1 Conv2d

Applies a 2D convolution over an input tensor which is typically of shape $(N, C_{in}, H_{in}, W_{in})$, where N is batch size, C_{in} is channel number, and (H_{in}, W_{in}) are height and width. For each batch of shape (C_{in}, H_{in}, W_{in}), the formula is defined as:

$$out_j = \sum_{i=0}^{C_{in}-1} ccor(W_{ij}, X_i) + b_j$$

where *ccor* is the cross-correlation operator, C_{in} is the input channel number, j ranges from 0 to $C_{out}-1$, W_{ij} corresponds to the i-th channel of the j-th filter and out_j corresponds to the j-th channel of the output. W_{ij} is a slice of kernel and it has shape (ks_h, ks_h), where ks_h and ks_h are the height and width of the convolution kernel. The full kernel has shape $(C_{out}, C_{in}//group, ks_h, ks_h)$, where group is the group number to split the input in the channel dimension.

If the 'pad_mode' is set to be "valid", the output height and width will be

$$\left\lfloor 1 + \frac{H_{in} + 2 \times padding - ks_h - (ks_h - 1) \times (dilation - 1)}{stride} \right\rfloor$$

and

$$\left\lfloor 1 + \frac{W_{in} + 2 \times padding - ks_h - (ks_h - 1) \times (dilation - 1)}{stride} \right\rfloor$$

respectively.

2 BatchNorm2d

Batch normalization layer over a 4D input.

Batch Normalization is widely used in convolutional networks. This layer applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) to avoid internal covariate shift as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift <https://arxiv.org/abs/1502.03167>_. It rescales and recenters the feature using a mini-batch of data and the learned parameters which can be described in the following formula.

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Note: The implementation of BatchNorm is different in graph mode and pynative mode, therefore that mode can not be changed after net was initialized. Note that the formula for updating the running_mean and running_var is

$$\hat{x}_{new} = (1 - momentum) \times x_t + momentum \times \hat{x}$$

, where $\hat{x}$ is the estimated statistic and x_t is the new observed value.

3 ReLU

Rectified Linear Unit activation function.

Applies the rectified linear unit function element-wise. It returns element-wise $\max(0, x)$, specially, the neurons with the negative output will be suppressed and the active neurons will stay the same.

4 SoftmaxCrossEntropyWithLogits

Computes softmax cross entropy between logits and labels.

Measures the distribution error between the probabilities of the input (computed with softmax function) and the target where the classes are mutually exclusive (only one class is positive) using cross entropy loss.

Typical input into this function is unnormalized scores and target of each class. Scores Tensor x is of shape (N, C) and target Tensor t is a Tensor of shape (N, C) which contains one-hot labels of length C.

For each instance N_i, the loss is given as:

$$\ell(x_i, t_i) = -\log\left(\frac{\exp(x_{t_i})}{\sum_j \exp(x_j)}\right) = -x_{t_i} + \log\left(\sum_j \exp(x_j)\right)$$

where x_i is a 1D score Tensor, t_i is a scalar.

Note: While the target classes are mutually exclusive, i.e., only one class is positive in the target, the predicted probabilities need not to be exclusive. It is only required that the predicted probability distribution of entry is a valid one.