nn.atrous\_conv2d\_transpose(  
    value,  
    filters,  
    output\_shape,  
    rate,  
    padding,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This operation is sometimes called "deconvolution" after [Deconvolutional Networks](https://www.matthewzeiler.com/mattzeiler/deconvolutionalnetworks.pdf), but is really the transpose (gradient) of atrous\_conv2d rather than an actual deconvolution.

#### Args:

* **value**: A 4-D Tensor of type float. It needs to be in the default NHWC format. Its shape is [batch, in\_height, in\_width, in\_channels].
* **filters**: A 4-D Tensor with the same type as value and shape [filter\_height, filter\_width, out\_channels, in\_channels]. filters' in\_channels dimension must match that of value. Atrous convolution is equivalent to standard convolution with upsampled filters with effective height filter\_height + (filter\_height - 1) \* (rate - 1) and effective width filter\_width + (filter\_width - 1) \* (rate - 1), produced by inserting rate - 1 zeros along consecutive elements across the filters' spatial dimensions.
* **output\_shape**: A 1-D Tensor of shape representing the output shape of the deconvolution op.
* **rate**: A positive int32. The stride with which we sample input values across the height and width dimensions. Equivalently, the rate by which we upsample the filter values by inserting zeros across the height and width dimensions. In the literature, the same parameter is sometimes called input stride or dilation.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm.
* **name**: Optional name for the returned tensor.

#### Returns:

A Tensor with the same type as value.

#### Raises:

* **ValueError**: If input/output depth does not match filters' shape, or if padding is other than 'VALID' or 'SAME', or if the rate is less than one, or if the output\_shape is not a tensor with 4 elements.

# tf.nn.avg\_pool

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool#aliases)

Performs the avg pooling on the input.

### Aliases:

* tf.compat.v1.nn.avg\_pool\_v2
* tf.compat.v2.nn.avg\_pool
* tf.nn.avg\_pool

tf.nn.avg\_pool(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Each entry in output is the mean of the corresponding size ksize window in value.

#### Args:

* **input**: Tensor of rank N+2, of shape [batch\_size] + input\_spatial\_shape + [num\_channels] if data\_format does not start with "NC" (default), or [batch\_size, num\_channels] + input\_spatial\_shape if data\_format starts with "NC". Pooling happens over the spatial dimensions only.
* **ksize**: An int or list of ints that has length 1, N or N+2. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1, N or N+2. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string. Specifies the channel dimension. For N=1 it can be either "NWC" (default) or "NCW", for N=2 it can be either "NHWC" (default) or "NCHW" and for N=3 either "NDHWC" (default) or "NCDHW".
* **name**: Optional name for the operation.

#### Returns:

A Tensor of format specified by data\_format. The average pooled output tensor.

# tf.nn.avg\_pool1d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool1d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool1d#aliases)

Performs the average pooling on the input.

### Aliases:

* tf.compat.v1.nn.avg\_pool1d
* tf.compat.v2.nn.avg\_pool1d
* tf.nn.avg\_pool1d

tf.nn.avg\_pool1d(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format='NWC',  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Each entry in output is the mean of the corresponding size ksize window in value.

Note internally this op reshapes and uses the underlying 2d operation.

#### Args:

* **input**: A 3-D Tensor of the format specified by data\_format.
* **ksize**: An int or list of ints that has length 1 or 3. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1 or 3. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: An optional string from: "NWC", "NCW". Defaults to "NWC".
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of format specified by data\_format. The max pooled output tensor.

# tf.nn.avg\_pool1d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool1d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool1d#aliases)

Performs the average pooling on the input.

### Aliases:

* tf.compat.v1.nn.avg\_pool1d
* tf.compat.v2.nn.avg\_pool1d
* tf.nn.avg\_pool1d

tf.nn.avg\_pool1d(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format='NWC',  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Each entry in output is the mean of the corresponding size ksize window in value.

Note internally this op reshapes and uses the underlying 2d operation.

#### Args:

* **input**: A 3-D Tensor of the format specified by data\_format.
* **ksize**: An int or list of ints that has length 1 or 3. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1 or 3. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: An optional string from: "NWC", "NCW". Defaults to "NWC".
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of format specified by data\_format. The max pooled output tensor.

# tf.nn.avg\_pool3d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool3d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/avg_pool3d#aliases)

Performs the average pooling on the input.

### Aliases:

* tf.compat.v1.nn.avg\_pool3d
* tf.compat.v2.nn.avg\_pool3d
* tf.nn.avg\_pool3d

tf.nn.avg\_pool3d(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format='NDHWC',  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Each entry in output is the mean of the corresponding size ksize window in value.

#### Args:

* **input**: A 5-D Tensor of shape [batch, height, width, channels] and type float32, float64, qint8, quint8, or qint32.
* **ksize**: An int or list of ints that has length 1, 3 or 5. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1, 3 or 5. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string. 'NDHWC' and 'NCDHW' are supported.
* **name**: Optional name for the operation.

#### Returns:

A Tensor with the same type as value. The average pooled output tensor.

# tf.nn.batch\_normalization

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/batch_normalization#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/batch_normalization#aliases)

Batch normalization.

### Aliases:

* tf.compat.v1.nn.batch\_normalization
* tf.compat.v2.nn.batch\_normalization
* tf.nn.batch\_normalization

tf.nn.batch\_normalization(  
    x,  
    mean,  
    variance,  
    offset,  
    scale,  
    variance\_epsilon,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

Normalizes a tensor by mean and variance, and applies (optionally) a scale γ to it, as well as an offset β:

γ(x−μ)σ+β

mean, variance, offset and scale are all expected to be of one of two shapes:

* In all generality, they can have the same number of dimensions as the input x, with identical sizes as x for the dimensions that are not normalized over (the 'depth' dimension(s)), and dimension 1 for the others which are being normalized over. mean and variance in this case would typically be the outputs of tf.nn.moments(..., keep\_dims=True) during training, or running averages thereof during inference.
* In the common case where the 'depth' dimension is the last dimension in the input tensor x, they may be one dimensional tensors of the same size as the 'depth' dimension. This is the case for example for the common [batch, depth] layout of fully-connected layers, and [batch, height, width, depth] for convolutions. mean and variance in this case would typically be the outputs of tf.nn.moments(..., keep\_dims=False) during training, or running averages thereof during inference.

See Source: [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift; S. Ioffe, C. Szegedy](http://arxiv.org/abs/1502.03167).

#### Args:

* **x**: Input Tensor of arbitrary dimensionality.
* **mean**: A mean Tensor.
* **variance**: A variance Tensor.
* **offset**: An offset Tensor, often denoted β in equations, or None. If present, will be added to the normalized tensor.
* **scale**: A scale Tensor, often denoted γ in equations, or None. If present, the scale is applied to the normalized tensor.
* **variance\_epsilon**: A small float number to avoid dividing by 0.
* **name**: A name for this operation (optional).

#### Returns:

the normalized, scaled, offset tensor.

# tf.nn.batch\_norm\_with\_global\_normalization

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/batch_norm_with_global_normalization#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/batch_norm_with_global_normalization#aliases)

Batch normalization.

### Aliases:

* tf.compat.v2.nn.batch\_norm\_with\_global\_normalization
* tf.nn.batch\_norm\_with\_global\_normalization

tf.nn.batch\_norm\_with\_global\_normalization(  
    input,  
    mean,  
    variance,  
    beta,  
    gamma,  
    variance\_epsilon,  
    scale\_after\_normalization,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

This op is deprecated. See [tf.nn.batch\_normalization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/batch_normalization).

#### Args:

* **input**: A 4D input Tensor.
* **mean**: A 1D mean Tensor with size matching the last dimension of t. This is the first output from tf.nn.moments, or a saved moving average thereof.
* **variance**: A 1D variance Tensor with size matching the last dimension of t. This is the second output from tf.nn.moments, or a saved moving average thereof.
* **beta**: A 1D beta Tensor with size matching the last dimension of t. An offset to be added to the normalized tensor.
* **gamma**: A 1D gamma Tensor with size matching the last dimension of t. If "scale\_after\_normalization" is true, this tensor will be multiplied with the normalized tensor.
* **variance\_epsilon**: A small float number to avoid dividing by 0.
* **scale\_after\_normalization**: A bool indicating whether the resulted tensor needs to be multiplied with gamma.
* **name**: A name for this operation (optional).

#### Returns:

A batch-normalized t.

# tf.nn.bias\_add

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/bias_add#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/bias_add#aliases)

Adds bias to value.

### Aliases:

* tf.compat.v1.nn.bias\_add
* tf.compat.v2.nn.bias\_add
* tf.nn.bias\_add

tf.nn.bias\_add(  
    value,  
    bias,  
    data\_format=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This is (mostly) a special case of [tf.add](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add) where bias is restricted to 1-D. Broadcasting is supported, so value may have any number of dimensions. Unlike [tf.add](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add), the type of bias is allowed to differ from value in the case where both types are quantized.

#### Args:

* **value**: A Tensor with type float, double, int64, int32, uint8, int16, int8, complex64, or complex128.
* **bias**: A 1-D Tensor with size matching the last dimension of value. Must be the same type as value unless value is a quantized type, in which case a different quantized type may be used.
* **data\_format**: A string. 'N...C' and 'NC...' are supported.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor with the same type as value.

# tf.nn.collapse\_repeated

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/collapse_repeated#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/collapse_repeated#aliases)

Merge repeated labels into single labels.

### Aliases:

* tf.compat.v1.nn.collapse\_repeated
* tf.compat.v2.nn.collapse\_repeated
* tf.nn.collapse\_repeated

tf.nn.collapse\_repeated(  
    labels,  
    seq\_length,  
    name=None  
)

Defined in [python/ops/ctc\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/ctc_ops.py).

#### Args:

* **labels**: Tensor of shape [batch, max value in seq\_length]
* **seq\_length**: Tensor of shape [batch], sequence length of each batch element.
* **name**: A name for this Op. Defaults to "collapse\_repeated\_labels".

#### Returns:

A tuple (collapsed\_labels, new\_seq\_length) where

* **collapsed\_labels**: Tensor of shape [batch, max\_seq\_length] with repeated labels collapsed and padded to max\_seq\_length, eg: [[A, A, B, B, A], [A, B, C, D, E]] => [[A, B, A, 0, 0], [A, B, C, D, E]]
* **new\_seq\_length**: int tensor of shape [batch] with new sequence lengths.

# tf.nn.compute\_accidental\_hits

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_accidental_hits#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_accidental_hits#aliases)

Compute the position ids in sampled\_candidates matching true\_classes.

### Aliases:

* tf.compat.v1.nn.compute\_accidental\_hits
* tf.compat.v2.nn.compute\_accidental\_hits
* tf.nn.compute\_accidental\_hits

tf.nn.compute\_accidental\_hits(  
    true\_classes,  
    sampled\_candidates,  
    num\_true,  
    seed=None,  
    name=None  
)

Defined in [python/ops/candidate\_sampling\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/candidate_sampling_ops.py).

In Candidate Sampling, this operation facilitates virtually removing sampled classes which happen to match target classes. This is done in Sampled Softmax and Sampled Logistic.

See our [Candidate Sampling Algorithms Reference](http://www.tensorflow.org/extras/candidate_sampling.pdf).

We presuppose that the sampled\_candidates are unique.

We call it an 'accidental hit' when one of the target classes matches one of the sampled classes. This operation reports accidental hits as triples (index, id, weight), where index represents the row number in true\_classes, id represents the position in sampled\_candidates, and weight is -FLOAT\_MAX.

The result of this op should be passed through a sparse\_to\_dense operation, then added to the logits of the sampled classes. This removes the contradictory effect of accidentally sampling the true target classes as noise classes for the same example.

#### Args:

* **true\_classes**: A Tensor of type int64 and shape [batch\_size, num\_true]. The target classes.
* **sampled\_candidates**: A tensor of type int64 and shape [num\_sampled]. The sampled\_candidates output of CandidateSampler.
* **num\_true**: An int. The number of target classes per training example.
* **seed**: An int. An operation-specific seed. Default is 0.
* **name**: A name for the operation (optional).

#### Returns:

* **indices**: A Tensor of type int32 and shape [num\_accidental\_hits]. Values indicate rows in true\_classes.
* **ids**: A Tensor of type int64 and shape [num\_accidental\_hits]. Values indicate positions in sampled\_candidates.
* **weights**: A Tensor of type float and shape [num\_accidental\_hits]. Each value is -FLOAT\_MAX.

# tf.nn.compute\_average\_loss

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_average_loss#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_average_loss#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_average_loss#used_in_the_tutorials)

Scales per-example losses with sample\_weights and computes their average.

### Aliases:

* tf.compat.v1.nn.compute\_average\_loss
* tf.compat.v2.nn.compute\_average\_loss
* tf.nn.compute\_average\_loss

tf.nn.compute\_average\_loss(  
    per\_example\_loss,  
    sample\_weight=None,  
    global\_batch\_size=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

### Used in the tutorials:

* [tf.distribute.Strategy with training loops](https://www.tensorflow.org/beta/tutorials/distribute/training_loops)

Usage with distribution strategy and custom training loop:

with strategy.scope():  
  def compute\_loss(labels, predictions, sample\_weight=None):  
  
    # If you are using a `Loss` class instead, set reduction to `NONE` so that  
    # we can do the reduction afterwards and divide by global batch size.  
    per\_example\_loss = tf.keras.losses.sparse\_categorical\_crossentropy(  
        labels, predictions)  
  
    # Compute loss that is scaled by sample\_weight and by global batch size.  
    return tf.compute\_average\_loss(  
        per\_example\_loss,  
        sample\_weight=sample\_weight,  
        global\_batch\_size=GLOBAL\_BATCH\_SIZE)

#### Args:

* **per\_example\_loss**: Per-example loss.
* **sample\_weight**: Optional weighting for each example.
* **global\_batch\_size**: Optional global batch size value. Defaults to (size of first dimension of losses) \* (number of replicas).

#### Returns:

Scalar loss value.

# tf.nn.compute\_average\_loss

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_average_loss#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_average_loss#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/compute_average_loss#used_in_the_tutorials)

Scales per-example losses with sample\_weights and computes their average.

### Aliases:

* tf.compat.v1.nn.compute\_average\_loss
* tf.compat.v2.nn.compute\_average\_loss
* tf.nn.compute\_average\_loss

tf.nn.compute\_average\_loss(  
    per\_example\_loss,  
    sample\_weight=None,  
    global\_batch\_size=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

### Used in the tutorials:

* [tf.distribute.Strategy with training loops](https://www.tensorflow.org/beta/tutorials/distribute/training_loops)

Usage with distribution strategy and custom training loop:

with strategy.scope():  
  def compute\_loss(labels, predictions, sample\_weight=None):  
  
    # If you are using a `Loss` class instead, set reduction to `NONE` so that  
    # we can do the reduction afterwards and divide by global batch size.  
    per\_example\_loss = tf.keras.losses.sparse\_categorical\_crossentropy(  
        labels, predictions)  
  
    # Compute loss that is scaled by sample\_weight and by global batch size.  
    return tf.compute\_average\_loss(  
        per\_example\_loss,  
        sample\_weight=sample\_weight,  
        global\_batch\_size=GLOBAL\_BATCH\_SIZE)

#### Args:

* **per\_example\_loss**: Per-example loss.
* **sample\_weight**: Optional weighting for each example.
* **global\_batch\_size**: Optional global batch size value. Defaults to (size of first dimension of losses) \* (number of replicas).

#### Returns:

Scalar loss value.

# tf.nn.conv1d\_transpose

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv1d_transpose#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv1d_transpose#aliases)

The transpose of conv1d.

