AI VIETNAM
All-in-One Course

Extra Class

UNet

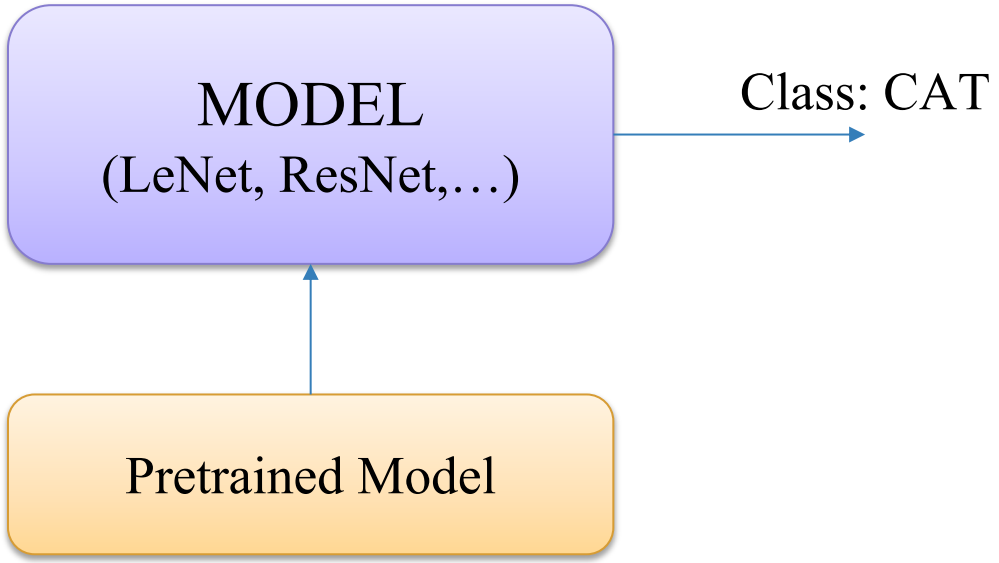
Nguyen Quoc Thai

Year 2023

CONTENT

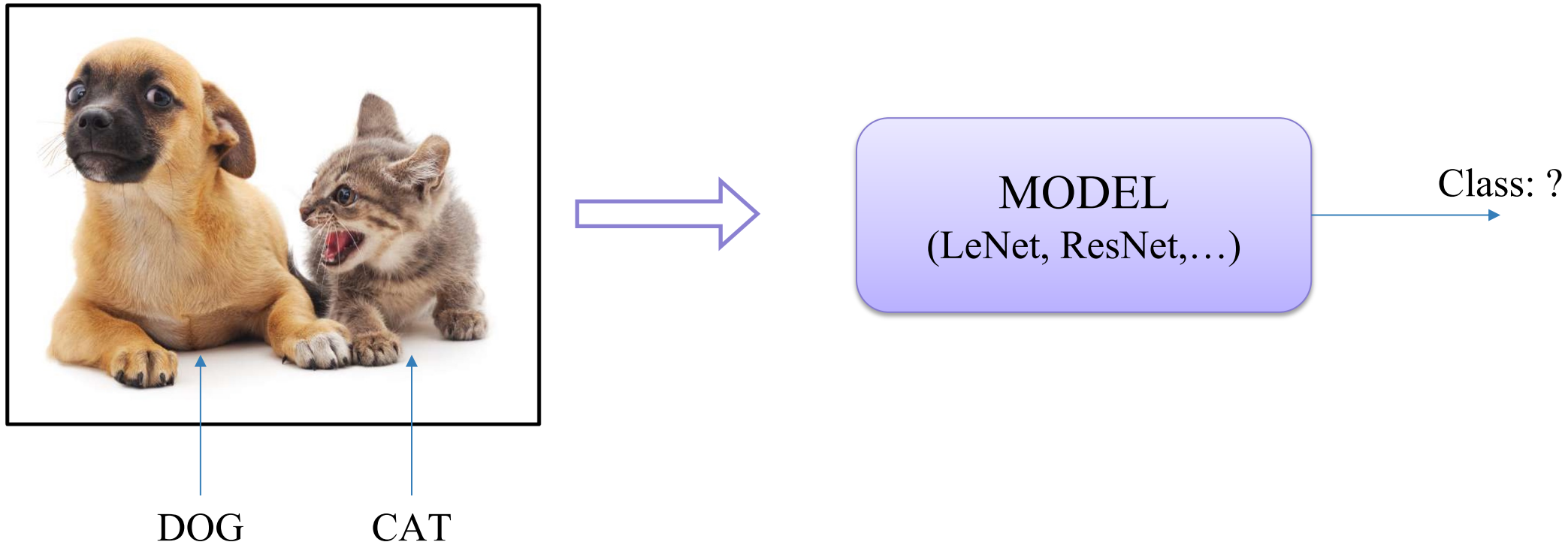
1 – Image Segmentation

Image Classification



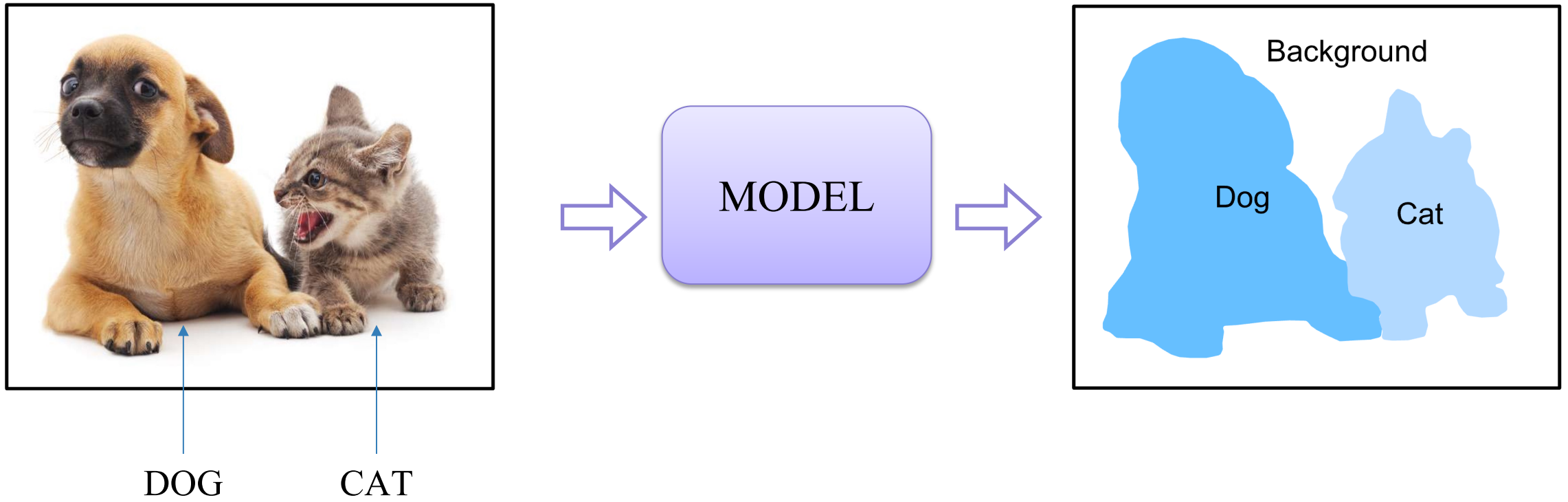
1 – Image Segmentation

Multiple Objects



