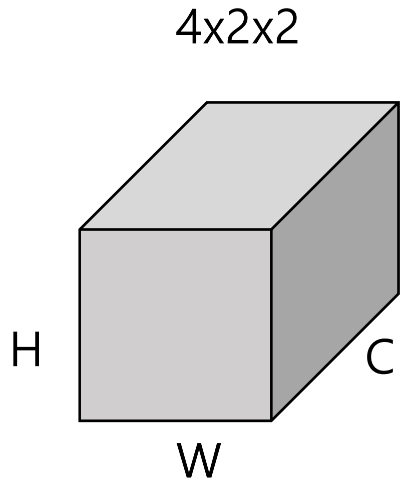
Sigmoid, Softmax in semantic segmentation

2021-02-06

이명오

Sigmoid



Dog

-3	-1
0	-2

Cat

4	1
-1	-3

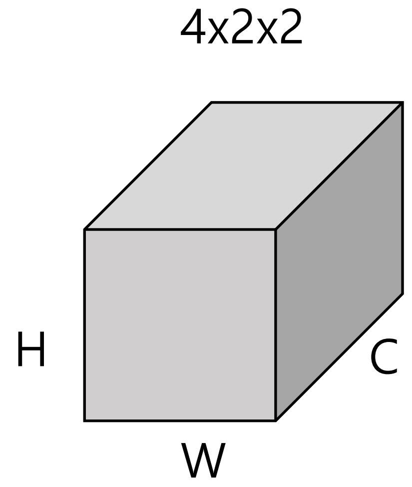
Person

-2	-5
3	-3

Background

-5	2
0	-1

Sigmoid



Dog

-3	-1
0	-2

Cat

4	1
-1	-3

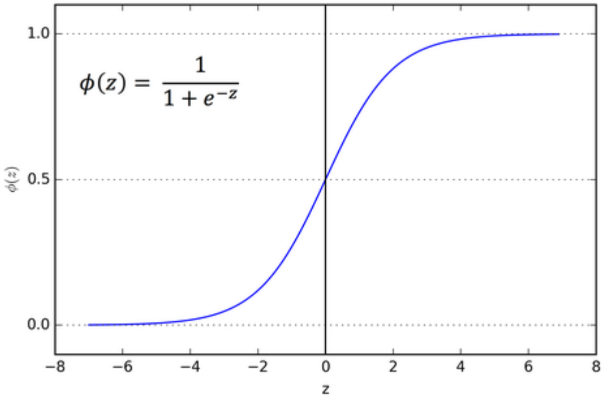
Person

-2	-5
3	-3

Background

-5	2
0	-1

Sigmoid



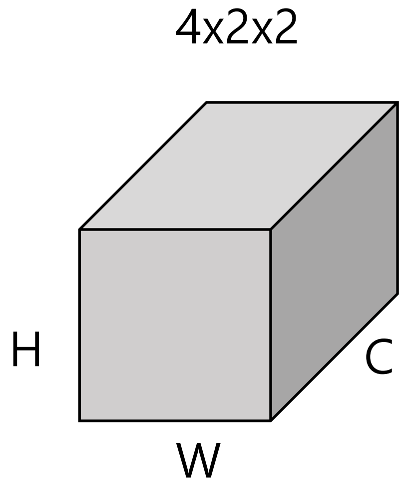
0.0474	0.2689
0.5	0.1192

0.9820	0.7311
0.2689	0.0474

0.1192	0.0067
0.9526	0.0474

0.0067	0.8808
0.5	0.2689

Sigmoid



Dog		Cat		Person		Background	
-3	-1	4	1	-2	-5	-5	2
0	-2	-1	-3	3	-3	0	-1

Sigmoid

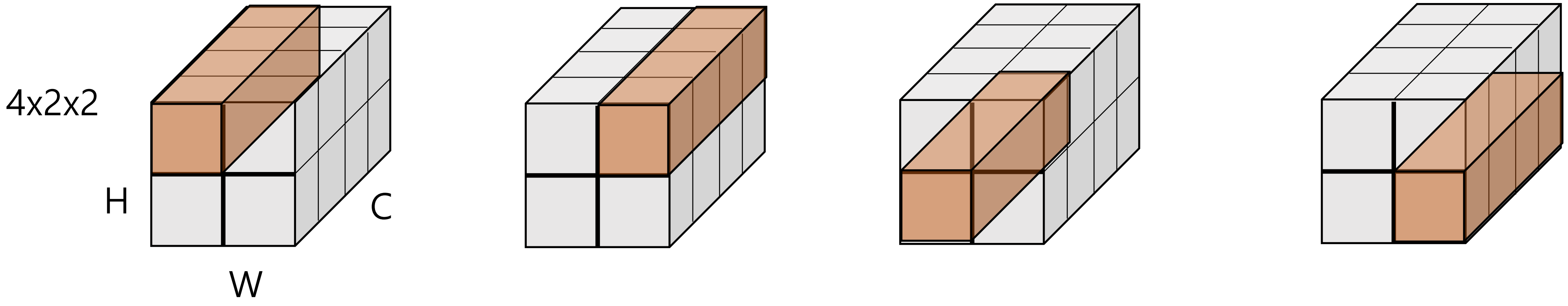
0.0474	0.2689	0.9820	$\boxed{0.7311}$	0.1192	0.0067	0.0067	$\boxed{0.8808}$
0.5	0.1192	0.2689	0.0474	0.9526	0.0474	0.5	0.2689

(0, 1) is Cat? or Background?

Softmax, along channel axis

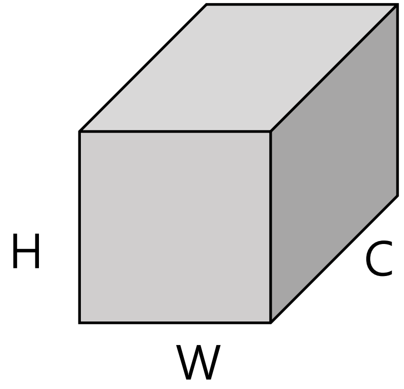
SOFTMAX

CLASS `torch.nn.Softmax(dim: Optional[int] = None)`



Softmax, along channel axis

4x2x2



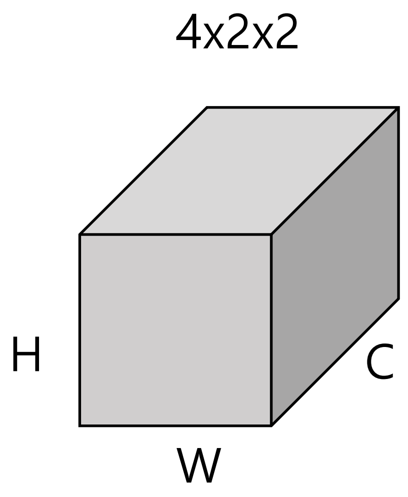
Dog		Cat		Person		Background	
-3	-1	4	1	-2	-5	-5	2
0	-2	-1	-3	3	-3	0	-1

Softmax, along channel axis

0.0009	0.0351	0.9965	0.2593	0.0025	0.0006	0.0001	0.7049
0.0445	0.2245	0.0164	0.0826	0.8945	0.0826	0.0445	0.6103