### Aliases:

* tf.compat.v1.nn.conv1d\_transpose
* tf.compat.v2.nn.conv1d\_transpose
* tf.nn.conv1d\_transpose

tf.nn.conv1d\_transpose(  
    input,  
    filters,  
    output\_shape,  
    strides,  
    padding='SAME',  
    data\_format='NWC',  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This operation is sometimes called "deconvolution" after [Deconvolutional Networks](https://www.matthewzeiler.com/mattzeiler/deconvolutionalnetworks.pdf), but is really the transpose (gradient) of conv1d rather than an actual deconvolution.

#### Args:

* **input**: A 3-D Tensor of type float and shape [batch, in\_width, in\_channels] for NWCdata format or [batch, in\_channels, in\_width] for NCW data format.
* **filters**: A 3-D Tensor with the same type as value and shape [filter\_width, output\_channels, in\_channels]. filter's in\_channels dimension must match that of value.
* **output\_shape**: A 1-D Tensor, containing three elements, representing the output shape of the deconvolution op.
* **strides**: An int or list of ints that has length 1 or 3. The number of entries by which the filter is moved right at each step.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string. 'NWC' and 'NCW' are supported.
* **dilations**: An int or list of ints that has length 1 or 3 which defaults to 1. The dilation factor for each dimension of input. If set to k > 1, there will be k-1 skipped cells between each filter element on that dimension. Dilations in the batch and depth dimensions must be 1.
* **name**: Optional name for the returned tensor.

#### Returns:

A Tensor with the same type as value.

#### Raises:

* **ValueError**: If input/output depth does not match filter's shape, if output\_shape is not at 3-element vector, if padding is other than 'VALID' or 'SAME', or if data\_format is invalid.

# tf.nn.conv2d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d#aliases)

Computes a 2-D convolution given 4-D input and filters tensors.

### Aliases:

* tf.compat.v2.nn.conv2d
* tf.nn.conv2d

tf.nn.conv2d(  
    input,  
    filters,  
    strides,  
    padding,  
    data\_format='NHWC',  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Given an input tensor of shape [batch, in\_height, in\_width, in\_channels] and a filter / kernel tensor of shape [filter\_height, filter\_width, in\_channels, out\_channels], this op performs the following:

1. Flattens the filter to a 2-D matrix with shape [filter\_height \* filter\_width \* in\_channels, output\_channels].
2. Extracts image patches from the input tensor to form a virtual tensor of shape [batch, out\_height, out\_width, filter\_height \* filter\_width \* in\_channels].
3. For each patch, right-multiplies the filter matrix and the image patch vector.

In detail, with the default NHWC format,

output[b, i, j, k] =  
    sum\_{di, dj, q} input[b, strides[1] \* i + di, strides[2] \* j + dj, q] \*  
                    filter[di, dj, q, k]

Must have strides[0] = strides[3] = 1. For the most common case of the same horizontal and vertices strides, strides = [1, stride, stride, 1].

#### Args:

* **input**: A Tensor. Must be one of the following types: half, bfloat16, float32, float64. A 4-D tensor. The dimension order is interpreted according to the value of data\_format, see below for details.
* **filters**: A Tensor. Must have the same type as input. A 4-D tensor of shape[filter\_height, filter\_width, in\_channels, out\_channels]
* **strides**: An int or list of ints that has length 1, 2 or 4. The stride of the sliding window for each dimension of input. If a single value is given it is replicated in the H and W dimension. By default the N and C dimensions are set to 1. The dimension order is determined by the value of data\_format, see below for details.
* **padding**: Either the string "SAME" or "VALID" indicating the type of padding algorithm to use, or a list indicating the explicit paddings at the start and end of each dimension. When explicit padding is used and data\_format is "NHWC", this should be in the form [[0, 0], [pad\_top, pad\_bottom], [pad\_left, pad\_right], [0, 0]]. When explicit padding used and data\_format is "NCHW", this should be in the form [[0, 0], [0, 0], [pad\_top, pad\_bottom], [pad\_left, pad\_right]].
* **data\_format**: An optional string from: "NHWC", "NCHW". Defaults to "NHWC". Specify the data format of the input and output data. With the default format "NHWC", the data is stored in the order of: [batch, height, width, channels]. Alternatively, the format could be "NCHW", the data storage order of: [batch, channels, height, width].
* **dilations**: An int or list of ints that has length 1, 2 or 4, defaults to 1. The dilation factor for each dimension ofinput. If a single value is given it is replicated in the H and W dimension. By default the N and C dimensions are set to 1. If set to k > 1, there will be k-1 skipped cells between each filter element on that dimension. The dimension order is determined by the value of data\_format, see above for details. Dilations in the batch and depth dimensions if a 4-d tensor must be 1.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.nn.conv2d\_transpose

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d_transpose#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d_transpose#aliases)

The transpose of conv2d.

### Aliases:

* tf.compat.v2.nn.conv2d\_transpose
* tf.nn.conv2d\_transpose

tf.nn.conv2d\_transpose(  
    input,  
    filters,  
    output\_shape,  
    strides,  
    padding='SAME',  
    data\_format='NHWC',  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This operation is sometimes called "deconvolution" after [Deconvolutional Networks](http://www.matthewzeiler.com/pubs/cvpr2010/cvpr2010.pdf), but is actually the transpose (gradient) of conv2d rather than an actual deconvolution.

#### Args:

* **input**: A 4-D Tensor of type float and shape [batch, height, width, in\_channels] for NHWC data format or [batch, in\_channels, height, width] for NCHW data format.
* **filters**: A 4-D Tensor with the same type as input and shape [height, width, output\_channels, in\_channels]. filter's in\_channels dimension must match that of input.
* **output\_shape**: A 1-D Tensor representing the output shape of the deconvolution op.
* **strides**: An int or list of ints that has length 1, 2 or 4. The stride of the sliding window for each dimension of input. If a single value is given it is replicated in the H and W dimension. By default the N and C dimensions are set to 0. The dimension order is determined by the value of data\_format, see below for details.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string. 'NHWC' and 'NCHW' are supported.
* **dilations**: An int or list of ints that has length 1, 2 or 4, defaults to 1. The dilation factor for each dimension ofinput. If a single value is given it is replicated in the H and W dimension. By default the N and C dimensions are set to 1. If set to k > 1, there will be k-1 skipped cells between each filter element on that dimension. The dimension order is determined by the value of data\_format, see above for details. Dilations in the batch and depth dimensions if a 4-d tensor must be 1.
* **name**: Optional name for the returned tensor.

#### Returns:

A Tensor with the same type as input.

#### Raises:

* **ValueError**: If input/output depth does not match filter's shape, or if padding is other than 'VALID' or 'SAME'.

# tf.nn.conv3d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv3d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv3d#aliases)

Computes a 3-D convolution given 5-D input and filters tensors.

### Aliases:

* tf.compat.v2.nn.conv3d
* tf.nn.conv3d

tf.nn.conv3d(  
    input,  
    filters,  
    strides,  
    padding,  
    data\_format='NDHWC',  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

In signal processing, cross-correlation is a measure of similarity of two waveforms as a function of a time-lag applied to one of them. This is also known as a sliding dot product or sliding inner-product.

Our Conv3D implements a form of cross-correlation.

#### Args:

* **input**: A Tensor. Must be one of the following types: half, bfloat16, float32, float64. Shape [batch, in\_depth, in\_height, in\_width, in\_channels].
* **filters**: A Tensor. Must have the same type as input. Shape [filter\_depth, filter\_height, filter\_width, in\_channels, out\_channels]. in\_channels must match between input and filters.
* **strides**: A list of ints that has length >= 5. 1-D tensor of length 5. The stride of the sliding window for each dimension of input. Must have strides[0] = strides[4] = 1.
* **padding**: A string from: "SAME", "VALID". The type of padding algorithm to use.
* **data\_format**: An optional string from: "NDHWC", "NCDHW". Defaults to "NDHWC". The data format of the input and output data. With the default format "NDHWC", the data is stored in the order of: [batch, in\_depth, in\_height, in\_width, in\_channels]. Alternatively, the format could be "NCDHW", the data storage order is: [batch, in\_channels, in\_depth, in\_height, in\_width].
* **dilations**: An optional list of ints. Defaults to [1, 1, 1, 1, 1]. 1-D tensor of length 5. The dilation factor for each dimension of input. If set to k > 1, there will be k-1 skipped cells between each filter element on that dimension. The dimension order is determined by the value of data\_format, see above for details. Dilations in the batch and depth dimensions must be 1.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.nn.conv3d\_transpose

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv3d_transpose#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv3d_transpose#aliases)

The transpose of conv3d.

### Aliases:

* tf.compat.v2.nn.conv3d\_transpose
* tf.nn.conv3d\_transpose

tf.nn.conv3d\_transpose(  
    input,  
    filters,  
    output\_shape,  
    strides,  
    padding='SAME',  
    data\_format='NDHWC',  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This operation is sometimes called "deconvolution" after [Deconvolutional Networks](http://www.matthewzeiler.com/pubs/cvpr2010/cvpr2010.pdf), but is actually the transpose (gradient) of conv2d rather than an actual deconvolution.

#### Args:

* **input**: A 5-D Tensor of type float and shape [batch, height, width, in\_channels] for NHWC data format or [batch, in\_channels, height, width] for NCHW data format.
* **filters**: A 5-D Tensor with the same type as value and shape [height, width, output\_channels, in\_channels]. filter's in\_channels dimension must match that of value.
* **output\_shape**: A 1-D Tensor representing the output shape of the deconvolution op.
* **strides**: An int or list of ints that has length 1, 3 or 5. The stride of the sliding window for each dimension of input. If a single value is given it is replicated in the D, H and W dimension. By default the N and C dimensions are set to 0. The dimension order is determined by the value of data\_format, see below for details.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string. 'NDHWC' and 'NCDHW' are supported.
* **dilations**: An int or list of ints that has length 1, 3 or 5, defaults to 1. The dilation factor for each dimension ofinput. If a single value is given it is replicated in the D, H and Wdimension. By default the N and C dimensions are set to 1. If set to k > 1, there will be k-1 skipped cells between each filter element on that dimension. The dimension order is determined by the value of data\_format, see above for details. Dilations in the batch and depth dimensions if a 5-d tensor must be 1.
* **name**: Optional name for the returned tensor.

#### Returns:

A Tensor with the same type as value.

# tf.nn.convolution

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution#aliases)

Computes sums of N-D convolutions (actually cross-correlation).

### Aliases:

* tf.compat.v2.nn.convolution
* tf.nn.convolution

tf.nn.convolution(  
    input,  
    filters,  
    strides=None,  
    padding='VALID',  
    data\_format=None,  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This also supports either output striding via the optional strides parameter or atrous convolution (also known as convolution with holes or dilated convolution, based on the French word "trous" meaning holes in English) via the optional dilations parameter. Currently, however, output striding is not supported for atrous convolutions.

Specifically, in the case that data\_format does not start with "NC", given a rank (N+2) input Tensor of shape

[num\_batches, input\_spatial\_shape[0], ..., input\_spatial\_shape[N-1], num\_input\_channels],

a rank (N+2) filters Tensor of shape

[spatial\_filter\_shape[0], ..., spatial\_filter\_shape[N-1], num\_input\_channels, num\_output\_channels],

an optional dilations tensor of shape [N](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/defaulting%20to%20%5B1%5D*N) specifying the filter upsampling/input downsampling rate, and an optional list of N strides (defaulting [1]\*N), this computes for each N-D spatial output position (x[0], ..., x[N-1]):

  output[b, x[0], ..., x[N-1], k] =  
      sum\_{z[0], ..., z[N-1], q}  
          filter[z[0], ..., z[N-1], q, k] \*  
          padded\_input[b,  
                       x[0]\*strides[0] + dilation\_rate[0]\*z[0],  
                       ...,  
                       x[N-1]\*strides[N-1] + dilation\_rate[N-1]\*z[N-1],  
                       q]

where b is the index into the batch, k is the output channel number, q is the input channel number, and z is the N-D spatial offset within the filter. Here, padded\_input is obtained by zero padding the input using an effective spatial filter shape of (spatial\_filter\_shape-1) \* dilation\_rate + 1 and output striding strides as described in the [comment here](https://tensorflow.org/api_guides/python/nn#Convolution).

In the case that data\_format does start with "NC", the input and output (but not the filters) are simply transposed as follows:

convolution(input, data\_format, \*\*kwargs) = tf.transpose(convolution(tf.transpose(input, [0] + range(2,N+2) + [1]), \*\*kwargs), [0, N+1] + range(1, N+1))

It is required that 1 <= N <= 3.

#### Args:

* **input**: An (N+2)-D Tensor of type T, of shape [batch\_size] + input\_spatial\_shape + [in\_channels] if data\_format does not start with "NC" (default), or [batch\_size, in\_channels] + input\_spatial\_shape if data\_format starts with "NC".
* **filters**: An (N+2)-D Tensor with the same type as input and shapespatial\_filter\_shape + [in\_channels, out\_channels].
* **padding**: A string, either "VALID" or "SAME". The padding algorithm.
* **strides**: Optional. Sequence of N ints >= 1. Specifies the output stride. Defaults to [1]\*N. If any value of strides is > 1, then all values of dilation\_rate must be 1.
* **dilations**: Optional. Sequence of N ints >= 1. Specifies the filter upsampling/input downsampling rate. In the literature, the same parameter is sometimes called input stride or dilation. The effective filter size used for the convolution will be spatial\_filter\_shape + (spatial\_filter\_shape - 1) \* (rate - 1), obtained by inserting (dilation\_rate[i]-1) zeros between consecutive elements of the original filter in each spatial dimension i. If any value of dilation\_rate is > 1, then all values of strides must be 1.
* **name**: Optional name for the returned tensor.
* **data\_format**: A string or None. Specifies whether the channel dimension of the input and output is the last dimension (default, or if data\_format does not start with "NC"), or the second dimension (if data\_format starts with "NC"). For N=1, the valid values are "NWC" (default) and "NCW". For N=2, the valid values are "NHWC" (default) and "NCHW". For N=3, the valid values are "NDHWC" (default) and "NCDHW".
* **filters**: Alias of filter.
* **dilations**: Alias of dilation\_rate.

#### Returns:

A Tensor with the same type as input of shape

`[batch\_size] + output\_spatial\_shape + [out\_channels]`

if data\_format is None or does not start with "NC", or

`[batch\_size, out\_channels] + output\_spatial\_shape`

if data\_format starts with "NC", where output\_spatial\_shape depends on the value of padding.

If padding == "SAME": output\_spatial\_shape[i] = ceil(input\_spatial\_shape[i] / strides[i])

If padding == "VALID": output\_spatial\_shape[i] = ceil((input\_spatial\_shape[i] - (spatial\_filter\_shape[i]-1) \* dilation\_rate[i]) / strides[i]).

#### Raises:

* **ValueError**: If input/output depth does not match filters shape, if padding is other than "VALID" or "SAME", or if data\_format is invalid.

# tf.nn.conv\_transpose

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv_transpose#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv_transpose#aliases)

The transpose of convolution.

### Aliases:

* tf.compat.v1.nn.conv\_transpose
* tf.compat.v2.nn.conv\_transpose
* tf.nn.conv\_transpose

tf.nn.conv\_transpose(  
    input,  
    filters,  
    output\_shape,  
    strides,  
    padding='SAME',  
    data\_format=None,  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This operation is sometimes called "deconvolution" after [Deconvolutional Networks](http://www.matthewzeiler.com/pubs/cvpr2010/cvpr2010.pdf), but is actually the transpose (gradient) of convolution rather than an actual deconvolution.

