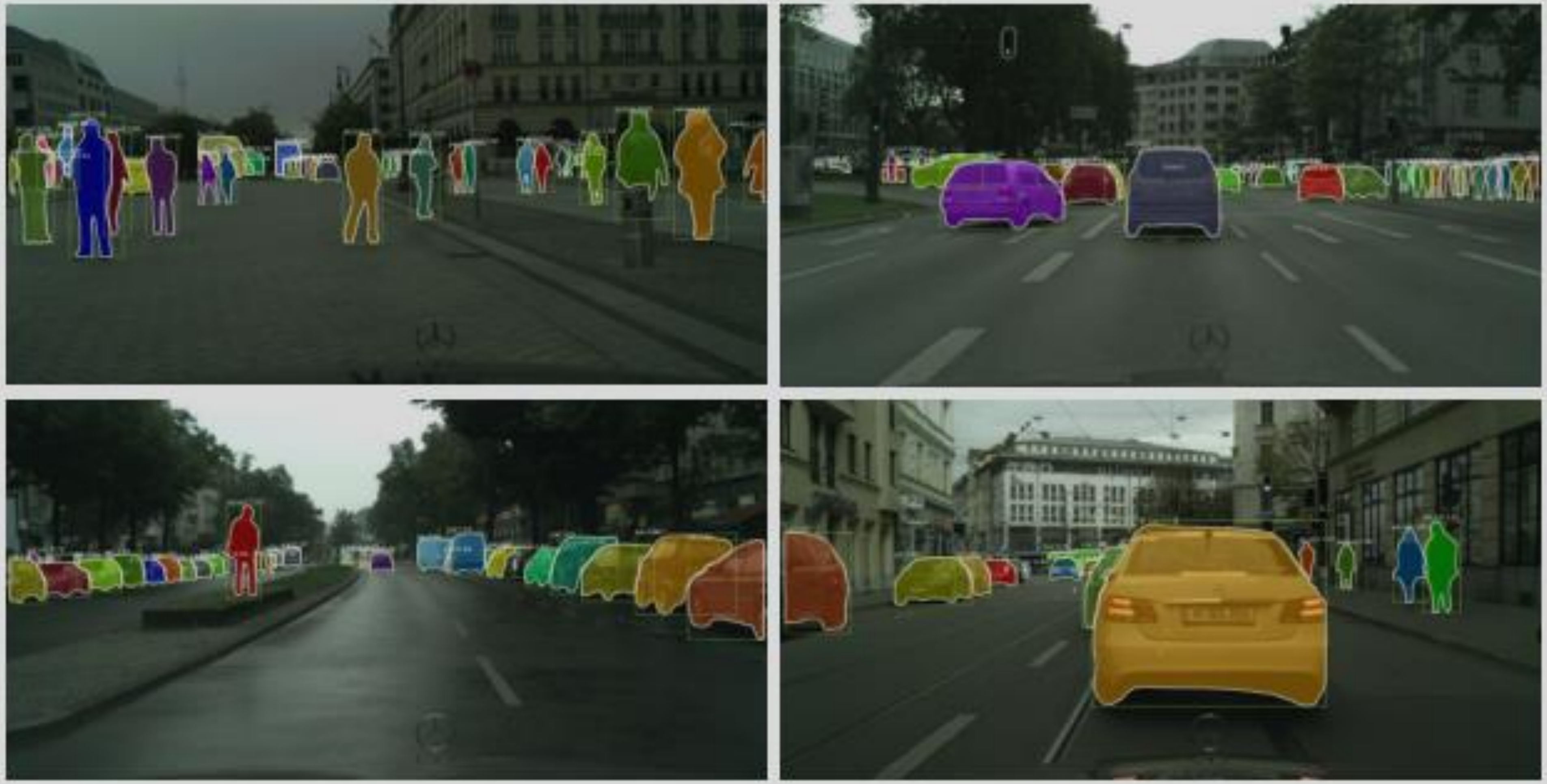
Computer Vision

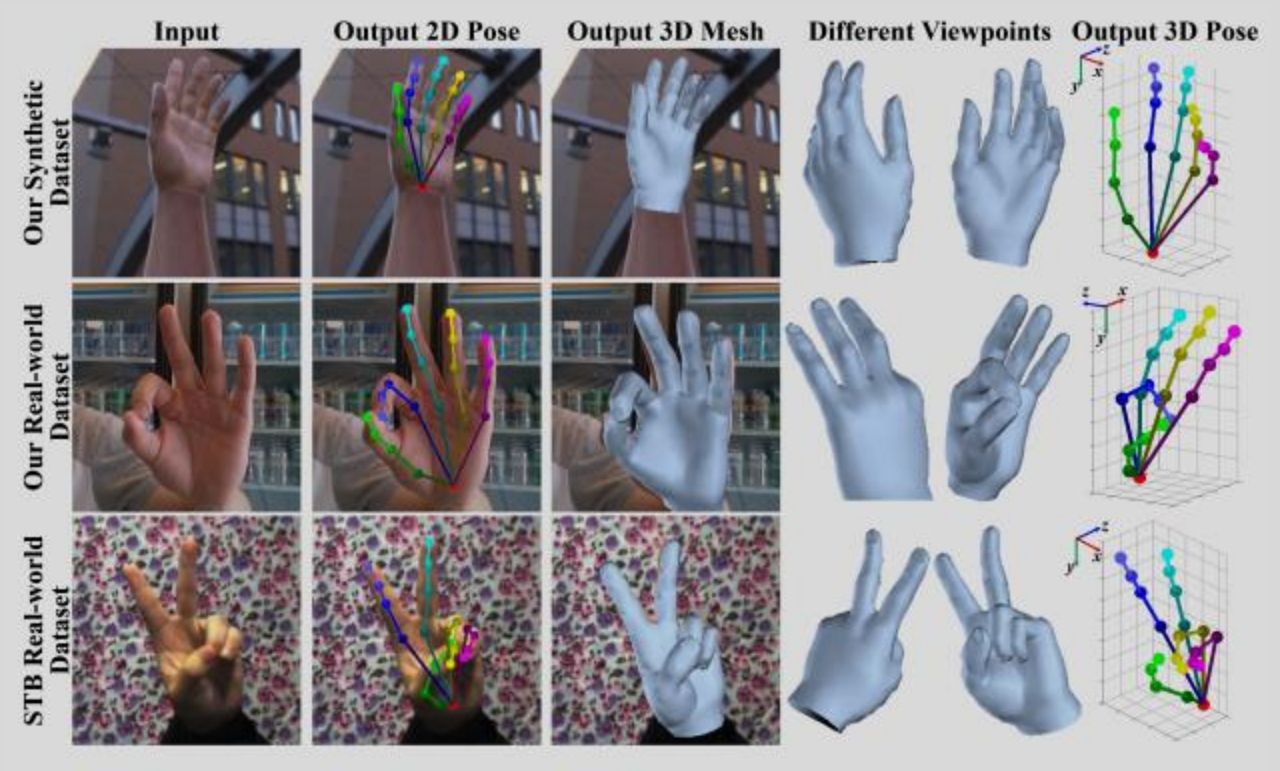
Lecture 05: Segmentation, Pose estimation

Computer vision applications

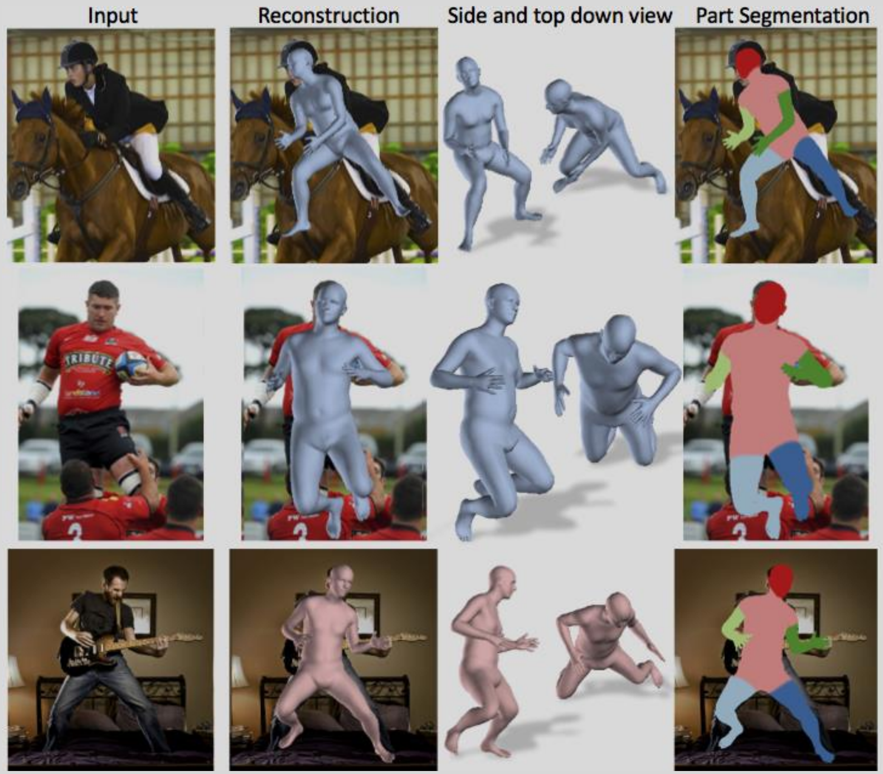


Detecting object locations and segmentation. [Mask RCNN ICCV’17]

