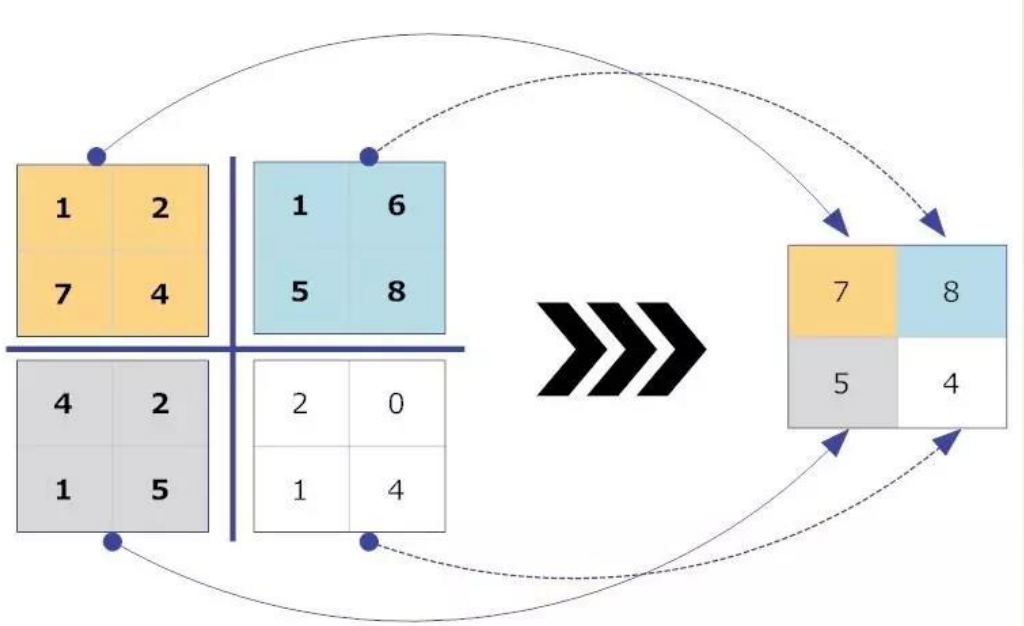
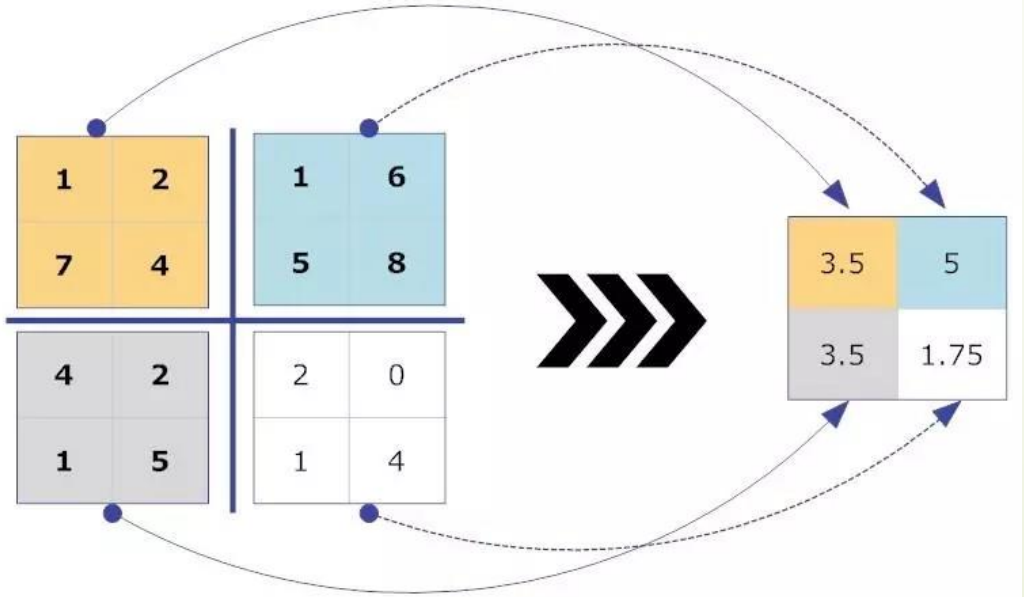
池化层：