#### Args:

* **input**: An N+2 dimensional Tensor of shape [batch\_size] + input\_spatial\_shape + [in\_channels] if data\_format does not start with "NC" (default), or [batch\_size, in\_channels] + input\_spatial\_shape if data\_format starts with "NC". It must be one of the following types: half, bfloat16, float32, float64.
* **filters**: An N+2 dimensional Tensor with the same type as input and shape spatial\_filter\_shape + [in\_channels, out\_channels].
* **output\_shape**: A 1-D Tensor representing the output shape of the deconvolution op.
* **strides**: An int or list of ints that has length 1, N or N+2. The stride of the sliding window for each dimension of input. If a single value is given it is replicated in the spatial dimensions. By default the N and C dimensions are set to 0. The dimension order is determined by the value of data\_format, see below for details.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string or None. Specifies whether the channel dimension of the input and output is the last dimension (default, or if data\_format does not start with "NC"), or the second dimension (if data\_format starts with "NC"). For N=1, the valid values are "NWC" (default) and "NCW". For N=2, the valid values are "NHWC" (default) and "NCHW". For N=3, the valid values are "NDHWC" (default) and "NCDHW".
* **dilations**: An int or list of ints that has length 1, N or N+2, defaults to 1. The dilation factor for each dimension ofinput. If a single value is given it is replicated in the spatial dimensions. By default the N and C dimensions are set to 1. If set to k > 1, there will be k-1 skipped cells between each filter element on that dimension. The dimension order is determined by the value of data\_format, see above for details.
* **name**: A name for the operation (optional). If not specified "conv\_transpose" is used.

#### Returns:

A Tensor with the same type as value.

# tf.nn.crelu

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/crelu#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/crelu#aliases)

Computes Concatenated ReLU.

### Aliases:

* tf.compat.v2.nn.crelu
* tf.nn.crelu

tf.nn.crelu(  
    features,  
    axis=-1,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Concatenates a ReLU which selects only the positive part of the activation with a ReLU which selects only the negative part of the activation. Note that as a result this non-linearity doubles the depth of the activations. Source: [Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units. W. Shang, et al.](https://arxiv.org/abs/1603.05201)

#### Args:

* **features**: A Tensor with type float, double, int32, int64, uint8, int16, or int8.
* **name**: A name for the operation (optional).
* **axis**: The axis that the output values are concatenated along. Default is -1.

#### Returns:

A Tensor with the same type as features.

# tf.nn.ctc\_beam\_search\_decoder

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_beam_search_decoder#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_beam_search_decoder#aliases)

Performs beam search decoding on the logits given in input.

### Aliases:

* tf.compat.v1.nn.ctc\_beam\_search\_decoder\_v2
* tf.compat.v2.nn.ctc\_beam\_search\_decoder
* tf.nn.ctc\_beam\_search\_decoder

tf.nn.ctc\_beam\_search\_decoder(  
    inputs,  
    sequence\_length,  
    beam\_width=100,  
    top\_paths=1  
)

Defined in [python/ops/ctc\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/ctc_ops.py).

**Note** The ctc\_greedy\_decoder is a special case of the ctc\_beam\_search\_decoder with top\_paths=1 and beam\_width=1 (but that decoder is faster for this special case).

#### Args:

* **inputs**: 3-D float Tensor, size [max\_time, batch\_size, num\_classes]. The logits.
* **sequence\_length**: 1-D int32 vector containing sequence lengths, having size [batch\_size].
* **beam\_width**: An int scalar >= 0 (beam search beam width).
* **top\_paths**: An int scalar >= 0, <= beam\_width (controls output size).

#### Returns:

A tuple (decoded, log\_probabilities) where

* **decoded**: A list of length top\_paths, where decoded[j] is a SparseTensor containing the decoded outputs:

decoded[j].indices: Indices matrix [total\_decoded\_outputs[j], 2]; The rows store: [batch, time].

decoded[j].values: Values vector, size [total\_decoded\_outputs[j]]. The vector stores the decoded classes for beam j.

decoded[j].dense\_shape: Shape vector, size (2). The shape values are: [batch\_size, max\_decoded\_length[j]].

* **log\_probability**: A float matrix [batch\_size, top\_paths] containing sequence log-probabilities.

# tf.nn.ctc\_greedy\_decoder

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_greedy_decoder#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_greedy_decoder#aliases)

Performs greedy decoding on the logits given in input (best path).

### Aliases:

* tf.compat.v1.nn.ctc\_greedy\_decoder
* tf.compat.v2.nn.ctc\_greedy\_decoder
* tf.nn.ctc\_greedy\_decoder

tf.nn.ctc\_greedy\_decoder(  
    inputs,  
    sequence\_length,  
    merge\_repeated=True  
)

Defined in [python/ops/ctc\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/ctc_ops.py).

**Note:** Regardless of the value of merge\_repeated, if the maximum index of a given time and batch corresponds to the blank index **(num\_classes - 1)**, no new element is emitted.

If merge\_repeated is True, merge repeated classes in output. This means that if consecutive logits' maximum indices are the same, only the first of these is emitted. The sequence A B B \* B \* B(where '\*' is the blank label) becomes

* A B B B if merge\_repeated=True.
* A B B B B if merge\_repeated=False.

#### Args:

* **inputs**: 3-D float Tensor sized [max\_time, batch\_size, num\_classes]. The logits.
* **sequence\_length**: 1-D int32 vector containing sequence lengths, having size [batch\_size].
* **merge\_repeated**: Boolean. Default: True.

#### Returns:

A tuple (decoded, neg\_sum\_logits) where

* **decoded**: A single-element list. decoded[0] is an SparseTensor containing the decoded outputs s.t.:

decoded.indices: Indices matrix (total\_decoded\_outputs, 2). The rows store: [batch, time].

decoded.values: Values vector, size (total\_decoded\_outputs). The vector stores the decoded classes.

decoded.dense\_shape: Shape vector, size (2). The shape values are: [batch\_size, max\_decoded\_length]

* **neg\_sum\_logits**: A float matrix (batch\_size x 1) containing, for the sequence found, the negative of the sum of the greatest logit at each timeframe.

# tf.nn.ctc\_loss

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_loss#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_loss#aliases)

Computes CTC (Connectionist Temporal Classification) loss.

### Aliases:

* tf.compat.v1.nn.ctc\_loss\_v2
* tf.compat.v2.nn.ctc\_loss
* tf.nn.ctc\_loss

tf.nn.ctc\_loss(  
    labels,  
    logits,  
    label\_length,  
    logit\_length,  
    logits\_time\_major=True,  
    unique=None,  
    blank\_index=None,  
    name=None  
)

Defined in [python/ops/ctc\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/ctc_ops.py).

This op implements the CTC loss as presented in the article:

[A. Graves, S. Fernandez, F. Gomez, J. Schmidhuber. Connectionist Temporal Classification: Labeling Unsegmented Sequence Data with Recurrent Neural Networks. ICML 2006, Pittsburgh, USA, pp. 369-376.](http://www.cs.toronto.edu/~graves/icml_2006.pdf)

#### Notes:

* Same as the "Classic CTC" in TensorFlow 1.x's tf.compat.v1.nn.ctc\_loss setting of preprocess\_collapse\_repeated=False, ctc\_merge\_repeated=True
* Labels may be supplied as either a dense, zero-padded tensor with a vector of label sequence lengths OR as a SparseTensor.
* On TPU and GPU:
  + Only dense padded labels are supported.
* On CPU:
  + Caller may use SparseTensor or dense padded labels but calling with a SparseTensor will be significantly faster.
* Default blank label is 0 rather num\_classes - 1, unless overridden by blank\_index.

#### Args:

* **labels**: tensor of shape [batch\_size, max\_label\_seq\_length] or SparseTensor
* **logits**: tensor of shape [frames, batch\_size, num\_labels], if logits\_time\_major == False, shape is [batch\_size, frames, num\_labels].
* **label\_length**: tensor of shape [batch\_size], None if labels is SparseTensor Length of reference label sequence in labels.
* **logit\_length**: tensor of shape [batch\_size] Length of input sequence in logits.
* **logits\_time\_major**: (optional) If True (default), logits is shaped [time, batch, logits]. If False, shape is [batch, time, logits]
* **unique**: (optional) Unique label indices as computed by ctc\_unique\_labels(labels). If supplied, enable a faster, memory efficient implementation on TPU.
* **blank\_index**: (optional) Set the class index to use for the blank label. Negative values will start from num\_classes, ie, -1 will reproduce the ctc\_loss behavior of using num\_classes - 1 for the blank symbol. There is some memory/performance overhead to switching from the default of 0 as an additional shifted copy of the logits may be created.
* **name**: A name for this Op. Defaults to "ctc\_loss\_dense".

#### Returns:

* **loss**: tensor of shape [batch\_size], negative log probabilities.

# tf.nn.ctc\_unique\_labels

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_unique_labels#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_unique_labels#aliases)

Get unique labels and indices for batched labels for [tf.nn.ctc\_loss](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_loss).

### Aliases:

* tf.compat.v1.nn.ctc\_unique\_labels
* tf.compat.v2.nn.ctc\_unique\_labels
* tf.nn.ctc\_unique\_labels

tf.nn.ctc\_unique\_labels(  
    labels,  
    name=None  
)

Defined in [python/ops/ctc\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/ctc_ops.py).

For use with [tf.nn.ctc\_loss](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/ctc_loss) optional argument unique: This op can be used to preprocess labels in input pipeline to for better speed/memory use computing the ctc loss on TPU.

#### Example:

ctc\_unique\_labels([[3, 4, 4, 3]]) -> unique labels padded with 0: [[3, 4, 0, 0]] indices of original labels in unique: [0, 1, 1, 0]

#### Args:

* **labels**: tensor of shape [batch\_size, max\_label\_length] padded with 0.
* **name**: A name for this Op. Defaults to "ctc\_unique\_labels".

#### Returns:

tuple of - unique labels, tensor of shape [batch\_size, max\_label\_length] - indices into unique labels, shape [batch\_size, max\_label\_length]

# tf.nn.depthwise\_conv2d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d#aliases)

Depthwise 2-D convolution.

### Aliases:

* tf.compat.v2.nn.depthwise\_conv2d
* tf.nn.depthwise\_conv2d

tf.nn.depthwise\_conv2d(  
    input,  
    filter,  
    strides,  
    padding,  
    data\_format=None,  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

Given a 4D input tensor ('NHWC' or 'NCHW' data formats) and a filter tensor of shape[filter\_height, filter\_width, in\_channels, channel\_multiplier] containing in\_channels convolutional filters of depth 1, depthwise\_conv2d applies a different filter to each input channel (expanding from 1 channel to channel\_multiplier channels for each), then concatenates the results together. The output has in\_channels \* channel\_multiplier channels.

In detail, with the default NHWC format,

output[b, i, j, k \* channel\_multiplier + q] = sum\_{di, dj}  
     filter[di, dj, k, q] \* input[b, strides[1] \* i + rate[0] \* di,  
                                     strides[2] \* j + rate[1] \* dj, k]

Must have strides[0] = strides[3] = 1. For the most common case of the same horizontal and vertical strides, strides = [1, stride, stride, 1]. If any value in rate is greater than 1, we perform atrous depthwise convolution, in which case all values in the strides tensor must be equal to 1.

#### Args:

* **input**: 4-D with shape according to data\_format.
* **filter**: 4-D with shape [filter\_height, filter\_width, in\_channels, channel\_multiplier].
* **strides**: 1-D of size 4. The stride of the sliding window for each dimension of input.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: The data format for input. Either "NHWC" (default) or "NCHW".
* **dilations**: 1-D of size 2. The dilation rate in which we sample input values across the heightand width dimensions in atrous convolution. If it is greater than 1, then all values of strides must be 1.
* **name**: A name for this operation (optional).

#### Returns:

A 4-D Tensor with shape according to data\_format. E.g., for "NHWC" format, shape is [batch, out\_height, out\_width, in\_channels \* channel\_multiplier].

# tf.nn.depthwise\_conv2d\_backprop\_filter

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d_backprop_filter#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d_backprop_filter#aliases)

Computes the gradients of depthwise convolution with respect to the filter.

### Aliases:

* tf.compat.v1.nn.depthwise\_conv2d\_backprop\_filter
* tf.compat.v1.nn.depthwise\_conv2d\_native\_backprop\_filter
* tf.compat.v2.nn.depthwise\_conv2d\_backprop\_filter
* tf.nn.depthwise\_conv2d\_backprop\_filter

tf.nn.depthwise\_conv2d\_backprop\_filter(  
    input,  
    filter\_sizes,  
    out\_backprop,  
    strides,  
    padding,  
    data\_format='NHWC',  
    dilations=[1, 1, 1, 1],  
    name=None  
)

Defined in generated file: python/ops/gen\_nn\_ops.py.

#### Args:

* **input**: A Tensor. Must be one of the following types: half, bfloat16, float32, float64. 4-D with shape based on data\_format. For example, if data\_format is 'NHWC' then input is a 4-D [batch, in\_height, in\_width, in\_channels] tensor.
* **filter\_sizes**: A Tensor of type int32. An integer vector representing the tensor shape of filter, where filter is a 4-D [filter\_height, filter\_width, in\_channels, depthwise\_multiplier] tensor.
* **out\_backprop**: A Tensor. Must have the same type as input. 4-D with shape based on data\_format. For example, if data\_format is 'NHWC' then out\_backprop shape is [batch, out\_height, out\_width, out\_channels]. Gradients w.r.t. the output of the convolution.
* **strides**: A list of ints. The stride of the sliding window for each dimension of the input of the convolution.
* **padding**: A string from: "SAME", "VALID". The type of padding algorithm to use.
* **data\_format**: An optional string from: "NHWC", "NCHW". Defaults to "NHWC". Specify the data format of the input and output data. With the default format "NHWC", the data is stored in the order of: [batch, height, width, channels]. Alternatively, the format could be "NCHW", the data storage order of: [batch, channels, height, width].
* **dilations**: An optional list of ints. Defaults to [1, 1, 1, 1]. 1-D tensor of length 4. The dilation factor for each dimension of input. If set to k > 1, there will be k-1 skipped cells between each filter element on that dimension. The dimension order is determined by the value of data\_format, see above for details. Dilations in the batch and depth dimensions must be 1.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.nn.depthwise\_conv2d\_backprop\_input

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d_backprop_input#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depthwise_conv2d_backprop_input#aliases)

Computes the gradients of depthwise convolution with respect to the input.

### Aliases:

* tf.compat.v1.nn.depthwise\_conv2d\_backprop\_input
* tf.compat.v1.nn.depthwise\_conv2d\_native\_backprop\_input
* tf.compat.v2.nn.depthwise\_conv2d\_backprop\_input
* tf.nn.depthwise\_conv2d\_backprop\_input

tf.nn.depthwise\_conv2d\_backprop\_input(  
    input\_sizes,  
    filter,  
    out\_backprop,  
    strides,  
    padding,  
    data\_format='NHWC',  
    dilations=[1, 1, 1, 1],  
    name=None  
)

Defined in generated file: python/ops/gen\_nn\_ops.py.

#### Args:

* **input\_sizes**: A Tensor of type int32. An integer vector representing the shape of input, based on data\_format. For example, if data\_format is 'NHWC' then input is a 4-D [batch, height, width, channels] tensor.
* **filter**: A Tensor. Must be one of the following types: half, bfloat16, float32, float64. 4-D with shape [filter\_height, filter\_width, in\_channels, depthwise\_multiplier].
* **out\_backprop**: A Tensor. Must have the same type as filter. 4-D with shape based on data\_format. For example, if data\_format is 'NHWC' then out\_backprop shape is [batch, out\_height, out\_width, out\_channels]. Gradients w.r.t. the output of the convolution.
* **strides**: A list of ints. The stride of the sliding window for each dimension of the input of the convolution.
* **padding**: A string from: "SAME", "VALID". The type of padding algorithm to use.
* **data\_format**: An optional string from: "NHWC", "NCHW". Defaults to "NHWC". Specify the data format of the input and output data. With the default format "NHWC", the data is stored in the order of: [batch, height, width, channels]. Alternatively, the format could be "NCHW", the data storage order of: [batch, channels, height, width].
* **dilations**: An optional list of ints. Defaults to [1, 1, 1, 1]. 1-D tensor of length 4. The dilation factor for each dimension of input. If set to k > 1, there will be k-1 skipped cells between each filter element on that dimension. The dimension order is determined by the value of data\_format, see above for details. Dilations in the batch and depth dimensions must be 1.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as filter.

