A Brief Introduction of Convolutional Neural Network

Mathematical definition of convolution

The convolution of function f and g is written as $f * g$ and defined as:

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

It is defined as the integral of the product of the two functions after one is reversed and shifted.

Proof of $f * g = g * f$:

Set $u = t - \tau$, we have $\tau = t - u$, so:

$$(f*g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\cdot d\tau$$

$$= \int_{+\infty}^{-\infty} f(t-u)\, g(u) \cdot d(t-u)$$

$$= \int_{+\infty}^{-\infty} f(t-u)\cdot g(u)\cdot d(-u)$$

$$= \int_{-\infty}^{+\infty} f(t-u)\cdot g(u)\cdot du$$

$$= (g * f)(t)$$

Convolution on 2D Image

Convolutional Neural Network is most commonly applied to analyze visual imagery.

Kernel size

The kernel is the number of pixels processed together. We usually use square kernel, e.g., 3×3 or 5×5. You can also define a 3×4 kernel.

Padding

Padding is the addition of (typically) 0-valued pixels on the borders of an image. When the kernel moves to the border of image, there are not enough pixels for convolution. Without padding, we will ignore the borders. If we add padding, we will do the convolution for borders with the missed parts filled with zeros.

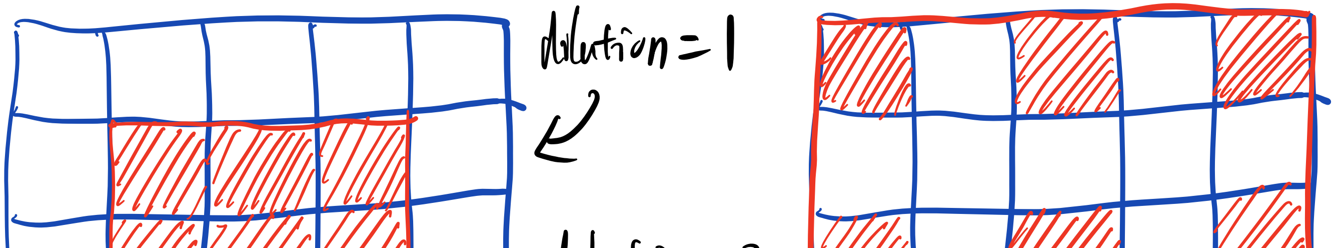
Stride

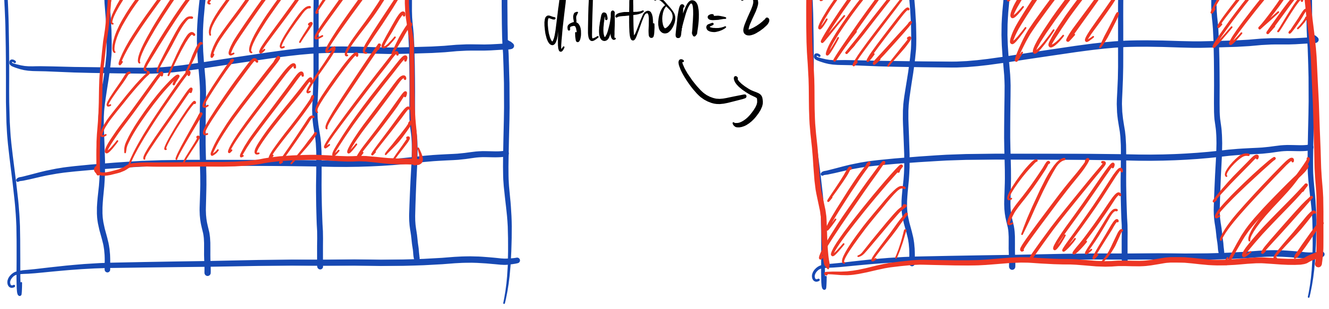
The stride is the number of pixels that the analysis window moves on each iteration. A stride of 2 means that each kernel is offset by 2 pixels from its predecessor.

