AI VIETNAM
All-in-One Course

Image Domain Conversion (Model Construction)

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

Motivation

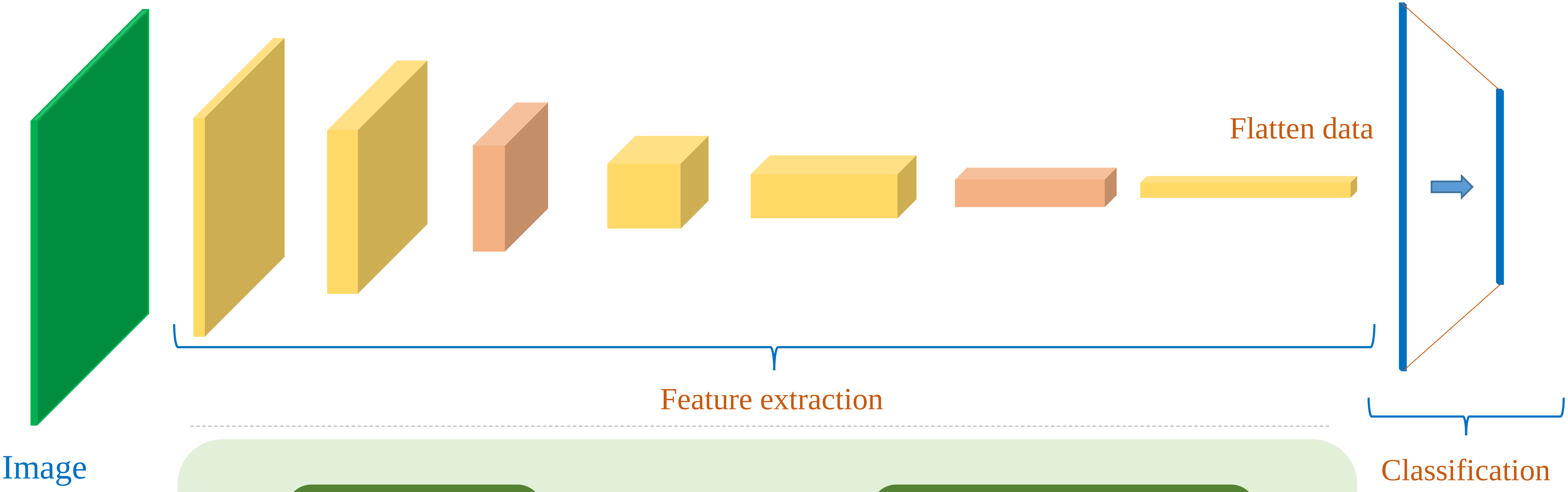


Image $\xrightarrow{\text{CNN}}$ Classification Information

Input Space → Output Space

Motivation

Image Classification

Image → CNN → Classification Information

Input Space | Output Space

Image Localization

Image → CNN → Localization Information

Input Space | Output Space

Image Detection

Image → CNN → Detection Information

Input Space | Output Space

...

Motivation

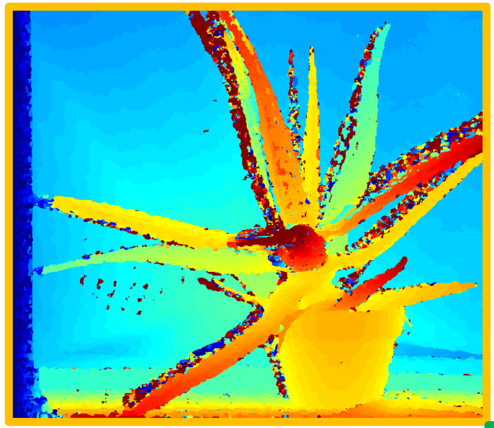


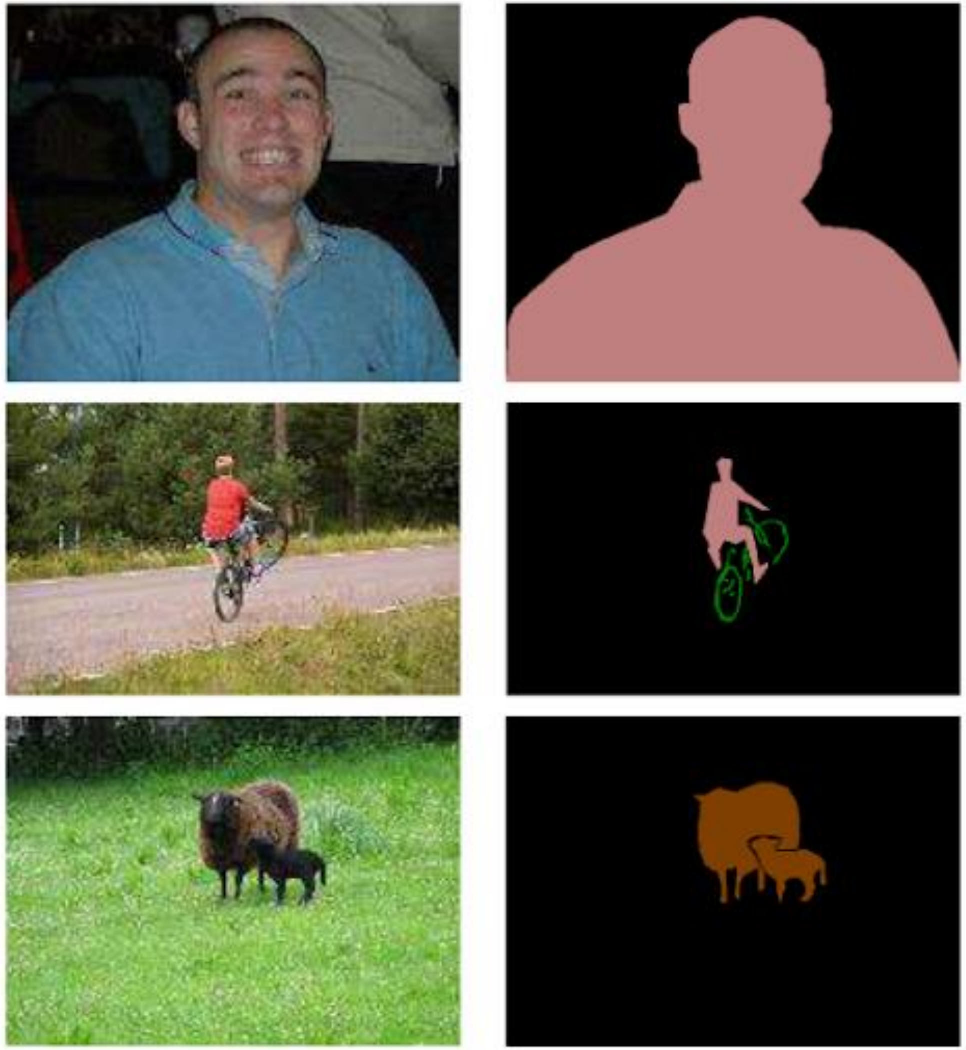
Image → **Image**

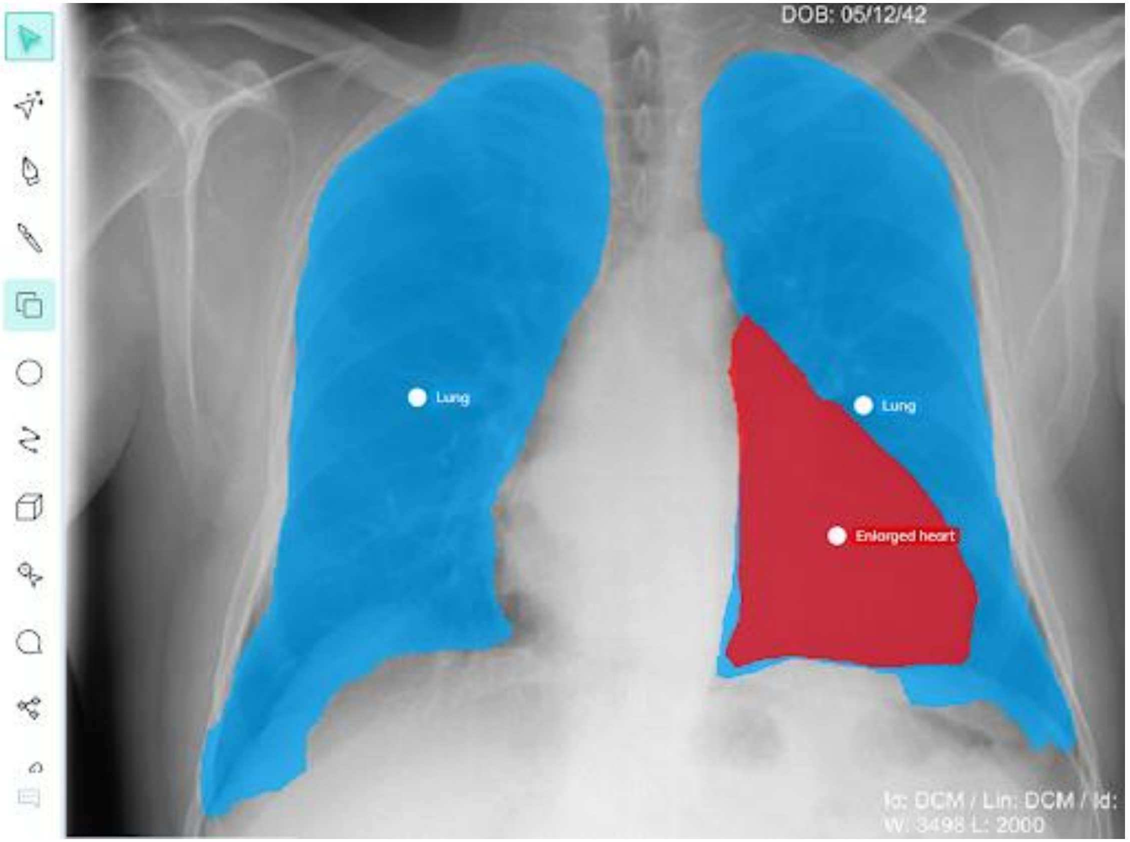
How?