# tf.nn.depth\_to\_space

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depth_to_space#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/depth_to_space#aliases)

DepthToSpace for tensors of type T.

### Aliases:

* tf.compat.v2.nn.depth\_to\_space
* tf.nn.depth\_to\_space

tf.nn.depth\_to\_space(  
    input,  
    block\_size,  
    data\_format='NHWC',  
    name=None  
)

Defined in [python/ops/array\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/array_ops.py).

Rearranges data from depth into blocks of spatial data. This is the reverse transformation of SpaceToDepth. More specifically, this op outputs a copy of the input tensor where values from the depth dimension are moved in spatial blocks to the height and width dimensions. The attr block\_size indicates the input block size and how the data is moved.

* Chunks of data of size block\_size \* block\_size from depth are rearranged into non-overlapping blocks of size block\_size x block\_size
* The width the output tensor is input\_depth \* block\_size, whereas the height is input\_height \* block\_size.
* The Y, X coordinates within each block of the output image are determined by the high order component of the input channel index.
* The depth of the input tensor must be divisible by block\_size \* block\_size.

The data\_format attr specifies the layout of the input and output tensors with the following options: "NHWC": [ batch, height, width, channels ] "NCHW": [ batch, channels, height, width ] "NCHW\_VECT\_C": qint8 [ batch, channels / 4, height, width, 4 ]

It is useful to consider the operation as transforming a 6-D Tensor. e.g. for data\_format = NHWC, Each element in the input tensor can be specified via 6 coordinates, ordered by decreasing memory layout significance as: n,iY,iX,bY,bX,oC (where n=batch index, iX, iY means X or Y coordinates within the input image, bX, bY means coordinates within the output block, oC means output channels). The output would be the input transposed to the following layout: n,iY,bY,iX,bX,oC

This operation is useful for resizing the activations between convolutions (but keeping all data), e.g. instead of pooling. It is also useful for training purely convolutional models.

For example, given an input of shape [1, 1, 1, 4], data\_format = "NHWC" and block\_size = 2:

x = [[[[1, 2, 3, 4]]]]

This operation will output a tensor of shape [1, 2, 2, 1]:

   [[[[1], [2]],  
     [[3], [4]]]]

Here, the input has a batch of 1 and each batch element has shape [1, 1, 4], the corresponding output will have 2x2 elements and will have a depth of 1 channel (1 = 4 / (block\_size \* block\_size)). The output element shape is [2, 2, 1].

For an input tensor with larger depth, here of shape [1, 1, 1, 12], e.g.

x = [[[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]]]]

This operation, for block size of 2, will return the following tensor of shape [1, 2, 2, 3]

   [[[[1, 2, 3], [4, 5, 6]],  
     [[7, 8, 9], [10, 11, 12]]]]

Similarly, for the following input of shape [1 2 2 4], and a block size of 2:

x =  [[[[1, 2, 3, 4],  
       [5, 6, 7, 8]],  
      [[9, 10, 11, 12],  
       [13, 14, 15, 16]]]]

the operator will return the following tensor of shape [1 4 4 1]:

x = [[[ [1],   [2],  [5],  [6]],  
      [ [3],   [4],  [7],  [8]],  
      [ [9],  [10], [13],  [14]],  
      [ [11], [12], [15],  [16]]]]

#### Args:

* **input**: A Tensor.
* **block\_size**: An int that is >= 2. The size of the spatial block, same as in Space2Depth.
* **data\_format**: An optional string from: "NHWC", "NCHW", "NCHW\_VECT\_C". Defaults to "NHWC".
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.nn.dilation2d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/dilation2d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/dilation2d#aliases)

Computes the grayscale dilation of 4-D input and 3-D filters tensors.

### Aliases:

* tf.compat.v2.nn.dilation2d
* tf.nn.dilation2d

tf.nn.dilation2d(  
    input,  
    filters,  
    strides,  
    padding,  
    data\_format,  
    dilations,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

The input tensor has shape [batch, in\_height, in\_width, depth] and the filters tensor has shape [filter\_height, filter\_width, depth], i.e., each input channel is processed independently of the others with its own structuring function. The output tensor has shape [batch, out\_height, out\_width, depth]. The spatial dimensions of the output tensor depend on the padding algorithm. We currently only support the default "NHWC" data\_format.

In detail, the grayscale morphological 2-D dilation is the max-sum correlation (for consistency with conv2d, we use unmirrored filters):

output[b, y, x, c] =  
   max\_{dy, dx} input[b,  
                      strides[1] \* y + rates[1] \* dy,  
                      strides[2] \* x + rates[2] \* dx,  
                      c] +  
                filters[dy, dx, c]

Max-pooling is a special case when the filter has size equal to the pooling kernel size and contains all zeros.

Note on duality: The dilation of input by the filters is equal to the negation of the erosion of -input by the reflected filters.

#### Args:

* **input**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64. 4-D with shape [batch, in\_height, in\_width, depth].
* **filters**: A Tensor. Must have the same type as input. 3-D with shape [filter\_height, filter\_width, depth].
* **strides**: A list of ints that has length >= 4. The stride of the sliding window for each dimension of the input tensor. Must be: [1, stride\_height, stride\_width, 1].
* **padding**: A string from: "SAME", "VALID". The type of padding algorithm to use.
* **data\_format**: A string, only "NCHW" is currently supported.
* **dilations**: A list of ints that has length >= 4. The input stride for atrous morphological dilation. Must be: [1, rate\_height, rate\_width, 1].
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.nn.dropout

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/dropout#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/dropout#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/dropout#used_in_the_guide)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/dropout#used_in_the_tutorials)

Computes dropout.

### Aliases:

* tf.compat.v2.nn.dropout
* tf.nn.dropout

tf.nn.dropout(  
    x,  
    rate,  
    noise\_shape=None,  
    seed=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

### Used in the guide:

* [Writing layers and models with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models)

### Used in the tutorials:

* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

With probability rate, drops elements of x. Input that are kept are scaled up by 1 / (1 - rate), otherwise outputs 0. The scaling is so that the expected sum is unchanged.

**Note:** The behavior of dropout has changed between TensorFlow 1.x and 2.x. When converting 1.x code, please use named arguments to ensure behavior stays consistent.

By default, each element is kept or dropped independently. If noise\_shape is specified, it must be[broadcastable](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html) to the shape of x, and only dimensions with noise\_shape[i] == shape(x)[i] will make independent decisions. For example, if shape(x) = [k, l, m, n] and noise\_shape = [k, 1, 1, n], each batch and channel component will be kept independently and each row and column will be kept or not kept together.

#### Args:

* **x**: A floating point tensor.
* **rate**: A scalar Tensor with the same type as x. The probability that each element is dropped. For example, setting rate=0.1 would drop 10% of input elements.
* **noise\_shape**: A 1-D Tensor of type int32, representing the shape for randomly generated keep/drop flags.
* **seed**: A Python integer. Used to create random seeds. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed)for behavior.
* **name**: A name for this operation (optional).

#### Returns:

A Tensor of the same shape of x.

#### Raises:

* **ValueError**: If rate is not in (0, 1] or if x is not a floating point tensor.

# tf.nn.elu

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/elu#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/elu#aliases)

Computes exponential linear: exp(features) - 1 if < 0, features otherwise.

### Aliases:

* tf.compat.v1.nn.elu
* tf.compat.v2.nn.elu
* tf.nn.elu

tf.nn.elu(  
    features,  
    name=None  
)

Defined in generated file: python/ops/gen\_nn\_ops.py.

See [Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)](http://arxiv.org/abs/1511.07289)

#### Args:

* **features**: A Tensor. Must be one of the following types: half, bfloat16, float32, float64.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as features.

# tf.nn.embedding\_lookup

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/embedding_lookup#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/embedding_lookup#aliases)

Looks up ids in a list of embedding tensors.

### Aliases:

* tf.compat.v2.nn.embedding\_lookup
* tf.nn.embedding\_lookup

tf.nn.embedding\_lookup(  
    params,  
    ids,  
    max\_norm=None,  
    name=None  
)

Defined in [python/ops/embedding\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/embedding_ops.py).

This function is used to perform parallel lookups on the list of tensors in params. It is a generalization of [tf.gather](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather), where params is interpreted as a partitioning of a large embedding tensor. paramsmay be a PartitionedVariable as returned by using tf.compat.v1.get\_variable() with a partitioner.

If len(params) > 1, each element id of ids is partitioned between the elements of paramsaccording to the partition\_strategy. In all strategies, if the id space does not evenly divide the number of partitions, each of the first (max\_id + 1) % len(params) partitions will be assigned one more id.

The partition\_strategy is always "div" currently. This means that we assign ids to partitions in a contiguous manner. For instance, 13 ids are split across 5 partitions as: [[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10], [11, 12]]

The results of the lookup are concatenated into a dense tensor. The returned tensor has shape shape(ids) + shape(params)[1:].

#### Args:

* **params**: A single tensor representing the complete embedding tensor, or a list of P tensors all of same shape except for the first dimension, representing sharded embedding tensors. Alternatively, a PartitionedVariable, created by partitioning along dimension 0. Each element must be appropriately sized for the 'div' partition\_strategy.
* **ids**: A Tensor with type int32 or int64 containing the ids to be looked up in params.
* **max\_norm**: If not None, each embedding is clipped if its l2-norm is larger than this value.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor with the same type as the tensors in params.

#### Raises:

* **ValueError**: If params is empty.

# tf.nn.embedding\_lookup\_sparse

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/embedding_lookup_sparse#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/embedding_lookup_sparse#aliases)

Computes embeddings for the given ids and weights.

### Aliases:

* tf.compat.v2.nn.embedding\_lookup\_sparse
* tf.nn.embedding\_lookup\_sparse

tf.nn.embedding\_lookup\_sparse(  
    params,  
    sp\_ids,  
    sp\_weights,  
    combiner=None,  
    max\_norm=None,  
    name=None  
)

Defined in [python/ops/embedding\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/embedding_ops.py).

This op assumes that there is at least one id for each row in the dense tensor represented by sp\_ids (i.e. there are no rows with empty features), and that all the indices of sp\_ids are in canonical row-major order.

It also assumes that all id values lie in the range [0, p0), where p0 is the sum of the size of params along dimension 0.

#### Args:

* **params**: A single tensor representing the complete embedding tensor, or a list of P tensors all of same shape except for the first dimension, representing sharded embedding tensors. Alternatively, a PartitionedVariable, created by partitioning along dimension 0. Each element must be appropriately sized for "div" partition\_strategy.
* **sp\_ids**: N x M SparseTensor of int64 ids where N is typically batch size and M is arbitrary.
* **sp\_weights**: either a SparseTensor of float / double weights, or None to indicate all weights should be taken to be 1. If specified, sp\_weights must have exactly the same shape and indices as sp\_ids.
* **combiner**: A string specifying the reduction op. Currently "mean", "sqrtn" and "sum" are supported. "sum" computes the weighted sum of the embedding results for each row. "mean" is the weighted sum divided by the total weight. "sqrtn" is the weighted sum divided by the square root of the sum of the squares of the weights.
* **max\_norm**: If not None, each embedding is clipped if its l2-norm is larger than this value, before combining.
* **name**: Optional name for the op.

#### Returns:

A dense tensor representing the combined embeddings for the sparse ids. For each row in the dense tensor represented by sp\_ids, the op looks up the embeddings for all ids in that row, multiplies them by the corresponding weight, and combines these embeddings as specified.

In other words, if

shape(combined params) = [p0, p1, ..., pm]

and

shape(sp\_ids) = shape(sp\_weights) = [d0, d1, ..., dn]

then

shape(output) = [d0, d1, ..., dn-1, p1, ..., pm].

For instance, if params is a 10x20 matrix, and sp\_ids / sp\_weights are

[0, 0]: id 1, weight 2.0  
[0, 1]: id 3, weight 0.5  
[1, 0]: id 0, weight 1.0  
[2, 3]: id 1, weight 3.0

with combiner="mean", then the output will be a 3x20 matrix where

output[0, :] = (params[1, :] \* 2.0 + params[3, :] \* 0.5) / (2.0 + 0.5)  
output[1, :] = (params[0, :] \* 1.0) / 1.0  
output[2, :] = (params[1, :] \* 3.0) / 3.0

#### Raises:

* **TypeError**: If sp\_ids is not a SparseTensor, or if sp\_weights is neither None nor SparseTensor.
* **ValueError**: If combiner is not one of {"mean", "sqrtn", "sum"}.

# tf.nn.erosion2d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/erosion2d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/erosion2d#aliases)

Computes the grayscale erosion of 4-D value and 3-D filters tensors.

### Aliases:

* tf.compat.v2.nn.erosion2d
* tf.nn.erosion2d

tf.nn.erosion2d(  
    value,  
    filters,  
    strides,  
    padding,  
    data\_format,  
    dilations,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

The value tensor has shape [batch, in\_height, in\_width, depth] and the filters tensor has shape [filters\_height, filters\_width, depth], i.e., each input channel is processed independently of the others with its own structuring function. The output tensor has shape [batch, out\_height, out\_width, depth]. The spatial dimensions of the output tensor depend on the padding algorithm. We currently only support the default "NHWC" data\_format.

In detail, the grayscale morphological 2-D erosion is given by:

output[b, y, x, c] =  
   min\_{dy, dx} value[b,  
                      strides[1] \* y - dilations[1] \* dy,  
                      strides[2] \* x - dilations[2] \* dx,  
                      c] -  
                filters[dy, dx, c]

Duality: The erosion of value by the filters is equal to the negation of the dilation of -value by the reflected filters.

#### Args:

* **value**: A Tensor. 4-D with shape [batch, in\_height, in\_width, depth].
* **filters**: A Tensor. Must have the same type as value. 3-D with shape [filters\_height, filters\_width, depth].
* **strides**: A list of ints that has length >= 4. 1-D of length 4. The stride of the sliding window for each dimension of the input tensor. Must be: [1, stride\_height, stride\_width, 1].
* **padding**: A string from: "SAME", "VALID". The type of padding algorithm to use.
* **data\_format**: A string, only "NHWC" is currently supported.
* **dilations**: A list of ints that has length >= 4. 1-D of length 4. The input stride for atrous morphological dilation. Must be: [1, rate\_height, rate\_width, 1].
* **name**: A name for the operation (optional). If not specified "erosion2d" is used.

#### Returns:

A Tensor. Has the same type as value. 4-D with shape [batch, out\_height, out\_width, depth].

#### Raises:

* **ValueError**: If the value depth does not match filters' shape, or if padding is other than 'VALID' or 'SAME'.

# tf.nn.fractional\_avg\_pool

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/fractional_avg_pool#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/fractional_avg_pool#aliases)

Performs fractional average pooling on the input.

### Aliases:

* tf.compat.v2.nn.fractional\_avg\_pool
* tf.nn.fractional\_avg\_pool

tf.nn.fractional\_avg\_pool(  
    value,  
    pooling\_ratio,  
    pseudo\_random=False,  
    overlapping=False,  
    seed=0,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Fractional average pooling is similar to Fractional max pooling in the pooling region generation step. The only difference is that after pooling regions are generated, a mean operation is performed instead of a max operation in each pooling region.