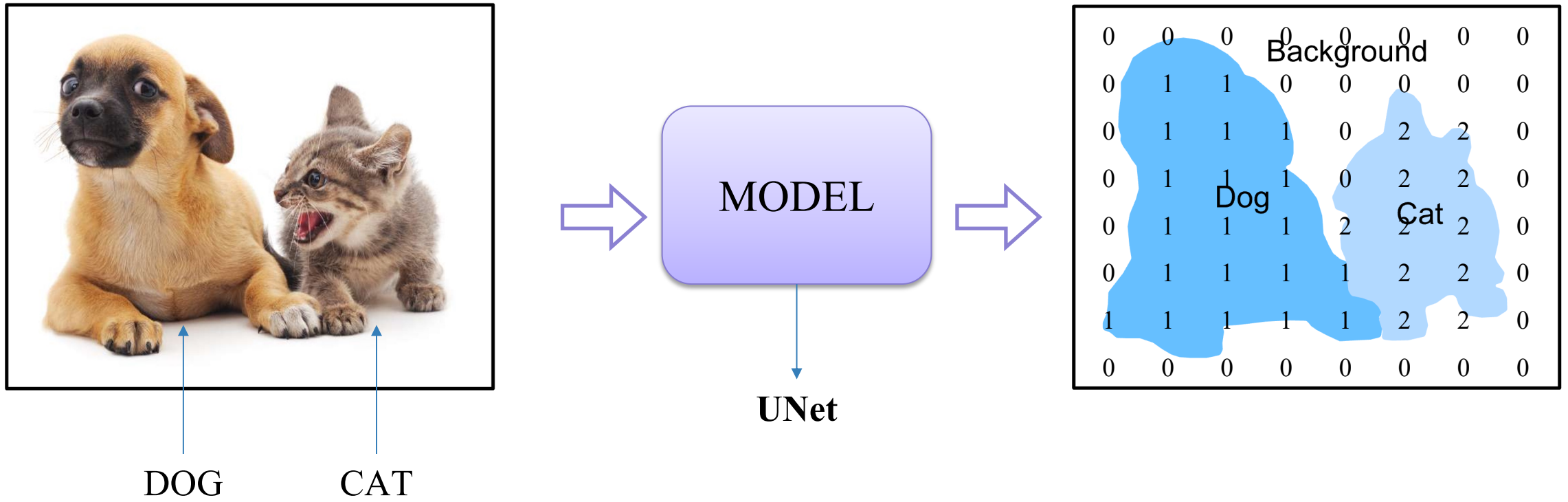
1 – Image Segmentation

Image Segmentation



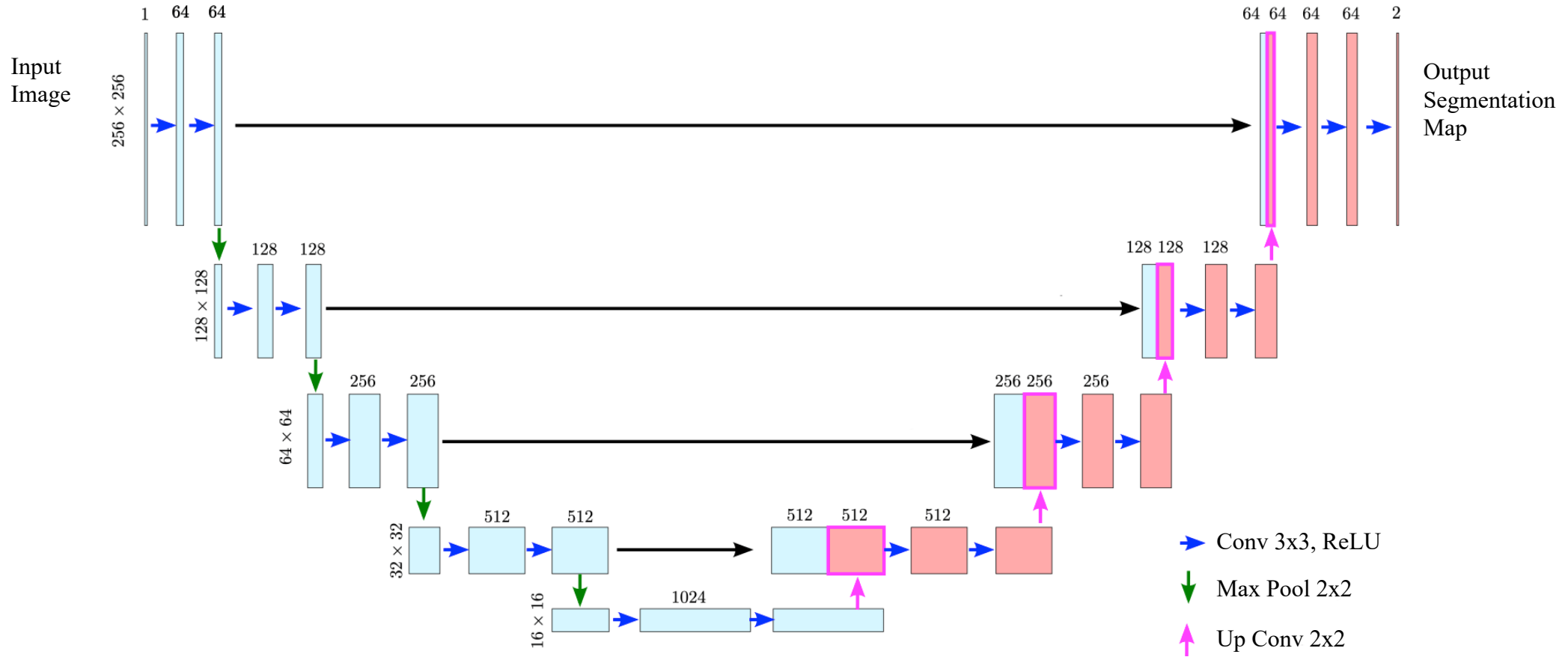
1 – Image Segmentation

Image Segmentation



1 – Image Segmentation

Unet Model



2 – Transposed Convolution

Review: Convolution

0	3	1	1
3	1	2	0
3	4	2	3
3	0	0	2

Input: M x N

Padding: (P, Q)