Computer vision applications

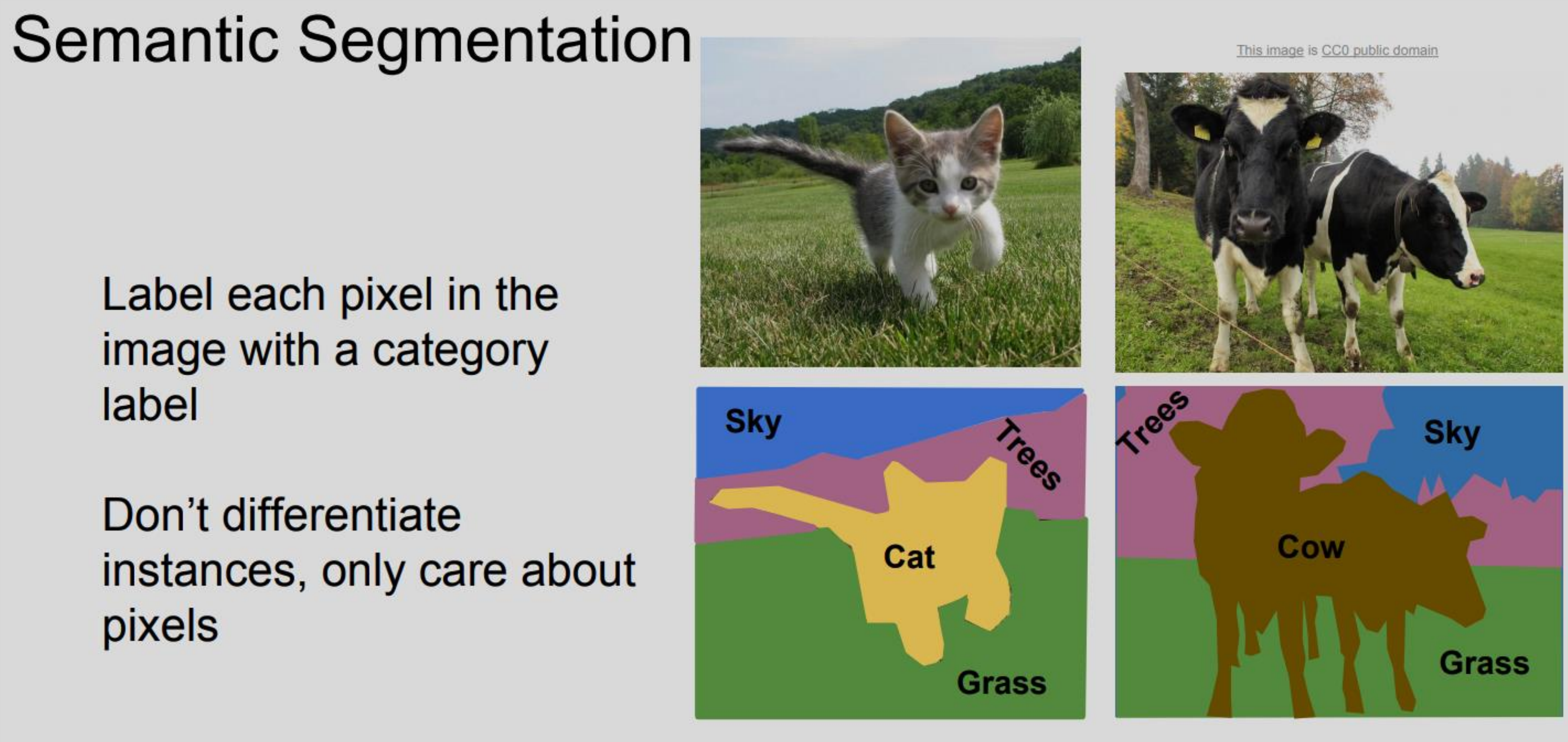


3D hand mesh reconstruction (Ge et al. CVPR'19)



3D human mesh reconstruction (Kanazawa et al. CVPR’18)

Semantic segmentation



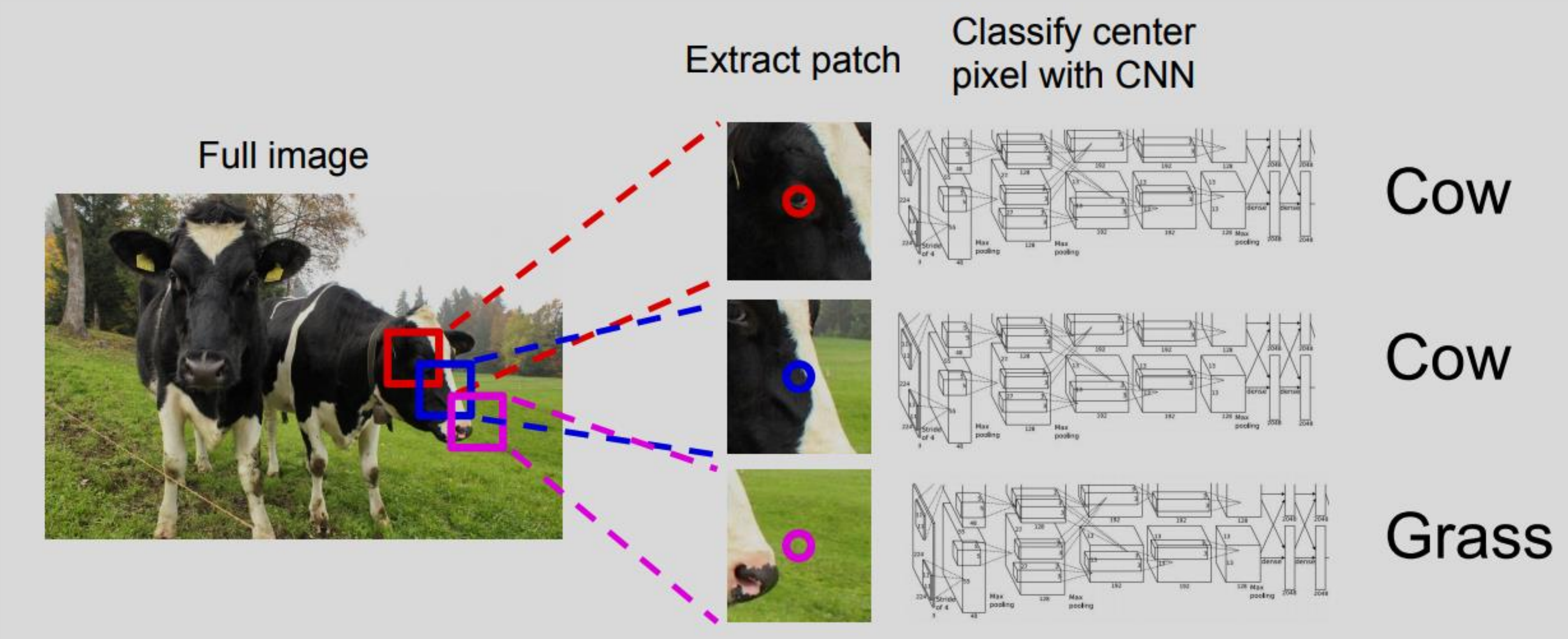
Semantic Segmentation

This image is CC0 public domain

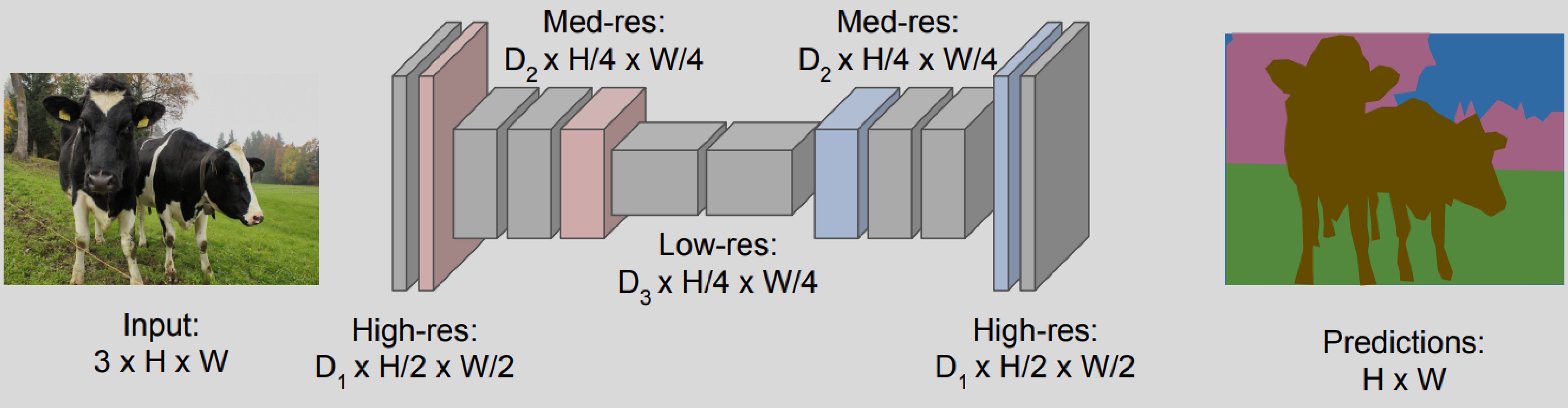
Label each pixel in the image with a category label