Input Space

Output Space

Image Segmentation

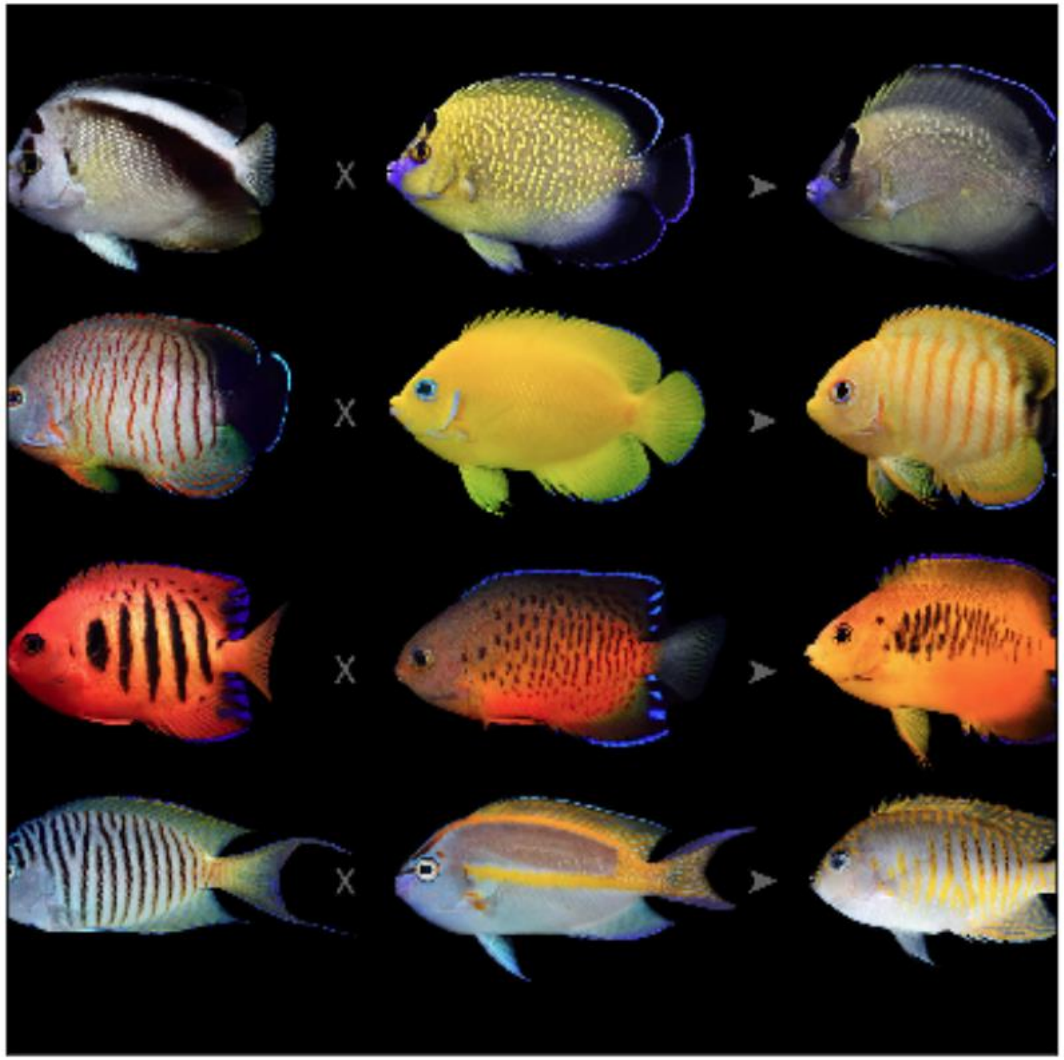




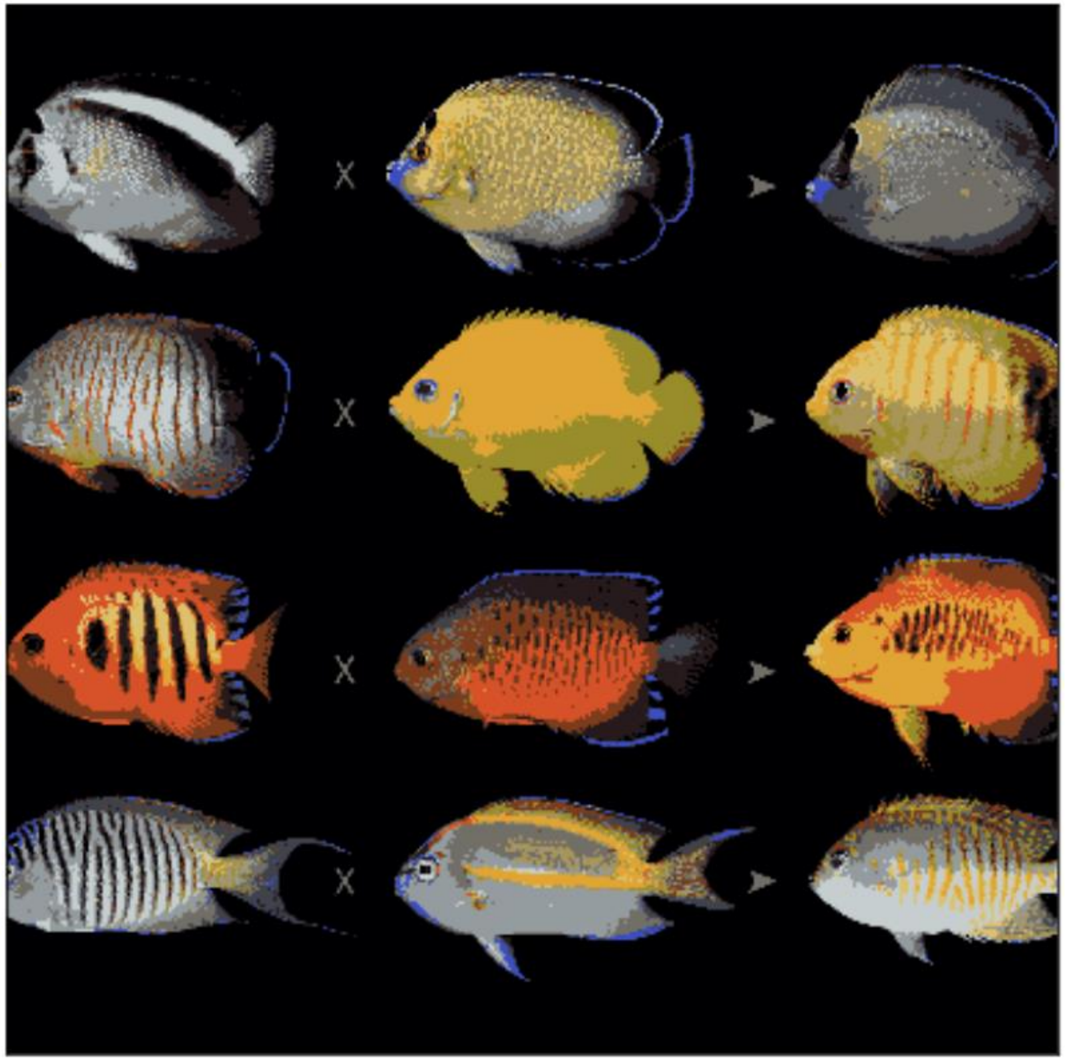
https://www.v7labs.com/blog/image-segmentation-guide

Segmentation

Traditional method



Original Image

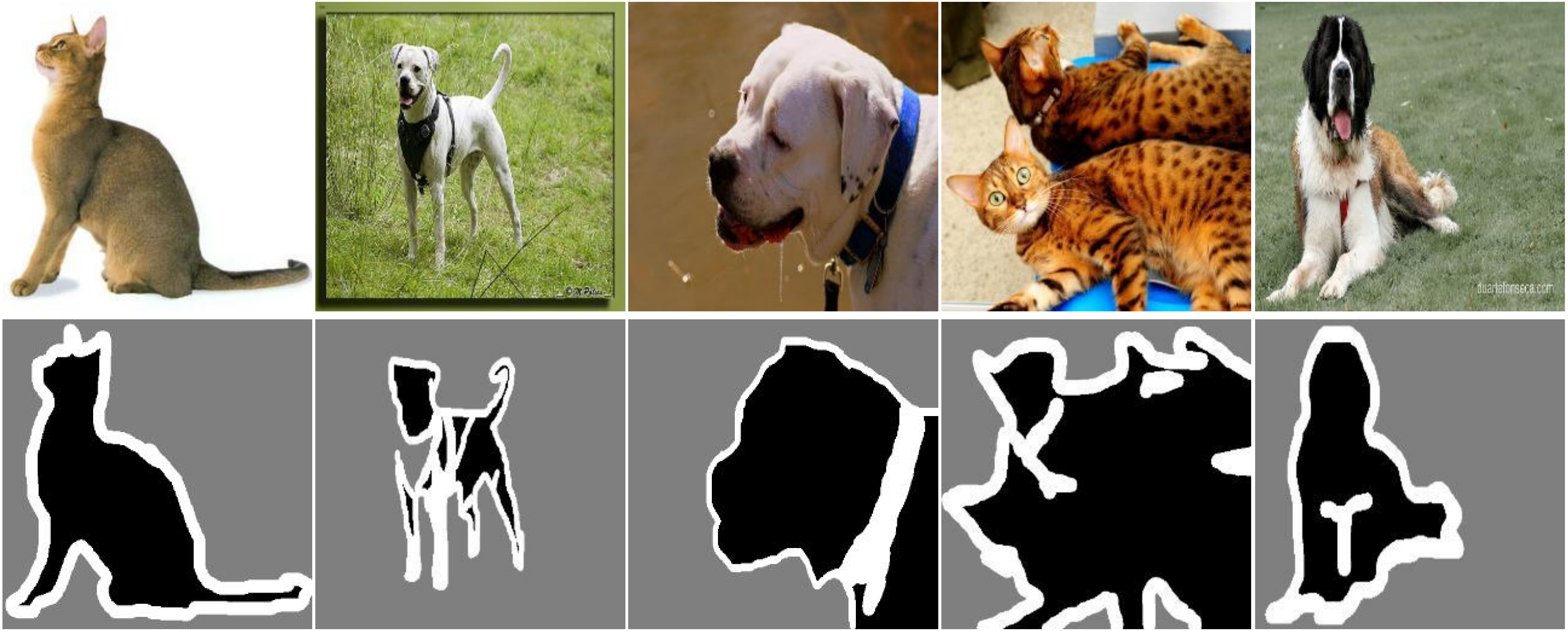


K-Mean Segmentation

Segmentation

Oxford-IIIT-Pet

https://www.robots.ox.ac.uk/~vgg/data/pets/

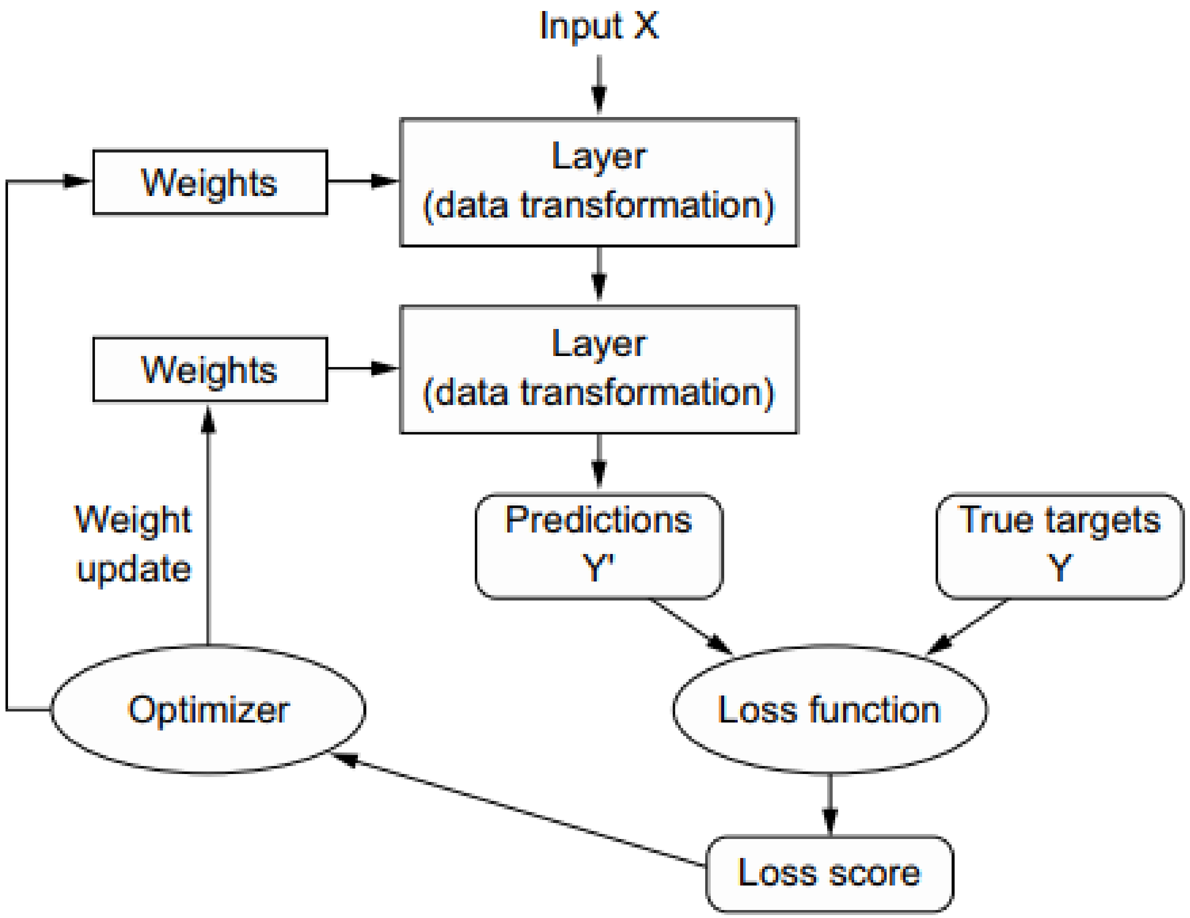


Discussion

Input/Output + Loss Function

Image Classification

Temperature Forecasting



Discussion

Input/Output + Loss Function

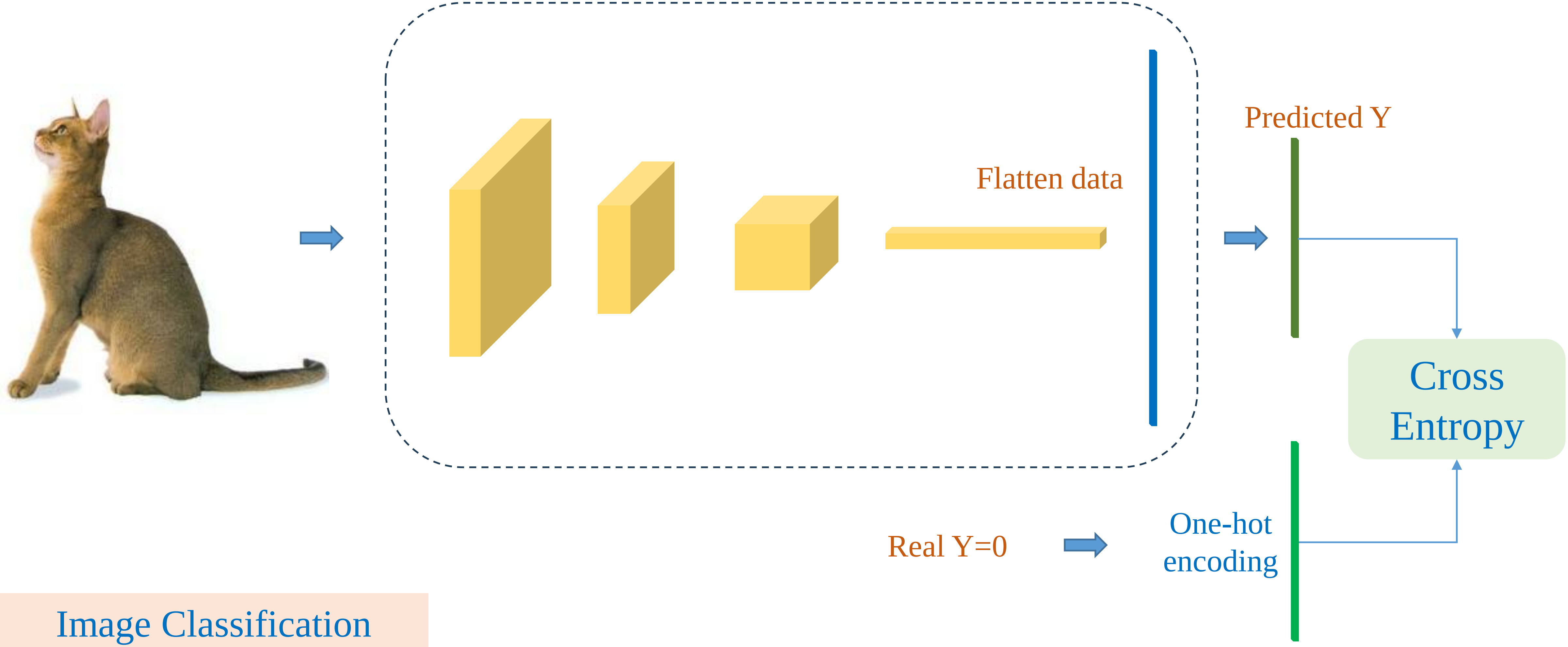


Image Classification

One-Hot Encoding and Cross-entropy

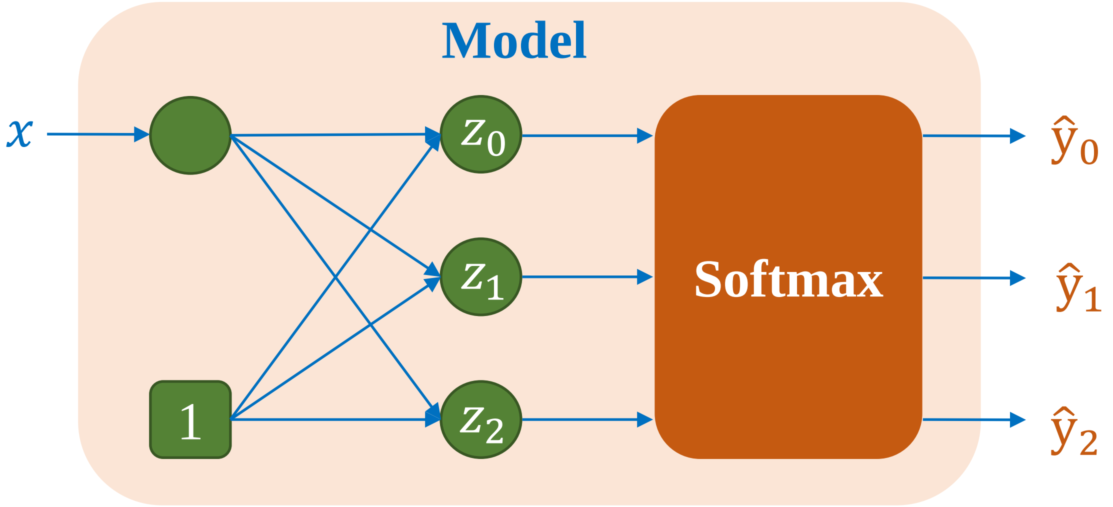
Review

Feature	Label
Petal_Length	Label
1.4	0
1.3	0
1.5	0
4.5	1
4.1	1
4.6	1
5.2	2
5.6	2
5.9	2