#### Args:

* **value**: A Tensor. 4-D with shape [batch, height, width, channels].
* **pooling\_ratio**: A list of floats that has length >= 4. Pooling ratio for each dimension of value, currently only supports row and col dimension and should be >= 1.0. For example, a valid pooling ratio looks like [1.0, 1.44, 1.73, 1.0]. The first and last elements must be 1.0 because we don't allow pooling on batch and channels dimensions. 1.44 and 1.73 are pooling ratio on height and width dimensions respectively.
* **pseudo\_random**: An optional bool. Defaults to False. When set to True, generates the pooling sequence in a pseudorandom fashion, otherwise, in a random fashion. Check paper [Benjamin Graham, Fractional Max-Pooling](http://arxiv.org/abs/1412.6071) for difference between pseudorandom and random.
* **overlapping**: An optional bool. Defaults to False. When set to True, it means when pooling, the values at the boundary of adjacent pooling cells are used by both cells. For example:index 0 1 2 3 4 value 20 5 16 3 7 If the pooling sequence is [0, 2, 4], then 16, at index 2 will be used twice. The result would be [20, 16] for fractional avg pooling.
* **seed**: An optional int. Defaults to 0. If set to be non-zero, the random number generator is seeded by the given seed. Otherwise it is seeded by a random seed.
* **name**: A name for the operation (optional).

#### Returns:

A tuple of Tensor objects (output, row\_pooling\_sequence, col\_pooling\_sequence). output: Output Tensor after fractional avg pooling. Has the same type as value. row\_pooling\_sequence: A Tensor of type int64. col\_pooling\_sequence: A Tensor of type int64.

# tf.nn.fractional\_max\_pool

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/fractional_max_pool#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/fractional_max_pool#aliases)

Performs fractional max pooling on the input.

### Aliases:

* tf.compat.v2.nn.fractional\_max\_pool
* tf.nn.fractional\_max\_pool

tf.nn.fractional\_max\_pool(  
    value,  
    pooling\_ratio,  
    pseudo\_random=False,  
    overlapping=False,  
    seed=0,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Fractional max pooling is slightly different than regular max pooling. In regular max pooling, you downsize an input set by taking the maximum value of smaller N x N subsections of the set (often 2x2), and try to reduce the set by a factor of N, where N is an integer. Fractional max pooling, as you might expect from the word "fractional", means that the overall reduction ratio N does not have to be an integer.

The sizes of the pooling regions are generated randomly but are fairly uniform. For example, let's look at the height dimension, and the constraints on the list of rows that will be pool boundaries.

First we define the following:

1. input\_row\_length : the number of rows from the input set
2. output\_row\_length : which will be smaller than the input
3. alpha = input\_row\_length / output\_row\_length : our reduction ratio
4. K = floor(alpha)
5. row\_pooling\_sequence : this is the result list of pool boundary rows

Then, row\_pooling\_sequence should satisfy:

1. a[0] = 0 : the first value of the sequence is 0
2. a[end] = input\_row\_length : the last value of the sequence is the size
3. K <= (a[i+1] - a[i]) <= K+1 : all intervals are K or K+1 size
4. length(row\_pooling\_sequence) = output\_row\_length+1

For more details on fractional max pooling, see this paper: [Benjamin Graham, Fractional Max-Pooling](http://arxiv.org/abs/1412.6071)

#### Args:

* **value**: A Tensor. 4-D with shape [batch, height, width, channels].
* **pooling\_ratio**: An int or list of ints that has length 1, 2 or 4. Pooling ratio for each dimension of value, currently only supports row and col dimension and should be >= 1.0. For example, a valid pooling ratio looks like [1.0, 1.44, 1.73, 1.0]. The first and last elements must be 1.0 because we don't allow pooling on batch and channels dimensions. 1.44 and 1.73 are pooling ratio on height and width dimensions respectively.
* **pseudo\_random**: An optional bool. Defaults to False. When set to True, generates the pooling sequence in a pseudorandom fashion, otherwise, in a random fashion. Check paper [Benjamin Graham, Fractional Max-Pooling](http://arxiv.org/abs/1412.6071) for difference between pseudorandom and random.
* **overlapping**: An optional bool. Defaults to False. When set to True, it means when pooling, the values at the boundary of adjacent pooling cells are used by both cells. For example:index 0 1 2 3 4 value 20 5 16 3 7 If the pooling sequence is [0, 2, 4], then 16, at index 2 will be used twice. The result would be [20, 16] for fractional max pooling.
* **seed**: An optional int. Defaults to 0. If set to be non-zero, the random number generator is seeded by the given seed. Otherwise it is seeded by a random seed.
* **name**: A name for the operation (optional).

#### Returns:

A tuple of Tensor objects (output, row\_pooling\_sequence, col\_pooling\_sequence). output: Output Tensor after fractional max pooling. Has the same type as value. row\_pooling\_sequence: A Tensor of type int64. col\_pooling\_sequence: A Tensor of type int64.

# tf.nn.l2\_loss

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/l2_loss#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/l2_loss#aliases)

L2 Loss.

### Aliases:

* tf.compat.v1.nn.l2\_loss
* tf.compat.v2.nn.l2\_loss
* tf.nn.l2\_loss

tf.nn.l2\_loss(  
    t,  
    name=None  
)

Defined in generated file: python/ops/gen\_nn\_ops.py.

Computes half the L2 norm of a tensor without the sqrt:

output = sum(t \*\* 2) / 2

#### Args:

* **t**: A Tensor. Must be one of the following types: half, bfloat16, float32, float64. Typically 2-D, but may have any dimensions.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as t.

# tf.nn.leaky\_relu

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/leaky_relu#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/leaky_relu#aliases)

Compute the Leaky ReLU activation function.

### Aliases:

* tf.compat.v1.nn.leaky\_relu
* tf.compat.v2.nn.leaky\_relu
* tf.nn.leaky\_relu

tf.nn.leaky\_relu(  
    features,  
    alpha=0.2,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Source: [Rectifier Nonlinearities Improve Neural Network Acoustic Models. AL Maas, AY Hannun, AY Ng - Proc. ICML, 2013](https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf).

#### Args:

* **features**: A Tensor representing preactivation values. Must be one of the following types: float16, float32, float64, int32, int64.
* **alpha**: Slope of the activation function at x < 0.
* **name**: A name for the operation (optional).

#### Returns:

The activation value.

# tf.nn.local\_response\_normalization

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/local_response_normalization#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/local_response_normalization#aliases)

Local Response Normalization.

### Aliases:

* tf.compat.v1.nn.local\_response\_normalization
* tf.compat.v1.nn.lrn
* tf.compat.v2.nn.local\_response\_normalization
* tf.compat.v2.nn.lrn
* tf.nn.local\_response\_normalization
* tf.nn.lrn

tf.nn.local\_response\_normalization(  
    input,  
    depth\_radius=5,  
    bias=1,  
    alpha=1,  
    beta=0.5,  
    name=None  
)

Defined in generated file: python/ops/gen\_nn\_ops.py.

The 4-D input tensor is treated as a 3-D array of 1-D vectors (along the last dimension), and each vector is normalized independently. Within a given vector, each component is divided by the weighted, squared sum of inputs within depth\_radius. In detail,

sqr\_sum[a, b, c, d] =  
    sum(input[a, b, c, d - depth\_radius : d + depth\_radius + 1] \*\* 2)  
output = input / (bias + alpha \* sqr\_sum) \*\* beta

For details, see [Krizhevsky et al., ImageNet classification with deep convolutional neural networks (NIPS 2012)](http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks).

#### Args:

* **input**: A Tensor. Must be one of the following types: half, bfloat16, float32. 4-D.
* **depth\_radius**: An optional int. Defaults to 5. 0-D. Half-width of the 1-D normalization window.
* **bias**: An optional float. Defaults to 1. An offset (usually positive to avoid dividing by 0).
* **alpha**: An optional float. Defaults to 1. A scale factor, usually positive.
* **beta**: An optional float. Defaults to 0.5. An exponent.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.nn.log\_poisson\_loss

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/log_poisson_loss#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/log_poisson_loss#aliases)

Computes log Poisson loss given log\_input.

### Aliases:

* tf.compat.v1.nn.log\_poisson\_loss
* tf.compat.v2.nn.log\_poisson\_loss
* tf.nn.log\_poisson\_loss

tf.nn.log\_poisson\_loss(  
    targets,  
    log\_input,  
    compute\_full\_loss=False,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

Gives the log-likelihood loss between the prediction and the target under the assumption that the target has a Poisson distribution. Caveat: By default, this is not the exact loss, but the loss minus a constant term [log(z!)]. That has no effect for optimization, but does not play well with relative loss comparisons. To compute an approximation of the log factorial term, specify compute\_full\_loss=True to enable Stirling's Approximation.

For brevity, let c = log(x) = log\_input, z = targets. The log Poisson loss is

  -log(exp(-x) \* (x^z) / z!)  
= -log(exp(-x) \* (x^z)) + log(z!)  
~ -log(exp(-x)) - log(x^z) [+ z \* log(z) - z + 0.5 \* log(2 \* pi \* z)]  
    [ Note the second term is the Stirling's Approximation for log(z!).  
      It is invariant to x and does not affect optimization, though  
      important for correct relative loss comparisons. It is only  
      computed when compute\_full\_loss == True. ]  
= x - z \* log(x) [+ z \* log(z) - z + 0.5 \* log(2 \* pi \* z)]  
= exp(c) - z \* c [+ z \* log(z) - z + 0.5 \* log(2 \* pi \* z)]

#### Args:

* **targets**: A Tensor of the same type and shape as log\_input.
* **log\_input**: A Tensor of type float32 or float64.
* **compute\_full\_loss**: whether to compute the full loss. If false, a constant term is dropped in favor of more efficient optimization.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of the same shape as log\_input with the componentwise logistic losses.

#### Raises:

* **ValueError**: If log\_input and targets do not have the same shape.

# tf.nn.log\_softmax

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/log_softmax#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/log_softmax#aliases)

Computes log softmax activations.

### Aliases:

* tf.compat.v2.math.log\_softmax
* tf.compat.v2.nn.log\_softmax
* tf.math.log\_softmax
* tf.nn.log\_softmax

tf.nn.log\_softmax(  
    logits,  
    axis=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

For each batch i and class j we have

logsoftmax = logits - log(reduce\_sum(exp(logits), axis))

#### Args:

* **logits**: A non-empty Tensor. Must be one of the following types: half, float32, float64.
* **axis**: The dimension softmax would be performed on. The default is -1 which indicates the last dimension.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as logits. Same shape as logits.

#### Raises:

* **InvalidArgumentError**: if logits is empty or axis is beyond the last dimension of logits.

# tf.nn.max\_pool

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool#aliases)

Performs the max pooling on the input.

### Aliases:

* tf.compat.v1.nn.max\_pool\_v2
* tf.compat.v2.nn.max\_pool
* tf.nn.max\_pool

tf.nn.max\_pool(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

#### Args:

* **input**: Tensor of rank N+2, of shape [batch\_size] + input\_spatial\_shape + [num\_channels] if data\_format does not start with "NC" (default), or [batch\_size, num\_channels] + input\_spatial\_shape if data\_format starts with "NC". Pooling happens over the spatial dimensions only.
* **ksize**: An int or list of ints that has length 1, N or N+2. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1, N or N+2. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string. Specifies the channel dimension. For N=1 it can be either "NWC" (default) or "NCW", for N=2 it can be either "NHWC" (default) or "NCHW" and for N=3 either "NDHWC" (default) or "NCDHW".
* **name**: Optional name for the operation.

#### Returns:

A Tensor of format specified by data\_format. The max pooled output tensor.

# tf.nn.max\_pool1d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool1d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool1d#aliases)

Performs the max pooling on the input.

### Aliases:

* tf.compat.v1.nn.max\_pool1d
* tf.compat.v2.nn.max\_pool1d
* tf.nn.max\_pool1d

tf.nn.max\_pool1d(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format='NWC',  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Note internally this op reshapes and uses the underlying 2d operation.

#### Args:

* **input**: A 3-D Tensor of the format specified by data\_format.
* **ksize**: An int or list of ints that has length 1 or 3. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1 or 3. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: An optional string from: "NWC", "NCW". Defaults to "NWC".
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of format specified by data\_format. The max pooled output tensor.

# tf.nn.max\_pool2d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool2d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool2d#aliases)

Performs the max pooling on the input.

### Aliases:

* tf.compat.v1.nn.max\_pool2d
* tf.compat.v2.nn.max\_pool2d
* tf.nn.max\_pool2d

tf.nn.max\_pool2d(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format='NHWC',  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

#### Args:

* **input**: A 4-D Tensor of the format specified by data\_format.
* **ksize**: An int or list of ints that has length 1, 2 or 4. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1, 2 or 4. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string. 'NHWC', 'NCHW' and 'NCHW\_VECT\_C' are supported.
* **name**: Optional name for the operation.

#### Returns:

A Tensor of format specified by data\_format. The max pooled output tensor.

# tf.nn.max\_pool3d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool3d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool3d#aliases)

Performs the max pooling on the input.

### Aliases:

* tf.compat.v1.nn.max\_pool3d
* tf.compat.v2.nn.max\_pool3d
* tf.nn.max\_pool3d

tf.nn.max\_pool3d(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format='NDHWC',  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

#### Args:

* **input**: A 5-D Tensor of the format specified by data\_format.
* **ksize**: An int or list of ints that has length 1, 3 or 5. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1, 3 or 5. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: An optional string from: "NDHWC", "NCDHW". Defaults to "NDHWC". The data format of the input and output data. With the default format "NDHWC", the data is stored in the order of: [batch, in\_depth, in\_height, in\_width, in\_channels]. Alternatively, the format could be "NCDHW", the data storage order is: [batch, in\_channels, in\_depth, in\_height, in\_width].
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of format specified by data\_format. The max pooled output tensor.

# tf.nn.max\_pool\_with\_argmax

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool_with_argmax#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/max_pool_with_argmax#aliases)

Performs max pooling on the input and outputs both max values and indices.

### Aliases:

* tf.compat.v2.nn.max\_pool\_with\_argmax
* tf.nn.max\_pool\_with\_argmax

tf.nn.max\_pool\_with\_argmax(  
    input,  
    ksize,  
    strides,  
    padding,  
    data\_format='NHWC',  
    output\_dtype=tf.dtypes.int64,  
    include\_batch\_in\_index=False,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

The indices in argmax are flattened, so that a maximum value at position [b, y, x, c] becomes flattened index: (y \* width + x) \* channels + c if include\_batch\_in\_index is False; ((b \* height + y) \* width + x) \* channels + c if include\_batch\_in\_index is True.

The indices returned are always in [0, height) x [0, width) before flattening, even if padding is involved and the mathematically correct answer is outside (either negative or too large). This is a bug, but fixing it is difficult to do in a safe backwards compatible way, especially due to flattening.

#### Args:

* **input**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64. 4-D with shape [batch, height, width, channels]. Input to pool over.
* **ksize**: An int or list of ints that has length 1, 2 or 4. The size of the window for each dimension of the input tensor.
* **strides**: An int or list of ints that has length 1, 2 or 4. The stride of the sliding window for each dimension of the input tensor.
* **padding**: A string from: "SAME", "VALID". The type of padding algorithm to use.
* **data\_format**: An optional string, must be set to "NHWC". Defaults to "NHWC". Specify the data format of the input and output data.
* **output\_dtype**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.int32, tf.int64. Defaults to [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64). The dtype of the returned argmax tensor.
* **include\_batch\_in\_index**: An optional boolean. Defaults to False. Whether to include batch dimension in flattened index of argmax.
* **name**: A name for the operation (optional).

#### Returns:

A tuple of Tensor objects (output, argmax).

* **output**: A Tensor. Has the same type as input.
* **argmax**: A Tensor of type output\_dtype.

# tf.nn.moments

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/moments#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/moments#aliases)

Calculates the mean and variance of x.

### Aliases:

* tf.compat.v2.nn.moments
* tf.nn.moments

tf.nn.moments(  
    x,  
    axes,  
    shift=None,  
    keepdims=False,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

The mean and variance are calculated by aggregating the contents of x across axes. If x is 1-D and axes = [0] this is just the mean and variance of a vector.