Don’t differentiate instances, only care about pixels

Semantic segmentation



Semantic segmentation



Input:
3 x H x W

High-res:
D_1 x H/2 x W/2

Med-res:
D_2 x H/4 x W/4

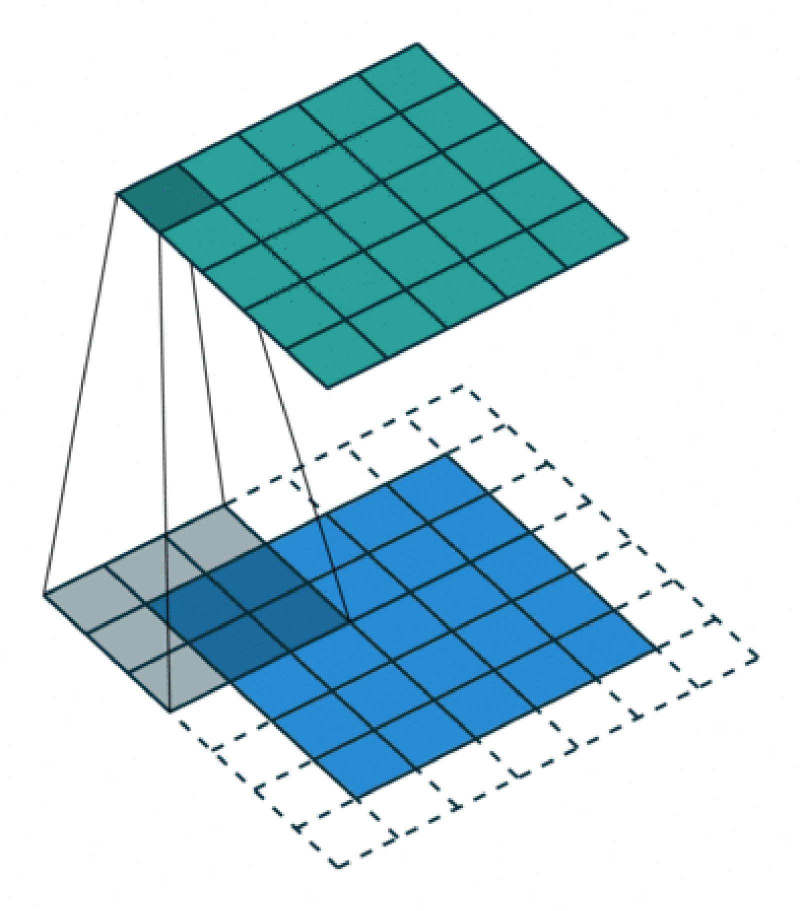
Low-res:
D_3 x H/4 x W/4

Med-res:
D_2 x H/4 x W/4

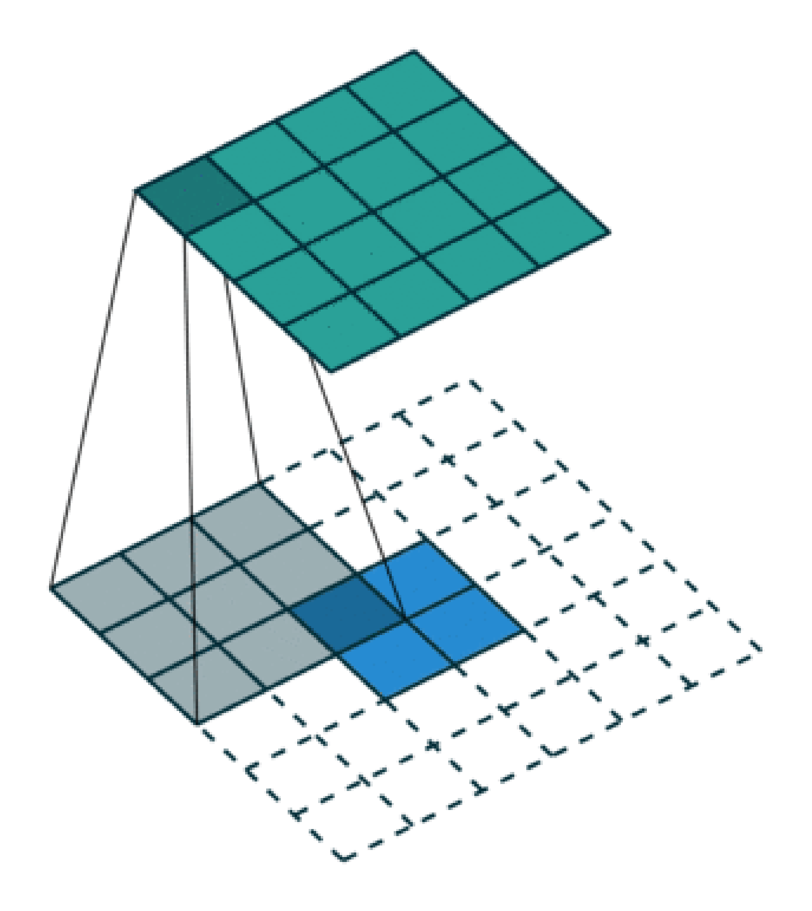
High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Transposed Convolution



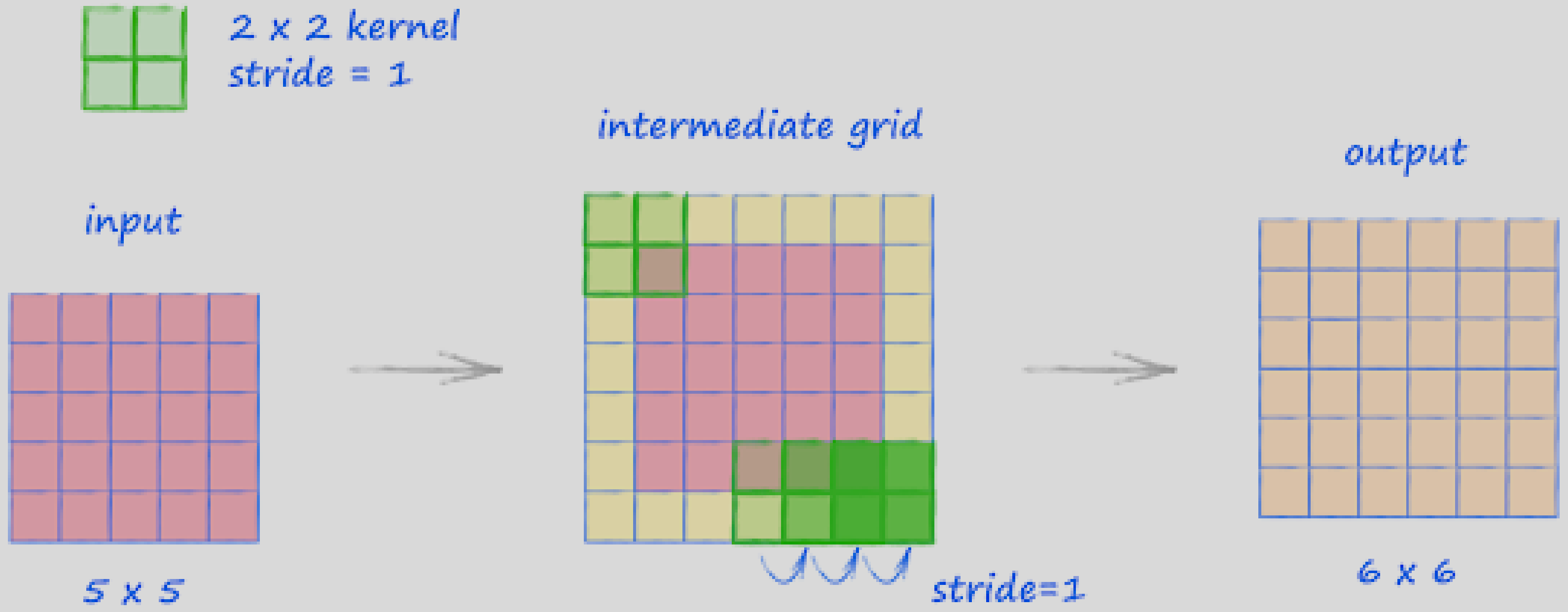
Convolution operation



Transposed convolution operation

Transposed Convolution

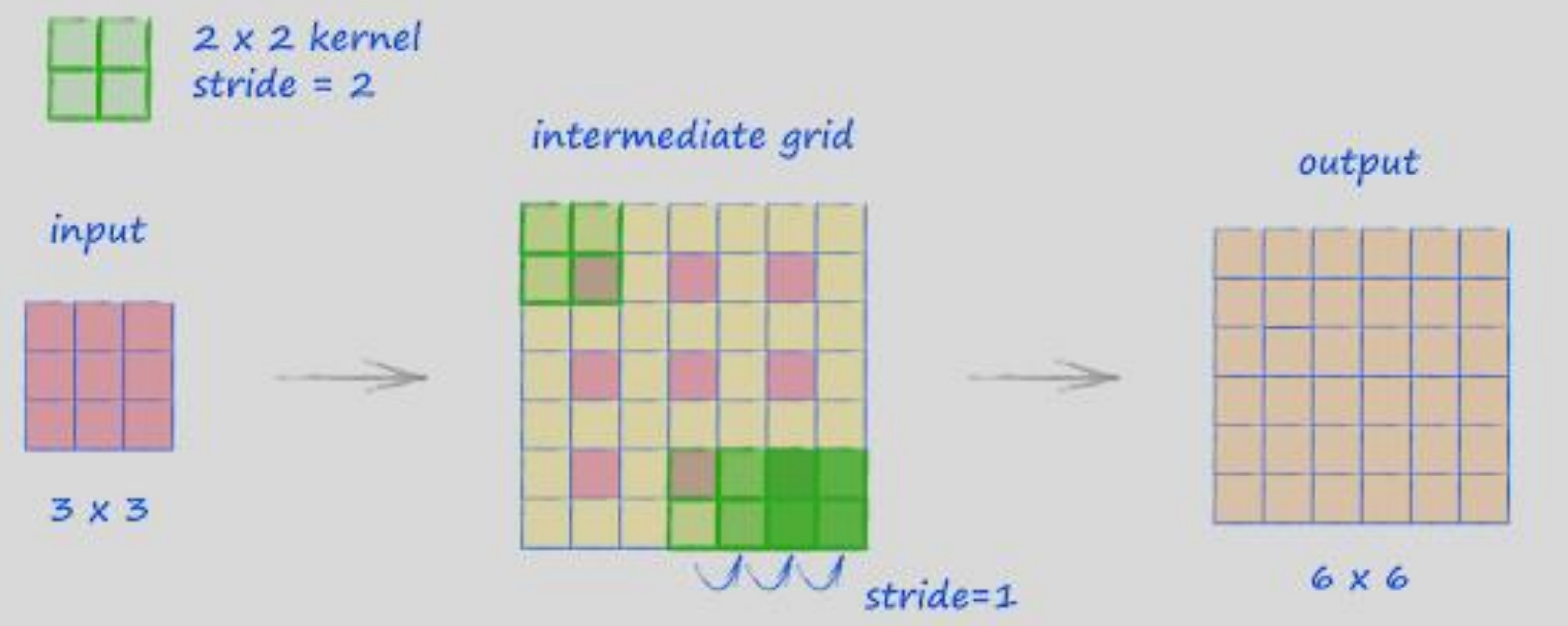
Transposed Convolution with 0 padding, stride 1, 2x2 kernel: Output_size = (input_size-1)*stride – 2*padding + kernel_size + output_padding



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

Transposed Convolution

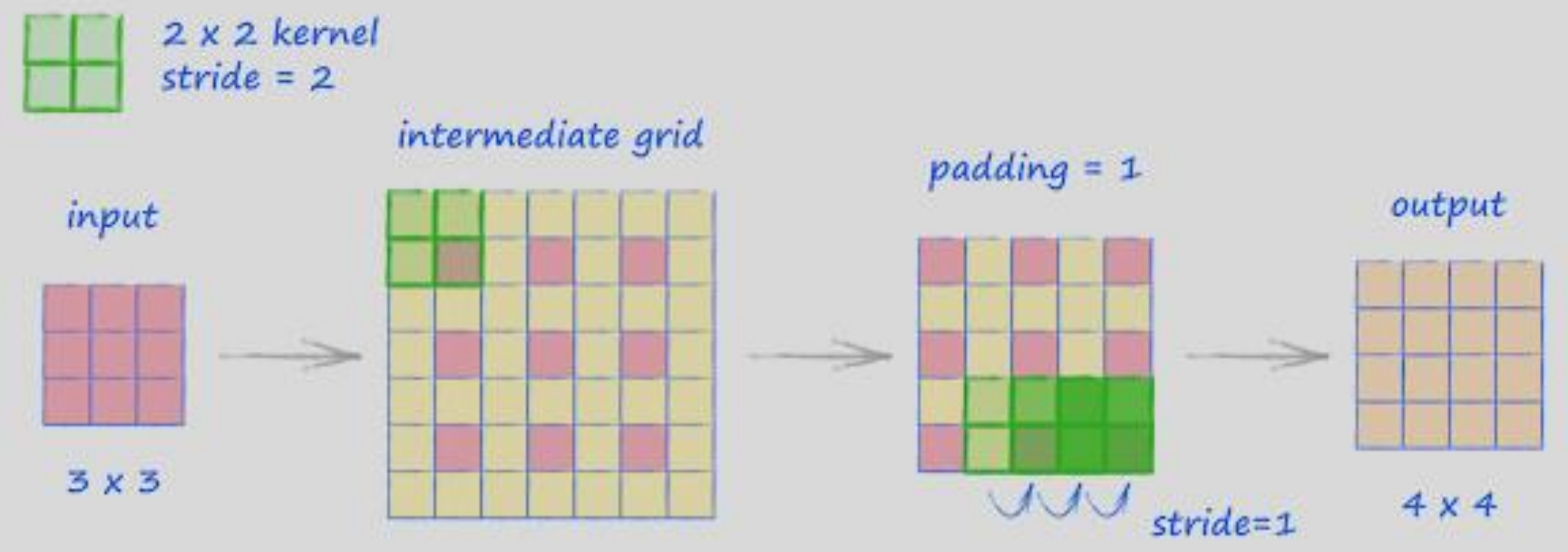
Transposed Convolution with 0 padding, stride 2, 2x2 kernel: Output_size = (input_size-1)*stride – 2*padding + kernel_size + output_padding



```
nn.ConvTranspose2d(in_channels, out_channels, kernel_size=2, stride=2)
```

