AI VIETNAM
AI Insight Course

Template Matching

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2020

Hệ số tương quan (correlation coefficient)

Công thức: Gọi x,y là hai biến ngẫu nhiên

$$\rho_{xy} = \frac{E[(x-\mu_x)[(y-\mu_y)]}{\sqrt{var(x)}\sqrt{var(y)}}$$

$$= \frac{n(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i)}{\sqrt{n\sum_i x_i^2 - (\sum_i x_i)^2}\sqrt{n\sum_i y_i^2 - (\sum_i y_i)^2}}$$

Tính chất 1

$$-1 \leq \rho_{xy} \leq 1$$

Tương quan nghịch ← | → **Tương quan thuận**

Tính chất 2

$$\rho_{xy} = \rho_{uv}$$

trong đó

$$u = ax + b$$

$$v = cy + d$$

Ví dụ 1

x = [7, 18, 29, 2, 10, 9, 9]

y = [1, 6, 12, 8, 6, 21, 10]

$$\rho_{xy} = \frac{E[(x-\mu_x)[(y-\mu_y)]}{\sqrt{var(x)}\sqrt{var(y)}}$$

$$= \frac{n*818 - 84*64}{\sqrt{n*1480 - 7056}\sqrt{n*822 - 4096}} = 0.149$$

Ví dụ 2

u=2*x-14 = [0, 22, 44, -10, 6, 4, 4]

v=y+2 = [3, 8, 14, 10, 8, 23, 12]

$$\rho_{uv} = \frac{E[(u-\mu_u)[(v-\mu_v)]}{\sqrt{var(u)}\sqrt{var(v)}}$$

$$= \frac{n*880 - 70*78}{\sqrt{n*2588 - 4900}\sqrt{n*1106 - 6084}} = 0.149$$

- Correlation coefficient (pearson r)

$$\rho_{XY} = \frac{\overbrace{Cov(X,Y)}^{\text{covariance}}}{\sigma_X \sigma_Y} \quad \sigma: \text{standard deviation}$$

$$= \frac{E[(X - E(X))(Y - E(Y))]}{\sigma_X \sigma_Y} \quad (1) \qquad E: \text{Expectation}$$

*) prove: with $\begin{cases} X' = aX + b \\ Y' = cY + d \end{cases}$

$$\rho_{XY} = \rho_{X'Y'}$$

- $E(X') = E(aX+b) = \frac{1}{n}\sum_i (aX_i + b)$

$$= \frac{1}{n}\left[a\sum_i X_i + nb\right]$$

$$= a\,\frac{1}{n}\sum_i X_i + b = aE(X) + b \quad (2)$$

- $Var(X') = Var(aX+b) = \frac{1}{n}\sum_i [(aX_i + b) - E(X')]^2$

$$= \frac{1}{n}\sum_i [aX_i + b - \underbrace{(aE(X) + b)}_{\text{from (2)}}]^2$$

$$= \frac{1}{n}\sum_i (aX_i - aE(X))^2$$

$$= a^2 \cdot \frac{1}{n}\sum_i (X_i - E(X))^2 = a^2 Var(X) \quad (3)$$

(1) $\Rightarrow \rho_{X'Y'} = \dfrac{E((X' - E(X'))(Y' - E(Y')))}{\sigma_{X'}\,\sigma_{Y'}}$ $\quad \sigma_X = \sqrt{Var(X)}$

$$= \frac{E\{[(aX+b) - (aE(X)+b)][(cY+d) - (cE(Y)+d)]\}}{\sqrt{a^2 Var(X)} \cdot \sqrt{c^2 Var(Y)}}$$

$$= \frac{E\{(aX - aE(X))(cY - cE(Y))\}}{\sqrt{a^2 Var(X)} \cdot \sqrt{c^2 Var(Y)}}$$