0	0	0	0	0	0
0	0	3	1	1	0
0	3	1	2	0	0
0	3	4	2	3	0
0	3	0	0	2	0
0	0	0	0	0	0

Shape: (M+2P) x (N+2Q)

*

Stride: (S, T)

1	1	1
1	1	1
0	1	0

Kernel: K x O

=

7	8
15	13

$$\left\lfloor \frac{M + 2P - K}{S} + 1 \right\rfloor \times \left\lfloor \frac{N + 2Q - O}{T} + 1 \right\rfloor$$

1

Bias

2 – Transposed Convolution

Review: Pooling

❖ Max Pooling

3	2	1	0	0	3
0	3	3	1	1	0
3	1	4	1	1	0
2	4	1	1	0	4
1	0	3	0	3	0
3	4	4	3	3	4

Input: 6 x 6

Kernel Size: 2
Stride: 2

3	3	3
4	4	4
4	4	4

Output: 3 x 3

❖ Average Pooling

3	2	1	0	0	3
0	3	3	1	1	0
3	1	4	1	1	0
2	4	1	1	0	4
1	0	3	0	3	0
3	4	4	3	3	4

Input: 6 x 6

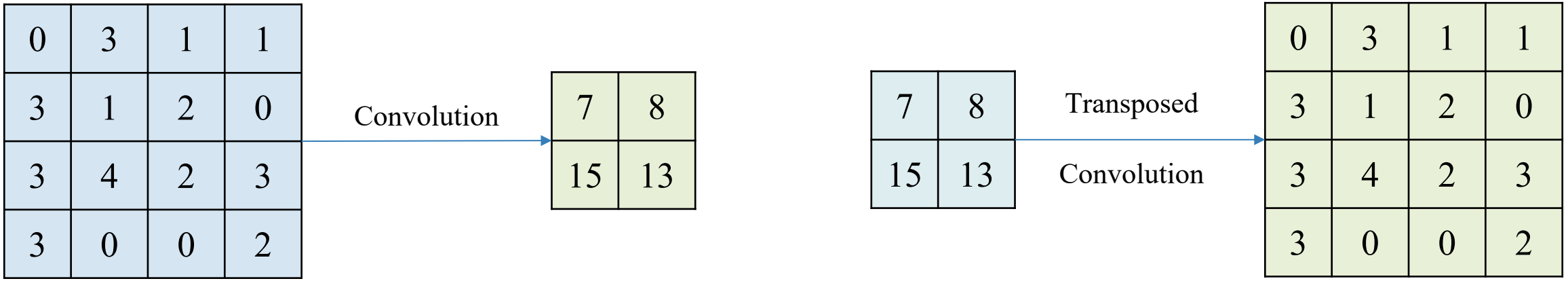
Kernel Size: (3, 2)
Stride: 2

2.0	1.7	0.8
1.8	1.6	1.3

Output: 2 x 3

2 – Transposed Convolution

Transposed Convolution



2 – Transposed Convolution

Transposed Convolution

Input

2	2
1	4

1	1
1	1

Kernel

2	2	
2	2	

+

	2	2
	2	2

+

1	1	
1	1	

+

	4	4
	4	4

2	4	2
3	9	6
1	5	4

2 – Transposed Convolution

Transposed Convolution – Demo

```
input = torch.randint(
    5, (1, 2, 2),
    dtype=torch.float32
)
input
```

```
tensor([[[2., 2.],
         [1., 4.]]])
```

```
conv_layer = nn.ConvTranspose2d(
    in_channels=1,
    out_channels=1,
    kernel_size=2,
    bias=False
)
```

```
conv_layer.weight.data = torch.ones(
    conv_layer.weight.data.shape
)
conv_layer.weight
```

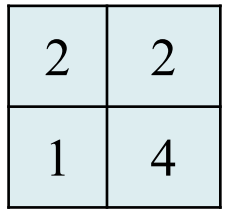
```
Parameter containing:
tensor([[[[1., 1.],
          [1., 1.]]]], requires_grad=True)
```

```
output = conv_layer(input)
output
```

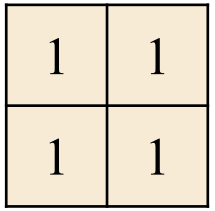
```
tensor([[[2., 4., 2.],
         [3., 9., 6.],
         [1., 5., 4.]]], grad_fn=<SqueezeBackward1>)
```

2 – Transposed Convolution

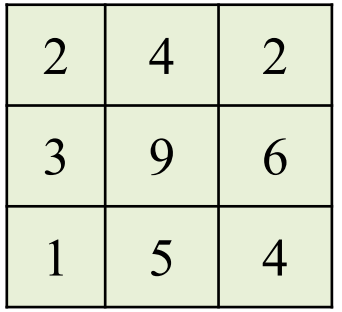
Transposed Convolution

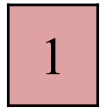


Input

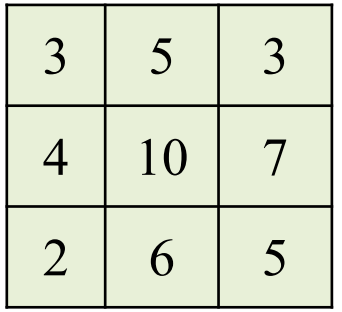


Kernel





Bias



2 – Transposed Convolution

Transposed Convolution – Demo

```
input = torch.randint(
    5, (1, 2, 2),
    dtype=torch.float32
)
input
```

```
tensor([[[2., 2.],
         [1., 4.]]])
```

```
conv_layer = nn.ConvTranspose2d(
    in_channels=1,
    out_channels=1,
    kernel_size=2,
    bias=True
)
```

```
conv_layer.weight.data = torch.ones(
    conv_layer.weight.data.shape
)
conv_layer.weight
```

```
Parameter containing:
tensor([[[[1., 1.],
          [1., 1.]]]], requires_grad=True)
```

```
conv_layer.bias =  nn.Parameter(
    torch.tensor([1], dtype=torch.float32)
)
conv_layer.bias
```