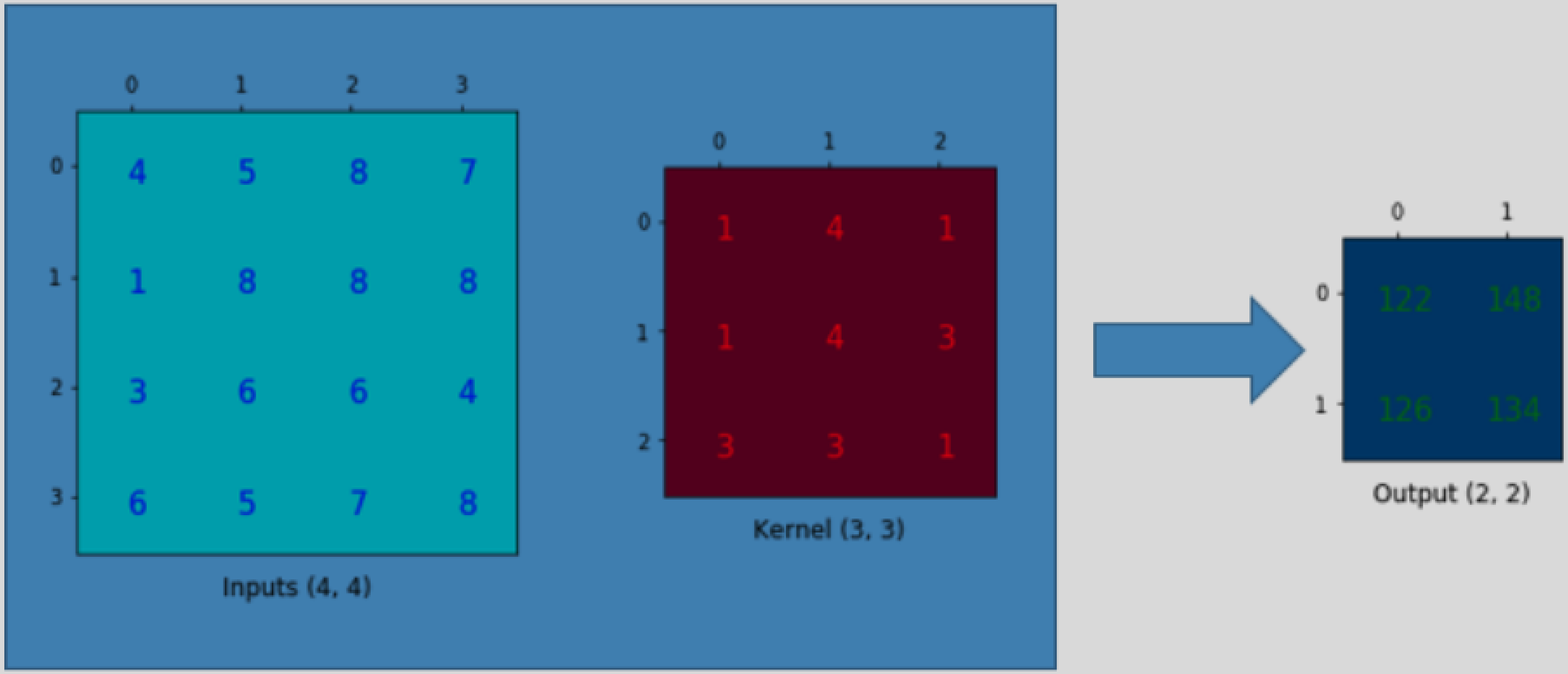
Transposed Convolution

Transposed Convolution with 1 padding, stride 2, 2x2 kernel: Output_size = (input_size-1)*stride – 2*padding + kernel_size + output_padding



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

Why called as Transpose Convoluton?



	0	1	2	3
0	4	5	8	7
1	1	8	8	8
2	3	6	6	4
3	6	5	7	8

Inputs (4, 4)

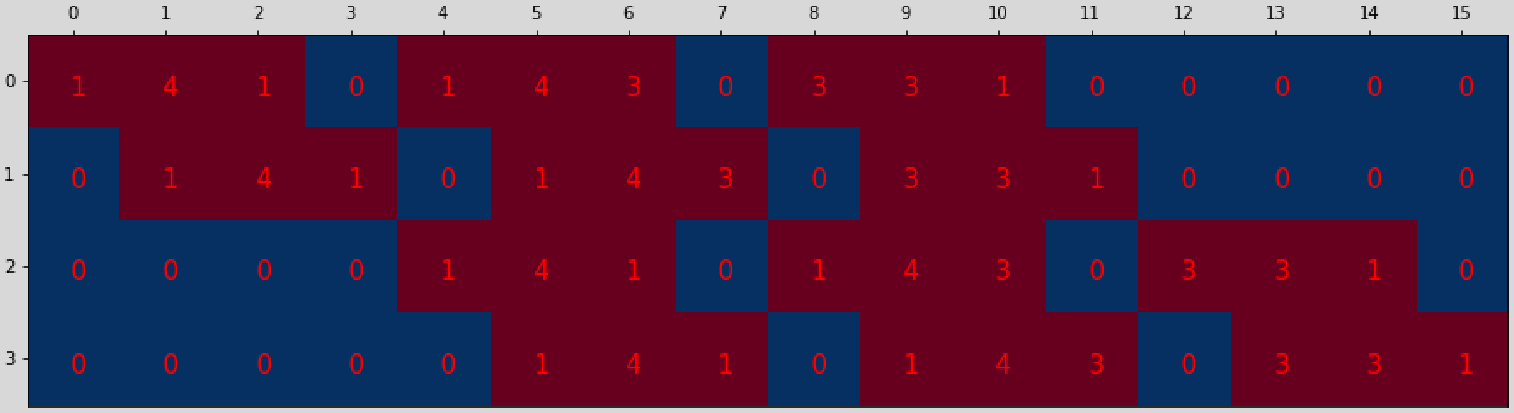
	0	1	2
0	1	4	1
1	1	4	3
2	3	3	1

Kernel (3, 3)

	0	1
0	122	148
1	126	134

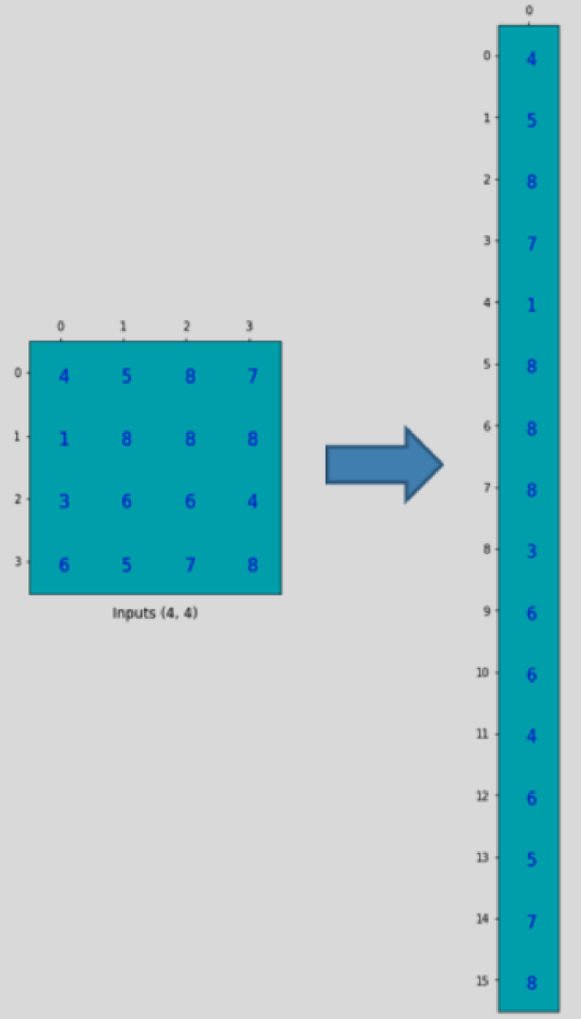
Output (2, 2)

Why called as Transpose Convoluton?



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
1	0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
2	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
3	0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

Convolution Matrix (4, 16)