$$= \frac{E[ac(X - E(X))(Y - E(Y))]}{ac\sqrt{Var(X)} \cdot \sqrt{Var(Y)}}$$

suppose $a, c > 0$

$$= \frac{ac\,E[(X - E(X))(Y - E(Y))]}{ac\sqrt{Var(X)} \cdot \sqrt{Var(Y)}}$$

$$= \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

$$\Rightarrow \rho_{X'Y'} = \rho_{XY}$$

Correlation Coefficient

```
def find_corr_x_y(x,y):                                          #1
    n = len(x)                                                   #2
    prod = []
    for xi,yi in zip(x,y):                                       #3
        prod.append(xi*yi)

    sum_prod_x_y = sum(prod)                                     #4

    sum_x = sum(x)
    sum_y = sum(y)

    squared_sum_x = sum_x**2
    squared_sum_y = sum_y**2

    x_square = []
    for xi in x:
        x_square.append(xi**2)
    x_square_sum = sum(x_square)

    y_square=[]
    for yi in y:
        y_square.append(yi**2)
    y_square_sum = sum(y_square)

    # Use formula to calculate correlation                       #5
    numerator = n*sum_prod_x_y - sum_x*sum_y
    denominator_term1 = n*x_square_sum - squared_sum_x
    denominator_term2 = n*y_square_sum - squared_sum_y
    denominator = (denominator_term1*denominator_term2)**0.5
    correlation = numerator/denominator

    return correlation
```

Ứng dụng cho patch matching

P_1 P_2 P_3 P_4

$\rho_{P_1P_2} = 0.55$

$\rho_{P_1P_3} = 0.23$ ➧ Ảnh P_2 giống với ảnh P_1 hơn so với P_3 và P_4

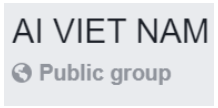
$\rho_{P_1P_4} = 0.30$

P_1 $P_2 = P_1+50$ $P_3 = 1.2P_1+10$

$\rho_{P_1P_2} = 0.9970$

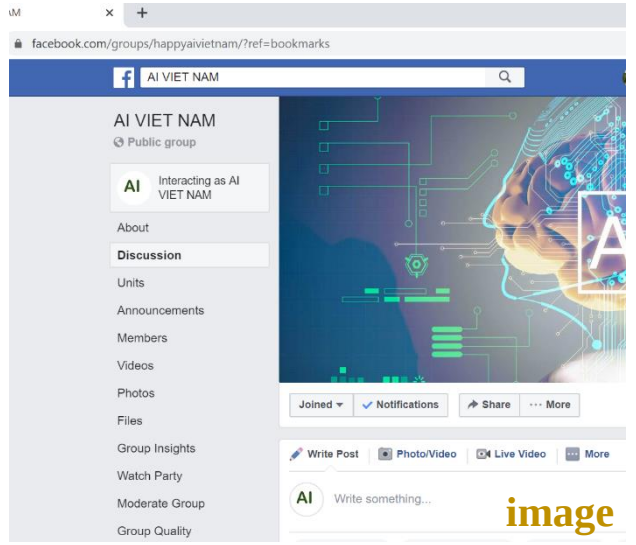
$\rho_{P_1P_3} = 0.9979$ ➧ ρ hoạt động tốt dưới sự thay đổi tuyến tính