#features = 1

#classes = 3

$y \in \{0,1,2\}$



One-hot encoding for label

$$\boldsymbol{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} \qquad y_i \in \{0,1\} \qquad \sum_i y_i = 1$$

$$y = 0 \rightarrow \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad y = 1 \rightarrow \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad y = 2 \rightarrow \boldsymbol{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

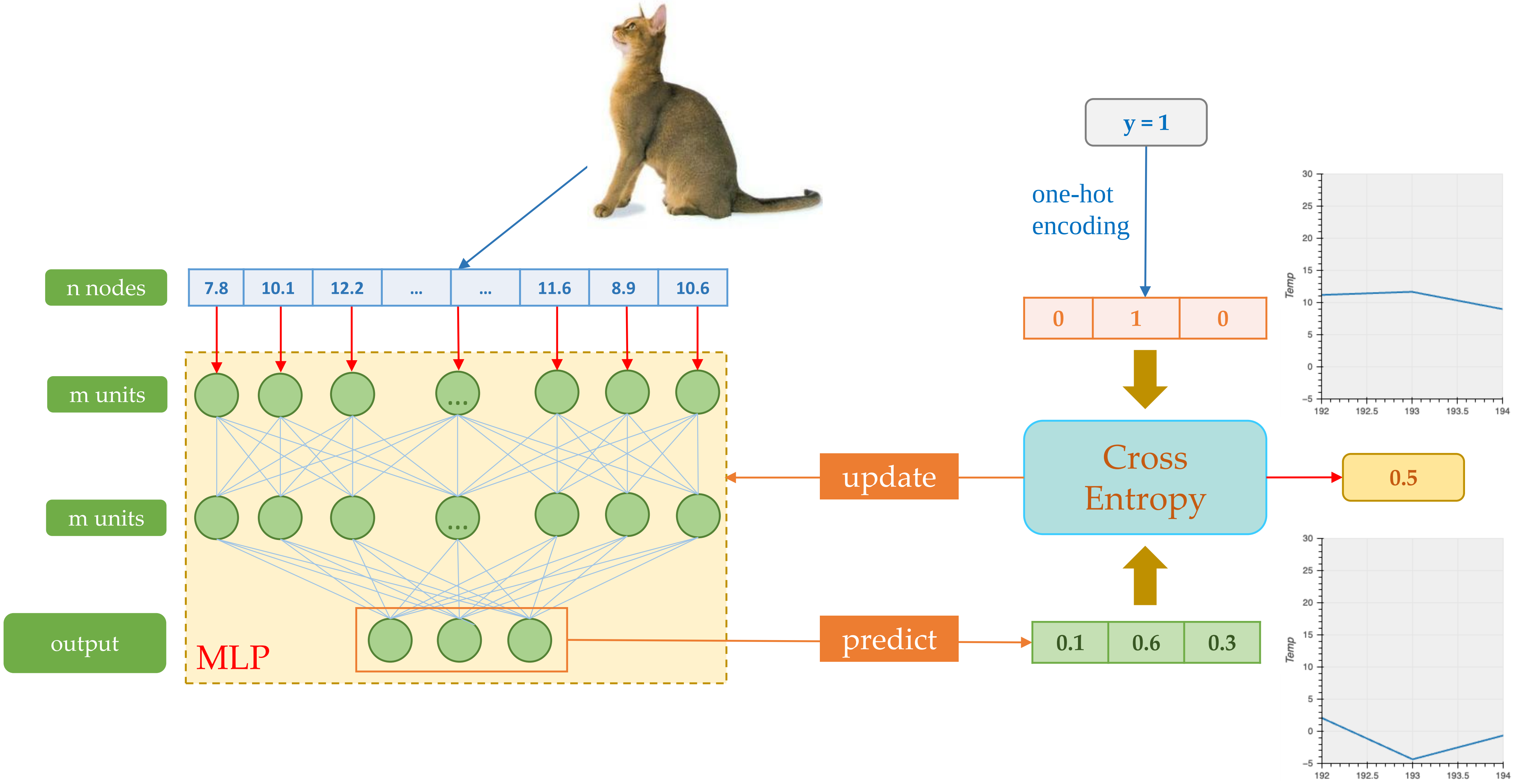
Loss function

$$L(\boldsymbol{\theta}) = -y_2 \log(\hat{y}_2) - y_1 \log(\hat{y}_1) - y_0 \log(\hat{y}_0)$$