**Note:** shift is currently not used; the true mean is computed and used.

When using these moments for batch normalization (see [tf.nn.batch\_normalization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/batch_normalization)):

* for so-called "global normalization", used with convolutional filters with shape [batch, height, width, depth], pass axes=[0, 1, 2].
* for simple batch normalization pass axes=[0] (batch only).

#### Args:

* **x**: A Tensor.
* **axes**: Array of ints. Axes along which to compute mean and variance.
* **shift**: Not used in the current implementation.
* **keepdims**: produce moments with the same dimensionality as the input.
* **name**: Name used to scope the operations that compute the moments.

#### Returns:

Two Tensor objects: mean and variance.

# tf.nn.nce\_loss

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/nce_loss#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/nce_loss#aliases)

Computes and returns the noise-contrastive estimation training loss.

### Aliases:

* tf.compat.v2.nn.nce\_loss
* tf.nn.nce\_loss

tf.nn.nce\_loss(  
    weights,  
    biases,  
    labels,  
    inputs,  
    num\_sampled,  
    num\_classes,  
    num\_true=1,  
    sampled\_values=None,  
    remove\_accidental\_hits=False,  
    name='nce\_loss'  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

See [Noise-contrastive estimation: A new estimation principle for unnormalized statistical models](http://www.jmlr.org/proceedings/papers/v9/gutmann10a/gutmann10a.pdf). Also see our [Candidate Sampling Algorithms Reference](https://www.tensorflow.org/extras/candidate_sampling.pdf)

A common use case is to use this method for training, and calculate the full sigmoid loss for evaluation or inference as in the following example:

if mode == "train":  
  loss = tf.nn.nce\_loss(  
      weights=weights,  
      biases=biases,  
      labels=labels,  
      inputs=inputs,  
      ...)  
elif mode == "eval":  
  logits = tf.matmul(inputs, tf.transpose(weights))  
  logits = tf.nn.bias\_add(logits, biases)  
  labels\_one\_hot = tf.one\_hot(labels, n\_classes)  
  loss = tf.nn.sigmoid\_cross\_entropy\_with\_logits(  
      labels=labels\_one\_hot,  
      logits=logits)  
  loss = tf.reduce\_sum(loss, axis=1)

**Note:** when doing embedding lookup on **weights** and **bias**, "div" partition strategy will be used. Support for other partition strategy will be added later.**Note:** By default this uses a log-uniform (Zipfian) distribution for sampling, so your labels must be sorted in order of decreasing frequency to achieve good results. For more details, see[**tf.random.log\_uniform\_candidate\_sampler**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/log_uniform_candidate_sampler).**Note:** In the case where **num\_true** > 1, we assign to each target class the target probability 1 / **num\_true** so that the target probabilities sum to 1 per-example.**Note:** It would be useful to allow a variable number of target classes per example. We hope to provide this functionality in a future release. For now, if you have a variable number of target classes, you can pad them out to a constant number by either repeating them or by padding with an otherwise unused class.

#### Args:

* **weights**: A Tensor of shape [num\_classes, dim], or a list of Tensor objects whose concatenation along dimension 0 has shape [num\_classes, dim]. The (possibly-partitioned) class embeddings.
* **biases**: A Tensor of shape [num\_classes]. The class biases.
* **labels**: A Tensor of type int64 and shape [batch\_size, num\_true]. The target classes.
* **inputs**: A Tensor of shape [batch\_size, dim]. The forward activations of the input network.
* **num\_sampled**: An int. The number of negative classes to randomly sample per batch. This single sample of negative classes is evaluated for each element in the batch.
* **num\_classes**: An int. The number of possible classes.
* **num\_true**: An int. The number of target classes per training example.
* **sampled\_values**: a tuple of (sampled\_candidates, true\_expected\_count,sampled\_expected\_count) returned by a \*\_candidate\_sampler function. (if None, we default to log\_uniform\_candidate\_sampler)
* **remove\_accidental\_hits**: A bool. Whether to remove "accidental hits" where a sampled class equals one of the target classes. If set to True, this is a "Sampled Logistic" loss instead of NCE, and we are learning to generate log-odds instead of log probabilities. See our [Candidate Sampling Algorithms Reference](https://www.tensorflow.org/extras/candidate_sampling.pdf). Default is False.
* **name**: A name for the operation (optional).

#### Returns:

A batch\_size 1-D tensor of per-example NCE losses.

# tf.nn.normalize\_moments

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/normalize_moments#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/normalize_moments#aliases)

Calculate the mean and variance of based on the sufficient statistics.

### Aliases:

* tf.compat.v1.nn.normalize\_moments
* tf.compat.v2.nn.normalize\_moments
* tf.nn.normalize\_moments

tf.nn.normalize\_moments(  
    counts,  
    mean\_ss,  
    variance\_ss,  
    shift,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

#### Args:

* **counts**: A Tensor containing the total count of the data (one value).
* **mean\_ss**: A Tensor containing the mean sufficient statistics: the (possibly shifted) sum of the elements to average over.
* **variance\_ss**: A Tensor containing the variance sufficient statistics: the (possibly shifted) squared sum of the data to compute the variance over.
* **shift**: A Tensor containing the value by which the data is shifted for numerical stability, or None if no shift was performed.
* **name**: Name used to scope the operations that compute the moments.

#### Returns:

Two Tensor objects: mean and variance.

# tf.nn.pool

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/pool#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/pool#aliases)

Performs an N-D pooling operation.

### Aliases:

* tf.compat.v2.nn.pool
* tf.nn.pool

tf.nn.pool(  
    input,  
    window\_shape,  
    pooling\_type,  
    strides=None,  
    padding='VALID',  
    data\_format=None,  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

In the case that data\_format does not start with "NC", computes for 0 <= b < batch\_size, 0 <= x[i] < output\_spatial\_shape[i], 0 <= c < num\_channels:

  output[b, x[0], ..., x[N-1], c] =  
    REDUCE\_{z[0], ..., z[N-1]}  
      input[b,  
            x[0] \* strides[0] - pad\_before[0] + dilation\_rate[0]\*z[0],  
            ...  
            x[N-1]\*strides[N-1] - pad\_before[N-1] + dilation\_rate[N-1]\*z[N-1],  
            c],

where the reduction function REDUCE depends on the value of pooling\_type, and pad\_before is defined based on the value of padding as described in the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution)for details. The reduction never includes out-of-bounds positions.

In the case that data\_format starts with "NC", the input and output are simply transposed as follows:

  pool(input, data\_format, \*\*kwargs) =  
    tf.transpose(pool(tf.transpose(input, [0] + range(2,N+2) + [1]),  
                      \*\*kwargs),  
                 [0, N+1] + range(1, N+1))

#### Args:

* **input**: Tensor of rank N+2, of shape [batch\_size] + input\_spatial\_shape + [num\_channels] if data\_format does not start with "NC" (default), or [batch\_size, num\_channels] + input\_spatial\_shape if data\_format starts with "NC". Pooling happens over the spatial dimensions only.
* **window\_shape**: Sequence of N ints >= 1.
* **pooling\_type**: Specifies pooling operation, must be "AVG" or "MAX".
* **strides**: Optional. Sequence of N ints >= 1. Defaults to [1]\*N. If any value of strides is > 1, then all values of dilation\_rate must be 1.
* **padding**: The padding algorithm, must be "SAME" or "VALID". Defaults to "SAME". See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: A string or None. Specifies whether the channel dimension of the input and output is the last dimension (default, or if data\_format does not start with "NC"), or the second dimension (if data\_format starts with "NC"). For N=1, the valid values are "NWC" (default) and "NCW". For N=2, the valid values are "NHWC" (default) and "NCHW". For N=3, the valid values are "NDHWC" (default) and "NCDHW".
* **dilations**: Optional. Dilation rate. List of N ints >= 1. Defaults to [1]\*N. If any value of dilation\_rate is > 1, then all values of strides must be 1.
* **name**: Optional. Name of the op.

#### Returns:

Tensor of rank N+2, of shape [batch\_size] + output\_spatial\_shape + [num\_channels]

if data\_format is None or does not start with "NC", or

[batch\_size, num\_channels] + output\_spatial\_shape

if data\_format starts with "NC", where output\_spatial\_shape depends on the value of padding:

If padding = "SAME": output\_spatial\_shape[i] = ceil(input\_spatial\_shape[i] / strides[i])

If padding = "VALID": output\_spatial\_shape[i] = ceil((input\_spatial\_shape[i] - (window\_shape[i] - 1) \* dilation\_rate[i]) / strides[i]).

#### Raises:

* **ValueError**: if arguments are invalid.

# tf.nn.relu

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/relu#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/relu#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/relu#used_in_the_guide)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/relu#used_in_the_tutorials)

Computes rectified linear: max(features, 0).

### Aliases:

* tf.compat.v1.nn.relu
* tf.compat.v2.nn.relu
* tf.nn.relu

tf.nn.relu(  
    features,  
    name=None  
)

Defined in generated file: python/ops/gen\_nn\_ops.py.

### Used in the guide:

* [Writing layers and models with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models)
* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [Custom layers](https://www.tensorflow.org/beta/tutorials/eager/custom_layers)
* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)

#### Args:

* **features**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64, qint8.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as features.

# tf.nn.relu6

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/relu6#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/relu6#aliases)

Computes Rectified Linear 6: min(max(features, 0), 6).

### Aliases:

* tf.compat.v1.nn.relu6
* tf.compat.v2.nn.relu6
* tf.nn.relu6

tf.nn.relu6(  
    features,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Source: [Convolutional Deep Belief Networks on CIFAR-10. A. Krizhevsky](http://www.cs.utoronto.ca/~kriz/conv-cifar10-aug2010.pdf)

#### Args:

* **features**: A Tensor with type float, double, int32, int64, uint8, int16, or int8.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor with the same type as features.

# tf.nn.RNNCellDeviceWrapper

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDeviceWrapper#top_of_page)
* [Class RNNCellDeviceWrapper](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDeviceWrapper#class_rnncelldevicewrapper)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDeviceWrapper#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDeviceWrapper#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDeviceWrapper#properties)

## Class RNNCellDeviceWrapper

Operator that ensures an RNNCell runs on a particular device.

### Aliases:

* Class tf.compat.v2.nn.RNNCellDeviceWrapper
* Class tf.nn.RNNCellDeviceWrapper

Defined in [python/keras/layers/rnn\_cell\_wrapper\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/keras/layers/rnn_cell_wrapper_v2.py).

## \_\_init\_\_

\_\_init\_\_(  
    \*args,  
    \*\*kwargs  
)

Construct a DeviceWrapper for cell with device device.

Ensures the wrapped cell is called with tf.device(device).

#### Args:

* **cell**: An instance of RNNCell.
* **device**: A device string or function, for passing to [tf.device](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/device).
* **\*\*kwargs**: dict of keyword arguments for base layer.

## Properties

### output\_size

### state\_size

## Methods

### get\_initial\_state

get\_initial\_state(  
    inputs=None,  
    batch\_size=None,  
    dtype=None  
)

### zero\_state

zero\_state(  
    batch\_size,  
    dtype  
)

# tf.nn.RNNCellDropoutWrapper

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDropoutWrapper#top_of_page)
* [Class RNNCellDropoutWrapper](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDropoutWrapper#class_rnncelldropoutwrapper)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDropoutWrapper#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDropoutWrapper#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellDropoutWrapper#properties)

## Class RNNCellDropoutWrapper

Operator adding dropout to inputs and outputs of the given cell.

### Aliases:

* Class tf.compat.v2.nn.RNNCellDropoutWrapper
* Class tf.nn.RNNCellDropoutWrapper

Defined in [python/keras/layers/rnn\_cell\_wrapper\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/keras/layers/rnn_cell_wrapper_v2.py).

## \_\_init\_\_

\_\_init\_\_(  
    \*args,  
    \*\*kwargs  
)

Create a cell with added input, state, and/or output dropout.

If variational\_recurrent is set to True (**NOT** the default behavior), then the same dropout mask is applied at every step, as described in: [A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. Y. Gal, Z. Ghahramani](https://arxiv.org/abs/1512.05287).

Otherwise a different dropout mask is applied at every time step.

Note, by default (unless a custom dropout\_state\_filter is provided), the memory state (ccomponent of any LSTMStateTuple) passing through a DropoutWrapper is never modified. This behavior is described in the above article.

#### Args:

* **cell**: an RNNCell, a projection to output\_size is added to it.
* **input\_keep\_prob**: unit Tensor or float between 0 and 1, input keep probability; if it is constant and 1, no input dropout will be added.
* **output\_keep\_prob**: unit Tensor or float between 0 and 1, output keep probability; if it is constant and 1, no output dropout will be added.
* **state\_keep\_prob**: unit Tensor or float between 0 and 1, output keep probability; if it is constant and 1, no output dropout will be added. State dropout is performed on the outgoing states of the cell. **Note** the state components to which dropout is applied when state\_keep\_prob is in (0, 1) are also determined by the argument dropout\_state\_filter\_visitor (e.g. by default dropout is never applied to the c component of an LSTMStateTuple).
* **variational\_recurrent**: Python bool. If True, then the same dropout pattern is applied across all time steps per run call. If this parameter is set, input\_size **must** be provided.
* **input\_size**: (optional) (possibly nested tuple of) TensorShape objects containing the depth(s) of the input tensors expected to be passed in to the DropoutWrapper. Required and used **iff**variational\_recurrent = True and input\_keep\_prob < 1.
* **dtype**: (optional) The dtype of the input, state, and output tensors. Required and used **iff**variational\_recurrent = True.
* **seed**: (optional) integer, the randomness seed.
* **dropout\_state\_filter\_visitor**: (optional), default: (see below). Function that takes any hierarchical level of the state and returns a scalar or depth=1 structure of Python booleans describing which terms in the state should be dropped out. In addition, if the function returns True, dropout is applied across this sublevel. If the function returns False, dropout is not applied across this entire sublevel. Default behavior: perform dropout on all terms except the memory (c) state of LSTMCellState objects, and don't try to apply dropout to TensorArrayobjects: def dropout\_state\_filter\_visitor(s): if isinstance(s, LSTMCellState): # Never perform dropout on the c state. return LSTMCellState(c=False, h=True) elif isinstance(s, TensorArray): return False return True
* **\*\*kwargs**: dict of keyword arguments for base layer.

#### Raises:

* **TypeError**: if cell is not an RNNCell, or keep\_state\_fn is provided but not callable.
* **ValueError**: if any of the keep\_probs are not between 0 and 1.

## Properties

### output\_size

### state\_size

### wrapped\_cell

## Methods

### get\_initial\_state

get\_initial\_state(  
    inputs=None,  
    batch\_size=None,  
    dtype=None  
)

### zero\_state

zero\_state(  
    batch\_size,  
    dtype  
)

# tf.nn.RNNCellResidualWrapper

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellResidualWrapper#top_of_page)
* [Class RNNCellResidualWrapper](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellResidualWrapper#class_rnncellresidualwrapper)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellResidualWrapper#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellResidualWrapper#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/RNNCellResidualWrapper#properties)

## Class RNNCellResidualWrapper

RNNCell wrapper that ensures cell inputs are added to the outputs.

### Aliases:

* Class tf.compat.v2.nn.RNNCellResidualWrapper
* Class tf.nn.RNNCellResidualWrapper

Defined in [python/keras/layers/rnn\_cell\_wrapper\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/keras/layers/rnn_cell_wrapper_v2.py).

## \_\_init\_\_

\_\_init\_\_(  
    \*args,  
    \*\*kwargs  
)

Constructs a ResidualWrapper for cell.

#### Args:

* **cell**: An instance of RNNCell.
* **residual\_fn**: (Optional) The function to map raw cell inputs and raw cell outputs to the actual cell outputs of the residual network. Defaults to calling nest.map\_structure on (lambda i, o: i + o), inputs and outputs.
* **\*\*kwargs**: dict of keyword arguments for base layer.