Ứng dụng vào template matching



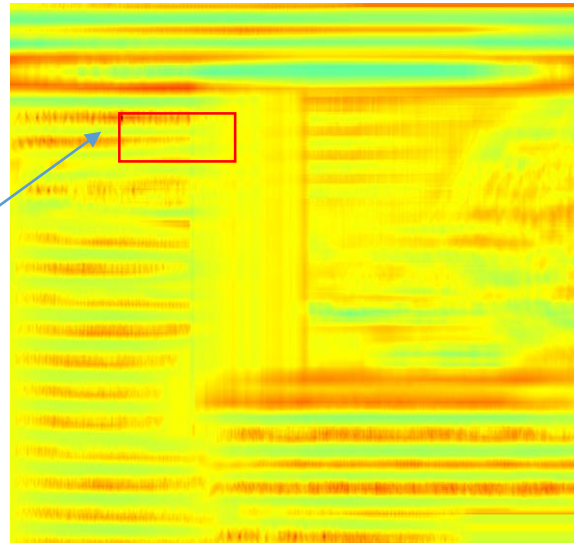
template

Tìm template có trong hình image

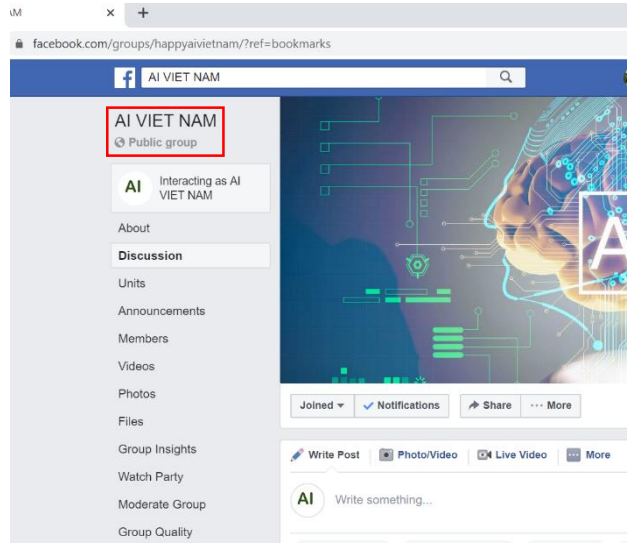


image

max



Map ρ cho từng pixel trong ảnh image



Kết quả

Ứng dụng cho patch matching

P_1 P_2 P_3 P_4

$\rho_{P_1P_2} = 0.55$

$\rho_{P_1P_3} = 0.23$ ➡ Ảnh P_2 giống với ảnh P_1 hơn so với P_3 và P_4

$\rho_{P_1P_4} = 0.30$

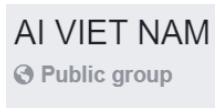
P_1 $P_2 = P_1+50$ $P_3 = 1.2P_1+10$

$\rho_{P_1P_2} = 0.9970$

$\rho_{P_1P_3} = 0.9979$ ➡ ρ hoạt động tốt dưới sự thay đổi tuyến tính

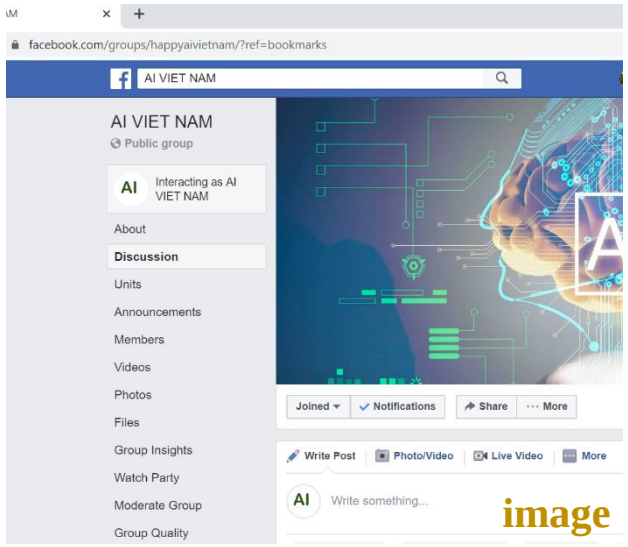
```python
# aivietnam.ai

import numpy as np
from PIL import Image

# load ảnh và chuyển về kiểu list
image1 = Image.open('images/img1.png')
image2 = Image.open('images/img2.png')
image3 = Image.open('images/img3.png')
image4 = Image.open('images/img4.png')

image1_list = np.asarray(image1).flatten().tolist()
image2_list = np.asarray(image2).flatten().tolist()
image3_list = np.asarray(image3).flatten().tolist()
image4_list = np.asarray(image4).flatten().tolist()


# tính correlation coefficient
corr_1_2 = find_corr_x_y(image1_list, image2_list)
corr_1_3 = find_corr_x_y(image1_list, image3_list)
corr_1_4 = find_corr_x_y(image1_list, image4_list)

print('corr_1_2:', corr_1_2)
print('corr_1_3:', corr_1_3)
print('corr_1_4:', corr_1_4)
```