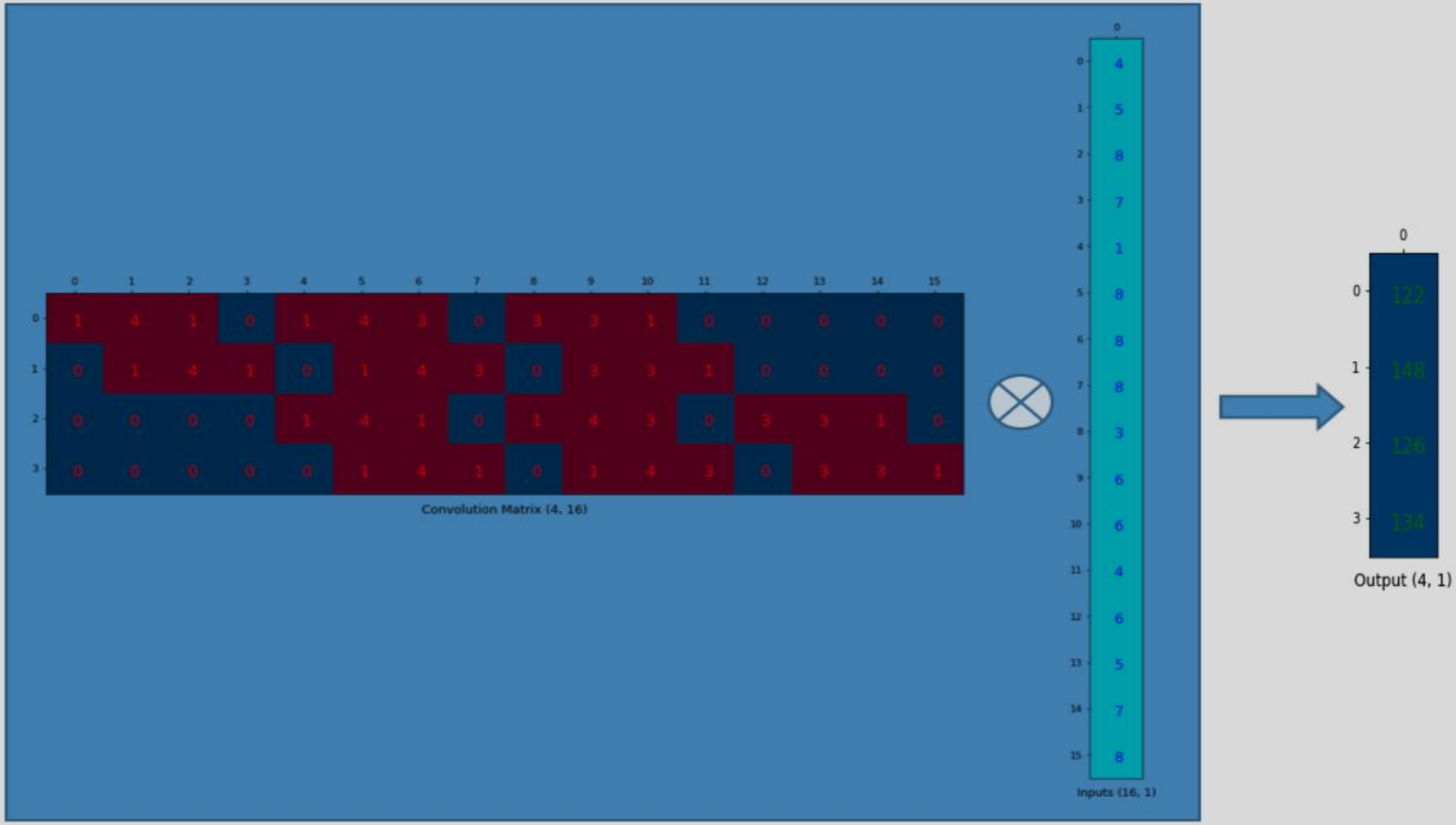
	0	1	2	3
0	4	5	8	7
1	1	8	8	8
2	3	6	6	4
3	6	5	7	8

Inputs (4, 4)

	0
0	4
1	5
2	8
3	7
4	1
5	8
6	8
7	8
8	3
9	6
10	6
11	4
12	6
13	5
14	7
15	8

Inputs (16, 1)

Why called as Transpose Convoluton?



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
1	0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
2	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
3	0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

Convolution Matrix (4, 16)

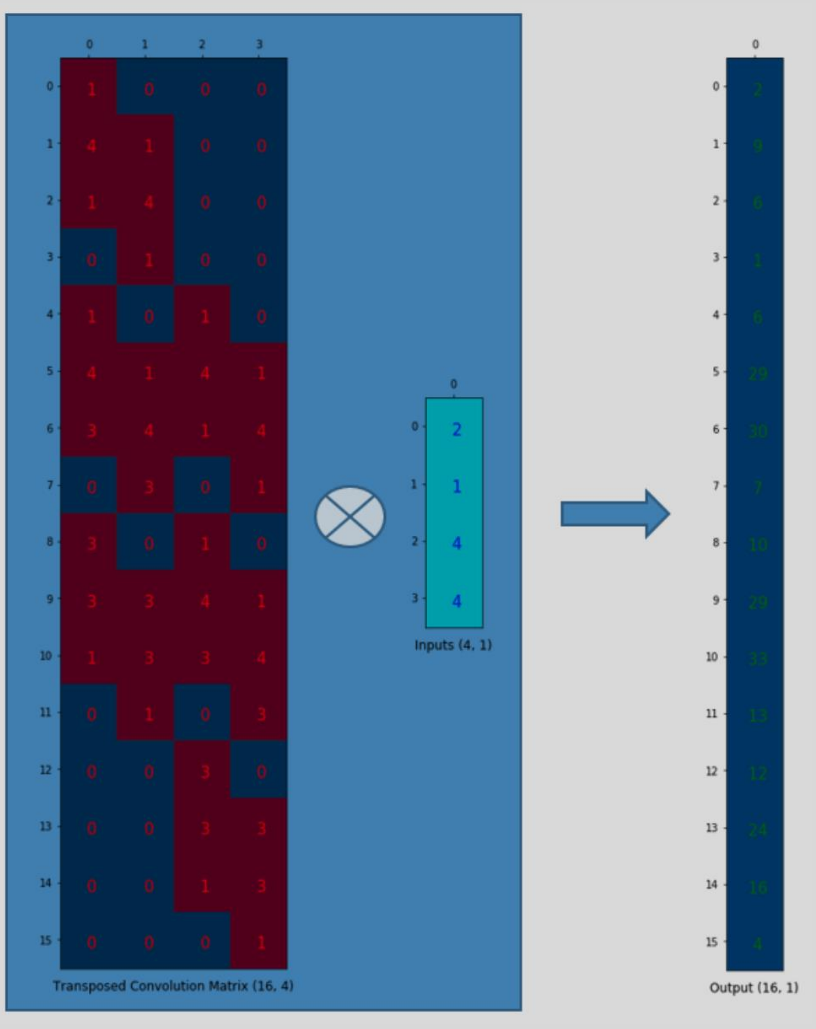
	0
0	4
1	5
2	8
3	7
4	1
5	8
6	8
7	8
8	3
9	6
10	6
11	4
12	6
13	5
14	7
15	8

Inputs (16, 1)

	0
0	122
1	148
2	126
3	134

Output (4, 1)

Why called as Transpose Convoluton?



	0	1	2	3
0	1	0	0	0
1	4	1	0	0
2	1	4	0	0
3	0	1	0	0
4	1	0	1	0
5	4	1	4	1
6	3	4	1	4
7	0	3	0	1
8	3	0	1	0
9	3	3	4	1
10	1	3	3	4
11	0	1	0	3
12	0	0	3	0
13	0	0	3	3
14	0	0	1	3
15	0	0	0	1

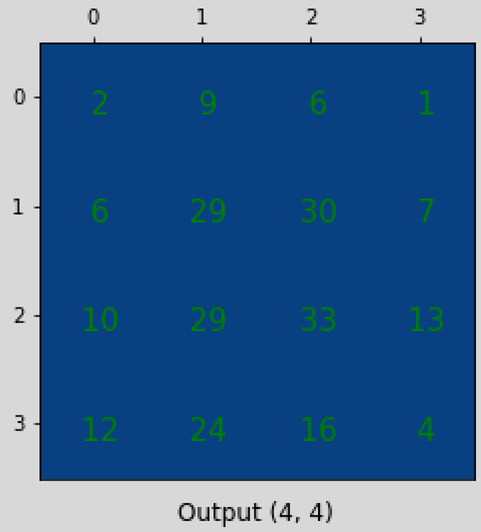
Transposed Convolution Matrix (16, 4)

	0
0	2
1	1
2	4
3	4

Inputs (4, 1)

	0
0	2
1	9
2	6
3	1
4	6
5	29
6	30
7	7
8	10
9	29
10	33
11	13
12	12
13	24
14	16
15	4

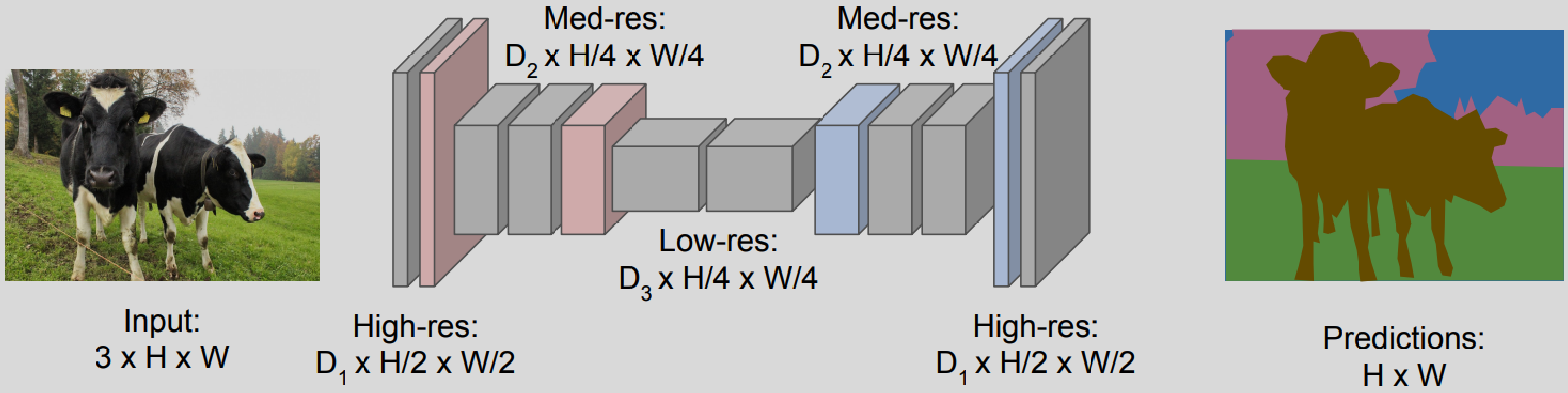
Output (16, 1)



	0	1	2	3
0	2	9	6	1
1	6	29	30	7
2	10	29	33	13
3	12	24	16	4

Output (4, 4)

Semantic segmentation



Encoder-decoder

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, 7)
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 32, 7),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU()
        )