$$= -\sum_i y_i \log(\hat{y}_i)$$

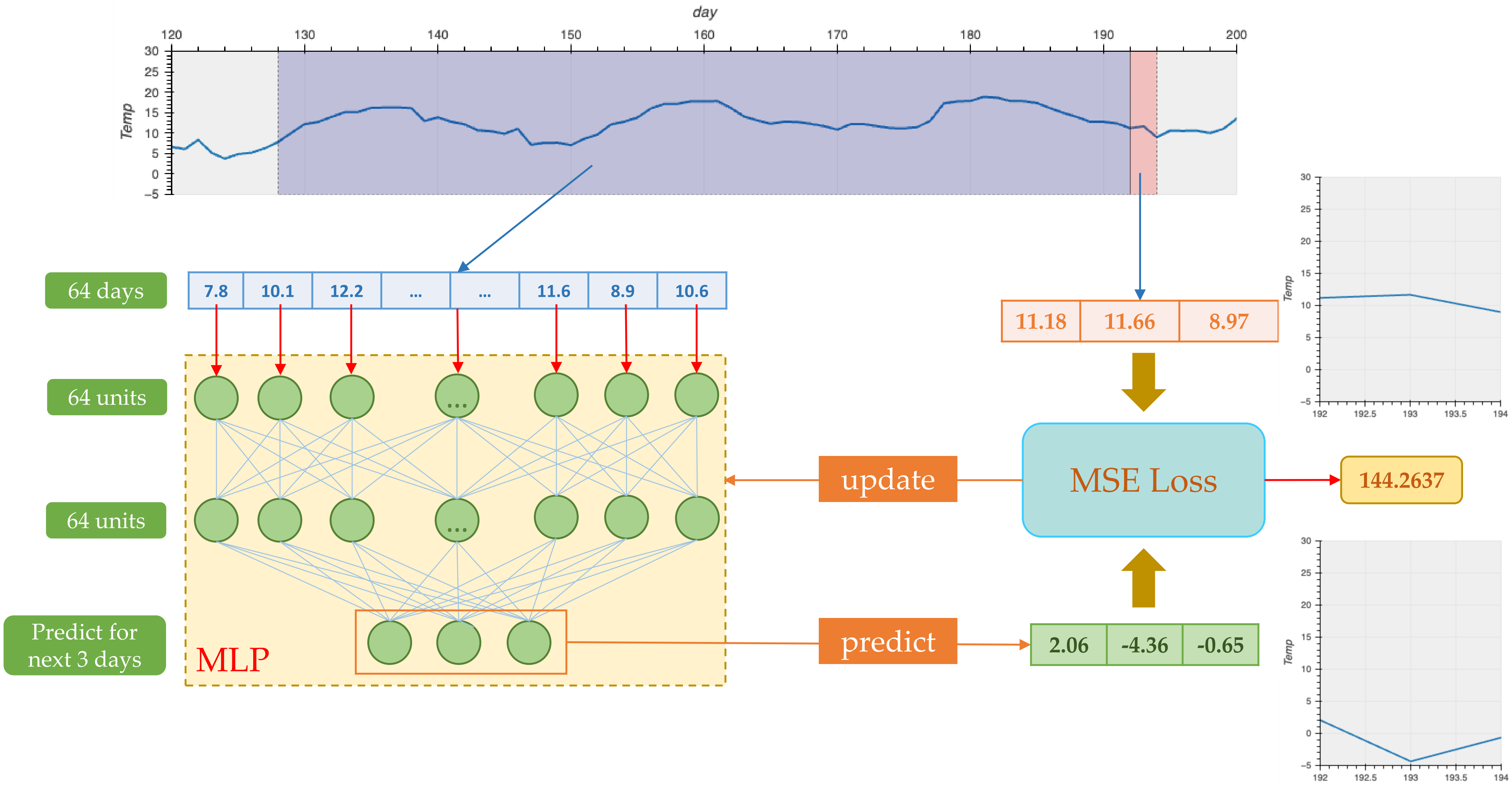
Discussion – Classification Problem

Input/Output + Loss Function



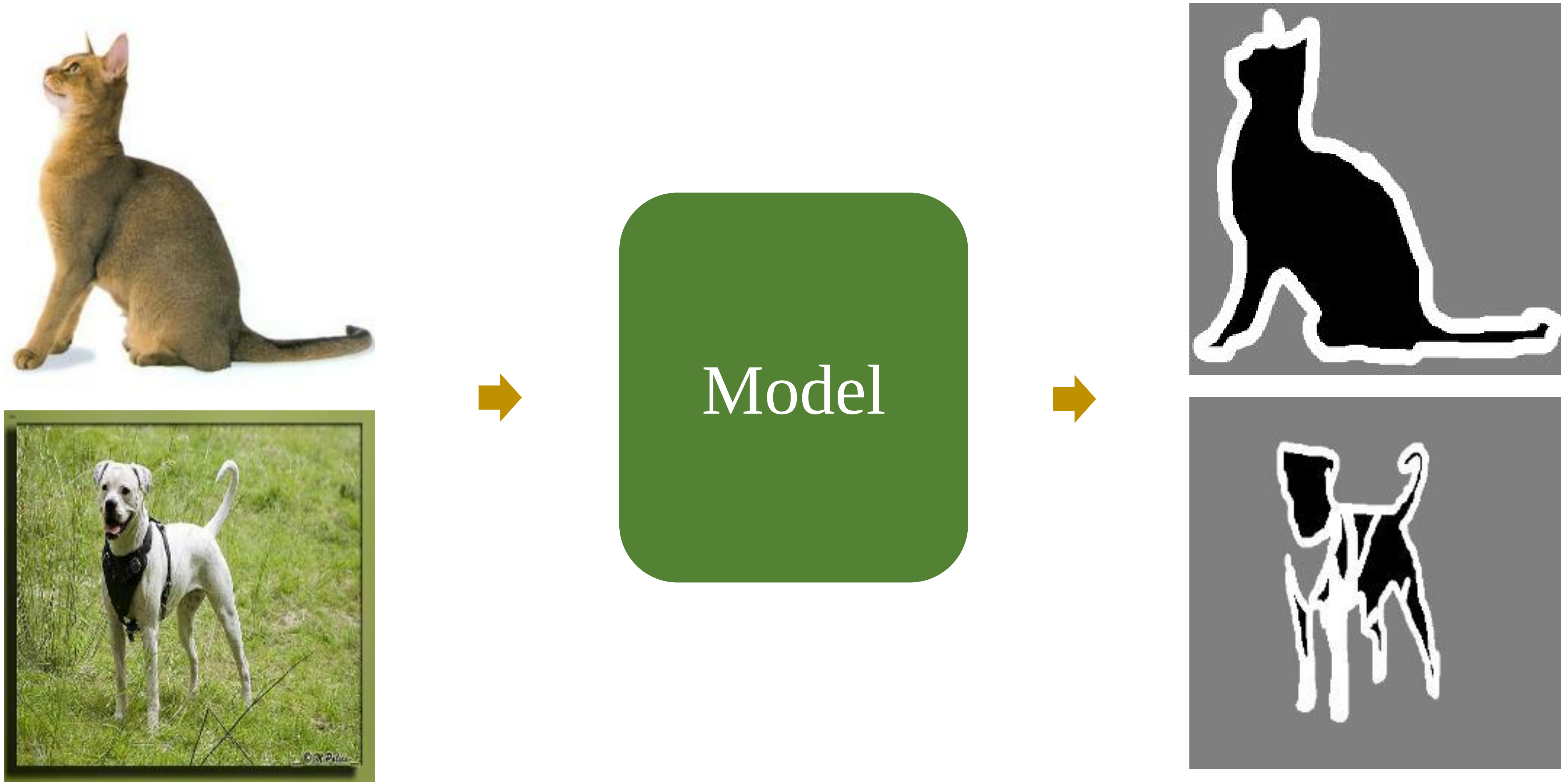
Discussion – Regression Problem

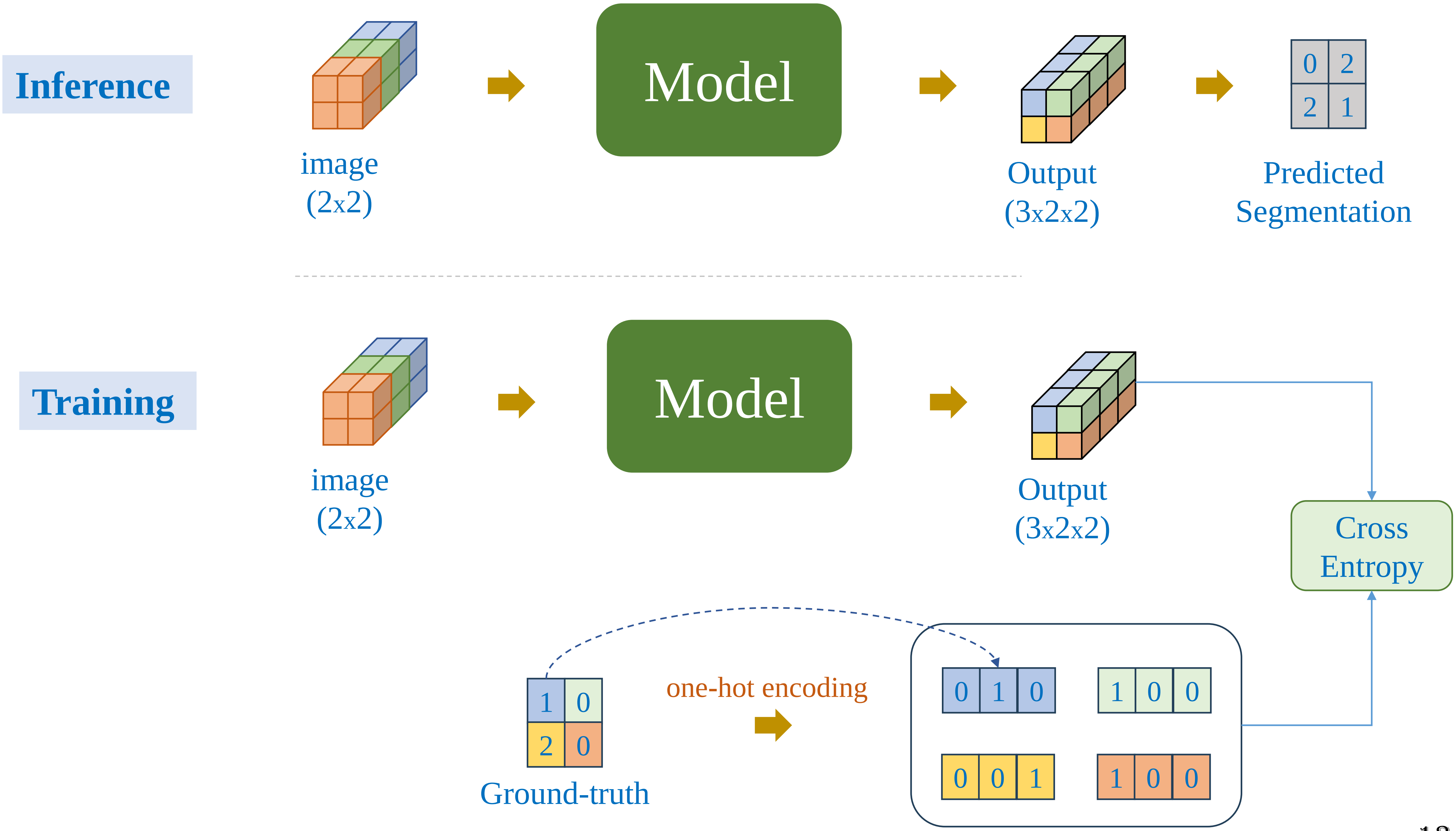
Input/Output + Loss Function



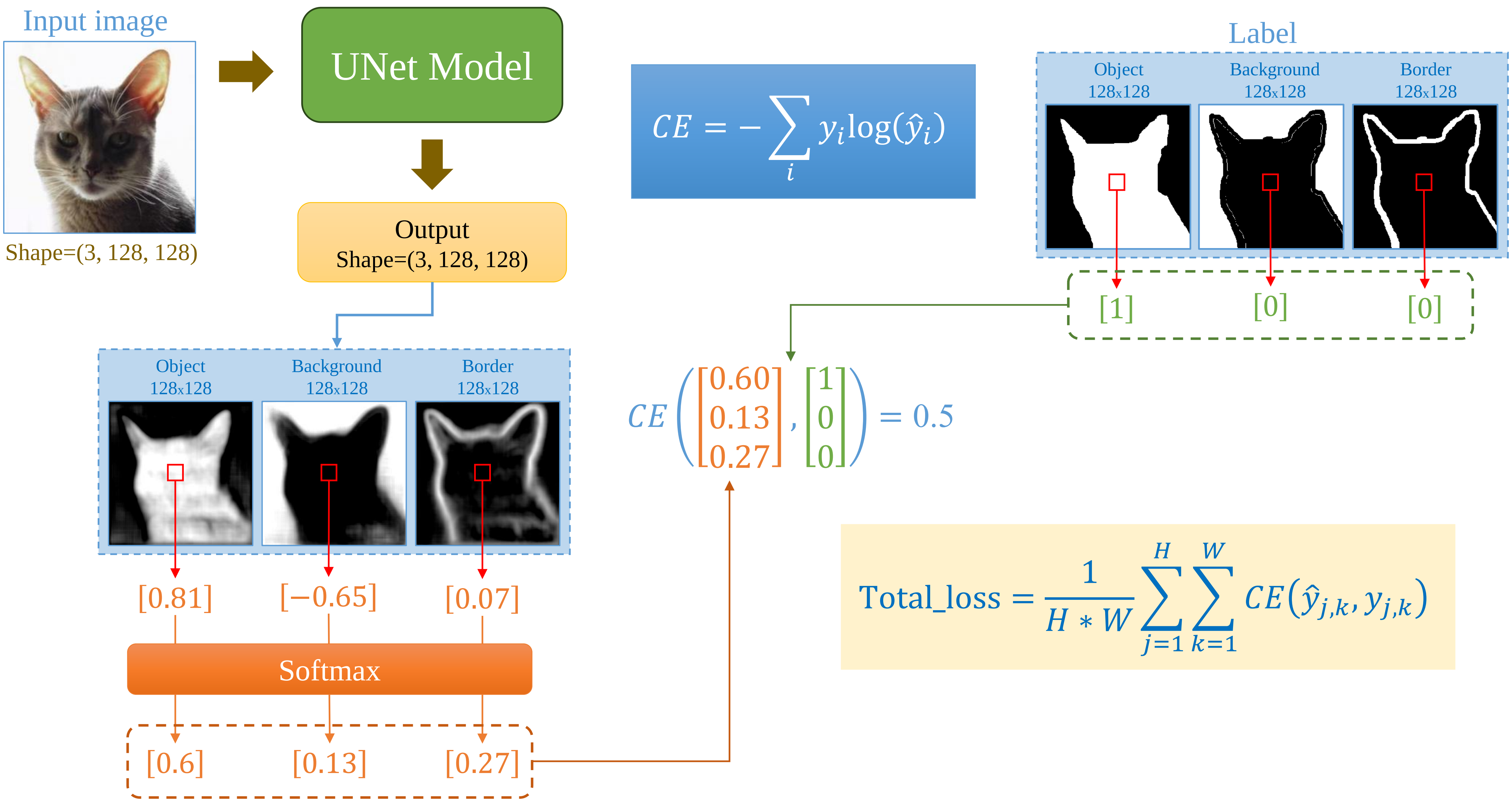
Discussion – Segmentation Problem

Input/Output + Loss Function

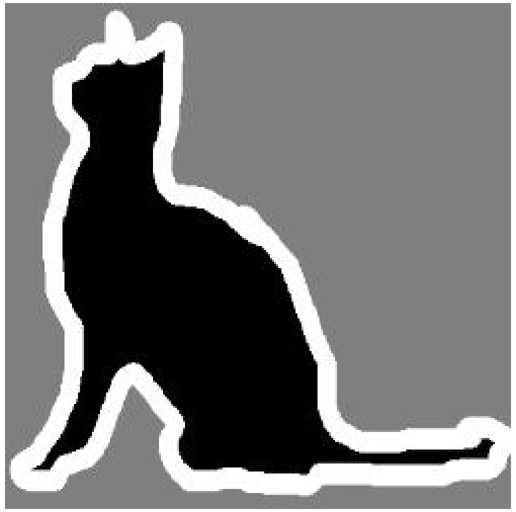



Inference
image
(2x2)
Model
Output
(3x2x2)
0 2
2 1
Predicted
Segmentation
Training
image
(2x2)
Model
Output
(3x2x2)
Cross
Entropy
1 0
2 0
Ground-truth
one-hot encoding
0 1 0
1 0 0
0 0 1
1 0 0

Loss Function



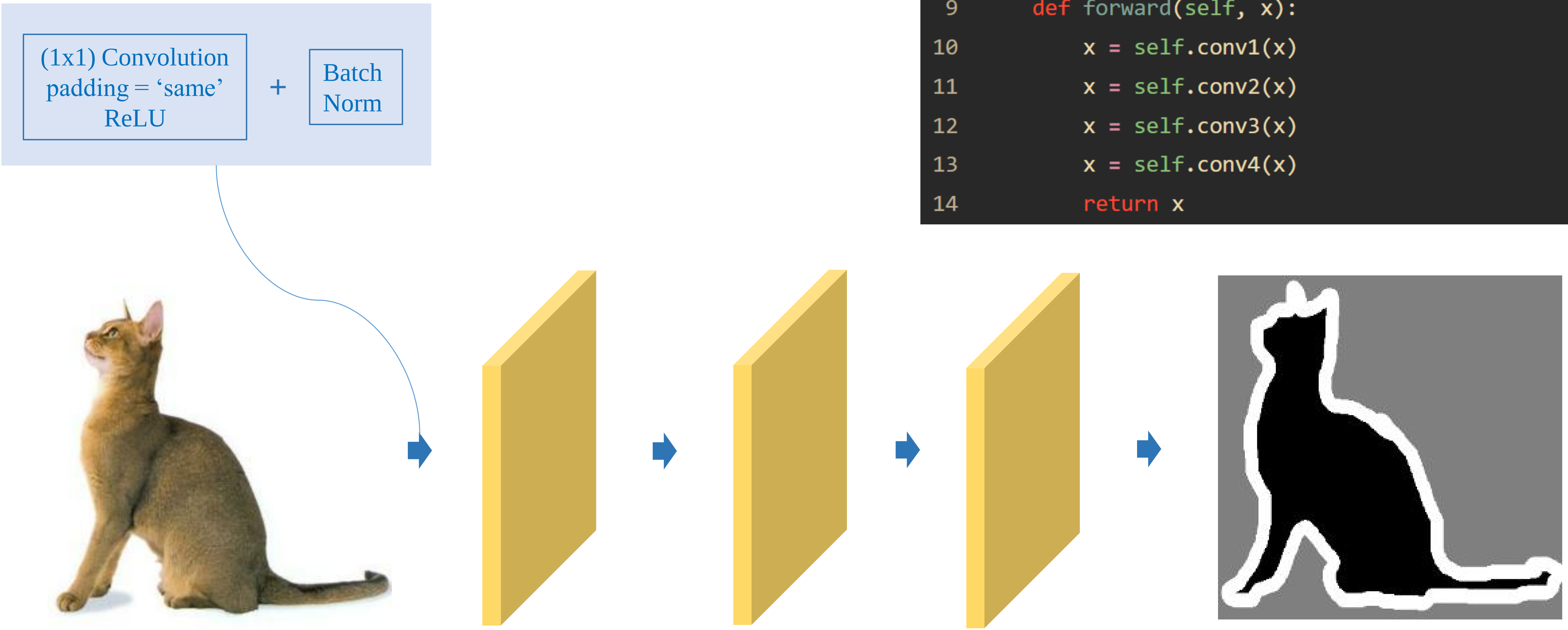
idea



Segmentation (1)

Pixel-wise and color-based method

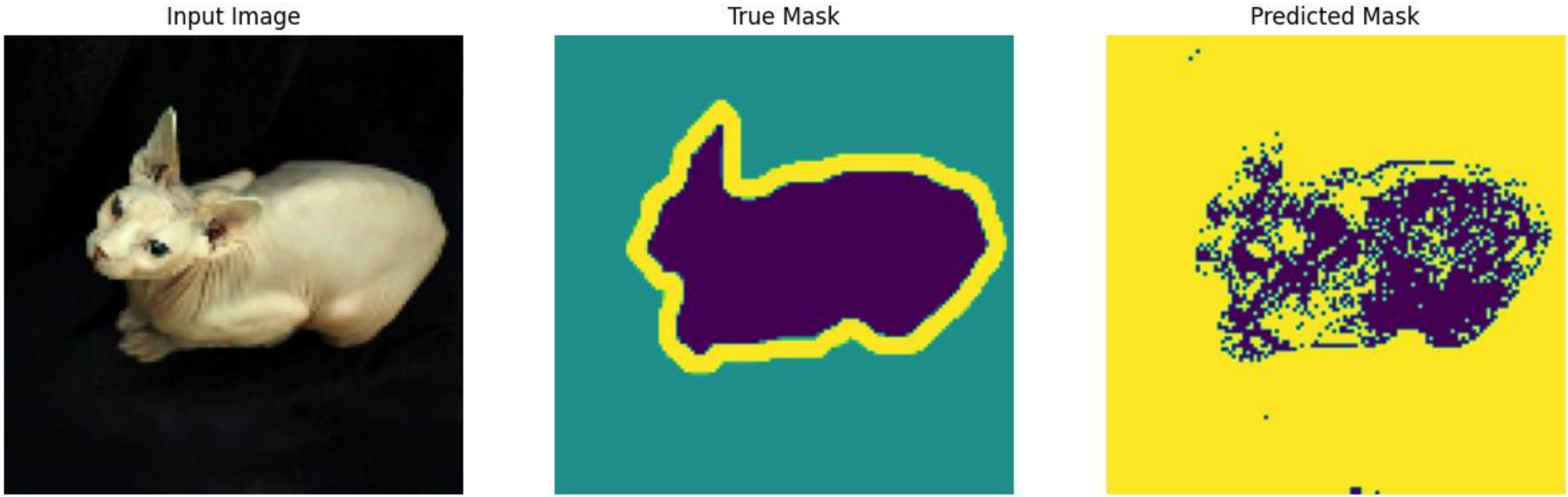
```python
class CNN_Segmentation(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = ConvBlock(3, 64, kernel_size=1)
        self.conv2 = ConvBlock(64, 64, kernel_size=1)
        self.conv3 = ConvBlock(64, 64, kernel_size=1)
        self.conv4 = nn.Conv2d(64, 3, kernel_size=1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        return x
```



Segmentation (1)