0.0009 + 0.9965 + 0.0025 + 0.0001 = 1

Softmax, along channel axis



Dog

-3	-1
0	-2

Cat

4	1
-1	-3

Person

-2	-5
3	-3

Background

-5	2
0	-1

Softmax, along channel axis

0.0009	0.0351
0.0445	0.2245

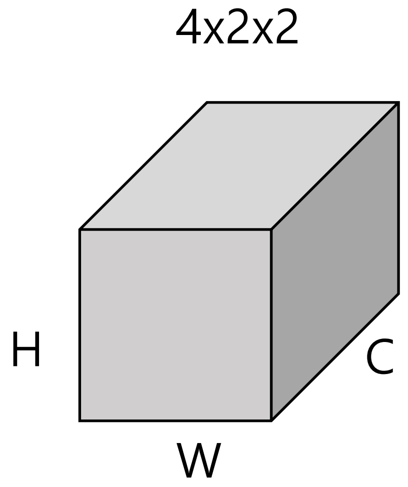
0.9965	0.2593
0.0164	0.0826

0.0025	0.0006
0.8945	0.0826

0.0001	0.7049
0.0445	0.6103

$$0.0351 + 0.2593 + 0.0006 + 0.7049 = 1$$

Softmax, along channel axis



Dog

-3	-1
0	-2

Cat

4	1
-1	-3

Person

-2	-5
3	-3

Background

-5	2
0	-1

Softmax, along channel axis

0.0009	0.0351
0.0445	0.2245

0.9965	0.2593
0.0164	0.0826

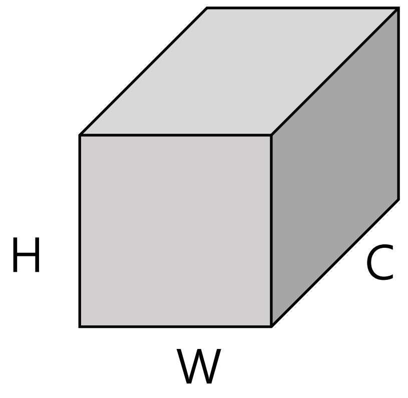
0.0025	0.0006
0.8945	0.0826

0.0001	0.7049
0.0445	0.6103

(0, 0) is Cat, (0, 1) is Bg, (1, 0) is Person, (1, 1) is Bg

Softmax, along Height axis

4x2x2



Dog		Cat		Person		Background	
-3	-1	4	1	-2	-5	-5	2
0	-2	-1	-3	3	-3	0	-1

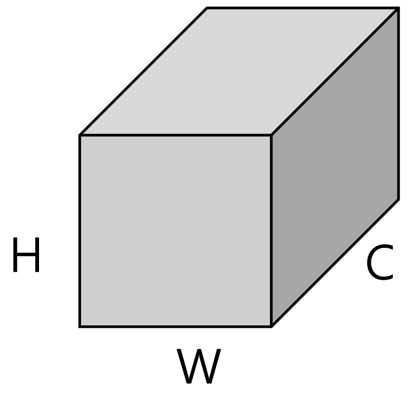
Softmax, along Height axis

0.0474	0.7311	0.9933	0.9820	0.0067	0.1192	0.0067	0.9526
0.9526	0.2689	0.0067	0.0180	0.9933	0.8808	0.9933	0.0474

0.0474 + 0.9526 = 1

Softmax, along Height axis

4x2x2



Dog		Cat		Person		Background	
-3	-1	4	1	-2	-5	-5	2
0	-2	-1	-3	3	-3	0	-1

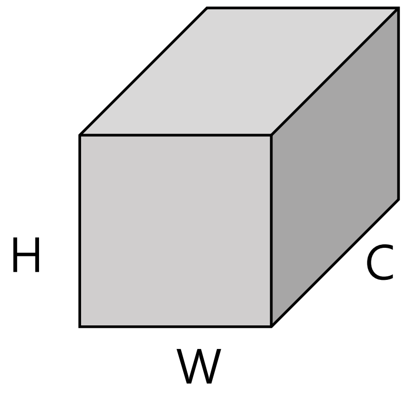
Softmax, along Height axis

0.0474	0.7311	0.9933	0.9820	0.0067	0.1192	0.0067	0.9526
0.9526	0.2689	0.0067	0.0180	0.9933	0.8808	0.9933	0.0474

$0.7311 + 0.2689 = 1$

Softmax, along Width axis

4x2x2



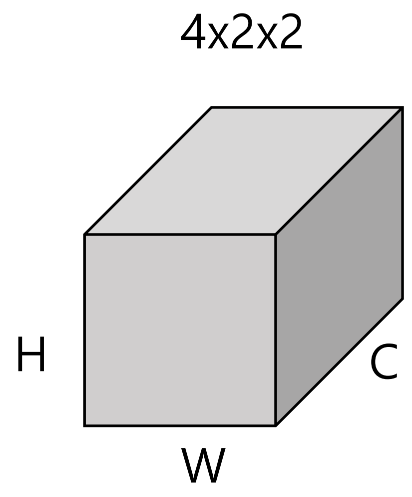
Dog		Cat		Person		Background	
-3	-1	4	1	-2	-5	-5	2
0	-2	-1	-3	3	-3	0	-1

Softmax, along Width axis

0.1192	0.8808	0.9526	0.0474	0.9526	0.0474	0.0009	0.9991
0.8808	0.1192	0.8808	0.1192	0.9975	0.0025	0.7311	0.2689

$0.1192 + 0.8808 = 1$

Softmax, along Width axis



Dog		Cat		Person		Background	
-3	-1	4	1	-2	-5	-5	2
0	-2	-1	-3	3	-3	0	-1

Softmax, along Width axis

Dog		Cat		Person		Background	
0.7311	0.2689	0.0474	0.9526	0.9933	0.0067	0.0180	0.9820
0.8808	0.1192	0.2689	0.7311	0.9991	0.0009	0.0009	0.9991

$0.8808 + 0.1192 = 1$

I was confused difference between sigmoid and softmax, because of this sentence.

3 ARCHITECTURE

SegNet has an encoder network and a corresponding decoder network, followed by a final pixelwise classification layer. This architecture is illustrated in Fig. 3. The encoder network consists of 13 convolutional layers which correspond to the first 13 convolutional layers in the VGG16 network [1] designed for object classification. We can therefore initialize the training process from weights trained for classification on large datasets [41]. We can also discard the fully connected layers in favour of retaining higher resolution feature maps at the deepest encoder output. This also reduces the number of parameters in the SegNet encoder network significantly (from 134M to 14.7M) as compared to other recent architectures [2], [4] (see. Table 6). Each encoder layer has a corresponding decoder layer and hence the decoder network has 13 layers. The final decoder output is fed to a multi-class soft-max classifier to produce class probabilities for each pixel independently.