（a）最大池化策略



（b）均值池化策略

池化结果：

TensorBoard IMAGES

Show actual image size

Brightness adjustment RESET

Contrast adjustment RESET

Runs

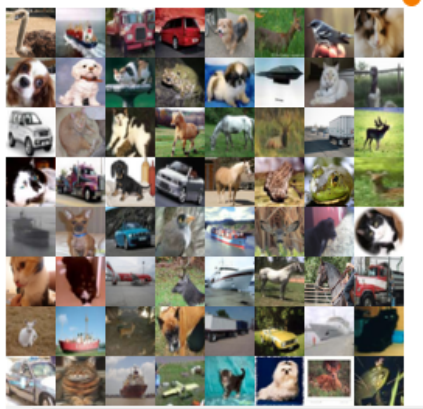
Write a regex to filter runs

TOGGLE ALL RUNS

logs_maxpool

input

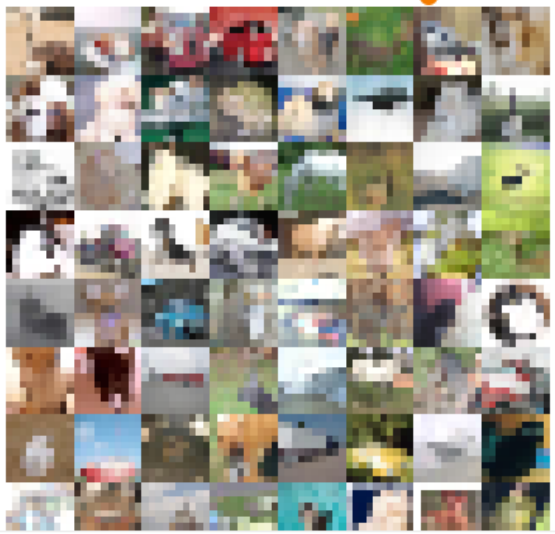
step 115 Wed Aug 04 2021 20:01:32 GMT+0800 (中国标准时间)



output

output

step 115 Wed Aug 04 2021 20:01:32 GMT+0800 (中国标准时间)



池化code：

max_pool2d(input, self.kernel_size, self.stride, self.padding, self.dilation, self.ceil_mode, self.return_indices)

`MaxPool2d` 这个类的实现十分简单。

我们先来看一下基本参数，一共六个：

1. `kernel_size`：表示做最大池化的窗口大小，可以是单个值，也可以是tuple元组
2. `stride`：步长，可以是单个值，也可以是tuple元组
3. `padding`：填充，可以是单个值，也可以是tuple元组
4. `dilation`：控制窗口中元素步幅
5. `return_indices`：布尔类型，返回最大值位置索引
6. `ceil_mode`：布尔类型，为True，用向上取整的方法，计算输出形状；默认是向下取整。

池化code：

max_pool2d(input, self.kernel_size, self.stride, self.padding, self.dilation, self.ceil_mode, self.return_indices)