Result

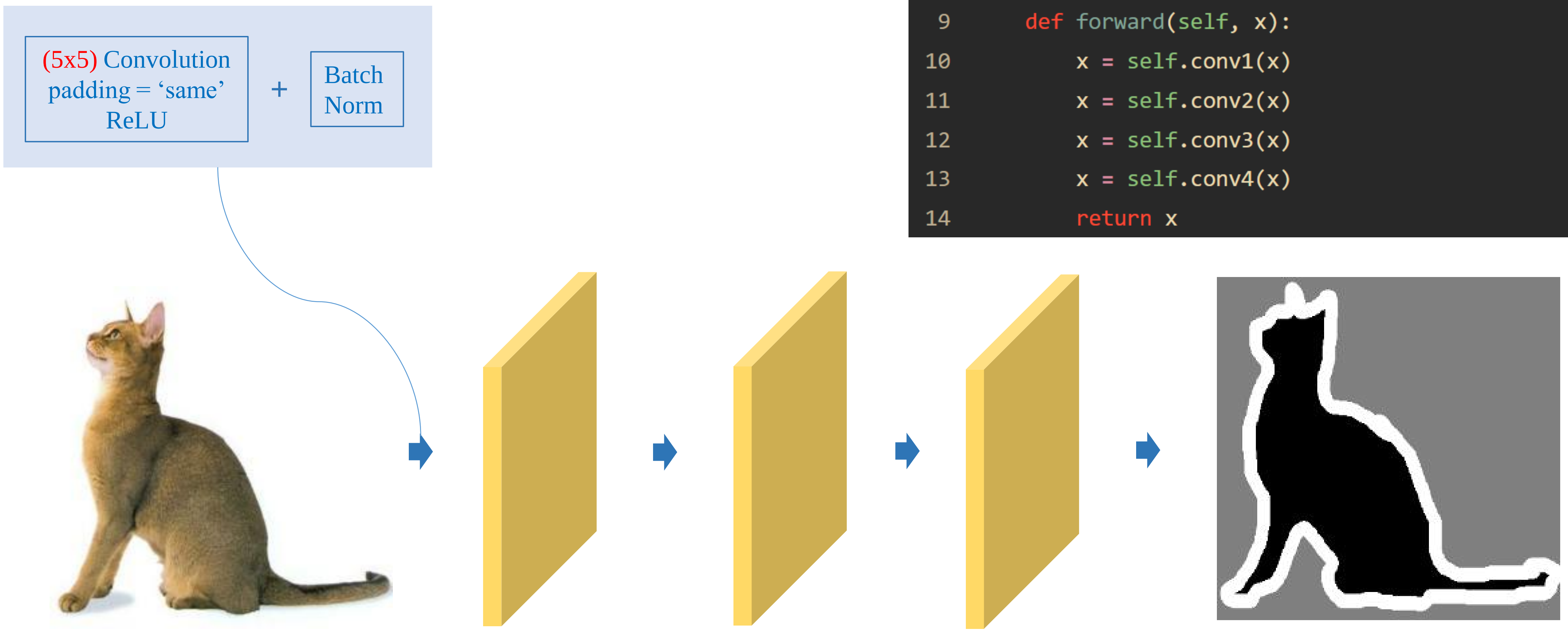
```python
for epoch in range(max_epoch):
    model.train()
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        loss.backward()
        optimizer.step()
```



Segmentation (2)

Increase kernel sizes

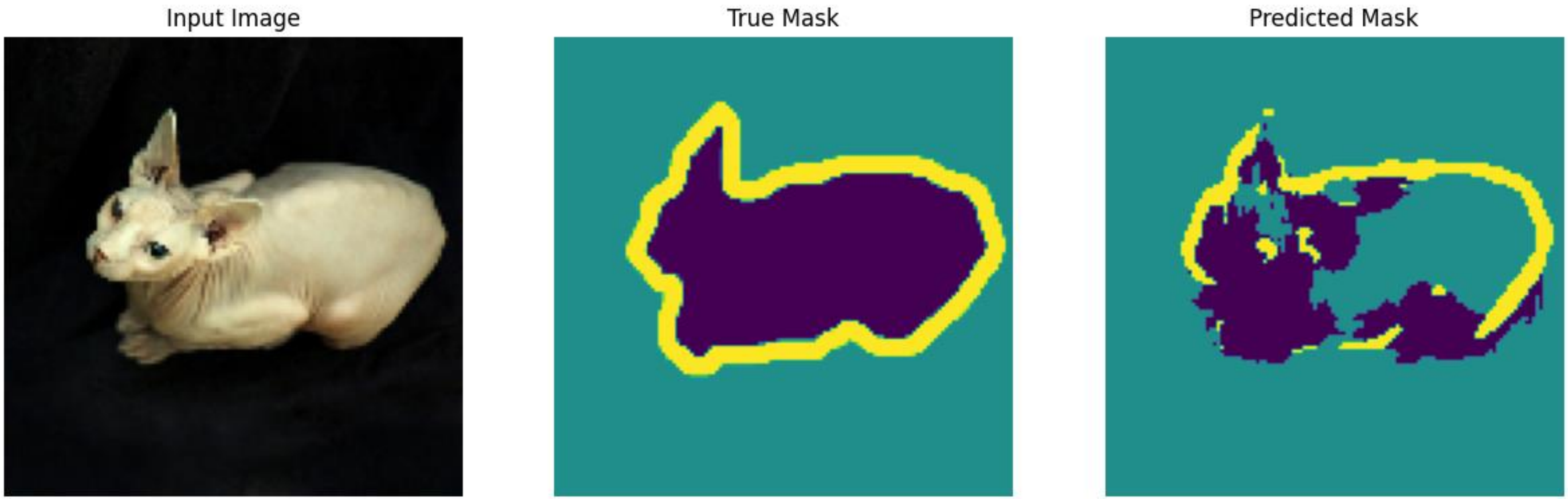
```python
class CNN_Segmentation(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = ConvBlock(3, 64, kernel_size=5)
        self.conv2 = ConvBlock(64, 64, kernel_size=5)
        self.conv3 = ConvBlock(64, 64, kernel_size=5)
        self.conv4 = nn.Conv2d(64, 3, kernel_size=5)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        return x
```



Segmentation (2)

Result

```python
for epoch in range(max_epoch):
    model.train()
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        loss.backward()
        optimizer.step()
```



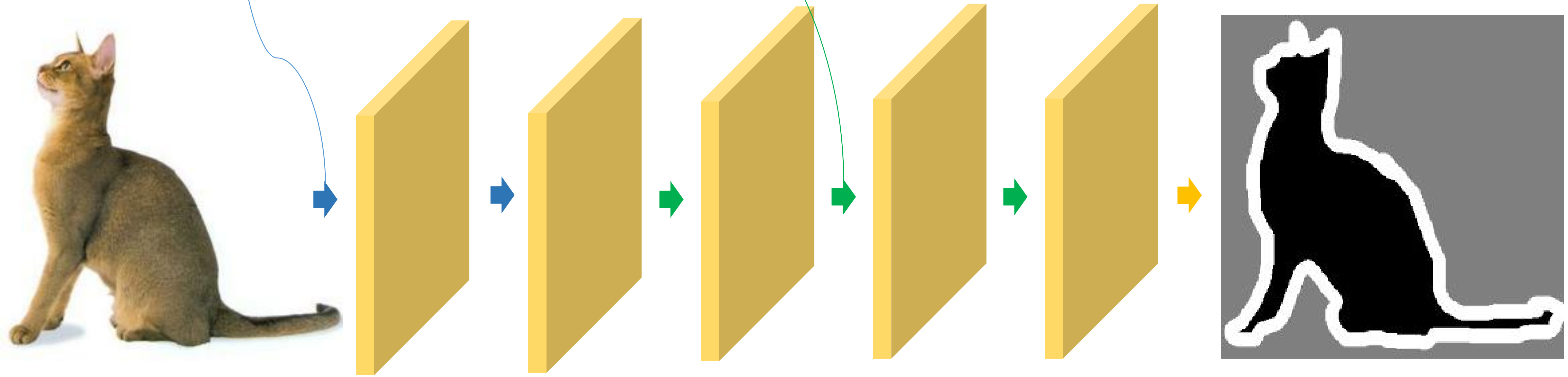
Segmentation (3)

Keep increasing

(9x9) Convolution
padding = 'same'
ReLU
+
Batch Norm

(5x5) Convolution
padding = 'same'
ReLU
+
Batch Norm

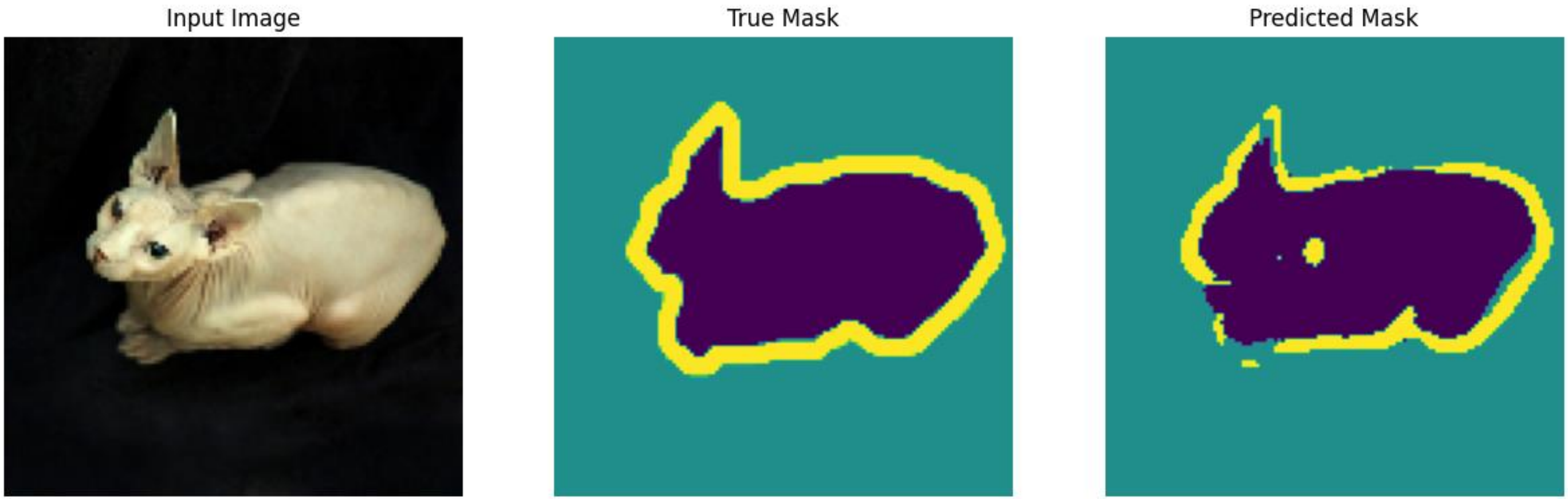
```python
class CNN_Segmentation(nn.Module):
    def __init__(self, n_channels, n_classes):
        super().__init__()
        self.conv1 = ConvBlock(3, 64, kernel_size=9)
        self.conv2 = ConvBlock(64, 64, kernel_size=9)
        self.conv3 = ConvBlock(64, 64, kernel_size=5)
        self.conv4 = ConvBlock(64, 64, kernel_size=5)
        self.conv5 = ConvBlock(64, 64, kernel_size=5)
        self.conv6 = nn.Conv2d(64, 3, kernel_size=3,
                               padding='same')
```



Segmentation (3)

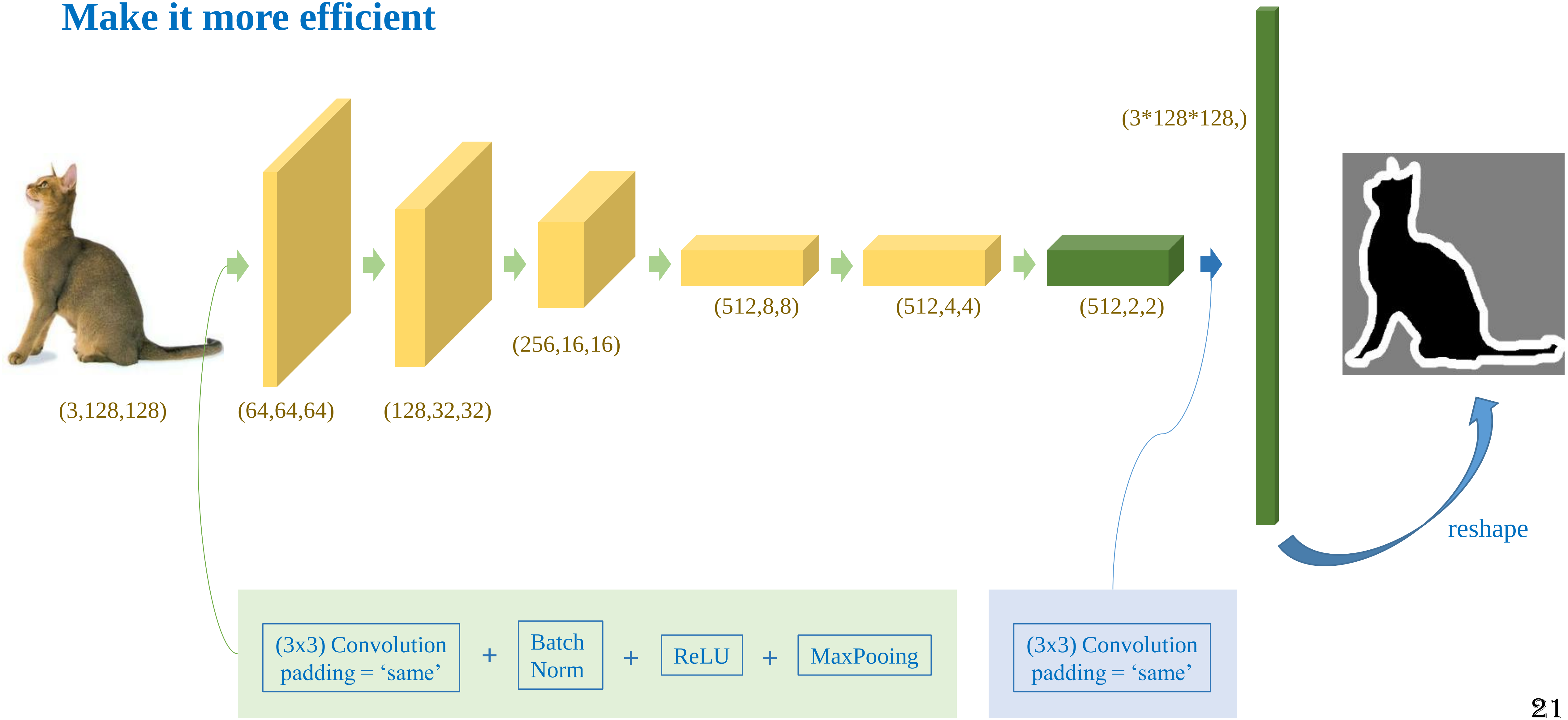
Result

```python
for epoch in range(max_epoch):
    model.train()
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        loss.backward()
        optimizer.step()
```