Ứng dụng vào template matching

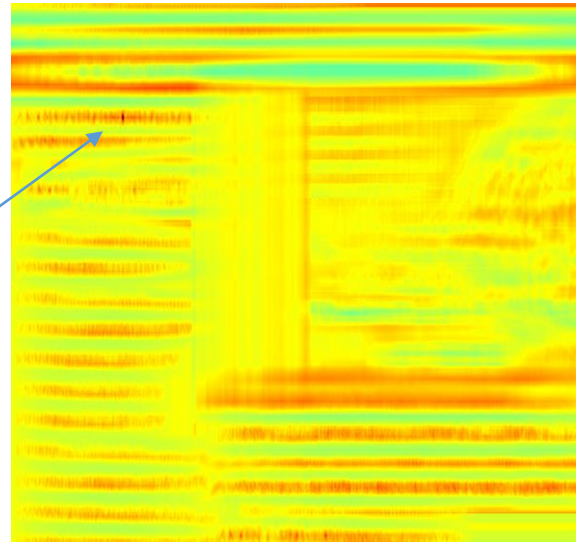


template

Tìm template có trong hình image

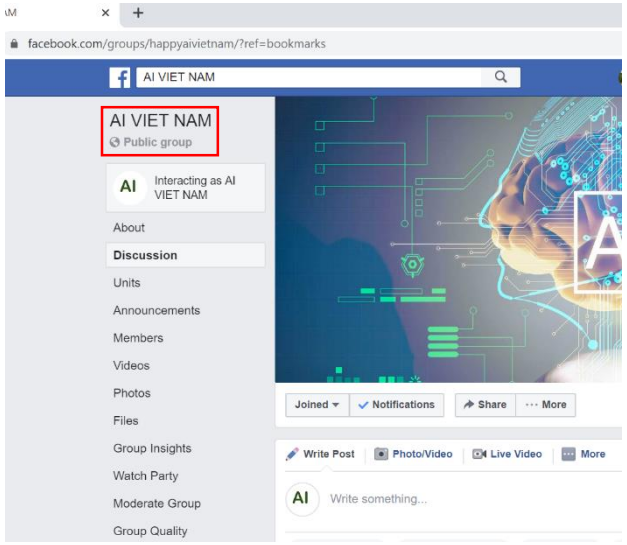


image



max

Map ρ cho từng pixel trong ảnh image



Kết quả

Template Matching

❖ Using pre-trained model

Template

Image

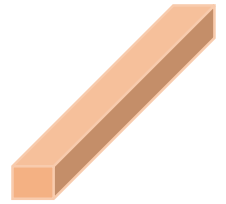
Template Matching

❖ Feature extraction

template

(1, 300, 300, 3)

VGG16
(Top=False)

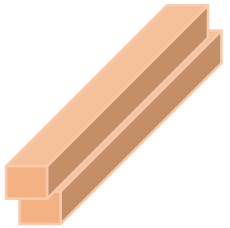


(1, 9, 9, 512)

image

(1, 800, 1000, 3)

VGG16
(Top=False)