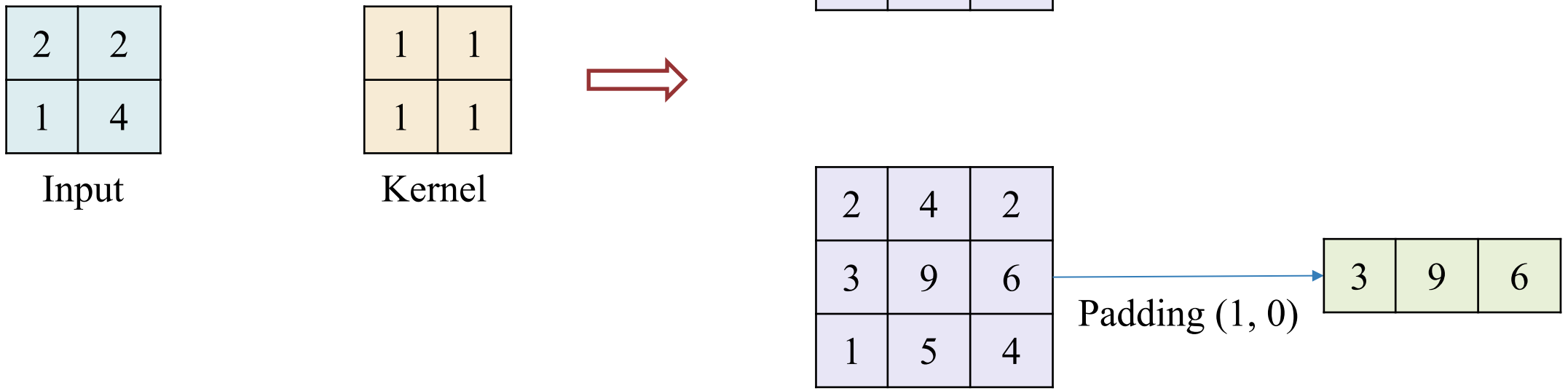
```
Parameter containing:
tensor([1.], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[ 3.,  5.,  3.],
         [ 4., 10.,  7.],
         [ 2.,  6.,  5.]]], grad_fn=<SqueezeBackward1>)
```

2 – Transposed Convolution

Padding



2 – Transposed Convolution

Padding – Demo

```
input = torch.randint(
    5, (1, 2, 2),
    dtype=torch.float32
)
input
```

```
tensor([[[2., 2.],
         [1., 4.]]])
```

```
conv_layer = nn.ConvTranspose2d(
    in_channels=1,
    out_channels=1,
    kernel_size=2,
    padding=1,
    bias=False
)
```

```
conv_layer.weight.data = torch.ones(
    conv_layer.weight.data.shape
)
conv_layer.weight
```

```
Parameter containing:
tensor([[[[1., 1.],
          [1., 1.]]]], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[9.]]], grad_fn=<SqueezeBackward1>)
```

```
conv_layer = nn.ConvTranspose2d(
    in_channels=1,
    out_channels=1,
    kernel_size=2,
    padding=(1, 0),
    bias=False
)
```

```
conv_layer.weight.data = torch.ones(
    conv_layer.weight.data.shape
)
conv_layer.weight
```

```
Parameter containing:
tensor([[[[1., 1.],
          [1., 1.]]]], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[3., 9., 6.]]], grad_fn=<SqueezeBackward1>)
```

2 – Transposed Convolution

Stride

2	2
1	4

Input

1	1
1	1

Kernel

Stride (1, 1)

2	4	2
3	9	6
1	5	4

Stride (2, 2)

2	2	2	2
2	2	2	2
1	1	4	4
1	1	4	4

2 – Transposed Convolution

Stride – Demo

```
input = torch.randint(
    5, (1, 2, 2),
    dtype=torch.float32
)
input
```

```
tensor([[[2., 2.],
         [1., 4.]]])
```

```
conv_layer = nn.ConvTranspose2d(
    in_channels=1,
    out_channels=1,
    kernel_size=2,
    stride=2,
    bias=False
)
```

```
conv_layer.weight.data = torch.ones(
    conv_layer.weight.data.shape
)
conv_layer.weight
```

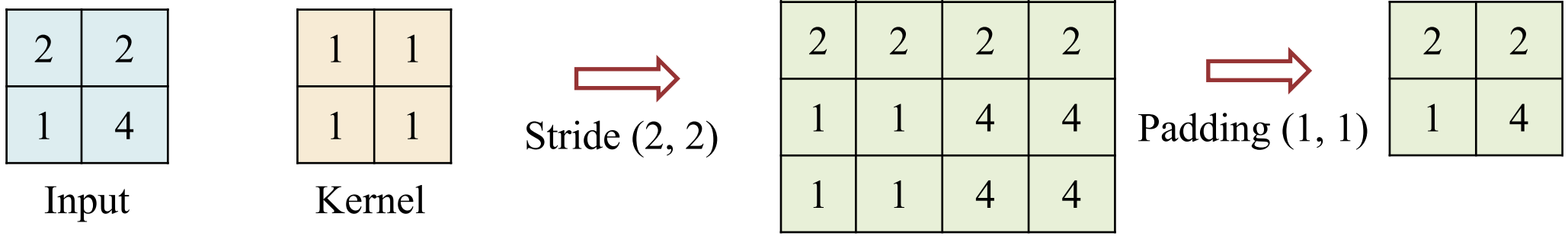
```
Parameter containing:
tensor([[[[1., 1.],
          [1., 1.]]]], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[2., 2., 2., 2.],
         [2., 2., 2., 2.],
         [1., 1., 4., 4.],
         [1., 1., 4., 4.]]], grad_fn=<SqueezeBackward1>)
```

2 – Transposed Convolution

Padding & Stride



2 – Transposed Convolution

Padding & Stride – Demo

```
input = torch.randint(
    5, (1, 2, 2),
    dtype=torch.float32
)
input
```

```
tensor([[[2., 2.],
         [1., 4.]]])
```

```
conv_layer = nn.ConvTranspose2d(
    in_channels=1,
    out_channels=1,
    kernel_size=2,
    stride=2,
    padding=1,
    bias=False
)
```

```
conv_layer.weight.data = torch.ones(
    conv_layer.weight.data.shape
)
conv_layer.weight
```