最大池化层输出形状计算

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

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

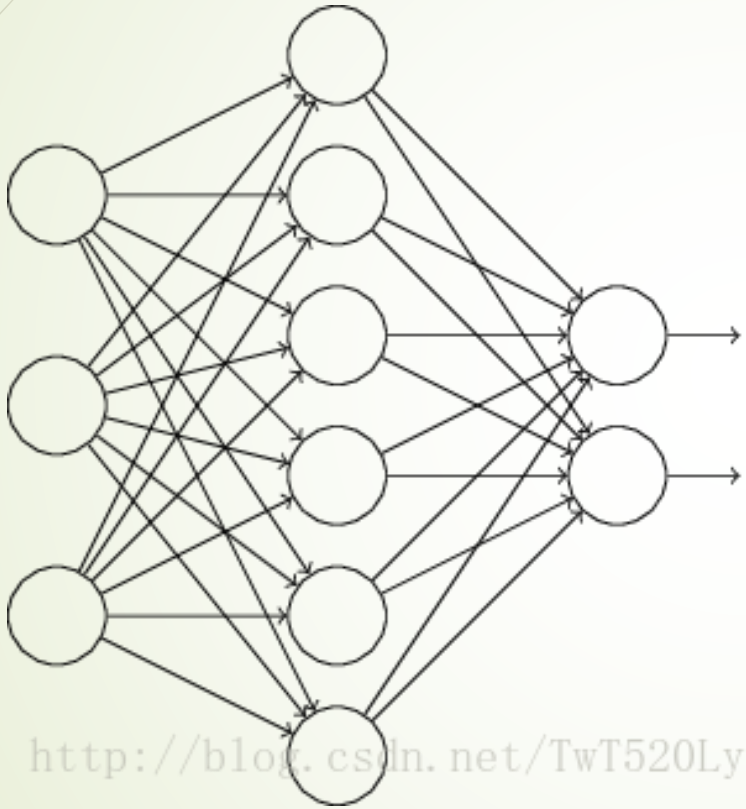
遗忘层：

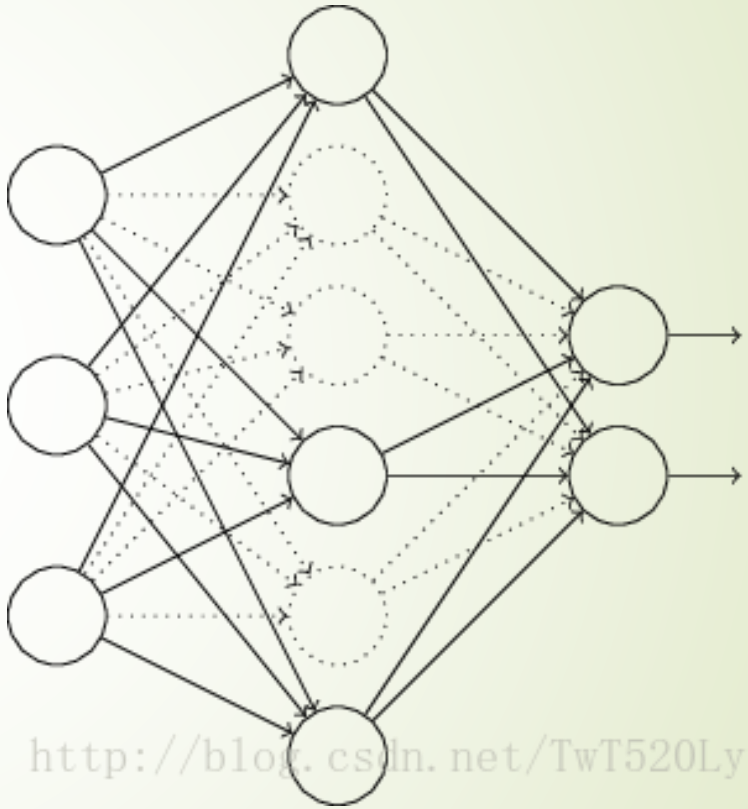
大规模的神经网络有两个缺点：费时、容易过拟合。

过拟合是很多机器学习的通病，过拟合了，得到的模型基本就废了。而为了解决过拟合问题，一般会采用ensemble方法，即训练多个模型做组合，此时，费时就成为一个大问题，不仅训练起来费时，测试起来多个模型也很费时。总之，几乎形成了一个死锁。

Dropout的出现很好的可以解决这个问题，每次做完dropout，相当于从原始的网络中找到一个更瘦的网络遗忘层。

Dropout：





池化code：

Dropout

`torch.nn.Dropout(p=0.5, inplace=False)`

- **p** – probability of an element to be zeroed. Default: 0.5
- **inplace** – If set to `True`, will do this operation in-place. Default: `False`

池化code：

当我们把 `nn.Dropout` 的 `inplace=True` 时，计算的结果就会替换掉原来的输入 `input`，如下：

```
input = torch.tensor([[1, 2, 3],
                      [4, 5, 6],
                      [7, 8, 9]], dtype=torch.float64)
input = torch.unsqueeze(input, 0)
m = nn.Dropout(p = 0.5, inplace=True)
output = m(input)

print("input: ", input)
print("output: ", output)
print("input: ", input)
'''
input:
tensor([[[1., 2., 3.],
         [4., 5., 6.],
         [7., 8., 9.]]], dtype=torch.float64)
output:
tensor([[[ 2.,  4.,  0.],
         [ 0., 10., 12.],
         [ 0., 16.,  0.]]], dtype=torch.float64)
input:
tensor([[[ 2.,  4.,  0.],
         [ 0., 10., 12.],
         [ 0., 16.,  0.]]], dtype=torch.float64)
'''
```