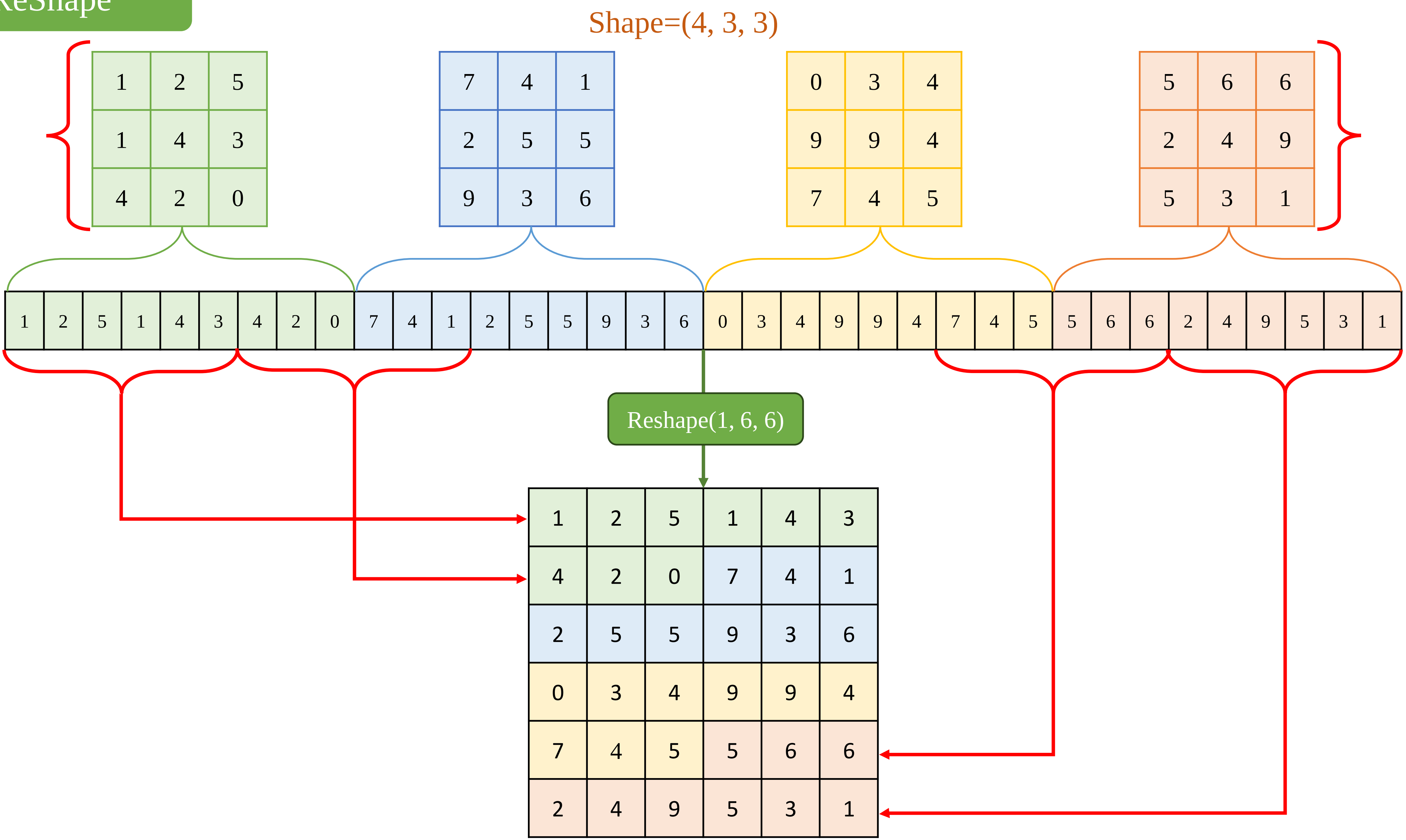


Segmentation (4)

Make it more efficient

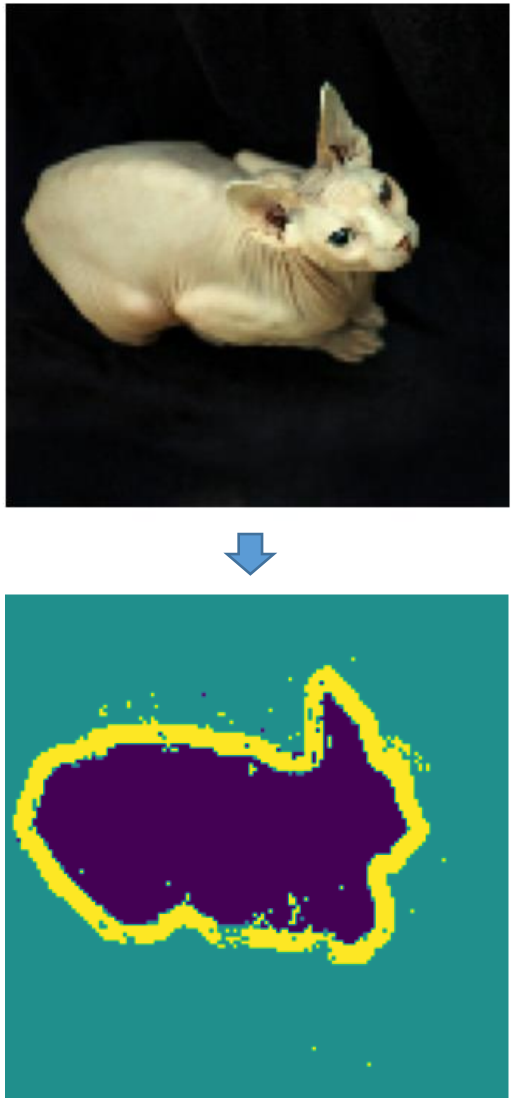


ReShape



Segmentation (4)

Make it more efficient



Problem?

```python
class FCN_Segmentation(nn.Module):
    def __init__(self):
        super().__init__()
        self.enc_1 = Encoder(3, 64)
        self.enc_2 = Encoder(64, 128)
        self.enc_3 = Encoder(128, 256)
        self.enc_4 = Encoder(256, 512)
        self.enc_5 = Encoder(512, 512)
        self.enc_6 = Encoder(512, 512)
        self.out_conv = nn.Conv2d(512, 128*128*3,
                                  kernel_size=2)

    def forward(self, x):
        x = self.enc_1(x)
        x = self.enc_2(x)
        x = self.enc_3(x)
        x = self.enc_4(x)
        x = self.enc_5(x)
        x = self.enc_6(x)
        x = self.out_conv(x)
        return torch.reshape(x, (-1, 3, 128, 128))
```

Further Reading

Fully Convolutional Networks

https://arxiv.org/pdf/1411.4038.pdf

Fully Convolutional Networks for Semantic Segmentation

Jonathan Long* Evan Shelhamer* Trevor Darrell
UC Berkeley
{jonlong,shelhamer,trevor}@cs.berkeley.edu

Abstract

Convolutional networks are powerful visual models that yield hierarchies of features. We show that convolutional networks by themselves, trained end-to-end, pixels-to-pixels, exceed the state-of-the-art in semantic segmentation. Our key insight is to build "fully convolutional" networks that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning. We define and detail the space of fully convolutional networks, explain their application to spatially dense prediction tasks, and draw connections to prior models. We adapt contemporary classification networks (AlexNet [19],

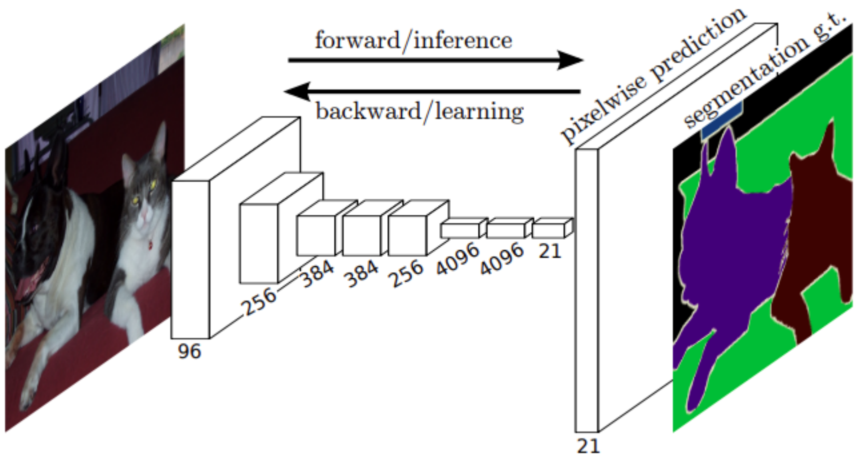
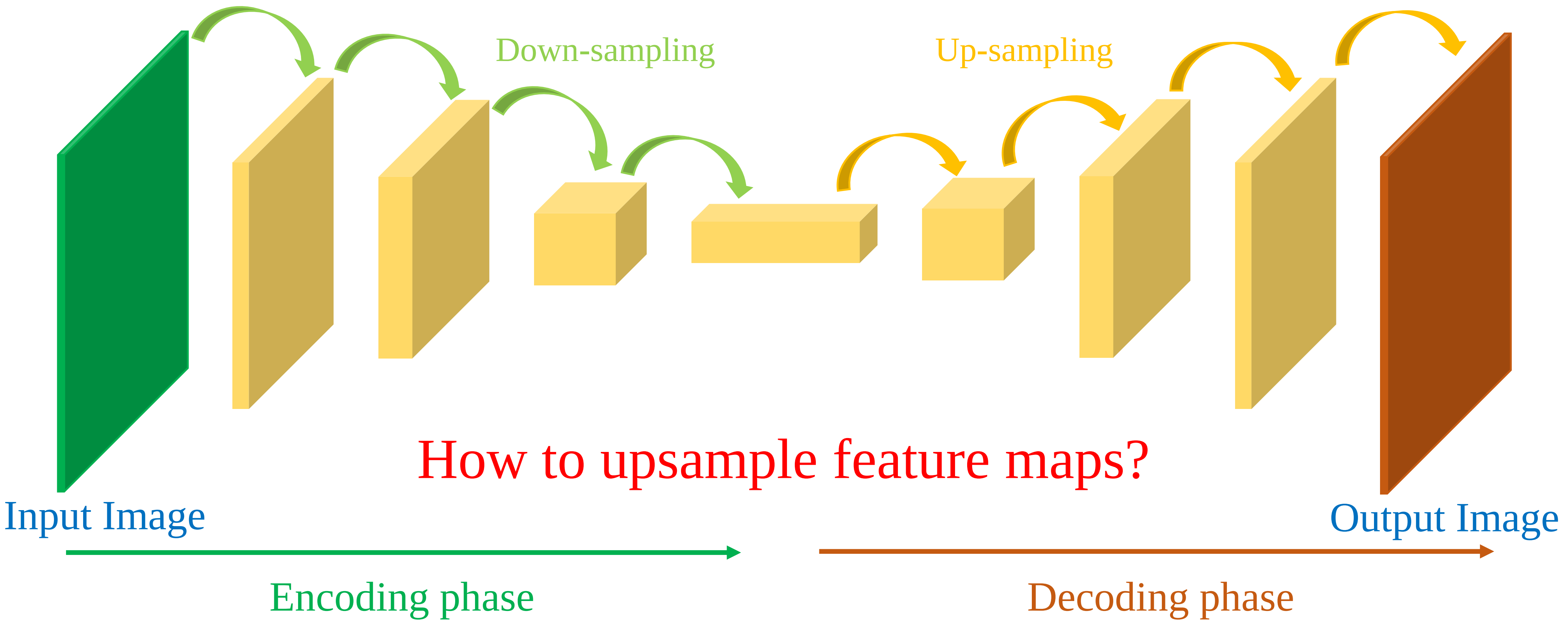


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

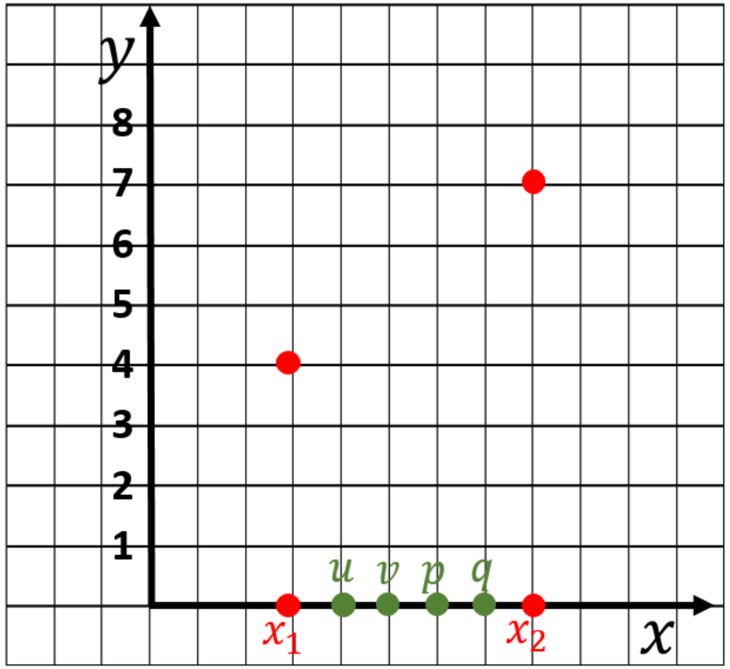
Segmentation (5)



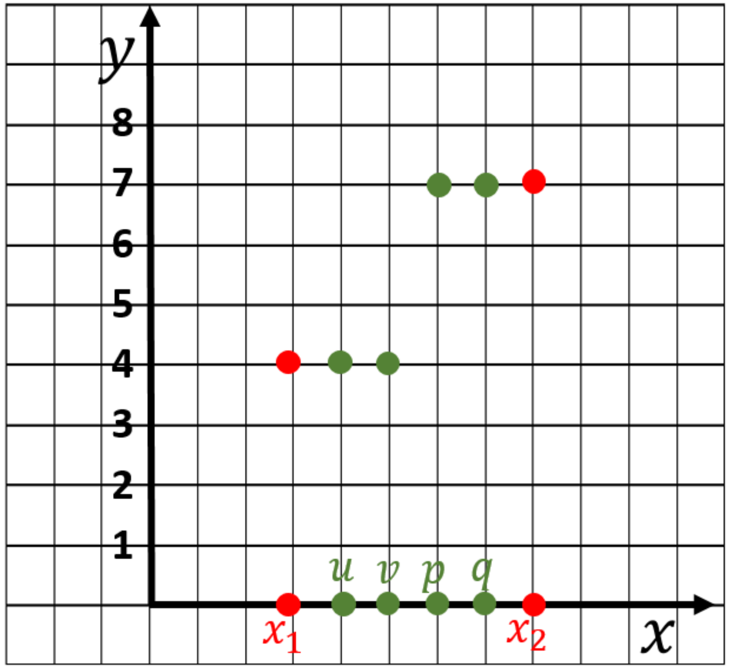
#channels increases	#channels decreases
resolution decreases	resolution increases

Segmentation (5)