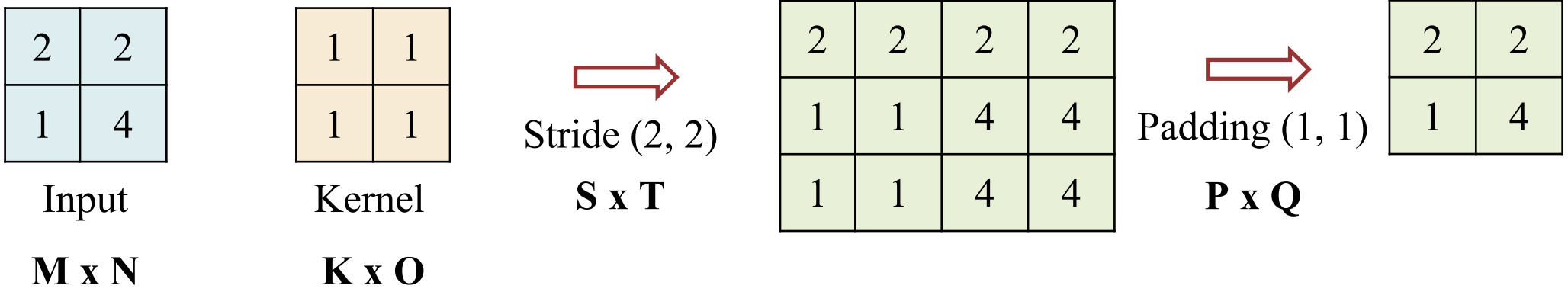
```
Parameter containing:
tensor([[[[1., 1.],
          [1., 1.]]]], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[2., 2.],
         [1., 4.]]], grad_fn=<SqueezeBackward1>)
```

2 – Transposed Convolution

Padding & Stride

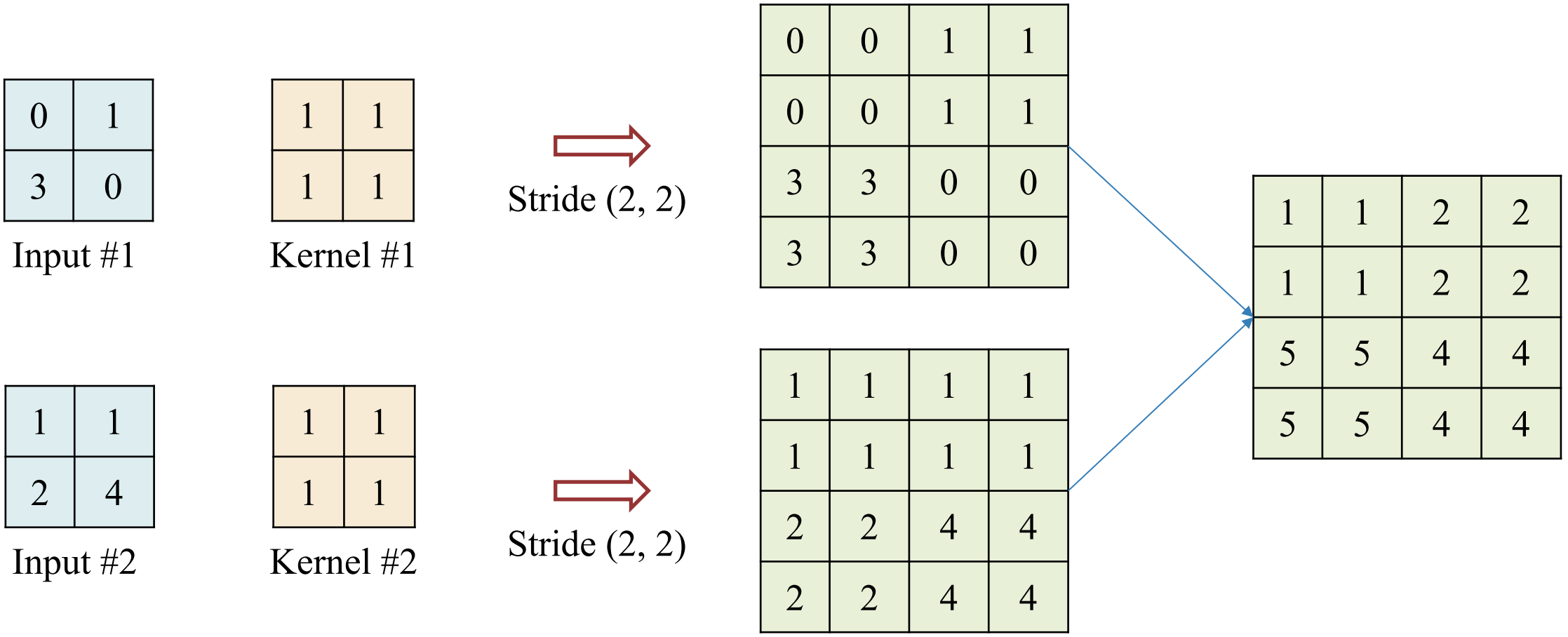


$$H_{out} = (M - 1) * S - 2P + (K - 1) + 1$$

$$W_{out} = (N - 1) * T - 2Q + (O - 1) + 1$$

2 – Transposed Convolution

Multiple Channels



2 – Transposed Convolution

Multiple Channels – Demo

```
input = torch.randint(
    5, (2, 2, 2),
    dtype=torch.float32
)
input
```

```
tensor([[[0., 1.],
         [3., 0.]],

        [[1., 1.],
         [2., 4.]]])
```

```
conv_layer = nn.ConvTranspose2d(
    in_channels=2,
    out_channels=1,
    kernel_size=2,
    stride=2,
    bias=False
)
```

```
conv_layer.weight.data = torch.ones(
    conv_layer.weight.data.shape
)
conv_layer.weight
```