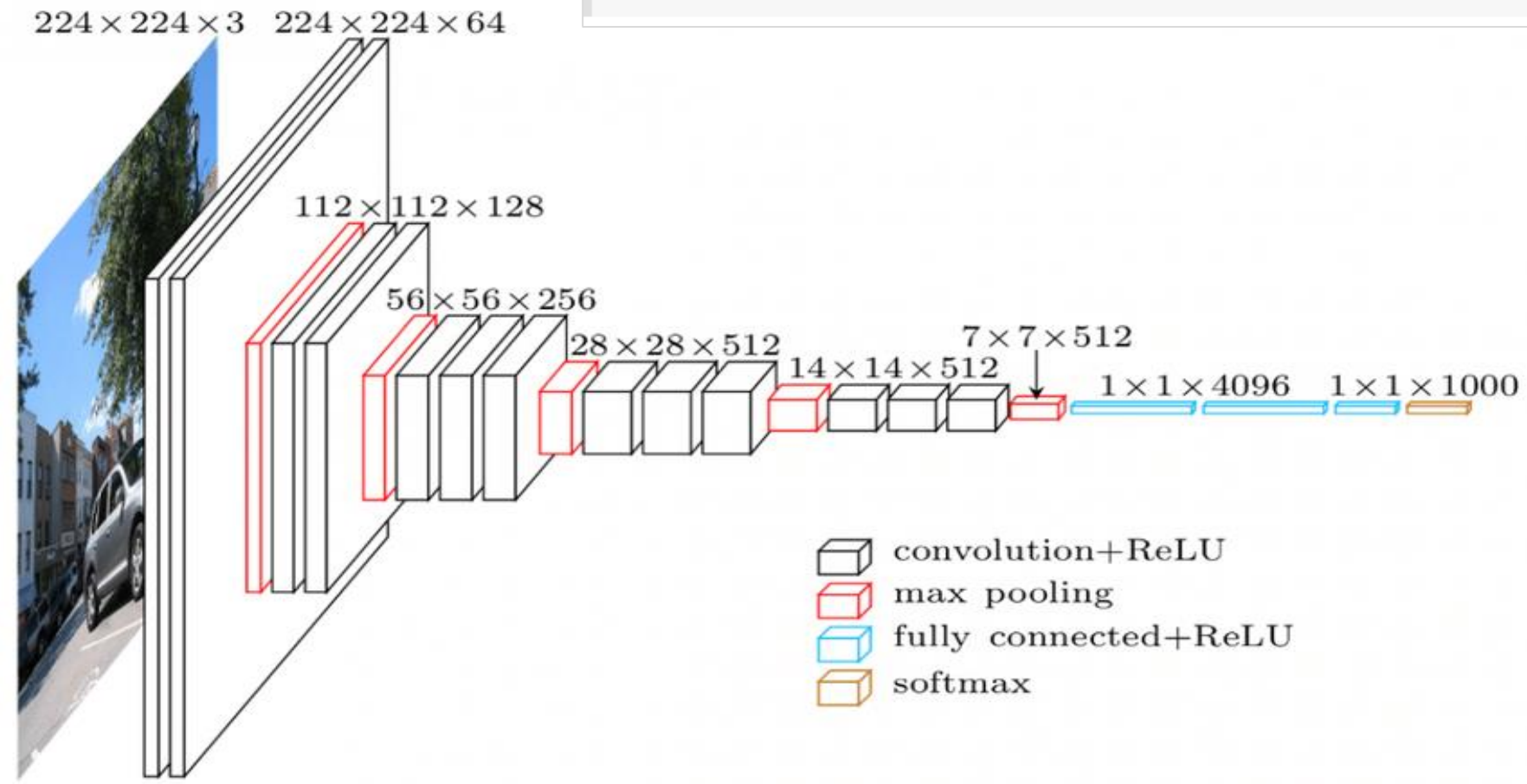


(1, 25, 31, 512)

Template Matching

❖ Get the VGG16 model

```
model = tf.keras.applications.VGG16(include_top=False,
                                    weights='imagenet',
                                    input_shape=(None, None, 3))
model.summary()
```



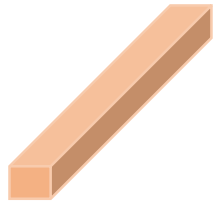
Template Matching

❖ Feature extraction

template

(1, 300, 300, 3)

VGG16
(Top=False)



(1, 9, 9, 512)

```
from tensorflow.keras.preprocessing import image as kimage

# load template
template = kimage.load_img(PATH+'template.jpg', target_size=(300, 300))

# add one more dim
template_dim = np.expand_dims(template, axis=0) # (1, 300, 300, 3)

# compute features
template_feature = model.predict(template_dim)  # (1, 9, 9, 3)
```

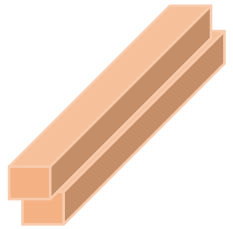
Template Matching

❖ Feature extraction

image

(1, 800, 1000, 3)