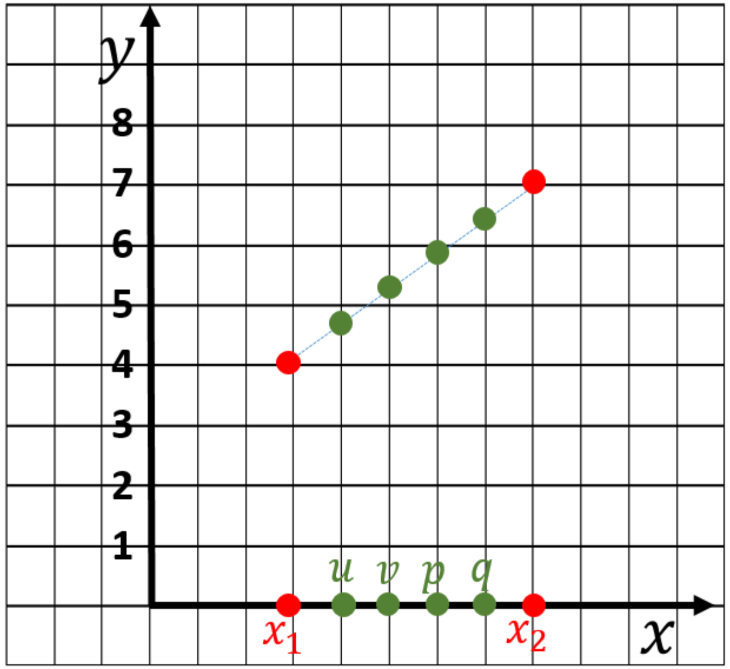
- **Image upsampling**
  - **Data interpolation**



Tìm giá trị cho các vị trí u, v, p và q



Nearest neighbor: Tính khoảng cách đến x_1 và x_2, và lấy giá trị của x gần hơn

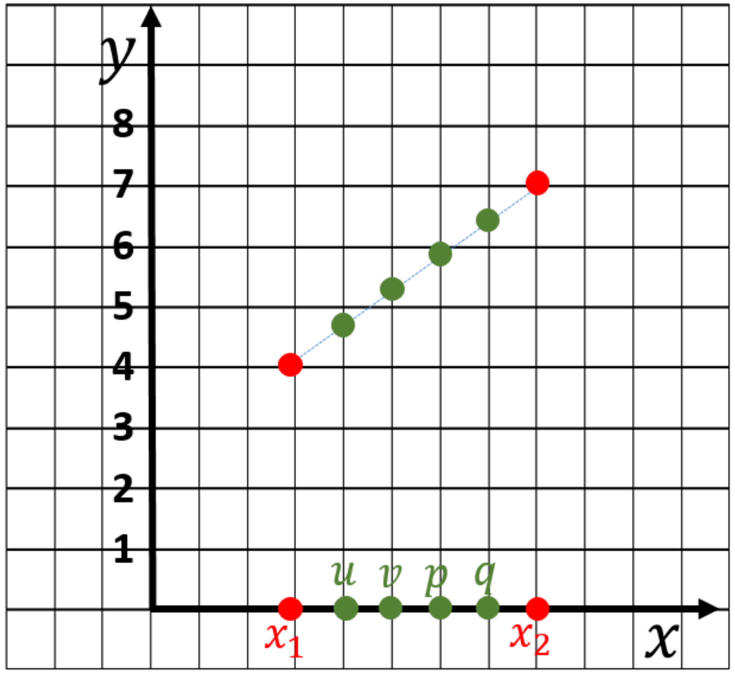


Nội suy theo hàm tuyến tính

Segmentation (5)

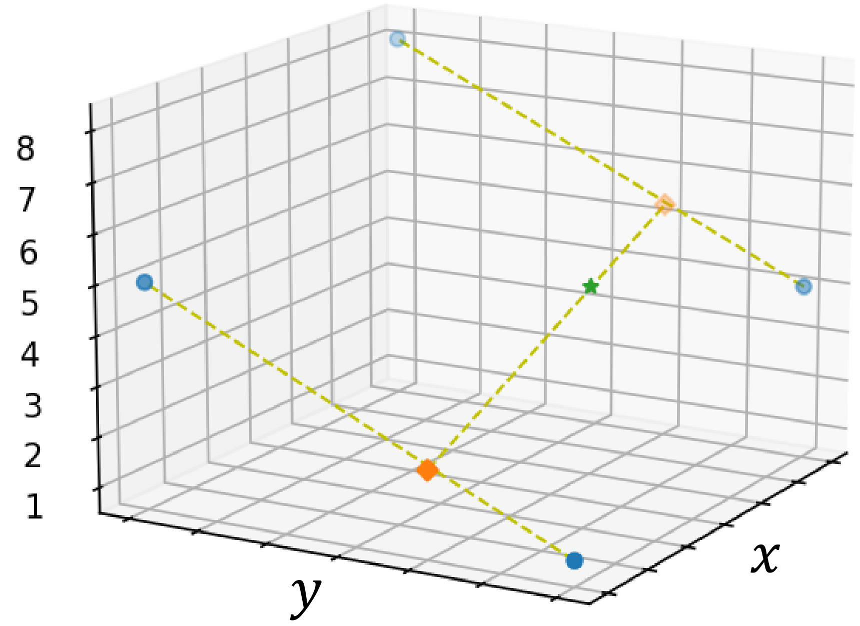
❖ Image upsampling

❖ Data interpolation



Nội suy theo hàm tuyến tính

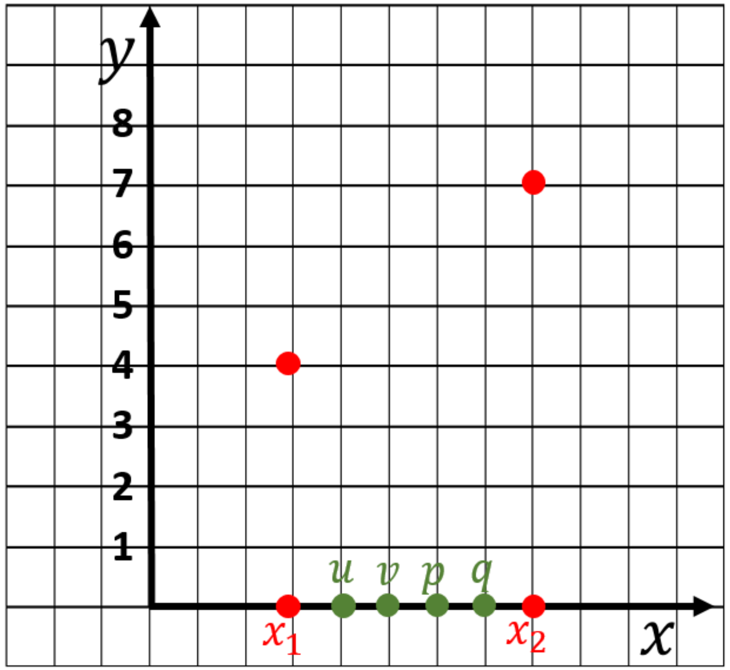
1.0			5.0
3.2	4.2		7.2
4.0			8.0



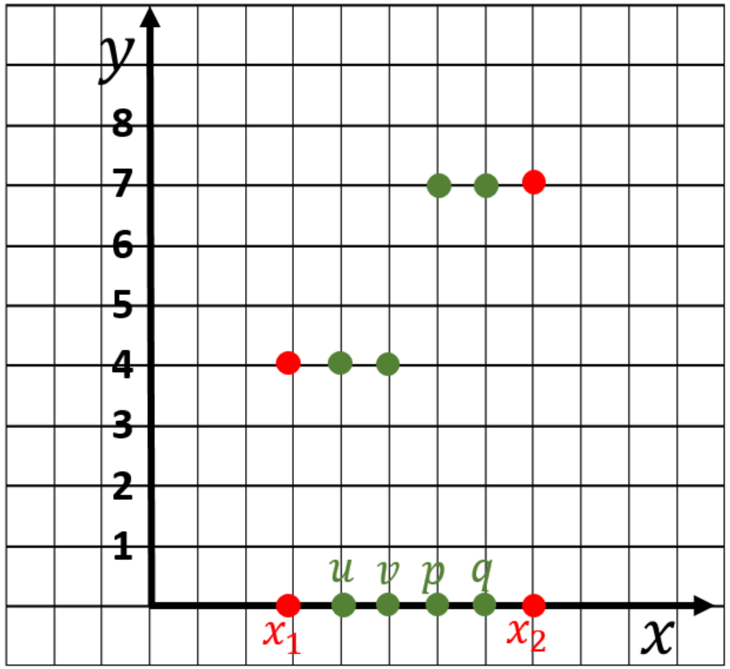
Bilinear interpolation

Segmentation (5)

❖ Nearest neighbor interpolation



Tìm giá trị cho các vị trí u, v, p và q



Nearest neighbor: Tính khoảng cách đến x_1 và x_2, và lấy giá trị của x gần hơn

data =

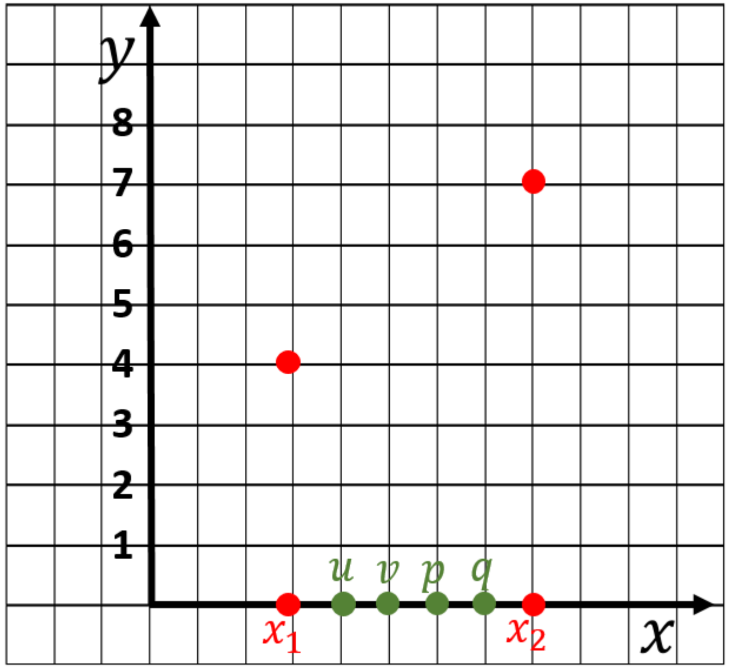
1	5
4	8

1	1	5	5
1	1	5	5
4	4	8	8
4	4	8	8

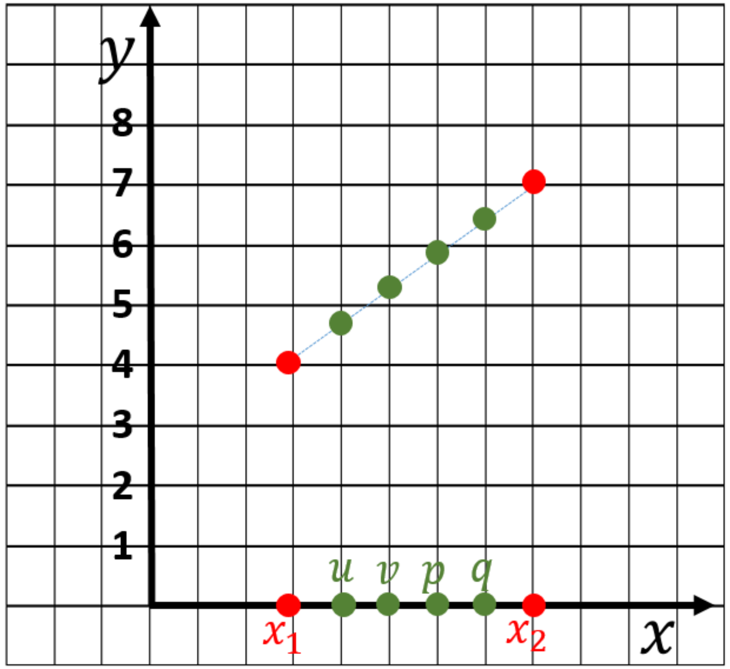
Nearest neighbor interpolation

Segmentation (5)

❖ Bilinear interpolation



Tìm giá trị cho các vị trí u, v, p và q



Nội suy theo hàm tuyến tính

data =

1	5
4	8

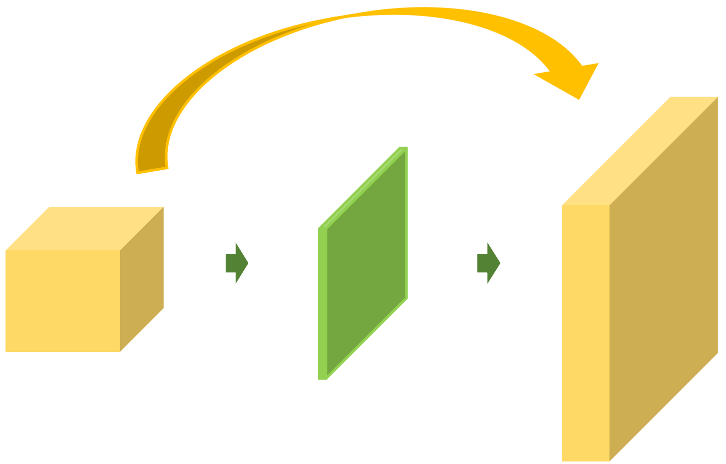
1.0	2.0	4.0	5.0
1.7	2.7	4.7	5.7
3.2	4.2	6.2	7.2
4.0	5.0	7.0	8.0