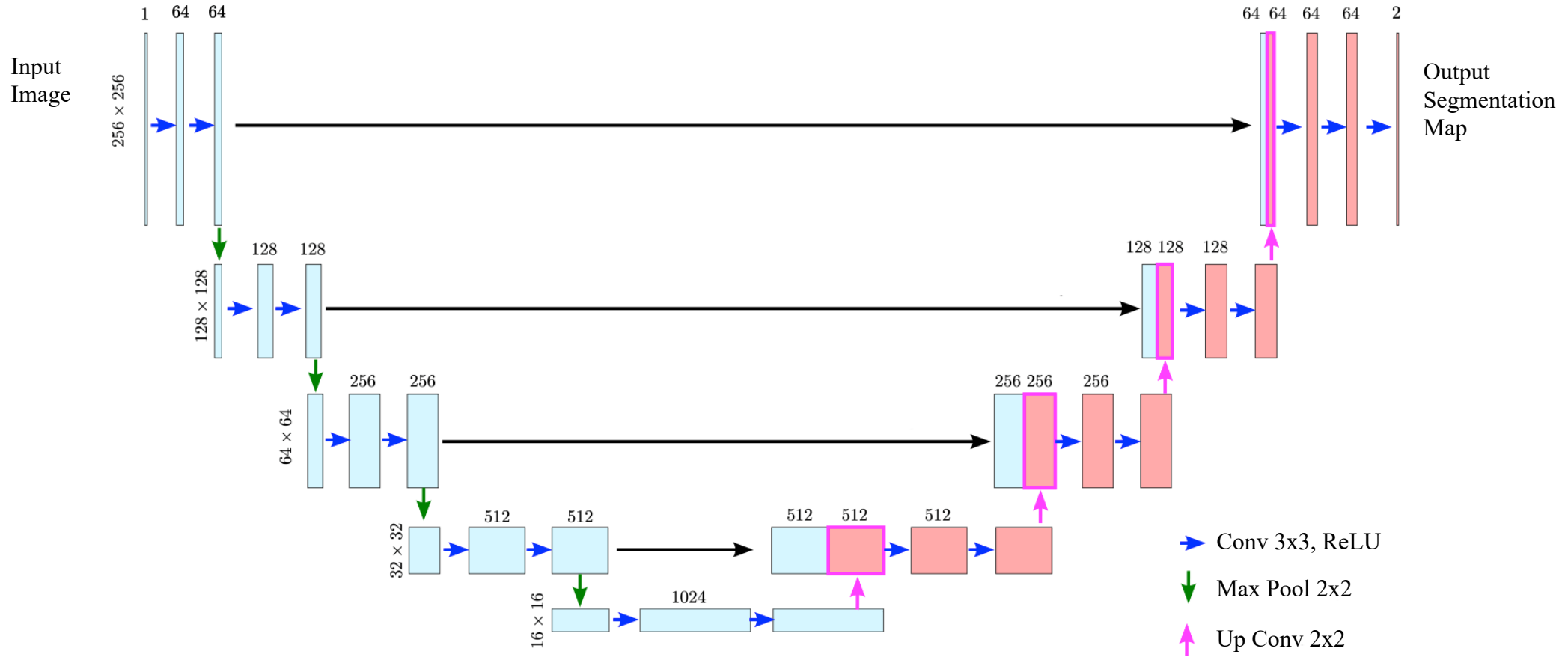
```
Parameter containing:
tensor([[[[1., 1.],
          [1., 1.]]],


        [[[1., 1.],
          [1., 1.]]]], requires_grad=True)
```

```
output = conv_layer(input)
output
```

```
tensor([[[1., 1., 2., 2.],
         [1., 1., 2., 2.],
         [5., 5., 4., 4.],
         [5., 5., 4., 4.]]], grad_fn=<SqueezeBackward1>)
```

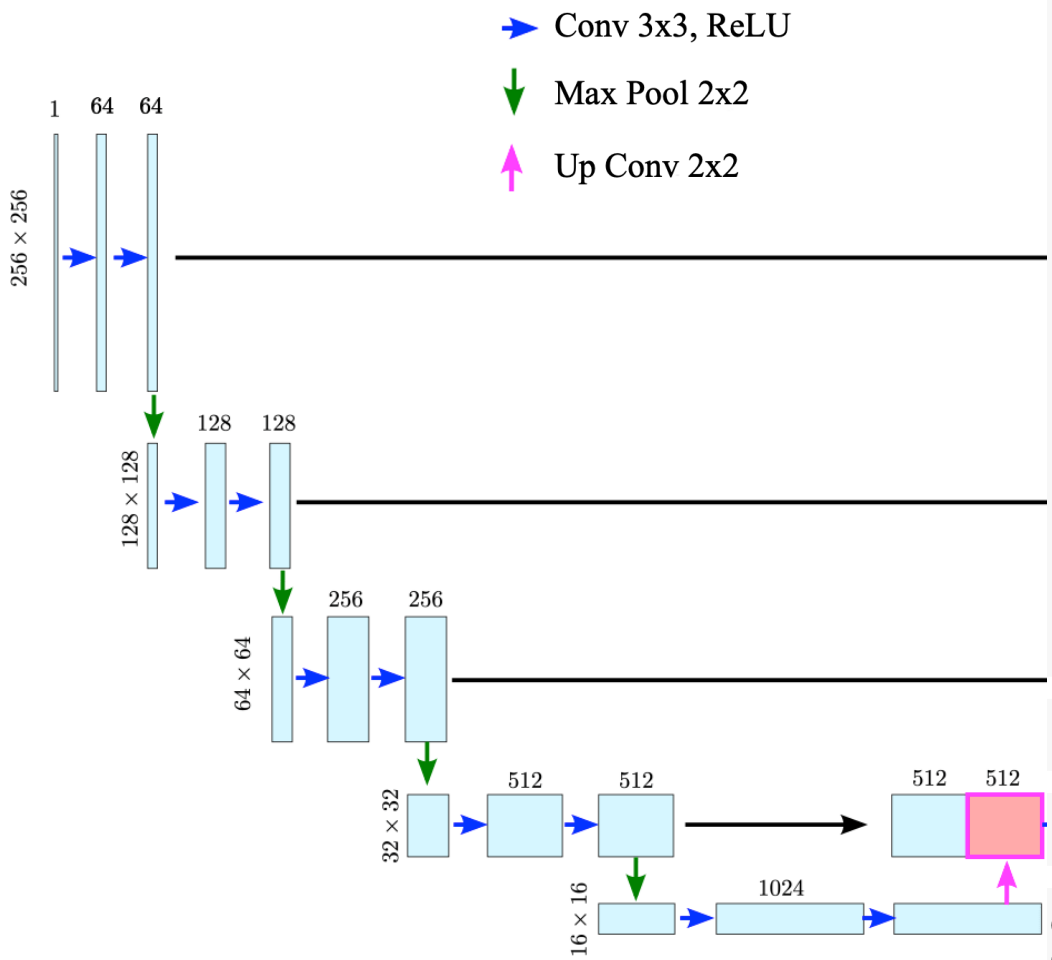
3 – UNet Model

UNet



3 – UNet Model

ConvBlock – Demo




```python
class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels) -> None:
        super().__init__()
        self.conv_block = nn.Sequential(
            nn.Conv2d(
                in_channels, out_channels, kernel_size=3, padding=1, bias=False
            ),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(
                out_channels, out_channels, kernel_size=3, padding=1, bias=False
            ),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        x = self.conv_block(x)
        return x
```

```python
input = torch.randint(5, (1, 1, 256, 256), dtype=torch.float32)
```