## Properties

### output\_size

### state\_size

## Methods

### get\_initial\_state

get\_initial\_state(  
    inputs=None,  
    batch\_size=None,  
    dtype=None  
)

### zero\_state

zero\_state(  
    batch\_size,  
    dtype  
)

# tf.nn.safe\_embedding\_lookup\_sparse

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/safe_embedding_lookup_sparse#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/safe_embedding_lookup_sparse#aliases)

Lookup embedding results, accounting for invalid IDs and empty features.

### Aliases:

* tf.compat.v2.nn.safe\_embedding\_lookup\_sparse
* tf.nn.safe\_embedding\_lookup\_sparse

tf.nn.safe\_embedding\_lookup\_sparse(  
    embedding\_weights,  
    sparse\_ids,  
    sparse\_weights=None,  
    combiner='mean',  
    default\_id=None,  
    max\_norm=None,  
    name=None  
)

Defined in [python/ops/embedding\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/embedding_ops.py).

The partitioned embedding in embedding\_weights must all be the same shape except for the first dimension. The first dimension is allowed to vary as the vocabulary size is not necessarily a multiple of P. embedding\_weights may be a PartitionedVariable as returned by usingtf.compat.v1.get\_variable() with a partitioner.

Invalid IDs (< 0) are pruned from input IDs and weights, as well as any IDs with non-positive weight. For an entry with no features, the embedding vector for default\_id is returned, or the 0-vector if default\_id is not supplied.

The ids and weights may be multi-dimensional. Embeddings are always aggregated along the last dimension.

**Note:** when doing embedding lookup on **embedding\_weights**, "div" partition strategy will be used. Support for other partition strategy will be added later.

#### Args:

* **embedding\_weights**: A list of P float Tensors or values representing partitioned embedding Tensors. Alternatively, a PartitionedVariable created by partitioning along dimension 0. The total unpartitioned shape should be [e\_0, e\_1, ..., e\_m], where e\_0 represents the vocab size and e\_1, ..., e\_m are the embedding dimensions.
* **sparse\_ids**: SparseTensor of shape [d\_0, d\_1, ..., d\_n] containing the ids. d\_0 is typically batch size.
* **sparse\_weights**: SparseTensor of same shape as sparse\_ids, containing float weights corresponding to sparse\_ids, or None if all weights are be assumed to be 1.0.
* **combiner**: A string specifying how to combine embedding results for each entry. Currently "mean", "sqrtn" and "sum" are supported, with "mean" the default.
* **default\_id**: The id to use for an entry with no features.
* **max\_norm**: If not None, all embeddings are l2-normalized to max\_norm before combining.
* **name**: A name for this operation (optional).

#### Returns:

Dense Tensor of shape [d\_0, d\_1, ..., d\_{n-1}, e\_1, ..., e\_m].

#### Raises:

* **ValueError**: if embedding\_weights is empty.

# tf.nn.sampled\_softmax\_loss

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sampled_softmax_loss#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sampled_softmax_loss#aliases)

Computes and returns the sampled softmax training loss.

### Aliases:

* tf.compat.v2.nn.sampled\_softmax\_loss
* tf.nn.sampled\_softmax\_loss

tf.nn.sampled\_softmax\_loss(  
    weights,  
    biases,  
    labels,  
    inputs,  
    num\_sampled,  
    num\_classes,  
    num\_true=1,  
    sampled\_values=None,  
    remove\_accidental\_hits=True,  
    seed=None,  
    name='sampled\_softmax\_loss'  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

This is a faster way to train a softmax classifier over a huge number of classes.

This operation is for training only. It is generally an underestimate of the full softmax loss.

A common use case is to use this method for training, and calculate the full sigmoid loss for evaluation or inference as in the following example:

if mode == "train":  
  loss = tf.nn.sampled\_softmax\_loss(  
      weights=weights,  
      biases=biases,  
      labels=labels,  
      inputs=inputs,  
      ...)  
elif mode == "eval":  
  logits = tf.matmul(inputs, tf.transpose(weights))  
  logits = tf.nn.bias\_add(logits, biases)  
  labels\_one\_hot = tf.one\_hot(labels, n\_classes)  
  loss = tf.nn.softmax\_cross\_entropy\_with\_logits(  
      labels=labels\_one\_hot,  
      logits=logits)

See our [Candidate Sampling Algorithms Reference](https://www.tensorflow.org/extras/candidate_sampling.pdf)

Also see Section 3 of [Jean et al., 2014](http://arxiv.org/abs/1412.2007) ([pdf](http://arxiv.org/pdf/1412.2007.pdf)) for the math.

**Note:** when doing embedding lookup on **weights** and **bias**, "div" partition strategy will be used. Support for other partition strategy will be added later.

#### Args:

* **weights**: A Tensor of shape [num\_classes, dim], or a list of Tensor objects whose concatenation along dimension 0 has shape [num\_classes, dim]. The (possibly-sharded) class embeddings.
* **biases**: A Tensor of shape [num\_classes]. The class biases.
* **labels**: A Tensor of type int64 and shape [batch\_size, num\_true]. The target classes. Note that this format differs from the labels argument of nn.softmax\_cross\_entropy\_with\_logits.
* **inputs**: A Tensor of shape [batch\_size, dim]. The forward activations of the input network.
* **num\_sampled**: An int. The number of classes to randomly sample per batch.
* **num\_classes**: An int. The number of possible classes.
* **num\_true**: An int. The number of target classes per training example.
* **sampled\_values**: a tuple of (sampled\_candidates, true\_expected\_count,sampled\_expected\_count) returned by a \*\_candidate\_sampler function. (if None, we default to log\_uniform\_candidate\_sampler)
* **remove\_accidental\_hits**: A bool. whether to remove "accidental hits" where a sampled class equals one of the target classes. Default is True.
* **seed**: random seed for candidate sampling. Default to None, which doesn't set the op-level random seed for candidate sampling.
* **name**: A name for the operation (optional).

#### Returns:

A batch\_size 1-D tensor of per-example sampled softmax losses.

# tf.nn.scale\_regularization\_loss

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/scale_regularization_loss#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/scale_regularization_loss#aliases)

Scales the sum of the given regularization losses by number of replicas.

### Aliases:

* tf.compat.v1.nn.scale\_regularization\_loss
* tf.compat.v2.nn.scale\_regularization\_loss
* tf.nn.scale\_regularization\_loss

tf.nn.scale\_regularization\_loss(regularization\_loss)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

Usage with distribution strategy and custom training loop:

with strategy.scope():  
  def compute\_loss(self, label, predictions):  
    per\_example\_loss = tf.keras.losses.sparse\_categorical\_crossentropy(  
        labels, predictions)  
  
    # Compute loss that is scaled by sample\_weight and by global batch size.  
    loss = tf.compute\_average\_loss(  
        per\_example\_loss,  
        sample\_weight=sample\_weight,  
        global\_batch\_size=GLOBAL\_BATCH\_SIZE)  
  
    # Add scaled regularization losses.  
    loss += tf.scale\_regularization\_loss(tf.nn.l2\_loss(weights))  
    return loss

#### Args:

* **regularization\_loss**: Regularization loss.

#### Returns:

Scalar loss value.

# tf.nn.selu

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/selu#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/selu#aliases)

Computes scaled exponential linear: scale \* alpha \* (exp(features) - 1)

### Aliases:

* tf.compat.v1.nn.selu
* tf.compat.v2.nn.selu
* tf.nn.selu

tf.nn.selu(  
    features,  
    name=None  
)

Defined in generated file: python/ops/gen\_nn\_ops.py.

if < 0, scale \* features otherwise.

To be used together with initializer = tf.variance\_scaling\_initializer(factor=1.0, mode='FAN\_IN'). For correct dropout, use tf.contrib.nn.alpha\_dropout.

See [Self-Normalizing Neural Networks](https://arxiv.org/abs/1706.02515)

#### Args:

* **features**: A Tensor. Must be one of the following types: half, bfloat16, float32, float64.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as features.

# tf.nn.separable\_conv2d

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/separable_conv2d#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/separable_conv2d#aliases)

2-D convolution with separable filters.

### Aliases:

* tf.compat.v2.nn.separable\_conv2d
* tf.nn.separable\_conv2d

tf.nn.separable\_conv2d(  
    input,  
    depthwise\_filter,  
    pointwise\_filter,  
    strides,  
    padding,  
    data\_format=None,  
    dilations=None,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

Performs a depthwise convolution that acts separately on channels followed by a pointwise convolution that mixes channels. Note that this is separability between dimensions [1, 2] and 3, not spatial separability between dimensions 1 and 2.

In detail, with the default NHWC format,

output[b, i, j, k] = sum\_{di, dj, q, r}  
    input[b, strides[1] \* i + di, strides[2] \* j + dj, q] \*  
    depthwise\_filter[di, dj, q, r] \*  
    pointwise\_filter[0, 0, q \* channel\_multiplier + r, k]

strides controls the strides for the depthwise convolution only, since the pointwise convolution has implicit strides of [1, 1, 1, 1]. Must have strides[0] = strides[3] = 1. For the most common case of the same horizontal and vertical strides, strides = [1, stride, stride, 1]. If any value in rate is greater than 1, we perform atrous depthwise convolution, in which case all values in the strides tensor must be equal to 1.

#### Args:

* **input**: 4-D Tensor with shape according to data\_format.
* **depthwise\_filter**: 4-D Tensor with shape [filter\_height, filter\_width, in\_channels, channel\_multiplier]. Contains in\_channels convolutional filters of depth 1.
* **pointwise\_filter**: 4-D Tensor with shape [1, 1, channel\_multiplier \* in\_channels, out\_channels]. Pointwise filter to mix channels after depthwise\_filter has convolved spatially.
* **strides**: 1-D of size 4. The strides for the depthwise convolution for each dimension of input.
* **padding**: A string, either 'VALID' or 'SAME'. The padding algorithm. See the "returns" section of [tf.nn.convolution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/convolution) for details.
* **data\_format**: The data format for input. Either "NHWC" (default) or "NCHW".
* **dilations**: 1-D of size 2. The dilation rate in which we sample input values across the heightand width dimensions in atrous convolution. If it is greater than 1, then all values of strides must be 1.
* **name**: A name for this operation (optional).

#### Returns:

A 4-D Tensor with shape according to 'data\_format'. For example, with data\_format="NHWC", shape is [batch, out\_height, out\_width, out\_channels].

# tf.nn.sigmoid\_cross\_entropy\_with\_logits

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits#used_in_the_tutorials)

Computes sigmoid cross entropy given logits.

### Aliases:

* tf.compat.v2.nn.sigmoid\_cross\_entropy\_with\_logits
* tf.nn.sigmoid\_cross\_entropy\_with\_logits

tf.nn.sigmoid\_cross\_entropy\_with\_logits(  
    labels=None,  
    logits=None,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

### Used in the tutorials:

* [Convolutional Variational Autoencoder](https://www.tensorflow.org/beta/tutorials/generative/cvae)

Measures the probability error in discrete classification tasks in which each class is independent and not mutually exclusive. For instance, one could perform multilabel classification where a picture can contain both an elephant and a dog at the same time.

For brevity, let x = logits, z = labels. The logistic loss is

  z \* -log(sigmoid(x)) + (1 - z) \* -log(1 - sigmoid(x))  
= z \* -log(1 / (1 + exp(-x))) + (1 - z) \* -log(exp(-x) / (1 + exp(-x)))  
= z \* log(1 + exp(-x)) + (1 - z) \* (-log(exp(-x)) + log(1 + exp(-x)))  
= z \* log(1 + exp(-x)) + (1 - z) \* (x + log(1 + exp(-x))  
= (1 - z) \* x + log(1 + exp(-x))  
= x - x \* z + log(1 + exp(-x))

For x < 0, to avoid overflow in exp(-x), we reformulate the above

  x - x \* z + log(1 + exp(-x))  
= log(exp(x)) - x \* z + log(1 + exp(-x))  
= - x \* z + log(1 + exp(x))

Hence, to ensure stability and avoid overflow, the implementation uses this equivalent formulation

max(x, 0) - x \* z + log(1 + exp(-abs(x)))

logits and labels must have the same type and shape.

#### Args:

* **labels**: A Tensor of the same type and shape as logits.
* **logits**: A Tensor of type float32 or float64.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of the same shape as logits with the componentwise logistic losses.

#### Raises:

* **ValueError**: If logits and labels do not have the same shape.

# tf.nn.softmax

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softmax#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softmax#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softmax#used_in_the_tutorials)

Computes softmax activations.

### Aliases:

* tf.compat.v2.math.softmax
* tf.compat.v2.nn.softmax
* tf.math.softmax
* tf.nn.softmax

tf.nn.softmax(  
    logits,  
    axis=None,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

### Used in the tutorials:

* [Custom training: walkthrough](https://www.tensorflow.org/beta/tutorials/eager/custom_training_walkthrough)
* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)
* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)

This function performs the equivalent of

softmax = tf.exp(logits) / tf.reduce\_sum(tf.exp(logits), axis)

#### Args:

* **logits**: A non-empty Tensor. Must be one of the following types: half, float32, float64.
* **axis**: The dimension softmax would be performed on. The default is -1 which indicates the last dimension.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type and shape as logits.

#### Raises:

* **InvalidArgumentError**: if logits is empty or axis is beyond the last dimension of logits.

# tf.nn.softmax\_cross\_entropy\_with\_logits

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softmax_cross_entropy_with_logits#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softmax_cross_entropy_with_logits#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softmax_cross_entropy_with_logits#used_in_the_guide)

Computes softmax cross entropy between logits and labels.

### Aliases:

* tf.compat.v2.nn.softmax\_cross\_entropy\_with\_logits
* tf.nn.softmax\_cross\_entropy\_with\_logits

tf.nn.softmax\_cross\_entropy\_with\_logits(  
    labels,  
    logits,  
    axis=-1,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

### Used in the guide:

* [Distributed training in TensorFlow](https://www.tensorflow.org/beta/guide/distribute_strategy)

Measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class). For example, each CIFAR-10 image is labeled with one and only one label: an image can be a dog or a truck, but not both.

**NOTE:** While the classes are mutually exclusive, their probabilities need not be. All that is required is that each row of labels is a valid probability distribution. If they are not, the computation of the gradient will be incorrect.

If using exclusive labels (wherein one and only one class is true at a time), see sparse\_softmax\_cross\_entropy\_with\_logits.

**WARNING:** This op expects unscaled logits, since it performs a softmax on logits internally for efficiency. Do not call this op with the output of softmax, as it will produce incorrect results.

A common use case is to have logits and labels of shape [batch\_size, num\_classes], but higher dimensions are supported, with the axis argument specifying the class dimension.

logits and labels must have the same dtype (either float16, float32, or float64).

Backpropagation will happen into both logits and labels. To disallow backpropagation into labels, pass label tensors through [tf.stop\_gradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stop_gradient) before feeding it to this function.

**Note that to avoid confusion, it is required to pass only named arguments to this function.**

#### Args:

* **labels**: Each vector along the class dimension should hold a valid probability distribution e.g. for the case in which labels are of shape [batch\_size, num\_classes], each row of labels[i] must be a valid probability distribution.
* **logits**: Per-label activations, typically a linear output. These activation energies are interpreted as unnormalized log probabilities.
* **axis**: The class dimension. Defaulted to -1 which is the last dimension.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor that contains the softmax cross entropy loss. Its type is the same as logits and its shape is the same as labels except that it does not have the last dimension of labels.

# tf.nn.softsign

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softsign#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/softsign#aliases)

Computes softsign: features / (abs(features) + 1).

### Aliases:

* tf.compat.v1.math.softsign
* tf.compat.v1.nn.softsign
* tf.compat.v2.math.softsign
* tf.compat.v2.nn.softsign
* tf.math.softsign
* tf.nn.softsign

tf.nn.softsign(  
    features,  
    name=None  
)

Defined in generated file: python/ops/gen\_nn\_ops.py.