Bilinear interpolation

How to Upsample Feature Maps

❖ Interpolation

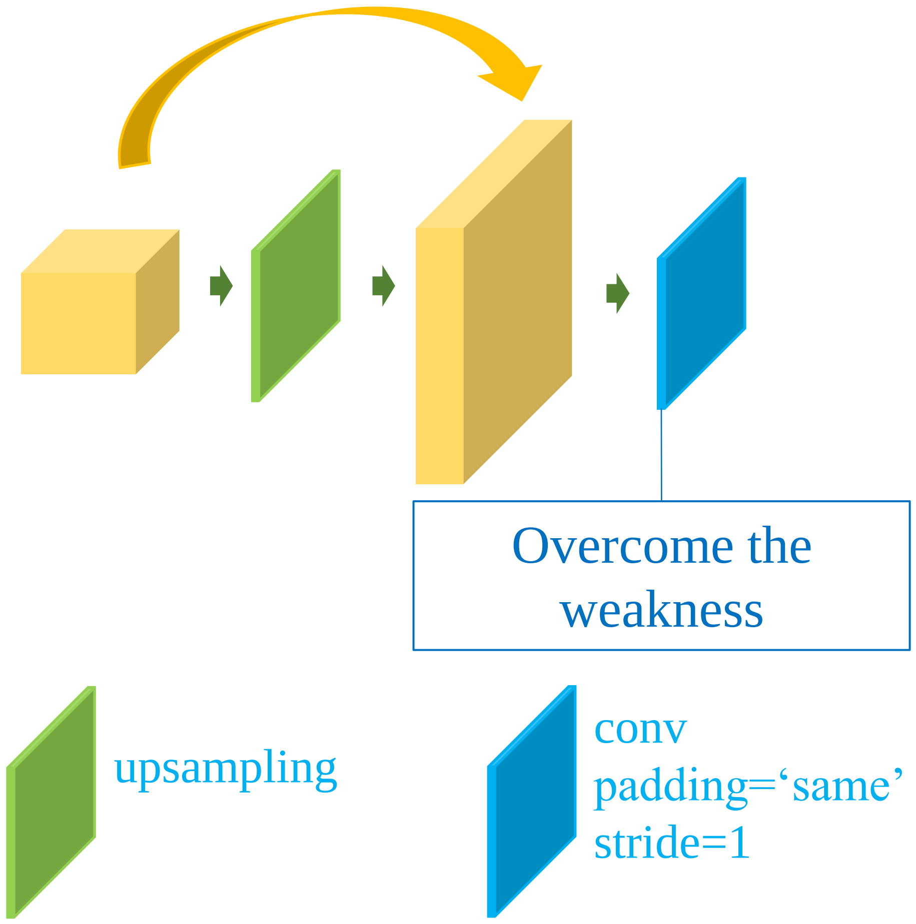
Naïve approach: Only use 'image upsampling'



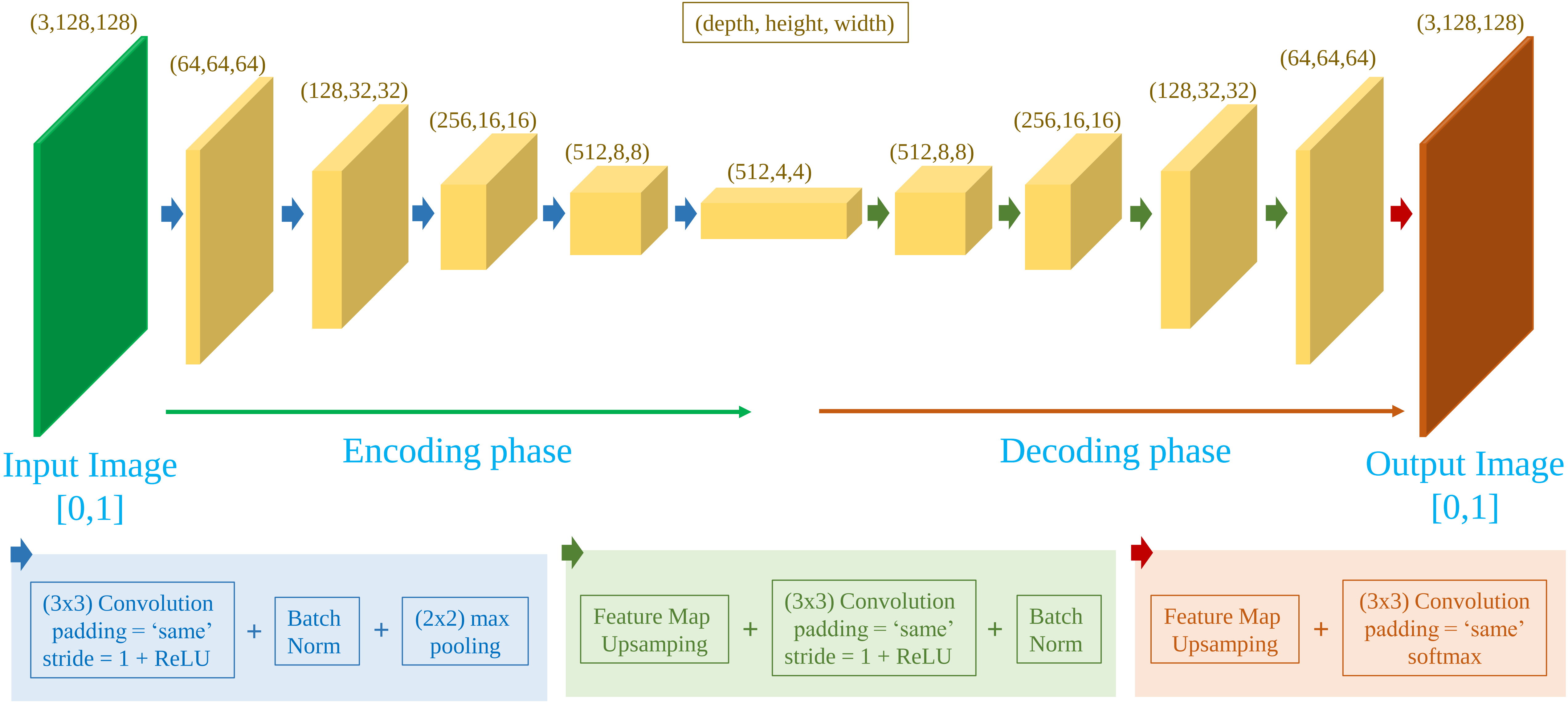
Output feature maps are lack of details

```
nn.Upsample(scale_factor=2,
            mode='bilinear', align_corners=True)
nn.Conv2d(in_channels, out_channels,
          kernel_size=3, padding=1)
```

Use 'image upsampling'+Conv



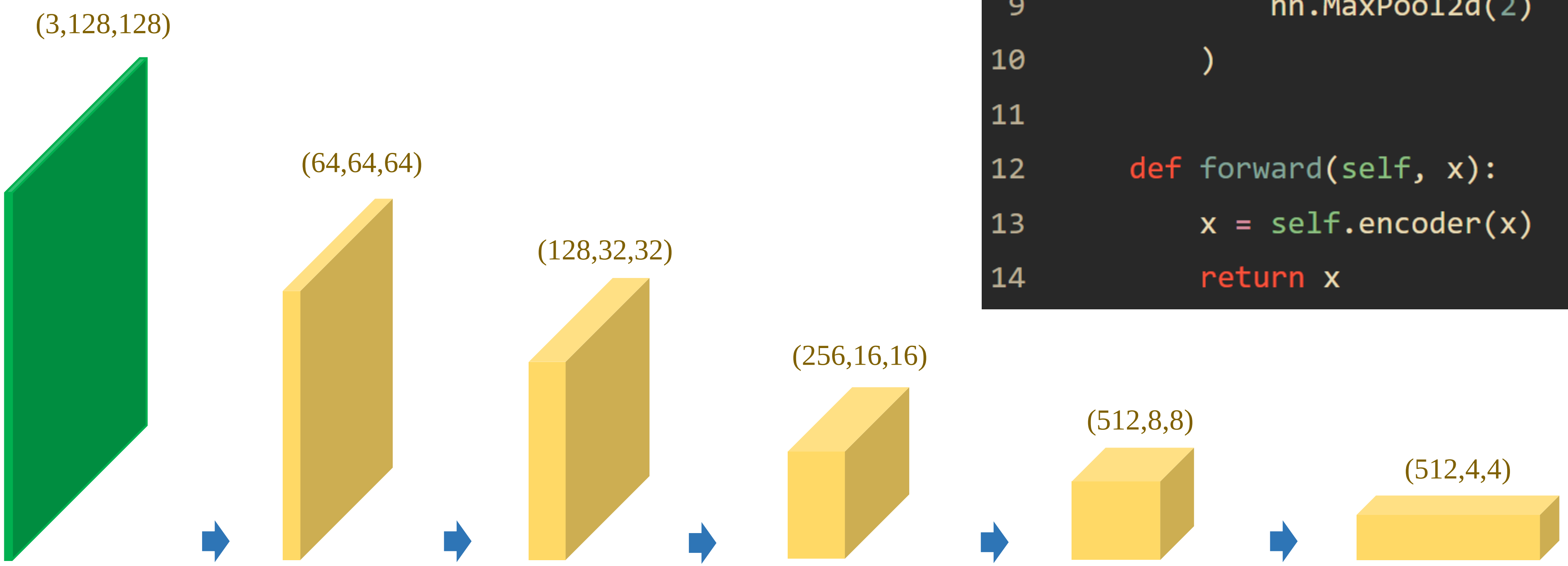
Segmentation (5)



Segmentation (5)

❖ **Encoding**

```python
class Encoder(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(in_channels, out_channels,
                      kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2)
        )

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



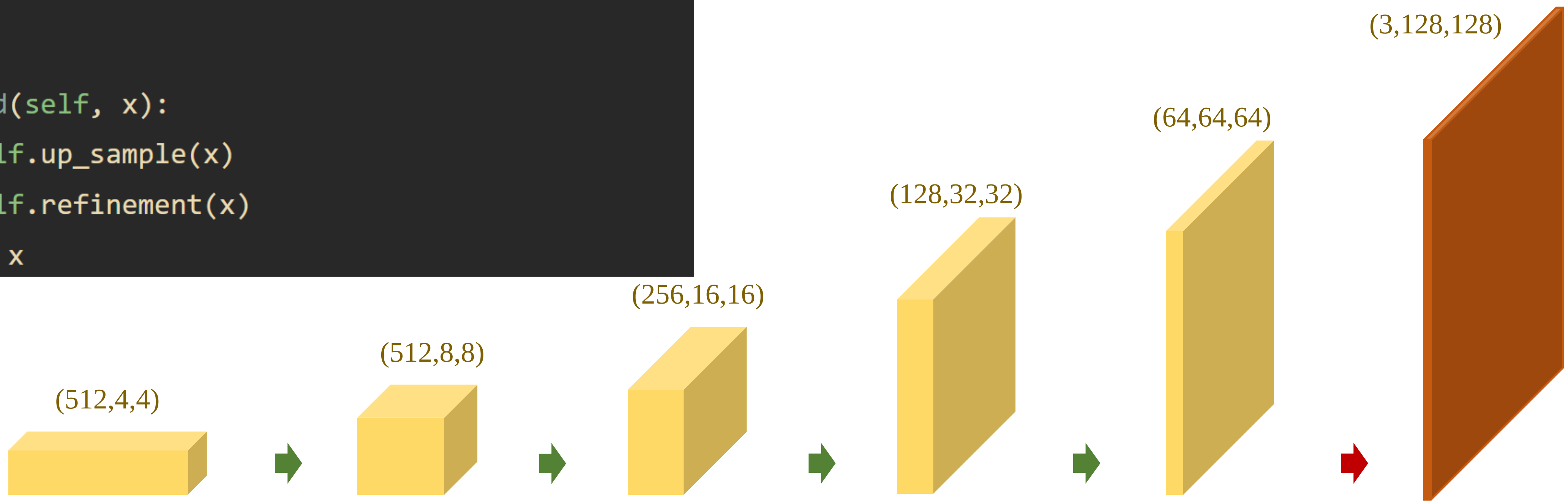
```python
class Decoder(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.up_sample = nn.Upsample(scale_factor=2,
                                     mode='bilinear',
                                     align_corners=True)
        self.refinement = nn.Sequential(
            nn.Conv2d(in_channels, out_channels,
                      kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

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

Segmentation (5)

❖ Decoding

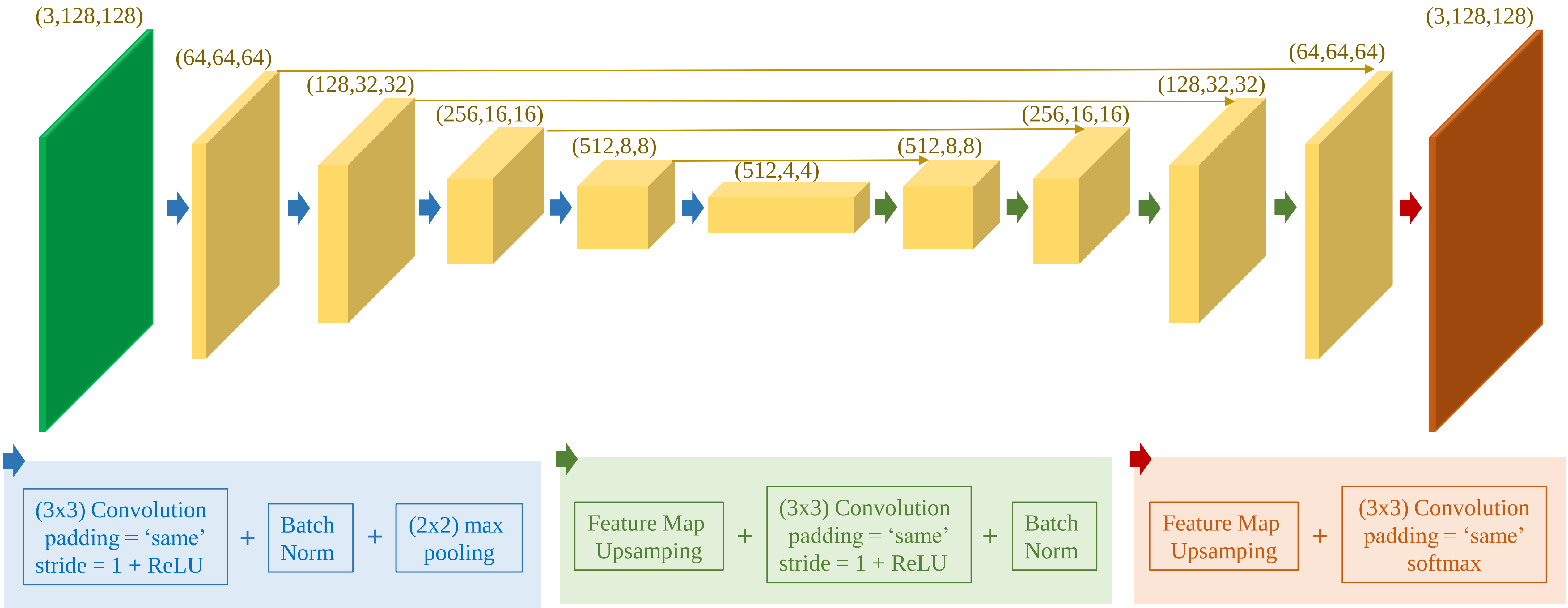
ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks, 2018

Perception-Oriented Single Image Super-Resolution using Optimal Objective Estimation, CVPR 2023



Segmentation (6)

Using skip connections



Segmentation (6)

With Skip Connections

Without Skip Connections