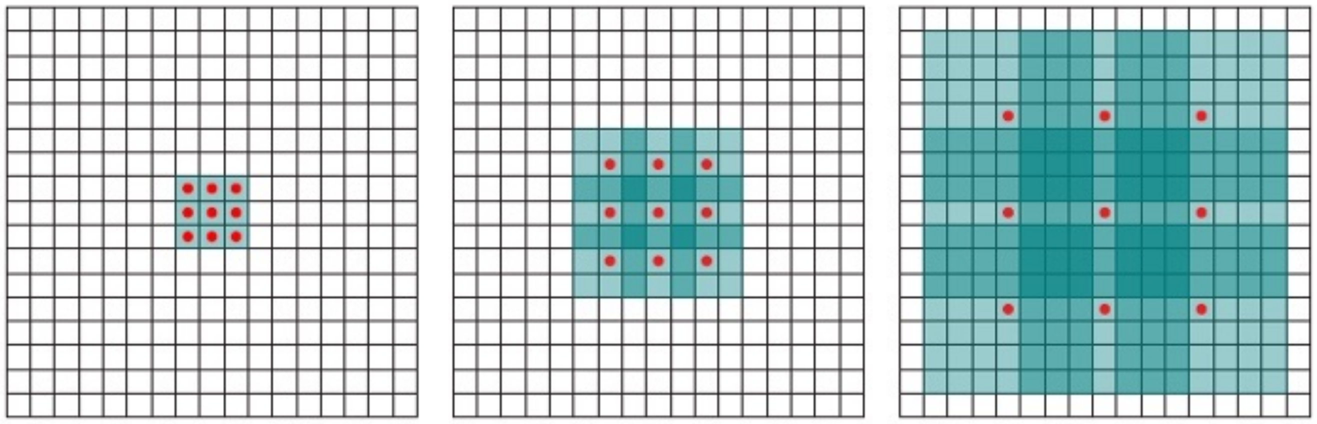
Dilation

Dilation involves ignoring pixels within a kernel. This reduces processing/memory potentially without significant signal loss. A dilation of 2 on a 3x3 kernel expands the kernel to ~~7x7~~ 5x5 while still processing 9 (evenly spaced) pixels.

When a convolutional layer is defined, we can compute the output dimension through following formulas:



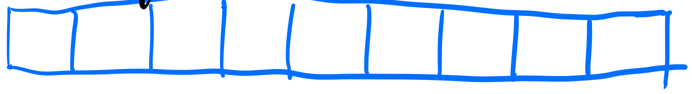




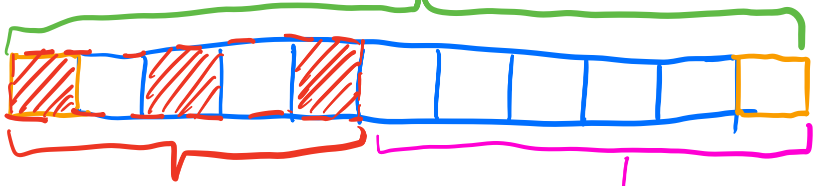
dilation = 1 → dilation = 2 → dilation = 4

Above figure illustrates the receptive fields of three convolutional layers with different dilation if we propagate data forward in them.

Compute the output dimension. Now let's just consider the width (the height is exactly the same).



$l_1 = W_{in} + 2 \times padding[1]$



$l_2 = dilation[1] \times (kernel\_size[1] - 1) + 1$

$l_3 = l_1 - l_2 = W_{in} + 2 \times padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1$

Q: How many steps can the kernel move?

A: $n = \left\lfloor \frac{l_3}{stride[1]} \right\rfloor$

So the width of output is: $\left\lfloor \frac{l_3}{stride[1]} \right\rfloor + 1$

[illegible]

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} + 2 \times \text{ padding } [0] - \text{ dilation } [0] \times (\text{ kernel\_size } [0] - 1) - 1}{\text{stride } [0]} + 1 \right\rfloor$$

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} + 2 \times \text{ padding } [1] - \text{ dilation } [1] \times (\text{ kernel\_size } [1] - 1) - 1}{\text{stride } [1]} + 1 \right\rfloor$$