from SegNet

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

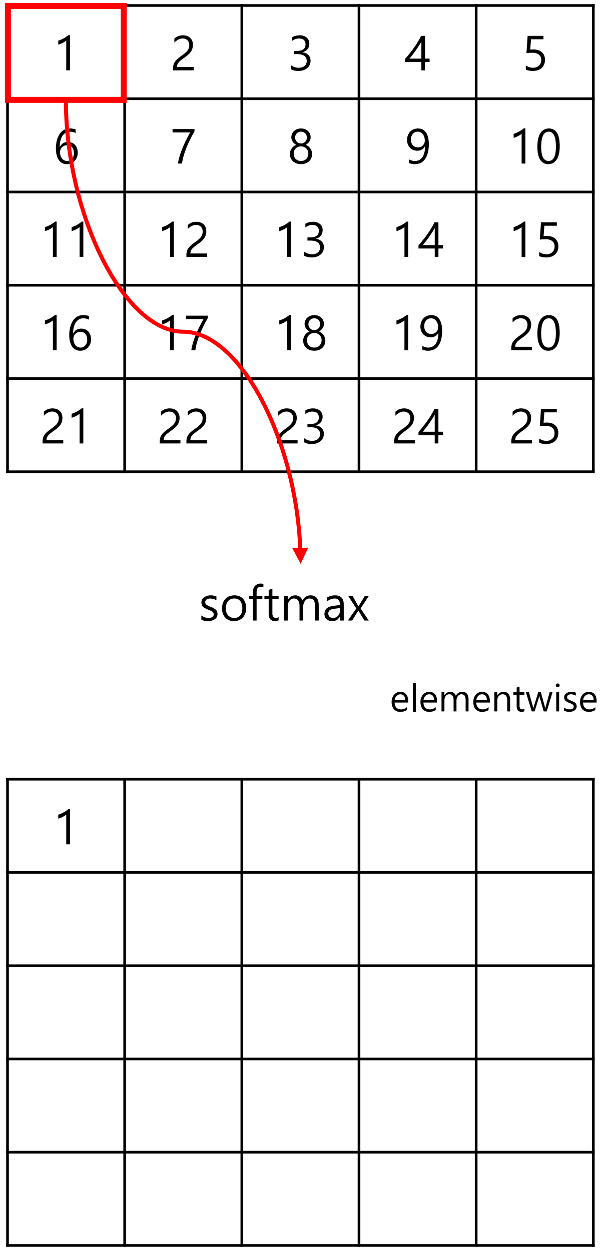
Independent pixel-wise softmax?

0.3	0.4	0.9	0.2	0.5
0.6	0.3	0.7	0.9	0.1
0.4	0.7	0.2	0.3	0.1
0.7	0.5	0.1	0.4	0.9
0.8	0.1	0.3	0.4	0.7

3 ARCHITECTURE

SegNet has an encoder network and a corresponding decoder network, followed by a final pixelwise classification layer. This architecture is illustrated in Fig. 3. The encoder network consists of 13 convolutional layers which correspond to the first 13 convolutional layers in the VGG16 network [1] designed for object classification. We can therefore initialize the training process from weights trained for classification on large datasets [41]. We can also discard the fully connected layers in favour of retaining higher resolution feature maps at the deepest encoder output. This also reduces the number of parameters in the SegNet encoder network significantly (from 134M to 14.7M) as compared to other recent architectures [2], [4] (see. Table 6). Each encoder layer has a corresponding decoder layer and hence the decoder network has 13 layers. The final decoder output is fed to a multi-class soft-max classifier to produce class probabilities for each pixel independently.

from SegNet



elementwise softmax output is 1

3 Architecture

SegNet has an encoder network and a corresponding decoder network, followed by a final pixelwise classification layer. This architecture is illustrated in Fig. 3. The encoder network consists of 13 convolutional layers which correspond to the first 13 convolutional layers in the VGG16 network [1] designed for object classification. We can therefore initialize the training process from weights trained for classification on large datasets [41]. We can also discard the fully connected layers in favour of retaining higher resolution feature maps at the deepest encoder output. This also reduces the number of parameters in the SegNet encoder network significantly (from 134M to 14.7M) as compared to other recent architectures [2], [4] (see. Table 6). Each encoder layer has a corresponding decoder layer and hence the decoder network has 13 layers. The final decoder output is fed to a multi-class soft-max classifier to produce class probabilities for each pixel independently.