VGG16
(Top=False)

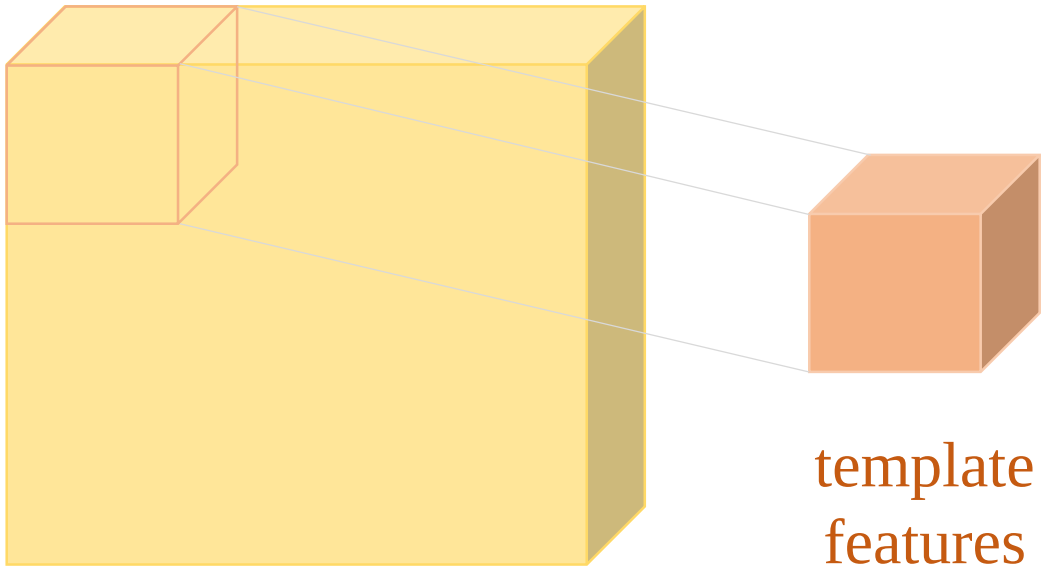


(1, 25, 31, 512)

```python
from tensorflow.keras.preprocessing import image as kimage

# load template
image = kimage.load_img(PATH+'image2.jpg', target_size=(800, 1000))

# add one more dim
image_dim = np.expand_dims(image, axis=0) # (1, 800, 1000, 3)

# compute features
image_feature = model.predict(image_dim)  # (1, 25, 31, 3)
```

Template Matching

❖ Compute similarity



template features

Image features

```python
# some parameters
side = 9
height_fm = 25
width_fm  = 31

# to store similarity values
sim_data = []
for i in range(height_fm-side+1):
    for j in range(width_fm-side+1):
        # get patch at (i,j)
        patch = image_feature[0,i:i+side,j:j+side,:]

        # reshape
        patch = np.reshape(patch, (1, -1))
        template_feature = np.reshape(template_feature, (1,-1))

        # compute cosine similarity
        sim = cosine_similarity(patch, template_feature)

        # save to a list
        sim_data.append((sim[0][0], i, j))
```