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

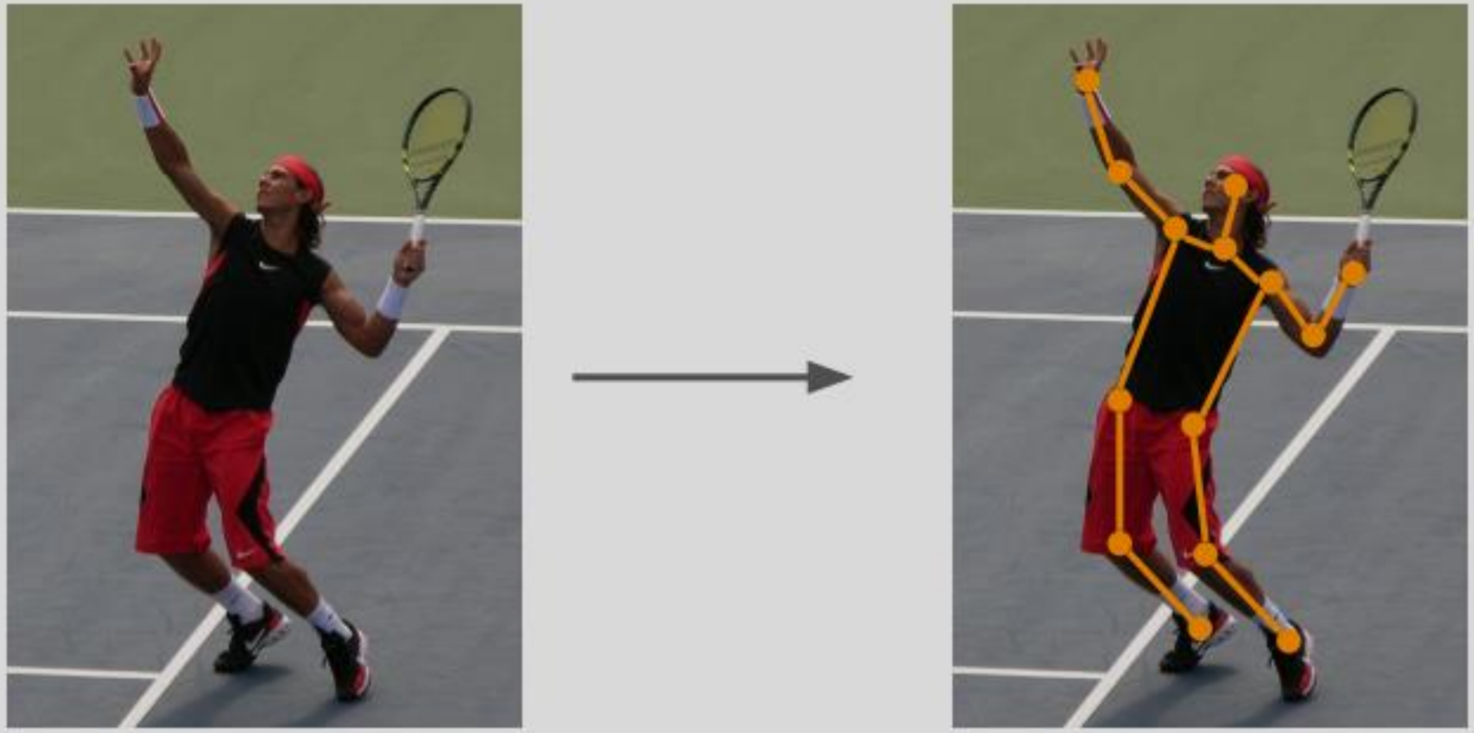
Encoder-decoder

```
class SegNet(nn.Module):
    def __init__(self, numObj):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, 7)
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 32, 7),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, numObj, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU()
        )

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

Encoder-decoder

```
numObj = 10
model = SegNet(numObj)
model.train()
criterion = torch.nn.CrossEntropyLoss()

for epoch in range(NUM_EPOCHS):

    for batch in train_dataloader:

        input = torch.autograd.Variable(batch['image'])
        target = torch.autograd.Variable(batch['mask'])

        predicted = model(input)
        output = torch.nn.functional.softmax(predicted, dim=1)

        optimizer.zero_grad()
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
```

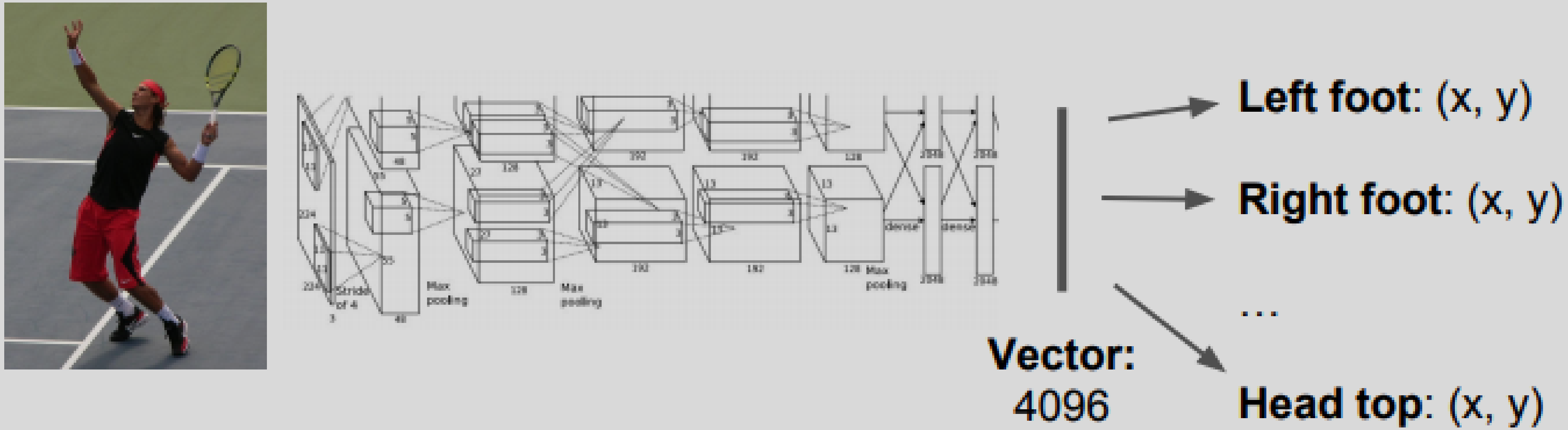
Pose estimation



Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

Pose estimation



Vector: 4096

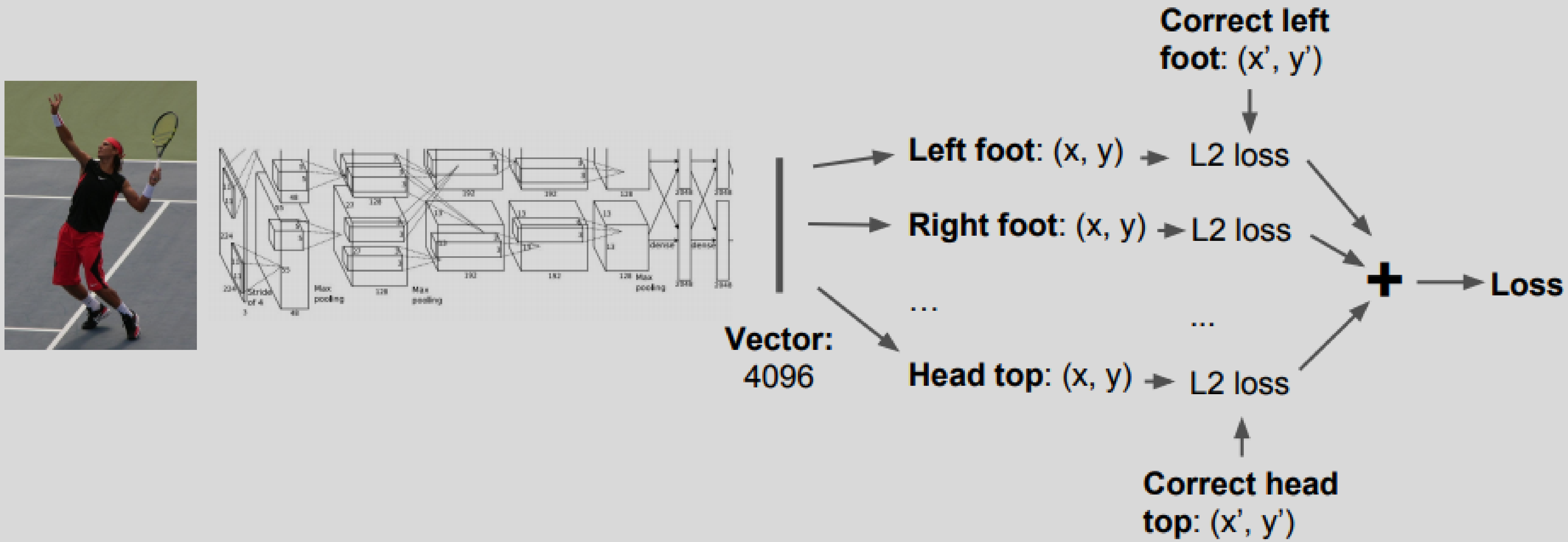
Left foot: (x, y)

Right foot: (x, y)

…

Head top: (x, y)

Pose estimation



Vector: 4096

Left foot: (x, y) → L2 loss

Right foot: (x, y) → L2 loss

…

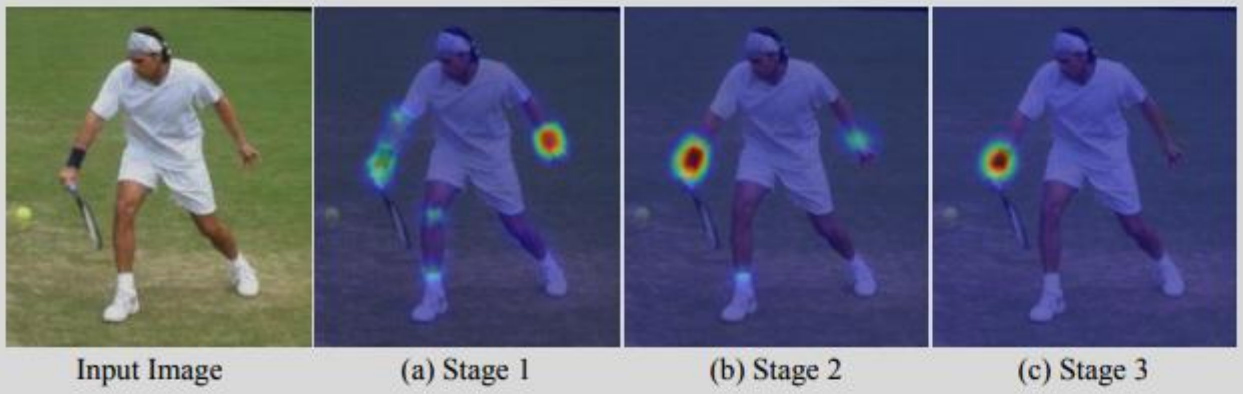
Head top: (x, y) → L2 loss

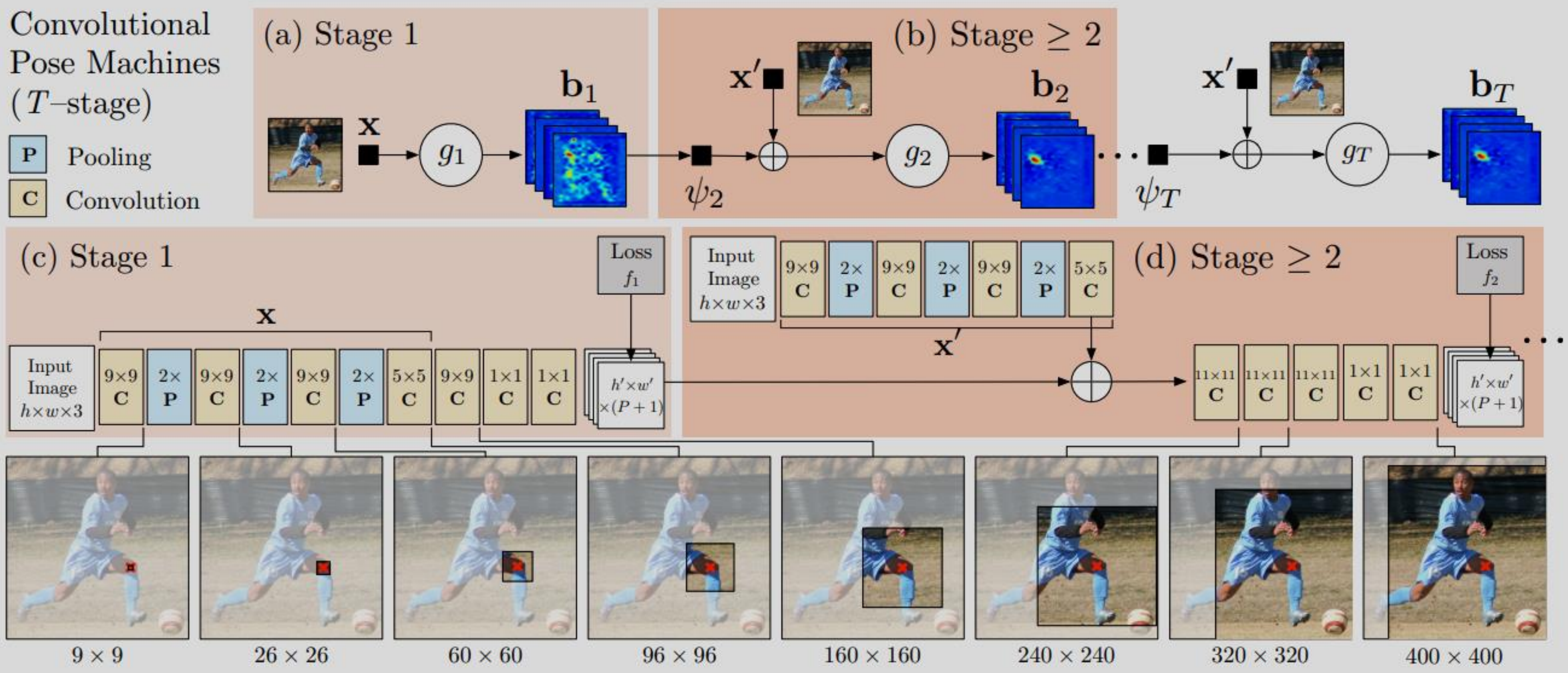
Correct left foot: (x', y')

Correct head top: (x', y')

+ → Loss

Pose estimation



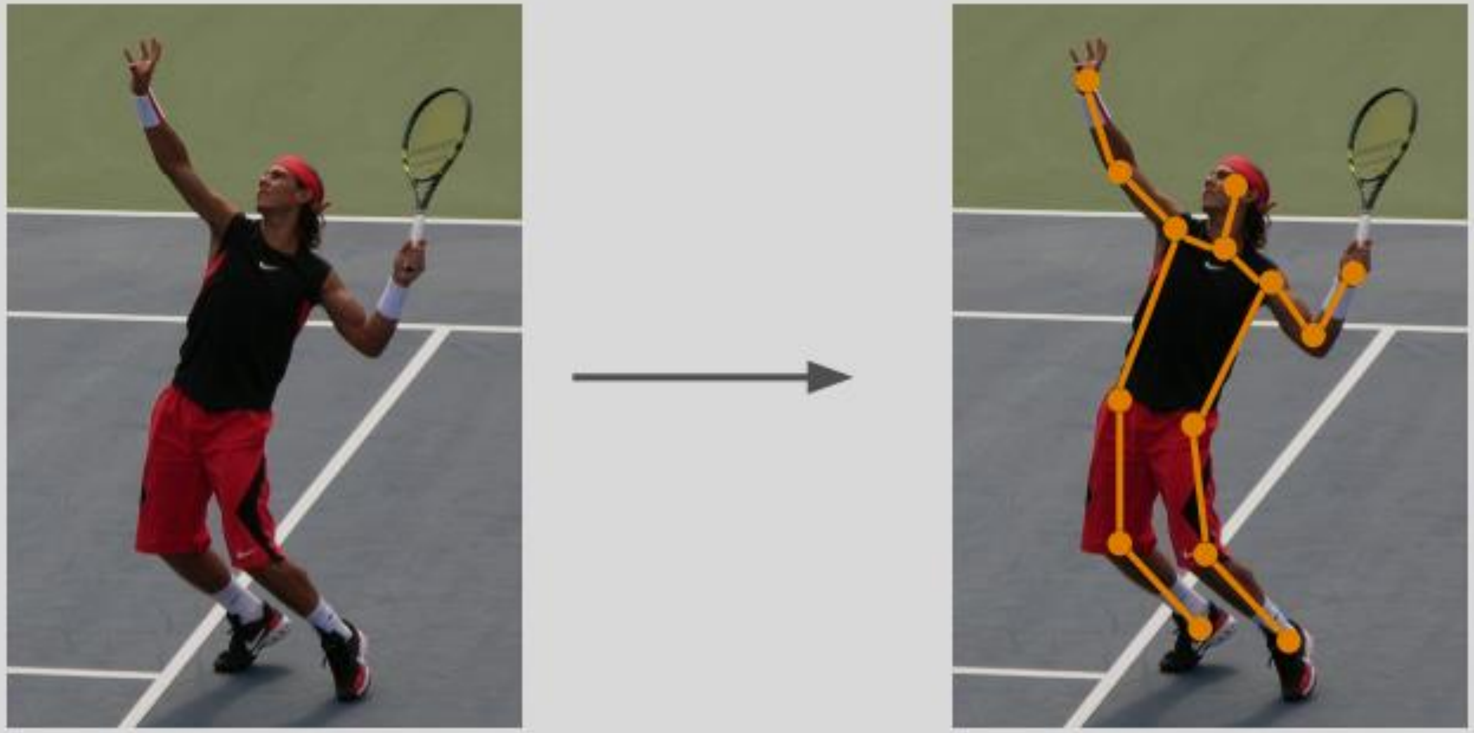


Pose estimation