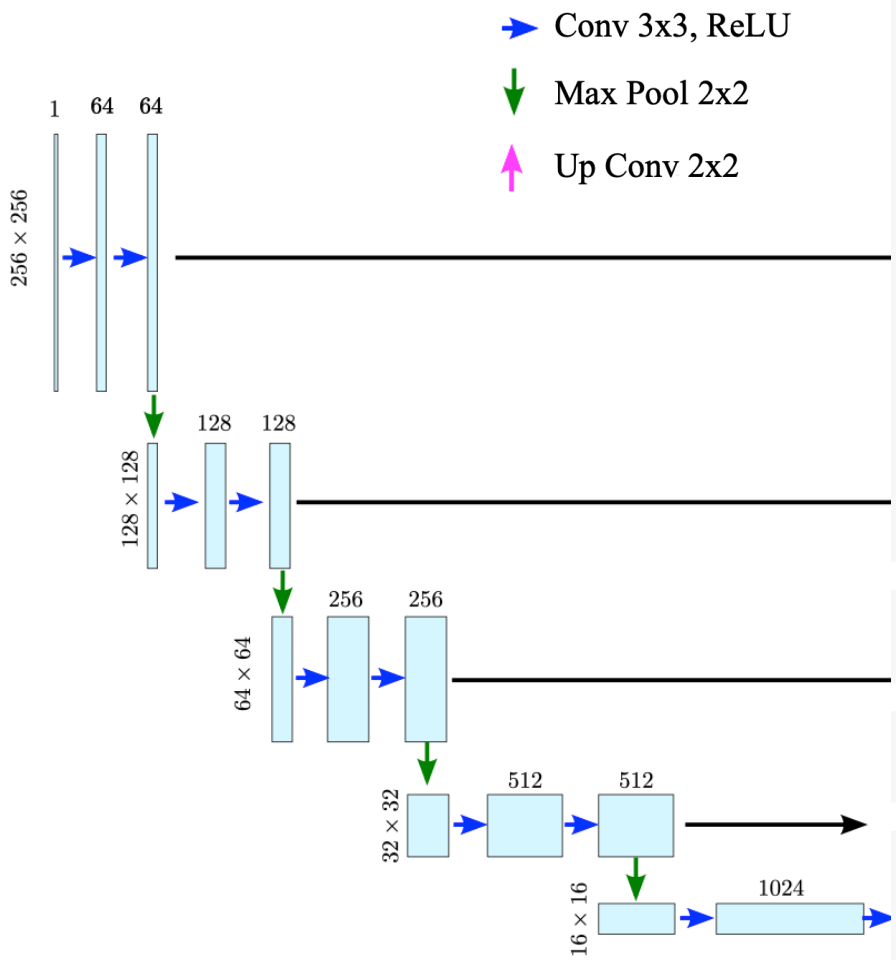
```python
in_conv = ConvBlock(in_channels=1, out_channels=64)
```

```python
output = in_conv(input)
output.shape
```

```
torch.Size([1, 64, 256, 256])
```

3 – UNet Model

Encoder – Demo




```python
# Down-Sample
class Encoder(nn.Module):
    def __init__(self, in_channels, out_channels) -> None:
        super().__init__()
        self.encoder = nn.Sequential(
            nn.MaxPool2d(2),
            ConvBlock(in_channels, out_channels)
        )

    def forward(self, x):
        x = self.encoder(x)
        return x
```

```python
encoder = Encoder(in_channels=64, out_channels=128)
```

```python
input = torch.randint(5, (1, 64, 256, 256), dtype=torch.float32)
```

```python
output = encoder(input)
output.shape
```

```
torch.Size([1, 128, 128, 128])
```

3 – UNet Model

Decoder – Demo

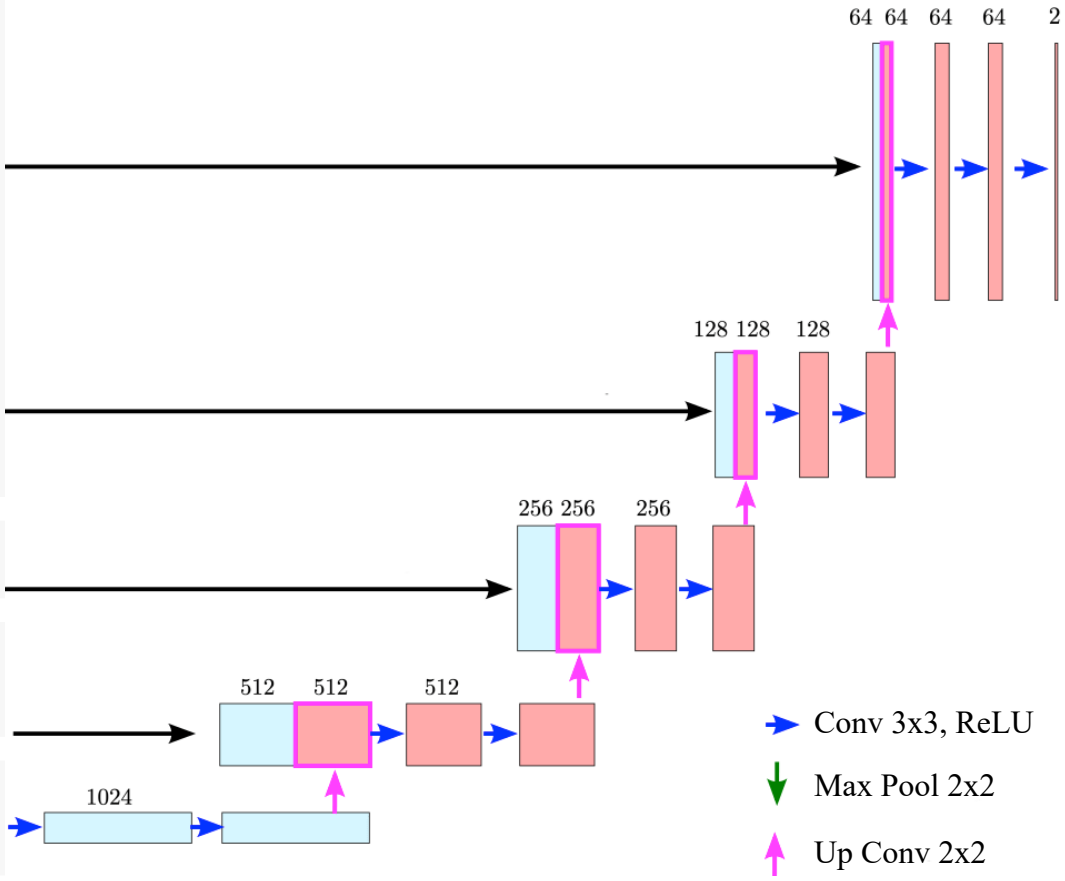
```python
# Up-Sample
class Decoder(nn.Module):
    def __init__(self, in_channels, out_channels) -> None:
        super().__init__()
        self.conv_trans = nn.ConvTranspose2d(
            in_channels, out_channels, kernel_size=4, stride=2, padding=1
        )
        self.conv_block = ConvBlock(in_channels, out_channels)

    def forward(self, x1, x2):
        x1  = self.conv_trans(x1)
        x = torch.cat([x2, x1], dim=1)
        x = self.conv_block(x)
        return x
```

```python
decoder = Decoder(in_channels=128, out_channels=64)
```

```python
x1 = torch.randint(5, (1, 128, 128, 128), dtype=torch.float32)
x2 = torch.randint(5, (1, 64, 256, 256), dtype=torch.float32)
```