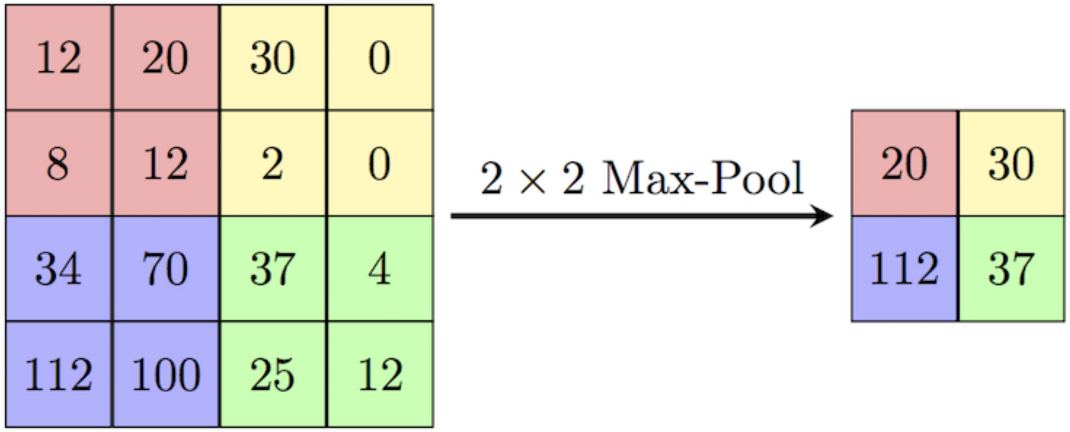
An interactive visualization of convolution: https://ezyang.github.io/convolution-visualizer/index.html

Pooling

Pooling can be used to reduce the dimension.

Max pooling calculates and propagates the maximum value of a given region.

Average pooling calculates and propagates the average value of a given region.



Loss

Cross-Entropy

Given probability distribution q and p, the cross-entropy of q relative to p is defined as:

$$H(p,q) = -\mathrm{E}_p[\log q]$$

For discrete probability distributions p and q, we have:

$$H(p,q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

In lecture3, we have the derivation of log-likelihood of Softmax as follows:

$$
\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{m} \log p\left(y^{(i)} \mid x^{(i)}; \theta\right) \\
&= \sum_{i=1}^{m} \log \prod_{l=1}^{k} p\left(y^{(i)} = l \mid x^{(i)}\right)^{\mathbf{1}\left\{y^{(i)}=l\right\}} \\
&= \sum_{i=1}^{m} \sum_{l=1}^{k} 1\left\{y^{(i)} = l\right\} \log p\left(y^{(i)} = l \mid x^{(i)}\right) \\
&= \sum_{i=1}^{m} \sum_{l=1}^{k} 1\left\{y^{(i)} = l\right\} \log \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}}
\end{aligned}
$$

We can see that maximizing the log-likelihood is equivalent to minimize the cross-entropy loss.

You may also have heard of KL-divergence:

$$
D_{\mathrm{KL}}(p \| q) = \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{p(x)}{q(x)}\right).
$$

We actually have:

$$
\begin{aligned}
D_{\mathrm{KL}}(p\|q) &= \sum_{x\in\mathcal{X}} p(x)\log\left(\frac{p(x)}{q(x)}\right) \\
&= \sum_{x\in\mathcal{X}} p(x)\log p(x) - \sum_{x\in\mathcal{X}} p(x)\log q(x) \\
&= -H(p) + H(p,q)
\end{aligned}
$$

Because p is the true probability, apparently minimizing KL-Divergence is equivalent to minimize the cross-entropy.

Typical architecture

Convolutional layer + Batch normalization/Pooling layer+ ReLU layer + Fully connected layer + Loss layer.

Training a image classifier using Pytorch

(See https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html).

Training dataset: CIFAR10 dataset.

Image dimension: $3\times 32\times 32$

Number of labels: 10

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        # input channel = 3
        # output channel = 6
        # kernel size = 3 (square kernel)
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        # input channel = 6
        # output channel = 16
        # kernel size = 5 (square kernel)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Loss layer:

```
import torch.optim as optim
# actually contains a softmax layer and cross entropy loss
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```