```python
class CPM2DPose(nn.Module):
    def __init__(self):
        super(CPM2DPose, self).__init__()

        self.relu = F.leaky_relu
        self.conv1_1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv1_2 = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv2_1 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv2_2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv3_1 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv3_2 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv3_3 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv3_4 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_1 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_2 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_3 = nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_4 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_5 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_6 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_7 = nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv5_1 = nn.Conv2d(128, 512, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv5_2 = nn.Conv2d(512, 21, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv6_1 = nn.Conv2d(149, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_2 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_3 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_4 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_5 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_6 = nn.Conv2d(128, 128, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv6_7 = nn.Conv2d(128, 21, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv7_1 = nn.Conv2d(149, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_2 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_3 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_4 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_5 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_6 = nn.Conv2d(128, 128, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv7_7 = nn.Conv2d(128, 21, kernel_size=1, stride=1, padding=0, bias=True)
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
```

```python
def forward(self, x):
    x = self.relu(self.conv1_1(x))
    x = self.relu(self.conv1_2(x))
    x = self.maxpool(x)
    x = self.relu(self.conv2_1(x))
    x = self.relu(self.conv2_2(x))
    x = self.maxpool(x)
    x = self.relu(self.conv3_1(x))
    x = self.relu(self.conv3_2(x))
    x = self.relu(self.conv3_3(x))
    x = self.relu(self.conv3_4(x))
    x = self.maxpool(x)
    x = self.relu(self.conv4_1(x))
    x = self.relu(self.conv4_2(x))
    x = self.relu(self.conv4_3(x))
    x = self.relu(self.conv4_4(x))
    x = self.relu(self.conv4_5(x))
    x = self.relu(self.conv4_6(x))
    encoding = self.relu(self.conv4_7(x))
    x = self.relu(self.conv5_1(encoding))
    scoremap = self.conv5_2(x)

    x = torch.cat([scoremap, encoding],1)
    x = self.relu(self.conv6_1(x))
    x = self.relu(self.conv6_2(x))
    x = self.relu(self.conv6_3(x))
    x = self.relu(self.conv6_4(x))
    x = self.relu(self.conv6_5(x))
    x = self.relu(self.conv6_6(x))
    scoremap = self.conv6_7(x)