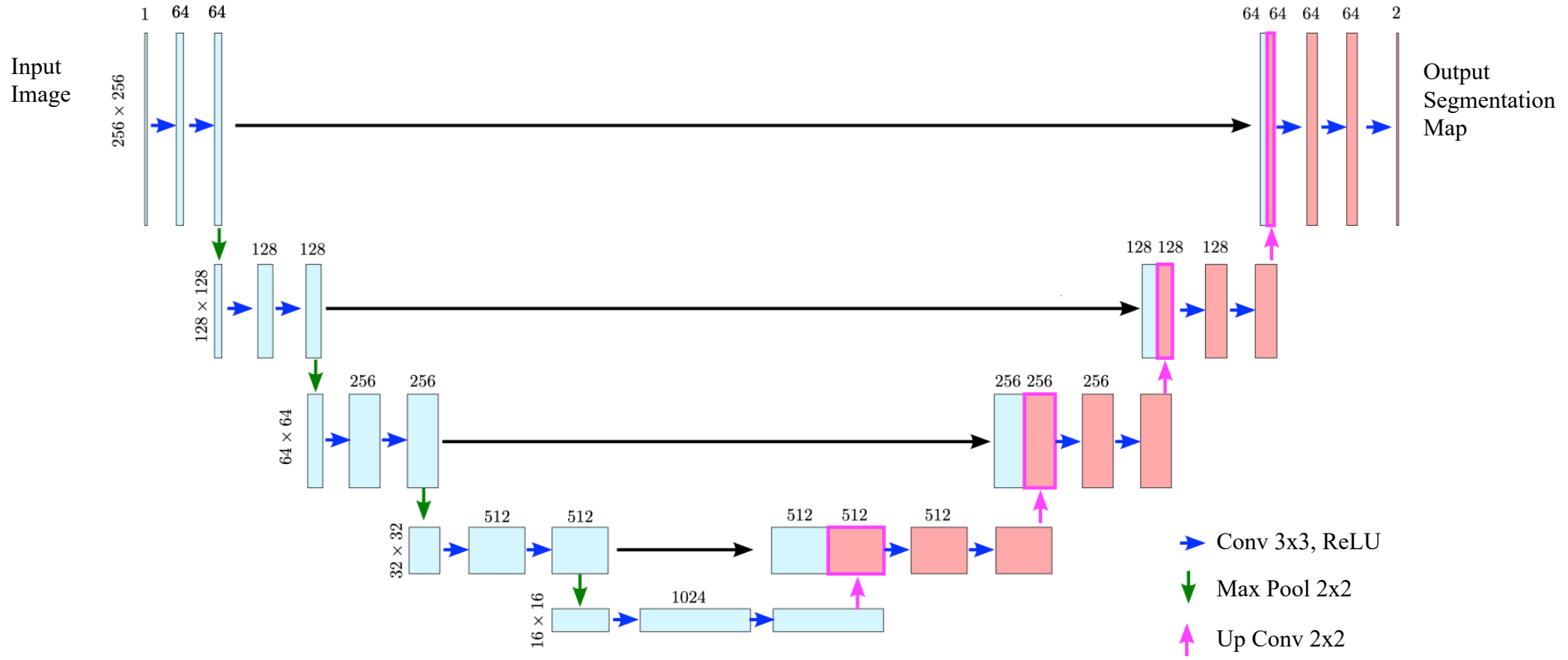
```python
output = decoder(x1, x2)
output.shape
```

```
torch.Size([1, 64, 256, 256])
```



3 – UNet Model

UNet – Demo



3 – UNet Model

UNet – Demo

```
class UNet(nn.Module):
    def __init__(self, n_channels, n_classes)
        super().__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes

        self.in_conv = ConvBlock(n_channels, 64)

        self.enc_1 = Encoder(64, 128)
        self.enc_2 = Encoder(128, 256)
        self.enc_3 = Encoder(256, 512)
        self.enc_4 = Encoder(512, 1024)

        self.dec_1 = Decoder(1024, 512)
        self.dec_2 = Decoder(512, 256)
        self.dec_3 = Decoder(256, 128)
        self.dec_4 = Decoder(128, 64)

        self.out_conv = nn.Conv2d(
            64, n_classes, kernel_size=1
        )
```

```
    def forward(self, x):
        x1 = self.in_conv(x)

        x2 = self.enc_1(x1)
        x3 = self.enc_2(x2)
        x4 = self.enc_3(x3)
        x5 = self.enc_4(x4)

        x = self.dec_1(x5, x4)
        x = self.dec_2(x, x3)
        x = self.dec_3(x, x2)
        x = self.dec_4(x, x1)

        x = self.out_conv(x)

        return x
```

```
model = UNet(n_channels=1, n_classes=2)
```

```
input = torch.randint(
    5, (4, 1, 256, 256), dtype=torch.float32
)
```

```
predictions = model(input)
```

```
predictions.shape
```

```
torch.Size([4, 2, 256, 256])
```

AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?