from SegNet

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

softmax

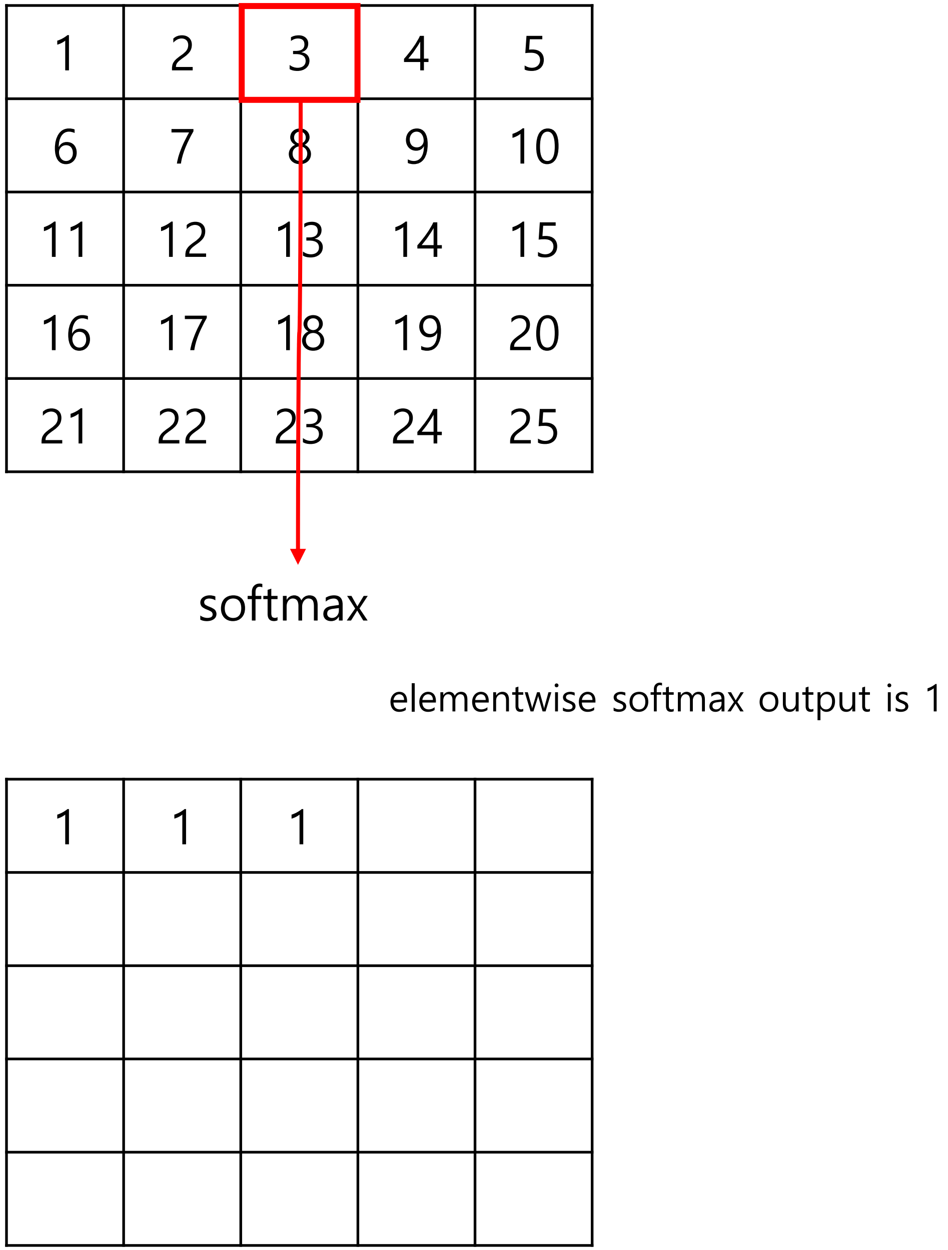
elementwise softmax output is 1

1	1			

3 ARCHITECTURE

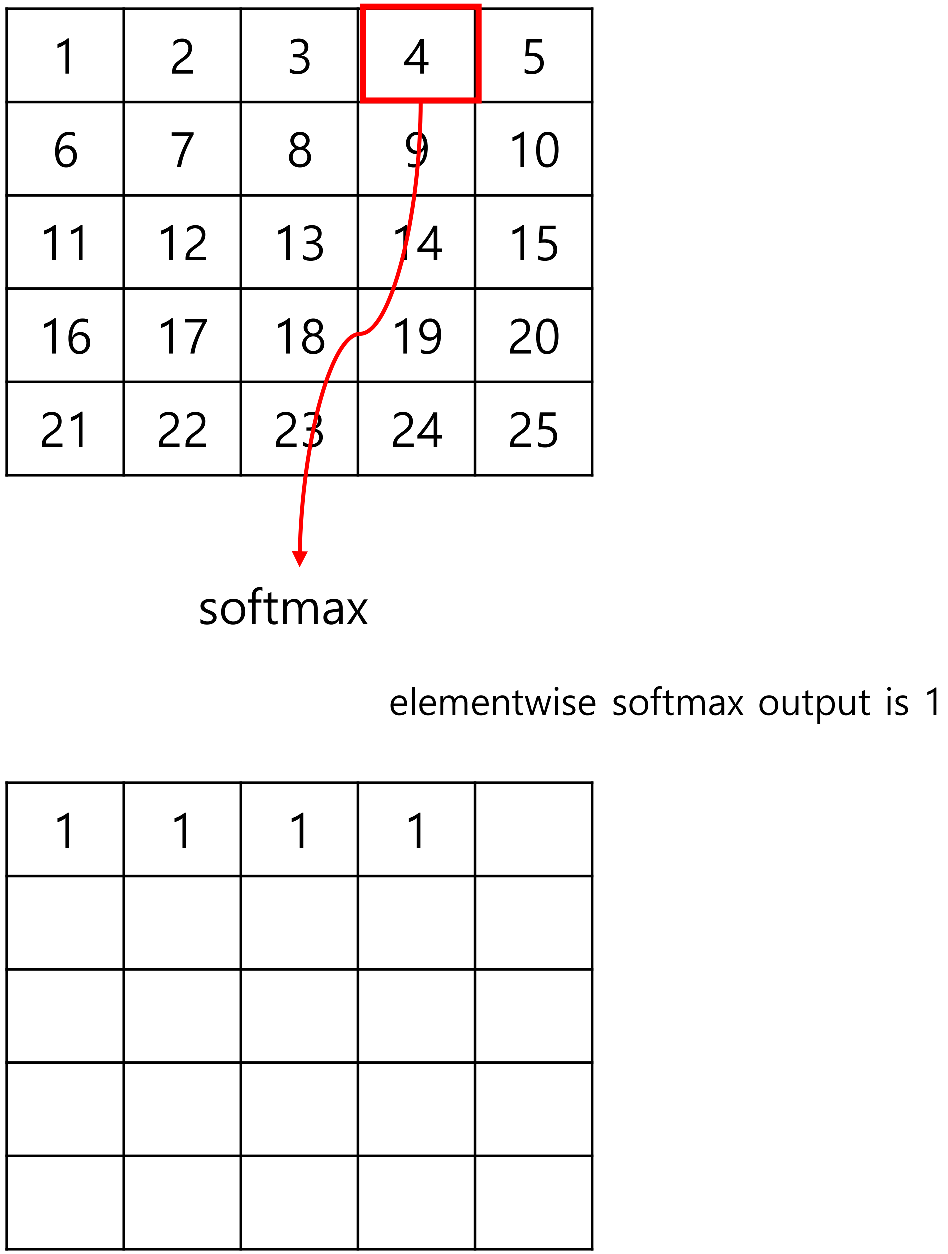
SegNet has an encoder network and a corresponding decoder network, followed by a final pixelwise classification layer. This architecture is illustrated in Fig. 3. The encoder network consists of 13 convolutional layers which correspond to the first 13 convolutional layers in the VGG16 network [1] designed for object classification. We can therefore initialize the training process from weights trained for classification on large datasets [41]. We can also discard the fully connected layers in favour of retaining higher resolution feature maps at the deepest encoder output. This also reduces the number of parameters in the SegNet encoder network significantly (from 134M to 14.7M) as compared to other recent architectures [2], [4] (see. Table 6). Each encoder layer has a corresponding decoder layer and hence the decoder network has 13 layers. The final decoder output is fed to a multi-class soft-max classifier to produce class probabilities for each pixel independently.