    x = torch.cat([scoremap, encoding], 1)
    x = self.relu(self.conv7_1(x))
    x = self.relu(self.conv7_2(x))
    x = self.relu(self.conv7_3(x))
    x = self.relu(self.conv7_4(x))
    x = self.relu(self.conv7_5(x))
    x = self.relu(self.conv7_6(x))
    x = self.conv7_7(x)
    return x
```

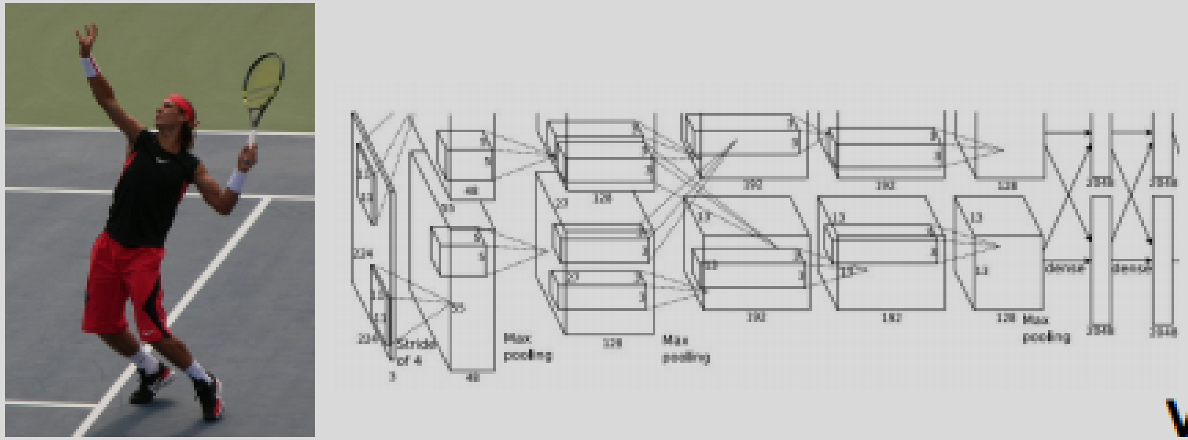
Pose estimation



Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

Pose estimation



Vector: 4096

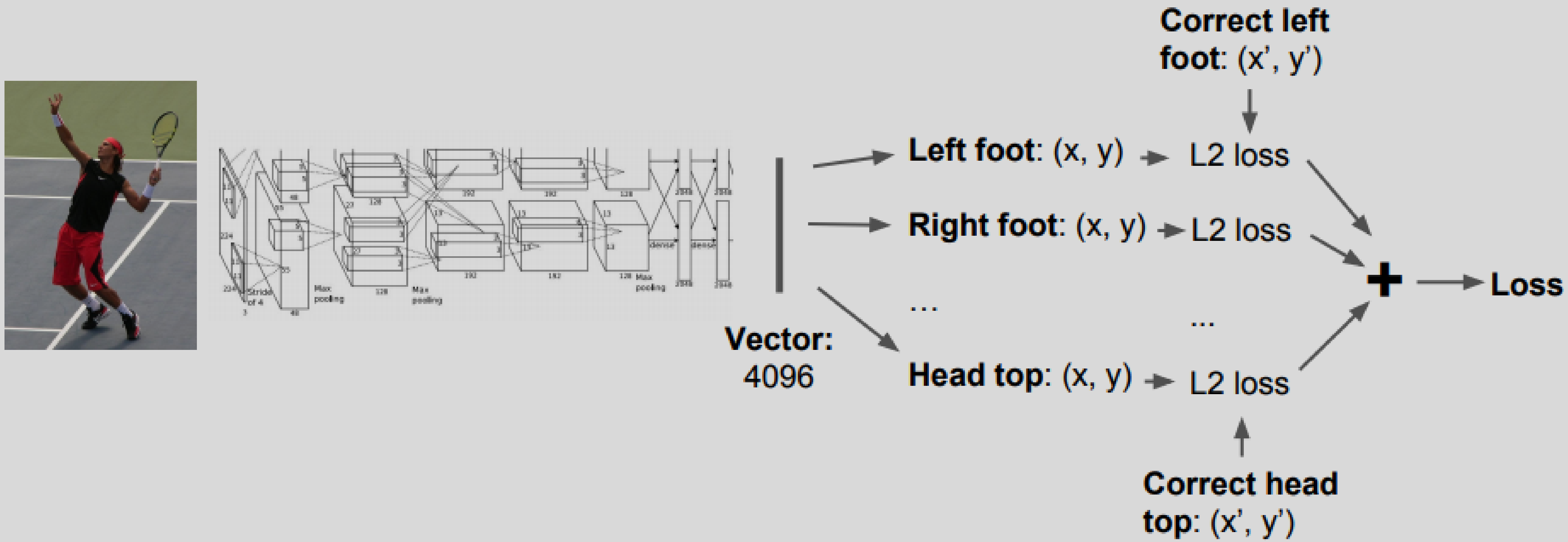
Left foot: (x, y)

Right foot: (x, y)

…

Head top: (x, y)

Pose estimation



Vector: 4096

Left foot: (x, y) → L2 loss

Right foot: (x, y) → L2 loss

…

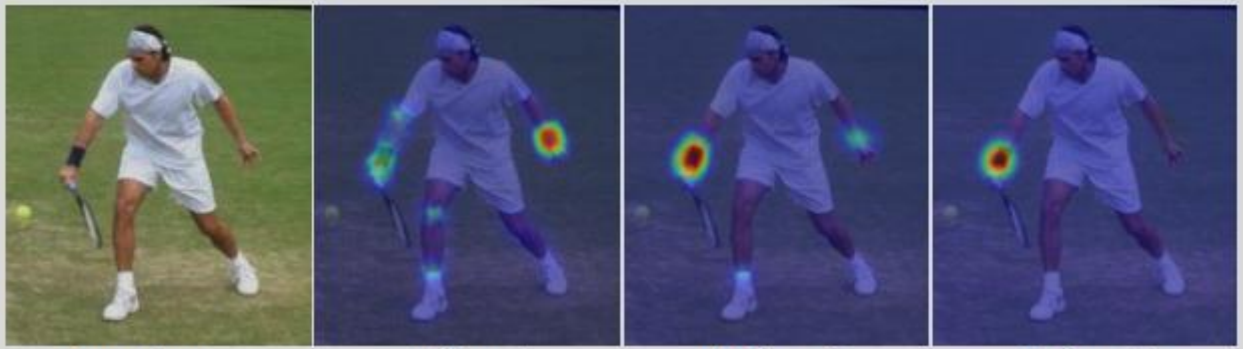
Head top: (x, y) → L2 loss

Correct left foot: (x', y')

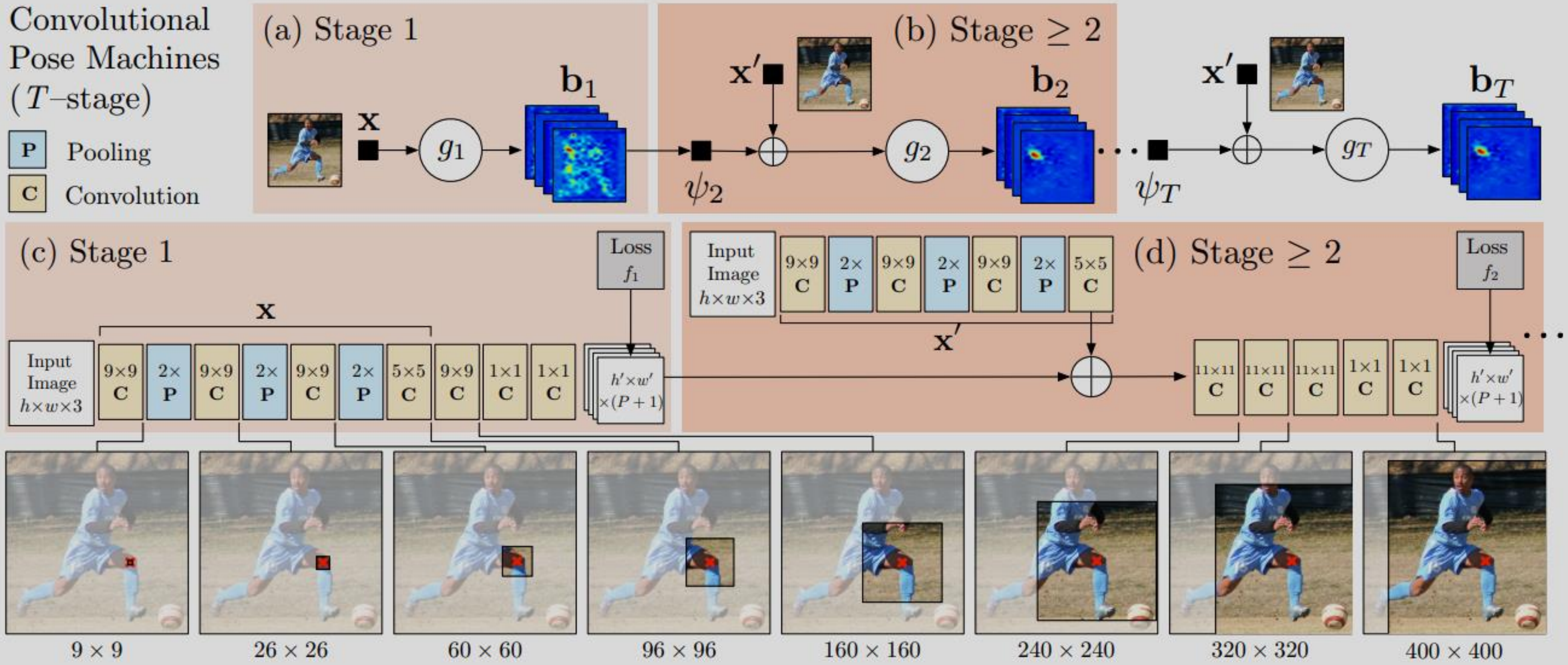
Correct head top: (x', y')

+ → **Loss**

Pose estimation



Input Image (a) Stage 1 (b) Stage 2 (c) Stage 3



Pose estimation

```
class CPM2DPose(nn.Module):
    def __init__(self):
        super(CPM2DPose, self).__init__()

        self.relu = F.leaky_relu
        self.conv1_1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv1_2 = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv2_1 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv2_2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv3_1 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv3_2 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv3_3 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv3_4 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_1 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_2 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_3 = nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_4 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_5 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_6 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv4_7 = nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv5_1 = nn.Conv2d(128, 512, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv5_2 = nn.Conv2d(512, 21, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv6_1 = nn.Conv2d(149, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_2 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_3 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_4 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_5 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv6_6 = nn.Conv2d(128, 128, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv6_7 = nn.Conv2d(128, 21, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv7_1 = nn.Conv2d(149, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_2 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_3 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_4 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_5 = nn.Conv2d(128, 128, kernel_size=7, stride=1, padding=3, bias=True)
        self.conv7_6 = nn.Conv2d(128, 128, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv7_7 = nn.Conv2d(128, 21, kernel_size=1, stride=1, padding=0, bias=True)
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
```

```
def forward(self, x):
    x = self.relu(self.conv1_1(x))
    x = self.relu(self.conv1_2(x))
    x = self.maxpool(x)
    x = self.relu(self.conv2_1(x))
    x = self.relu(self.conv2_2(x))
    x = self.maxpool(x)
    x = self.relu(self.conv3_1(x))
    x = self.relu(self.conv3_2(x))
    x = self.relu(self.conv3_3(x))
    x = self.relu(self.conv3_4(x))
    x = self.maxpool(x)
    x = self.relu(self.conv4_1(x))
    x = self.relu(self.conv4_2(x))
    x = self.relu(self.conv4_3(x))
    x = self.relu(self.conv4_4(x))
    x = self.relu(self.conv4_5(x))
    x = self.relu(self.conv4_6(x))
    encoding = self.relu(self.conv4_7(x))
    x = self.relu(self.conv5_1(encoding))
    scoremap = self.conv5_2(x)

    x = torch.cat([scoremap, encoding],1)
    x = self.relu(self.conv6_1(x))
    x = self.relu(self.conv6_2(x))
    x = self.relu(self.conv6_3(x))
    x = self.relu(self.conv6_4(x))
    x = self.relu(self.conv6_5(x))
    x = self.relu(self.conv6_6(x))
    scoremap = self.conv6_7(x)

    x = torch.cat([scoremap, encoding], 1)
    x = self.relu(self.conv7_1(x))
    x = self.relu(self.conv7_2(x))
    x = self.relu(self.conv7_3(x))
    x = self.relu(self.conv7_4(x))
    x = self.relu(self.conv7_5(x))
    x = self.relu(self.conv7_6(x))
    x = self.conv7_7(x)
    return x
```

Thank you!

UNIST

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

2007