#### Args:

* **features**: A Tensor. Must be one of the following types: half, bfloat16, float32, float64.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as features.

# tf.nn.space\_to\_depth

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/space_to_depth#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/space_to_depth#aliases)

SpaceToDepth for tensors of type T.

### Aliases:

* tf.compat.v2.nn.space\_to\_depth
* tf.nn.space\_to\_depth

tf.nn.space\_to\_depth(  
    input,  
    block\_size,  
    data\_format='NHWC',  
    name=None  
)

Defined in [python/ops/array\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/array_ops.py).

Rearranges blocks of spatial data, into depth. More specifically, this op outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimension. The attr block\_size indicates the input block size.

* Non-overlapping blocks of size block\_size x block size are rearranged into depth at each location.
* The depth of the output tensor is block\_size \* block\_size \* input\_depth.
* The Y, X coordinates within each block of the input become the high order component of the output channel index.
* The input tensor's height and width must be divisible by block\_size.

The data\_format attr specifies the layout of the input and output tensors with the following options: "NHWC": [ batch, height, width, channels ] "NCHW": [ batch, channels, height, width ] "NCHW\_VECT\_C": qint8 [ batch, channels / 4, height, width, 4 ]

It is useful to consider the operation as transforming a 6-D Tensor. e.g. for data\_format = NHWC, Each element in the input tensor can be specified via 6 coordinates, ordered by decreasing memory layout significance as: n,oY,bY,oX,bX,iC (where n=batch index, oX, oY means X or Y coordinates within the output image, bX, bY means coordinates within the input block, iC means input channels). The output would be a transpose to the following layout: n,oY,oX,bY,bX,iC

This operation is useful for resizing the activations between convolutions (but keeping all data), e.g. instead of pooling. It is also useful for training purely convolutional models.

For example, given an input of shape [1, 2, 2, 1], data\_format = "NHWC" and block\_size = 2:

x = [[[[1], [2]],  
      [[3], [4]]]]

This operation will output a tensor of shape [1, 1, 1, 4]:

[[[[1, 2, 3, 4]]]]

Here, the input has a batch of 1 and each batch element has shape [2, 2, 1], the corresponding output will have a single element (i.e. width and height are both 1) and will have a depth of 4 channels (1 \* block\_size \* block\_size). The output element shape is [1, 1, 4].

For an input tensor with larger depth, here of shape [1, 2, 2, 3], e.g.

x = [[[[1, 2, 3], [4, 5, 6]],  
      [[7, 8, 9], [10, 11, 12]]]]

This operation, for block\_size of 2, will return the following tensor of shape [1, 1, 1, 12]

[[[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]]]]

Similarly, for the following input of shape [1 4 4 1], and a block size of 2:

x = [[[[1],   [2],  [5],  [6]],  
      [[3],   [4],  [7],  [8]],  
      [[9],  [10], [13],  [14]],  
      [[11], [12], [15],  [16]]]]

the operator will return the following tensor of shape [1 2 2 4]:

x = [[[[1, 2, 3, 4],  
       [5, 6, 7, 8]],  
      [[9, 10, 11, 12],  
       [13, 14, 15, 16]]]]

#### Args:

* **input**: A Tensor.
* **block\_size**: An int that is >= 2. The size of the spatial block.
* **data\_format**: An optional string from: "NHWC", "NCHW", "NCHW\_VECT\_C". Defaults to "NHWC".
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sparse_softmax_cross_entropy_with_logits#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sparse_softmax_cross_entropy_with_logits#aliases)

Computes sparse softmax cross entropy between logits and labels.

### Aliases:

* tf.compat.v2.nn.sparse\_softmax\_cross\_entropy\_with\_logits
* tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits

tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(  
    labels,  
    logits,  
    name=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

Measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class). For example, each CIFAR-10 image is labeled with one and only one label: an image can be a dog or a truck, but not both.

**NOTE:** For this operation, the probability of a given label is considered exclusive. That is, soft classes are not allowed, and the labels vector must provide a single specific index for the true class for each row of logits (each minibatch entry). For soft softmax classification with a probability distribution for each entry, see softmax\_cross\_entropy\_with\_logits\_v2.

**WARNING:** This op expects unscaled logits, since it performs a softmax on logits internally for efficiency. Do not call this op with the output of softmax, as it will produce incorrect results.

A common use case is to have logits of shape [batch\_size, num\_classes] and have labels of shape [batch\_size], but higher dimensions are supported, in which case the dim-th dimension is assumed to be of size num\_classes. logits must have the dtype of float16, float32, or float64, and labels must have the dtype of int32 or int64.

**Note that to avoid confusion, it is required to pass only named arguments to this function.**

#### Args:

* **labels**: Tensor of shape [d\_0, d\_1, ..., d\_{r-1}] (where r is rank of labels and result) and dtype int32 or int64. Each entry in labels must be an index in [0, num\_classes). Other values will raise an exception when this op is run on CPU, and return NaNfor corresponding loss and gradient rows on GPU.
* **logits**: Unscaled log probabilities of shape [d\_0, d\_1, ..., d\_{r-1}, num\_classes] and dtype float16, float32, or float64.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of the same shape as labels and of the same type as logits with the softmax cross entropy loss.

#### Raises:

* **ValueError**: If logits are scalars (need to have rank >= 1) or if the rank of the labels is not equal to the rank of the logits minus one.

# tf.nn.sufficient\_statistics

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sufficient_statistics#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/sufficient_statistics#aliases)

Calculate the sufficient statistics for the mean and variance of x.

### Aliases:

* tf.compat.v2.nn.sufficient\_statistics
* tf.nn.sufficient\_statistics

tf.nn.sufficient\_statistics(  
    x,  
    axes,  
    shift=None,  
    keepdims=False,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

These sufficient statistics are computed using the one pass algorithm on an input that's optionally shifted. See: https://en.wikipedia.org/wiki/Algorithms\_for\_calculating\_variance#Computing\_shifted\_data

#### Args:

* **x**: A Tensor.
* **axes**: Array of ints. Axes along which to compute mean and variance.
* **shift**: A Tensor containing the value by which to shift the data for numerical stability, or Noneif no shift is to be performed. A shift close to the true mean provides the most numerically stable results.
* **keepdims**: produce statistics with the same dimensionality as the input.
* **name**: Name used to scope the operations that compute the sufficient stats.

#### Returns:

Four Tensor objects of the same type as x:

* the count (number of elements to average over).
* the (possibly shifted) sum of the elements in the array.
* the (possibly shifted) sum of squares of the elements in the array.
* the shift by which the mean must be corrected or None if shift is None.

# tf.nn.weighted\_cross\_entropy\_with\_logits

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/weighted_cross_entropy_with_logits#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/weighted_cross_entropy_with_logits#aliases)

Computes a weighted cross entropy.

### Aliases:

* tf.compat.v2.nn.weighted\_cross\_entropy\_with\_logits
* tf.nn.weighted\_cross\_entropy\_with\_logits

tf.nn.weighted\_cross\_entropy\_with\_logits(  
    labels,  
    logits,  
    pos\_weight,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

This is like sigmoid\_cross\_entropy\_with\_logits() except that pos\_weight, allows one to trade off recall and precision by up- or down-weighting the cost of a positive error relative to a negative error.

The usual cross-entropy cost is defined as:

labels \* -log(sigmoid(logits)) +  
    (1 - labels) \* -log(1 - sigmoid(logits))

A value pos\_weights > 1 decreases the false negative count, hence increasing the recall. Conversely setting pos\_weights < 1 decreases the false positive count and increases the precision. This can be seen from the fact that pos\_weight is introduced as a multiplicative coefficient for the positive labels term in the loss expression:

labels \* -log(sigmoid(logits)) \* pos\_weight +  
    (1 - labels) \* -log(1 - sigmoid(logits))

For brevity, let x = logits, z = labels, q = pos\_weight. The loss is:

  qz \* -log(sigmoid(x)) + (1 - z) \* -log(1 - sigmoid(x))  
= qz \* -log(1 / (1 + exp(-x))) + (1 - z) \* -log(exp(-x) / (1 + exp(-x)))  
= qz \* log(1 + exp(-x)) + (1 - z) \* (-log(exp(-x)) + log(1 + exp(-x)))  
= qz \* log(1 + exp(-x)) + (1 - z) \* (x + log(1 + exp(-x))  
= (1 - z) \* x + (qz +  1 - z) \* log(1 + exp(-x))  
= (1 - z) \* x + (1 + (q - 1) \* z) \* log(1 + exp(-x))

Setting l = (1 + (q - 1) \* z), to ensure stability and avoid overflow, the implementation uses

(1 - z) \* x + l \* (log(1 + exp(-abs(x))) + max(-x, 0))

logits and labels must have the same type and shape.

#### Args:

* **labels**: A Tensor of the same type and shape as logits.
* **logits**: A Tensor of type float32 or float64.
* **pos\_weight**: A coefficient to use on the positive examples.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of the same shape as logits with the componentwise weighted logistic losses.

#### Raises:

* **ValueError**: If logits and labels do not have the same shape.

# tf.nn.weighted\_moments

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/weighted_moments#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/weighted_moments#aliases)

Returns the frequency-weighted mean and variance of x.

### Aliases:

* tf.compat.v2.nn.weighted\_moments
* tf.nn.weighted\_moments

tf.nn.weighted\_moments(  
    x,  
    axes,  
    frequency\_weights,  
    keepdims=False,  
    name=None  
)

Defined in [python/ops/nn\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_impl.py).

#### Args:

* **x**: A tensor.
* **axes**: 1-d tensor of int32 values; these are the axes along which to compute mean and variance.
* **frequency\_weights**: A tensor of positive weights which can be broadcast with x.
* **keepdims**: Produce moments with the same dimensionality as the input.
* **name**: Name used to scope the operation.

#### Returns:

Two tensors: weighted\_mean and weighted\_variance.

# tf.nn.with\_space\_to\_batch

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/with_space_to_batch#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/with_space_to_batch#aliases)

Performs op on the space-to-batch representation of input.

### Aliases:

* tf.compat.v1.nn.with\_space\_to\_batch
* tf.compat.v2.nn.with\_space\_to\_batch
* tf.nn.with\_space\_to\_batch

tf.nn.with\_space\_to\_batch(  
    input,  
    dilation\_rate,  
    padding,  
    op,  
    filter\_shape=None,  
    spatial\_dims=None,  
    data\_format=None  
)

Defined in [python/ops/nn\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/nn_ops.py).

This has the effect of transforming sliding window operations into the corresponding "atrous" operation in which the input is sampled at the specified dilation\_rate.

In the special case that dilation\_rate is uniformly 1, this simply returns:

op(input, num\_spatial\_dims, padding)

Otherwise, it returns:

batch\_to\_space\_nd( op(space\_to\_batch\_nd(input, adjusted\_dilation\_rate, adjusted\_paddings), num\_spatial\_dims, "VALID") adjusted\_dilation\_rate, adjusted\_crops),

where:

adjusted\_dilation\_rate is an int64 tensor of shape [max(spatialdims)], adjusted{paddings,crops} are int64 tensors of shape [max(spatial\_dims), 2]

defined as follows:

We first define two int64 tensors paddings and crops of shape [num\_spatial\_dims, 2] based on the value of padding and the spatial dimensions of the input:

If padding = "VALID", then:

paddings, crops = required\_space\_to\_batch\_paddings( input\_shape[spatial\_dims], dilation\_rate)

If padding = "SAME", then:

dilated\_filter\_shape = filter\_shape + (filter\_shape - 1) \* (dilation\_rate - 1)

paddings, crops = required\_space\_to\_batch\_paddings( input\_shape[spatial\_dims], dilation\_rate, [(dilated\_filter\_shape - 1) // 2, dilated\_filter\_shape - 1 - (dilated\_filter\_shape - 1) // 2])

Because space\_to\_batch\_nd and batch\_to\_space\_nd assume that the spatial dimensions are contiguous starting at the second dimension, but the specified spatial\_dims may not be, we must adjust dilation\_rate, paddings and crops in order to be usable with these operations. For a given dimension, if the block size is 1, and both the starting and ending padding and crop amounts are 0, then space\_to\_batch\_nd effectively leaves that dimension alone, which is what is needed for dimensions not part of spatial\_dims. Furthermore, space\_to\_batch\_nd and batch\_to\_space\_ndhandle this case efficiently for any number of leading and trailing dimensions.

For 0 <= i < len(spatial\_dims), we assign:

adjusted\_dilation\_rate[spatial\_dims[i] - 1] = dilation\_rate[i] adjusted\_paddings[spatial\_dims[i] - 1, :] = paddings[i, :] adjusted\_crops[spatial\_dims[i] - 1, :] = crops[i, :]

All unassigned values of adjusted\_dilation\_rate default to 1, while all unassigned values of adjusted\_paddings and adjusted\_crops default to 0.

Note in the case that dilation\_rate is not uniformly 1, specifying "VALID" padding is equivalent to specifying padding = "SAME" with a filter\_shape of [1]\*N.

Advanced usage. Note the following optimization: A sequence of with\_space\_to\_batch operations with identical (not uniformly 1) dilation\_rate parameters and "VALID" padding

net = with\_space\_to\_batch(net, dilation\_rate, "VALID", op\_1) ... net = with\_space\_to\_batch(net, dilation\_rate, "VALID", op\_k)

can be combined into a single with\_space\_to\_batch operation as follows:

def combined\_op(converted\_input, num\_spatial\_dims, \_): result = op\_1(converted\_input, num\_spatial\_dims, "VALID") ... result = op\_k(result, num\_spatial\_dims, "VALID")

net = with\_space\_to\_batch(net, dilation\_rate, "VALID", combined\_op)

This eliminates the overhead of k-1 calls to space\_to\_batch\_nd and batch\_to\_space\_nd.

Similarly, a sequence of with\_space\_to\_batch operations with identical (not uniformly 1) dilation\_rate parameters, "SAME" padding, and odd filter dimensions

net = with\_space\_to\_batch(net, dilation\_rate, "SAME", op\_1, filter\_shape\_1) ... net = with\_space\_to\_batch(net, dilation\_rate, "SAME", op\_k, filter\_shape\_k)

can be combined into a single with\_space\_to\_batch operation as follows:

def combined\_op(converted\_input, num\_spatial\_dims, \_): result = op\_1(converted\_input, num\_spatial\_dims, "SAME") ... result = op\_k(result, num\_spatial\_dims, "SAME")

net = with\_space\_to\_batch(net, dilation\_rate, "VALID", combined\_op)

#### Args:

* **input**: Tensor of rank > max(spatial\_dims).
* **dilation\_rate**: int32 Tensor of known shape [num\_spatial\_dims].
* **padding**: str constant equal to "VALID" or "SAME"
* **op**: Function that maps (input, num\_spatial\_dims, padding) -> output
* **filter\_shape**: If padding = "SAME", specifies the shape of the convolution kernel/pooling window as an integer Tensor of shape [>=num\_spatial\_dims]. If padding = "VALID", filter\_shape is ignored and need not be specified.
* **spatial\_dims**: Monotonically increasing sequence of num\_spatial\_dims integers (which are >= 1) specifying the spatial dimensions of input and output. Defaults to: range(1, num\_spatial\_dims+1).
* **data\_format**: A string or None. Specifies whether the channel dimension of the input and output is the last dimension (default, or if data\_format does not start with "NC"), or the second dimension (if data\_format starts with "NC"). For N=1, the valid values are "NWC" (default) and "NCW". For N=2, the valid values are "NHWC" (default) and "NCHW". For N=3, the valid values are "NDHWC" (default) and "NCDHW".

#### Returns:

The output Tensor as described above, dimensions will vary based on the op provided.

#### Raises:

* **ValueError**: if padding is invalid or the arguments are incompatible.
* **ValueError**: if spatial\_dims are invalid.