from SegNet

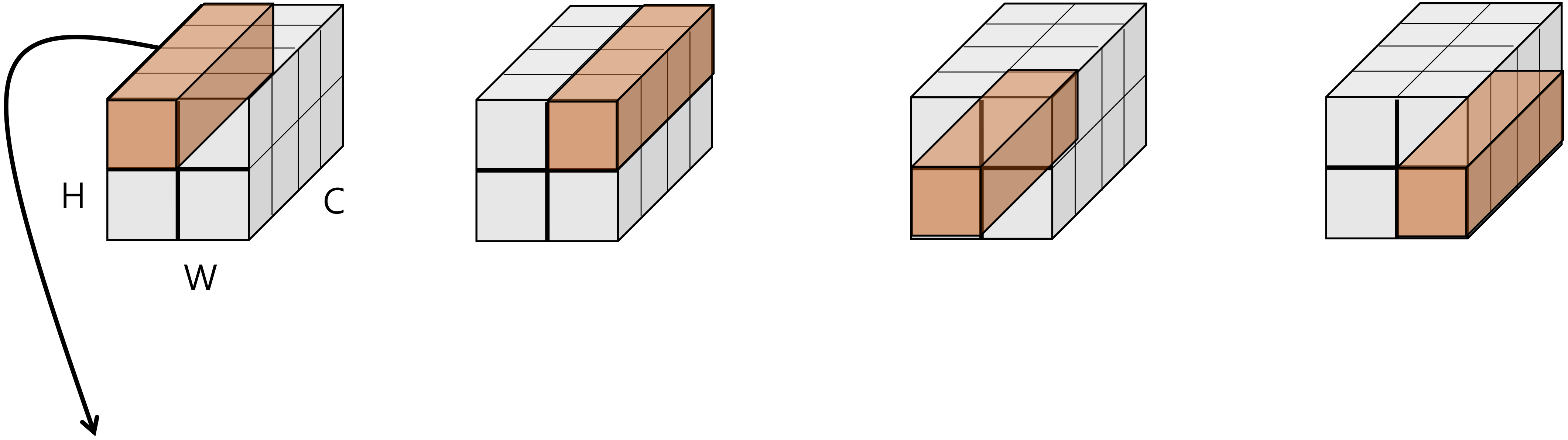


3 ARCHITECTURE

SegNet has an encoder network and a corresponding decoder network, followed by a final pixelwise classification layer. This architecture is illustrated in Fig. 3. The encoder network consists of 13 convolutional layers which correspond to the first 13 convolutional layers in the VGG16 network [1] designed for object classification. We can therefore initialize the training process from weights trained for classification on large datasets [41]. We can also discard the fully connected layers in favour of retaining higher resolution feature maps at the deepest encoder output. This also reduces the number of parameters in the SegNet encoder network significantly (from 134M to 14.7M) as compared to other recent architectures [2], [4] (see. Table 6). Each encoder layer has a corresponding decoder layer and hence the decoder network has 13 layers. The final decoder output is fed to a multi-class soft-max classifier to produce class probabilities for each pixel independently.



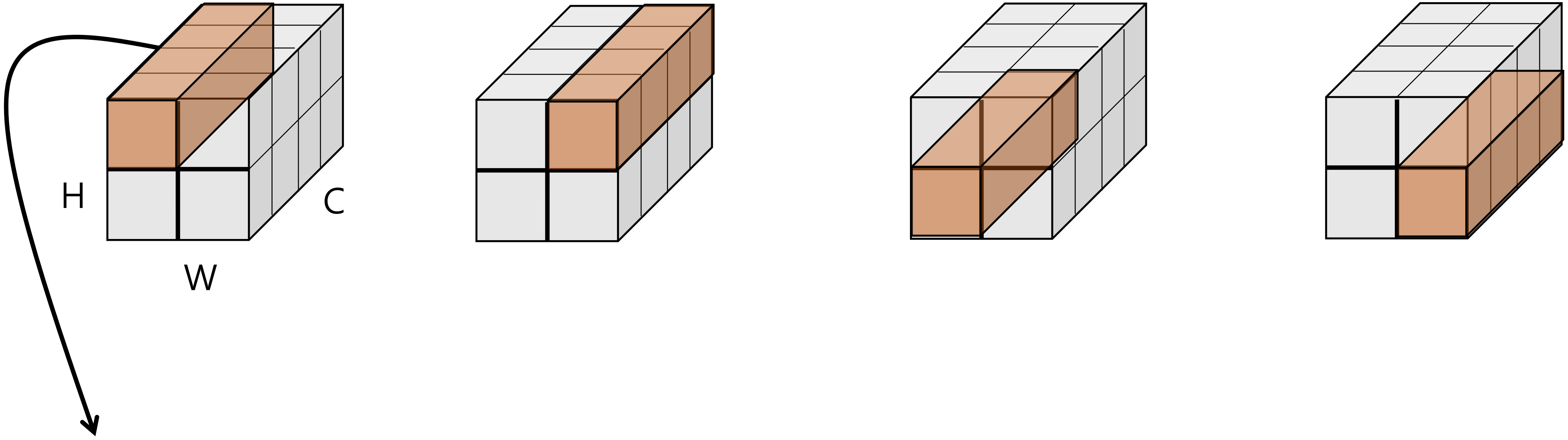
Therefore, use softmax along channel axis in semantic segmentation task



[0.0009, 0.9965, 0.0025, 0.0001]

Dog Cat Person Background

Therefore, use softmax along channel axis in semantic segmentation task

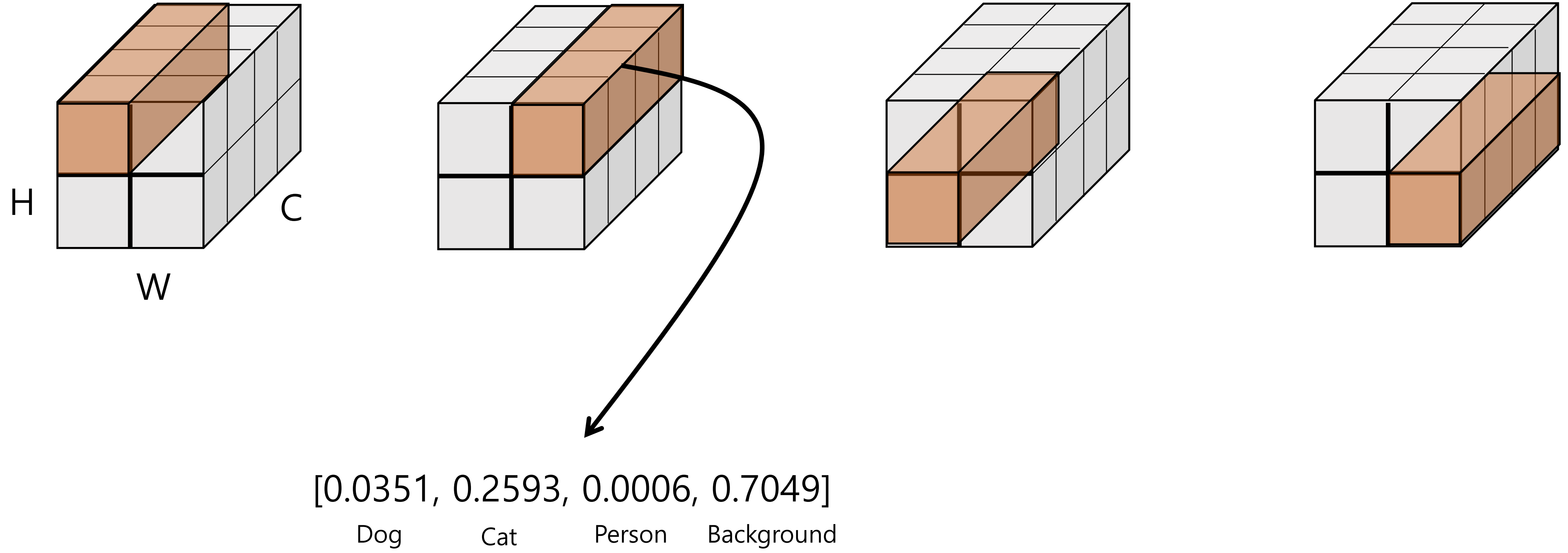


[0.0009, 0.9965, 0.0025, 0.0001]

