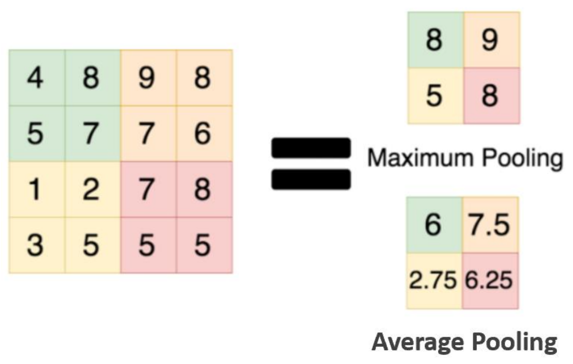
Assignment 3

LI XINYA (G2004358J)

Function 1&2:

torch.nn.MaxPool2d & torch.nn.AvgPool2d



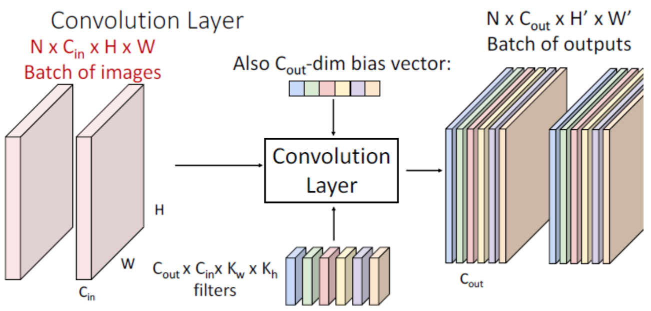
Pooling algorithm:

Input: $C \times H \times W$;

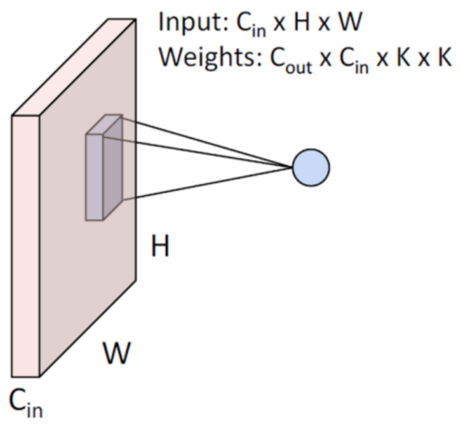
Hyperparameters: Kernel size: K; Stride: S; Pooling function (max, avg);

Output: $C \times H' \times W'$, where $H' = \frac{H-K}{S} + 1$; $W' = \frac{W-K}{S} + 1$.

Function 3: torch.nn.Conv2d



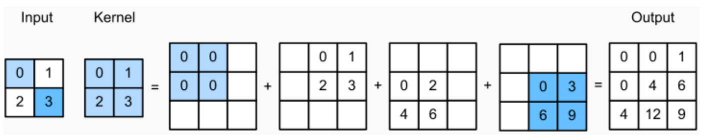
2D Convolution:



If input size is defined as (N, C_{in}, H, W) and output size is defined as $(N, C_{out}, H_{out}, W_{out})$, the output value is:

$$out(N_i, C_{out j}) = bias(C_{out j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out j}, k) * input(N_i, k)$$

Function 4: torch.nn.ConvTranspose2d



Transposed Convolutions are used to upsample the input feature map to a desired output feature map using some learnable parameters. The basic operation of a transposed convolution is explained in the example below: The input shape is 2×2 and needs to be unsampled to 3×3. Then we define a kernel of size 2×2 with stride = 1 and padding = 0. And then take the upper left element of the input and multiply it with every element of the kernel. Similarly, repeat the former operation for all other elements in the input. As you can see in the figure, some of the unsampled elements are overlapping, then add the elements of the over-lapping positions. The output will finally have the required dimensions of 3×3.

Function 5: torch.flatten

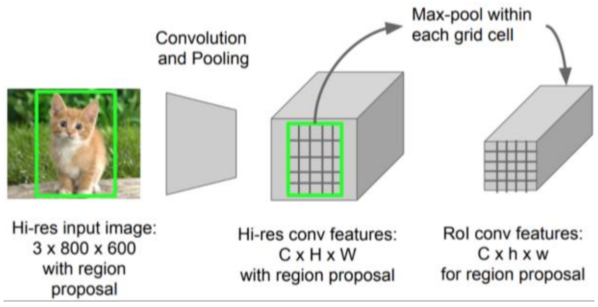
The flatten function is able to reshape the input data into a one-dimensional tensor. The order of elements in input is unchanged. The function returns original objects. If no dimensions are flattened, then the original object input is returned.

Function 6: torch.sigmoid

The sigmoid function is defined as:

$$Out_i = \frac{1}{1 + e^{-input_i}}$$

Function 7: torchvision.ops.roi_pool


Convolution and Pooling
Max-pool within each grid cell
Hi-res input image: 3 x 800 x 600 with region proposal
Hi-res conv features: C x H x W with region proposal
RoI conv features: C x h x w for region proposal

ROI-pooling a type of max-pooling to convert features in the projected region of the image of any size, $h \times w$, into a small fixed window, $H \times W$. The input region is divided into $H \times W$ grids, approximately every sub-window of size $h/H \times w/W$. Then apply max-pooling in each grid. The ROI pooling layer can achieve significant acceleration of training and testing, and improve detection accuracy.

Function 8: torch.nn.functional.batch_norm

Input: $x: N \times D$;

Learnable scale and shift parameters: $\gamma, \beta: D$;

Per-channel mean, shape is D: $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$;

Per-channel std, shape is D: $\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$;

Normalized $x: N \times D$: $\hat{x}_{i,j} = \frac{x_{i,j} - \eta_j}{\sqrt{\sigma_j^2 + \varepsilon}}$;

Output: $y: N \times D$: $y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$.

What's more, if learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function.

Function 9: torch.nn.functional.cross_entropy

This function is useful when training a classification problem with C classed. If provided, the optional argument weight should be a 1D Tensor assigning weight to each of the classes.

The input is expected to contain raw, unnormalized scores for each class. It needs to be a tensor of size $(minibatch, C)$ or $(minibatch, C, d_1, d_2, \dots, d_K)$ with $K \geq 1$ for the K-dimensional case. And the class index is in range $[0, C-1]$ as the target for each value of a 1D tensor of size minibatch.

The loss can be defined as:

$loss(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$;

Or if the weight is added:

$loss(x, class) = weight[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right)$;

And the lossed are averaged across observations for each minibatch.

Function 10: torch.nn.functional.mse_loss

This function is able to measure the mean squared error between each element in the input x and target y.

The unreduced loss can be described as:

$$l(x, y) = L = \{l_n, \dots, l_n\}^T, l_n = (x_n - y_n)^2$$

where N is the batch size, and

$$l(x, y) = \begin{cases} mean(L), if\ reduction =' mean' \\ sum(L), if\ reduction =' sum' \end{cases}$$

x and y are tensors of arbitrary shapes with a total of n element each.

Conclusion:

After comparation, torch_out and my_out are equal up to small numerical errors.