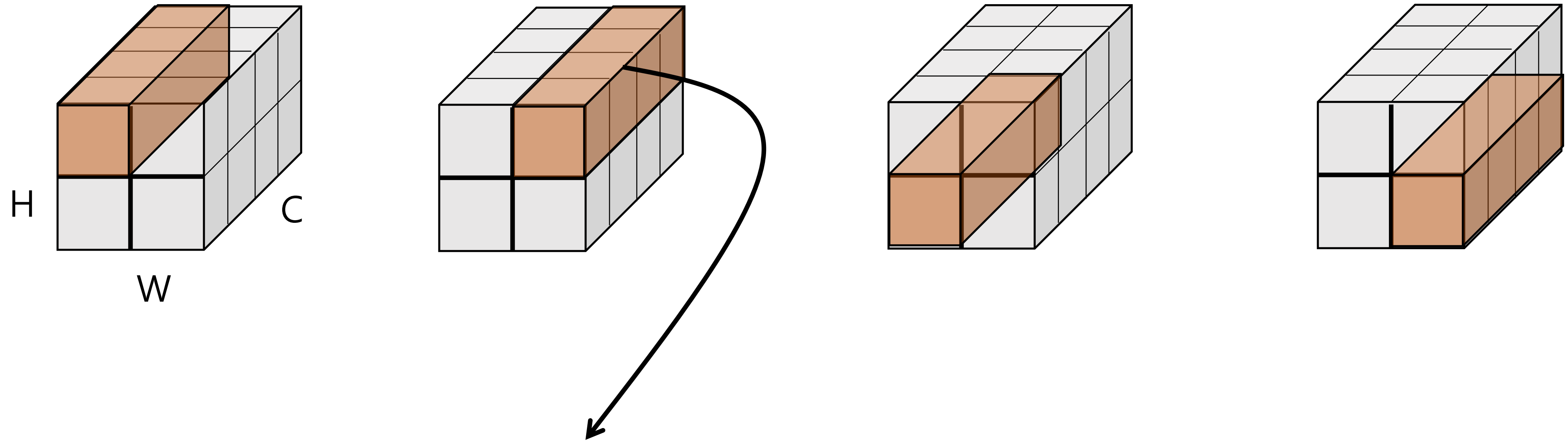
Dog Cat Person Background

=> (0, 0) pixel is Cat

Therefore, use softmax along channel axis in semantic segmentation task



Therefore, use softmax along channel axis in semantic segmentation task



[0.0351, 0.2593, 0.0006, $\boxed{\text{0.7049]}}$
Dog Cat Person Background

=> (0, 1) pixel is Background

Where is softmax?

```python
class UNet(nn.Module):

    def __init__(self, n_class):
        super().__init__()

        self.dconv_down1 = double_conv(3, 64)
        self.dconv_down2 = double_conv(64, 128)
        self.dconv_down3 = double_conv(128, 256)
        self.dconv_down4 = double_conv(256, 512)

        self.maxpool = nn.MaxPool2d(2)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)

        self.dconv_up3 = double_conv(256 + 512, 256)
        self.dconv_up2 = double_conv(128 + 256, 128)
        self.dconv_up1 = double_conv(128 + 64, 64)

        self.conv_last = nn.Conv2d(64, n_class, 1)
```

```python
    def forward(self, x):
        conv1 = self.dconv_down1(x)
        x = self.maxpool(conv1)

        conv2 = self.dconv_down2(x)
        x = self.maxpool(conv2)

        conv3 = self.dconv_down3(x)
        x = self.maxpool(conv3)

        x = self.dconv_down4(x)

        x = self.upsample(x)
        x = torch.cat([x, conv3], dim=1)

        x = self.dconv_up3(x)
        x = self.upsample(x)
        x = torch.cat([x, conv2], dim=1)

        x = self.dconv_up2(x)
        x = self.upsample(x)
        x = torch.cat([x, conv1], dim=1)

        x = self.dconv_up1(x)

        out = self.conv_last(x)

        return out
```

source : https://github.com/usuyama/pytorch-unet/blob/master/pytorch_unet.py

Pytorch CrossEntropyLoss combines Softmax!

CROSSENTROPYLOSS

CLASS `torch.nn.CrossEntropyLoss(weight: Optional[torch.Tensor] = None, size_average=None, ignore_index: int = -100, reduce=None, reduction: str = 'mean')` [SOURCE]

This criterion combines $\boxed{\texttt{nn.LogSoftmax()} \text{ and } \texttt{nn.NLLLoss()}}$ in one single class.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

The *input* is expected to contain raw, unnormalized scores for each class.

input has to be a Tensor of size either $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case (described later).

This criterion expects a class index in the range $[0, C-1]$ as the *target* for each value of a 1D tensor of size *minibatch*; if *ignore_index* is specified, this criterion also accepts this class index (this index may not necessarily be in the class range).

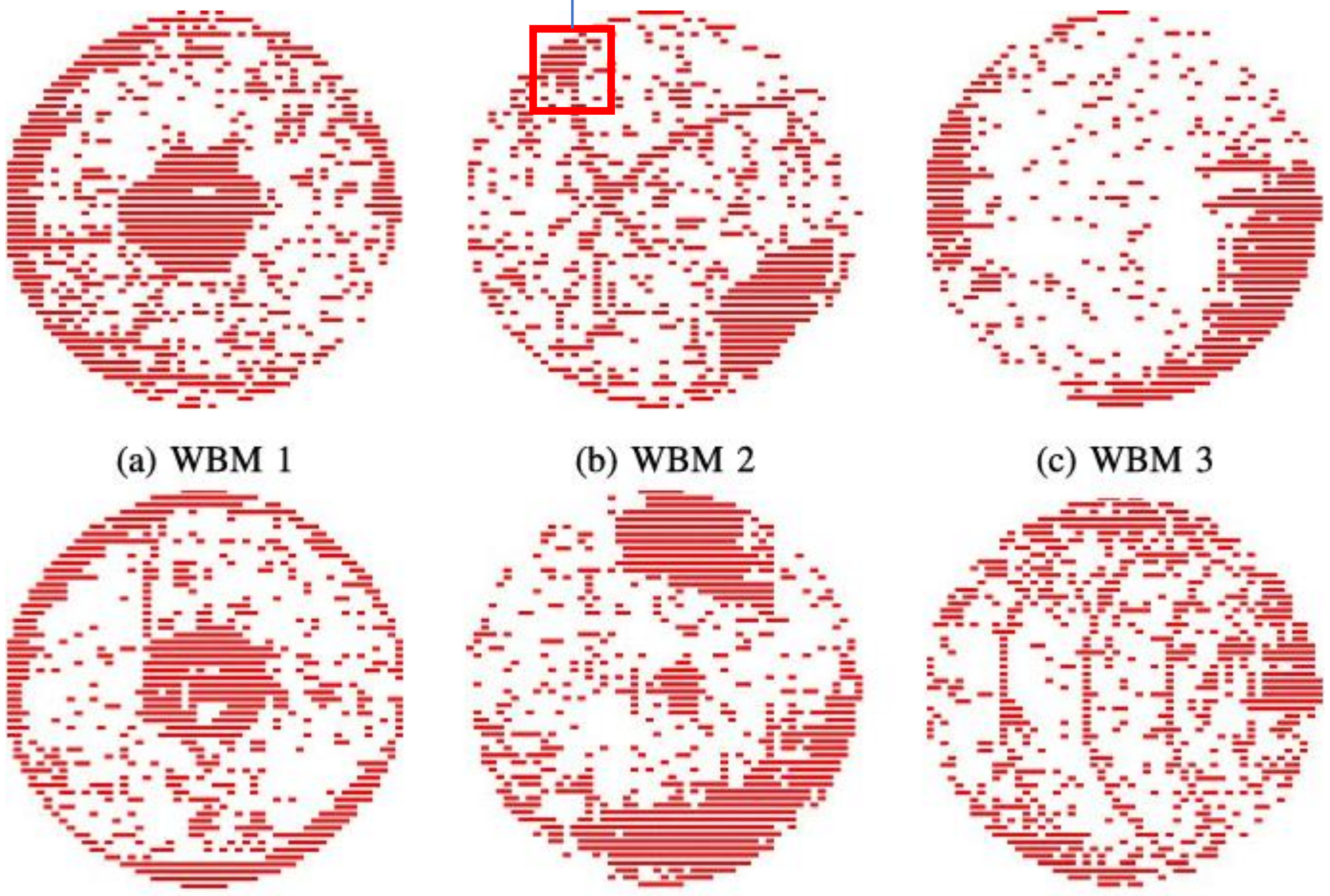
The loss can be described as:

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

source : https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html?highlight=crossentropy#torch.nn.CrossEntropyLoss

When is Sigmoid?

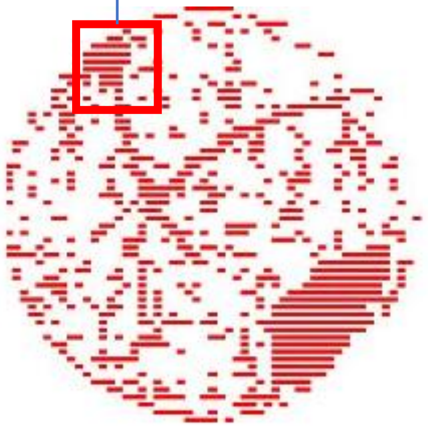
겹치는 클래스가 존재하는데 둘 다 찾는게 중요



(a) WBM 1 (b) WBM 2 (c) WBM 3

(d) WBM 4 (e) WBM 5 (f) WBM 6

When is Sigmoid?



(b) WBM 2

-2	-1
0	2

라인 불량

-4	-3
-1	-3

서클 불량

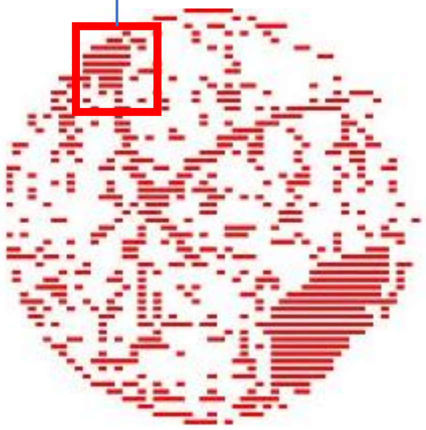
5	5
3	-3

외각 불량

2	1
0	2

존 불량

When is Sigmoid?



(b) WBM 2

-2	-1
0	2

라인 불량

-4	-3
-1	-3

서클 불량

5	5
3	-3

외각 불량

2	1
0	2

존 불량

Sigmoid

0.12	0.27
0.5	0.88

라인 불량

0.02	0.05
0.27	0.05

서클 불량

0.99	0.99
0.95	0.04

외각 불량

0.88	0.73
0.5	0.88

존 불량

When is Sigmoid?

외곽과 존 불량 모두 발생

0.12	0.27
0.5	0.88

라인 불량

0.02	0.05
0.27	0.05

서클 불량

0.99	0.99
0.95	0.04

외각 불량

0.88	0.73
0.5	0.88

존 불량

라인과 존 불량 모두 발생

정답 라벨과 Binary Cross Entropy 계산

0	0
0	1

라인 불량

0	0
0	0

서클 불량

1	1
1	0

외각 불량

1	1
1	1

존 불량

How to apply softamx along (height x width) axis?

Dog

-3	-1
0	-2

Cat

4	1
-1	-3

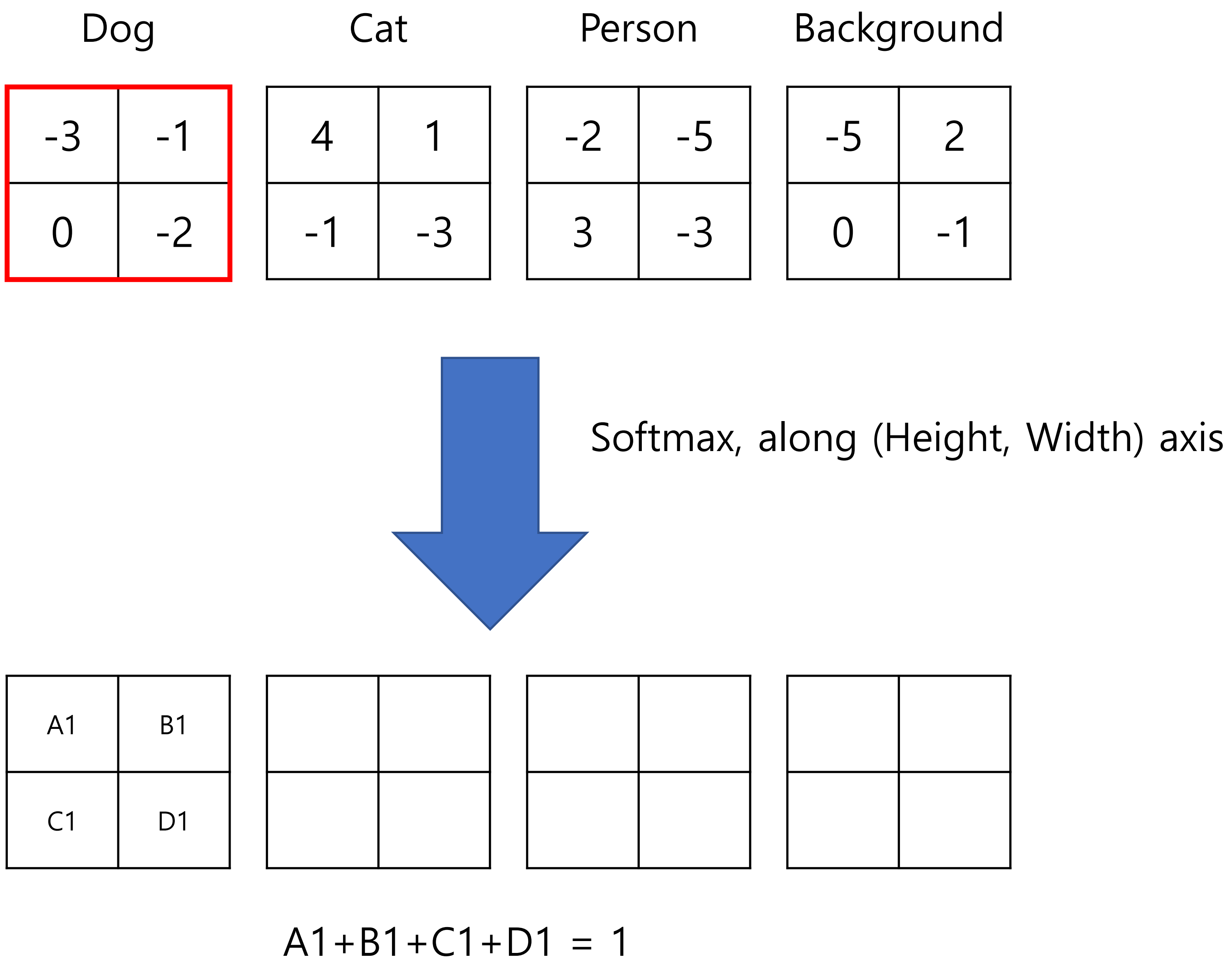
Person

-2	-5
3	-3

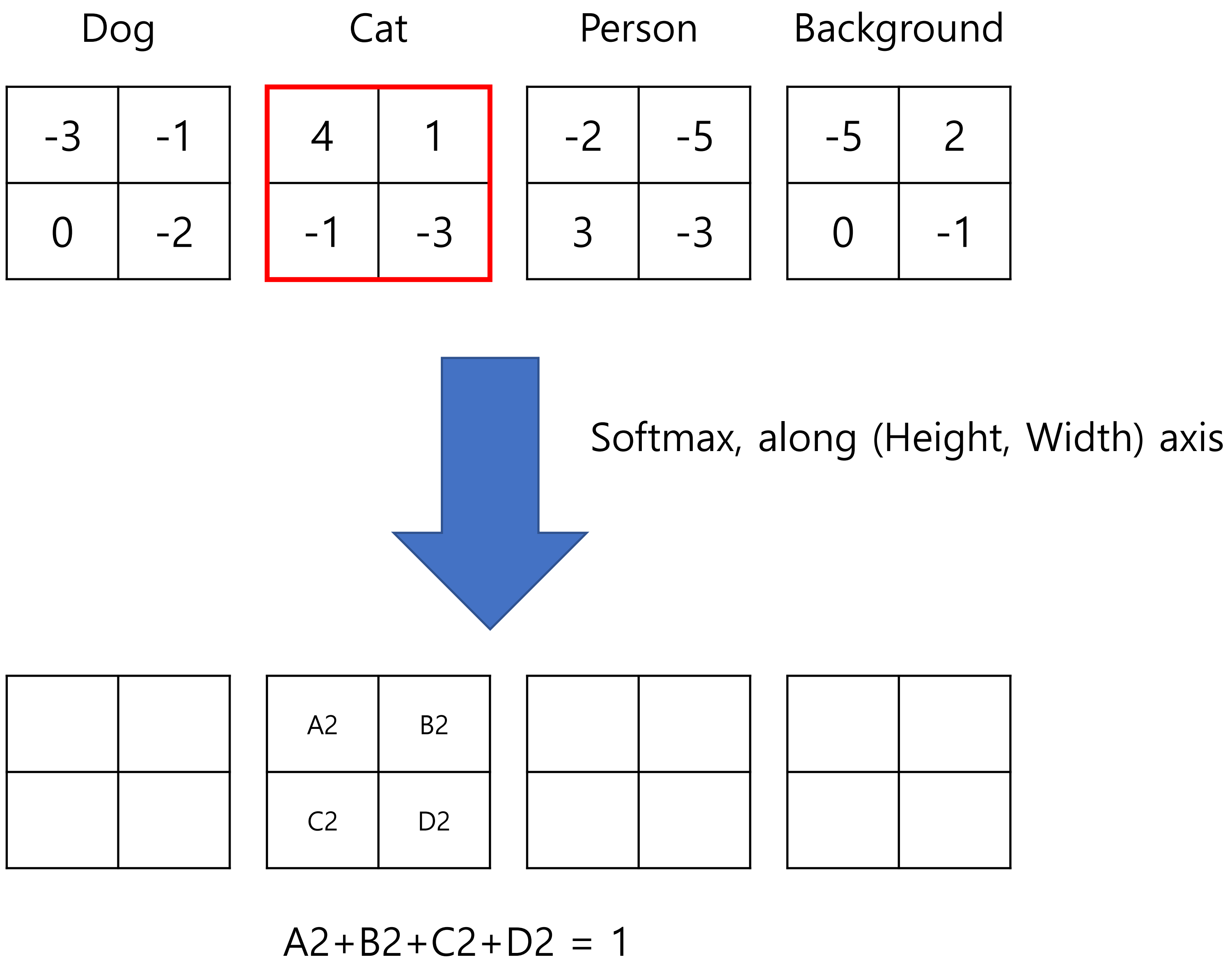
Background

-5	2
0	-1

Softmax, along (Height, Width) axis



Dog
Cat
Person
Background
-3
-1
0
-2
4
1
-1
-3
-2
-5
3
-3
-5
2
0
-1
Softmax, along (Height, Width) axis
A1
B1
C1
D1
A1+B1+C1+D1 = 1


Dog
Cat
Person
Background
-3 -1
0 -2
4 1
-1 -3
-2 -5
3 -3
-5 2
0 -1
Softmax, along (Height, Width) axis
A2 B2
C2 D2
A2+B2+C2+D2 = 1

SOFTMAX2D

CLASS `torch.nn.Softmax2d` [SOURCE]

Applies SoftMax over features to each spatial location.

When given an image of `Channels x Height x Width`, it will apply *Softmax* to each location $(Channels, h_i, w_j)$

Shape:

- Input: (N, C, H, W)
- Output: (N, C, H, W) (same shape as input)

Returns

a Tensor of the same dimension and shape as the input with values in the range [0, 1]

Examples:

```
>>> m = nn.Softmax2d()
>>> # you softmax over the 2nd dimension
>>> input = torch.randn(2, 3, 12, 13)
>>> output = m(input)
```

Softmax result is same as Softmax2d

```
[133] x = torch.tensor([[[-3, -1],
                         [0, -2]],

                        [[4, 1],
                         [-1, -3]],

                        [[-2, -5],
                         [3, -3]],

                        [[-5, 2],
                         [0, -1]]], dtype=torch.float64)

      x, x.size()

      (tensor([[[-3., -1.],
                [ 0., -2.]],

               [[ 4.,  1.],
                [-1., -3.]],

               [[-2., -5.],
                [ 3., -3.]],

               [[-5.,  2.],
                [ 0., -1.]]], dtype=torch.float64), torch.Size([4, 2, 2]))
```

```
[134] x = x.unsqueeze(0)
      x.size()

      torch.Size([1, 4, 2, 2])
```

```
[136] softmax_channel = torch.nn.Softmax(dim=1)

      output_softmax_channel = softmax_channel(x)
      output_softmax_channel

      tensor([[[[0.0009, 0.0351],
                [0.0445, 0.2245]],

               [[0.9965, 0.2593],
                [0.0164, 0.0826]],

               [[0.0025, 0.0006],
                [0.8945, 0.0826]],

               [[0.0001, 0.7049],
                [0.0445, 0.6103]]]], dtype=torch.float64)
```

```
[140] softmax_height_width = torch.nn.Softmax2d()

      output_softmax_height_width = softmax_height_width(x)
      output_softmax_height_width

      tensor([[[[0.0009, 0.0351],
                [0.0445, 0.2245]],

               [[0.9965, 0.2593],
                [0.0164, 0.0826]],

               [[0.0025, 0.0006],
                [0.8945, 0.0826]],

               [[0.0001, 0.7049],
                [0.0445, 0.6103]]]], dtype=torch.float64)
```