Template Matching

❖ Locate Object Position

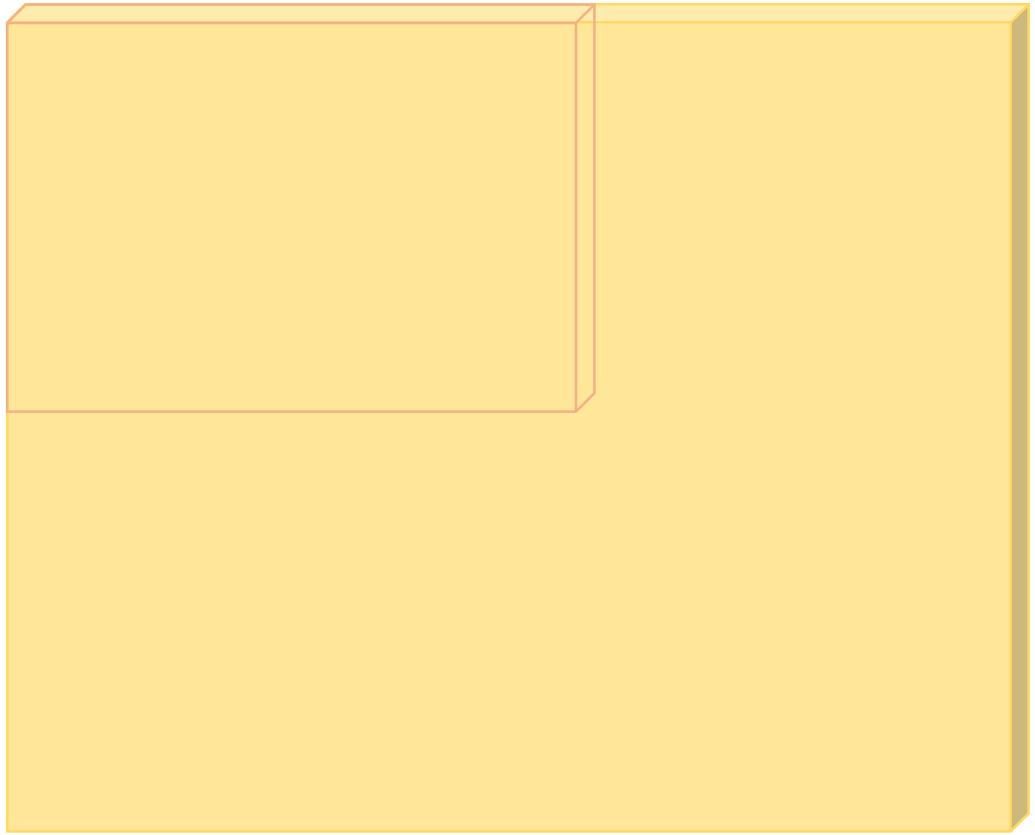


Image
(1, 800, 1000, 3)

```
scale_height = 800//25
scale_width  = 1000//31
```

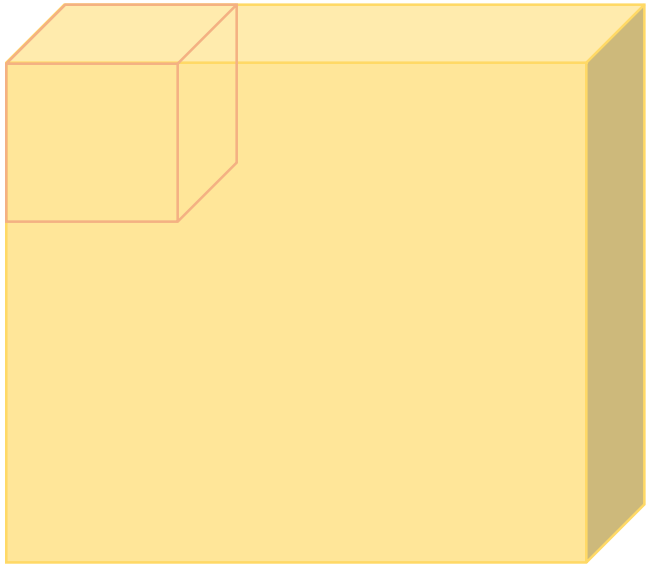


Image features
(1, 25, 31, 512)

Template Matching

❖ Case study 1: Absolute difference

```
distances = []
for i in range(height-side+1):
  for j in range(width-side+1):
    patch = image[i:i+side,j:j+side,:]
    dis = np.sum(np.absolute(template - patch))
    distances.append((dis, i, j))

print(len(distances))
print(distances[0][0])
```

Template

Image

Template Matching

❖ Case study 2: Cosine similarity

```
sim_data = []
for i in range(height-side+1):
  for j in range(width-side+1):
    patch = image_dim[0,i:i+side,j:j+side,:]
    sim = cosine_similarity(np.reshape(patch, (1, -1)),
                            np.reshape(template_dim, (1,-1)))
    sim_data.append((sim[0][0], i, j))

print(len(sim_data))
```

Template

Image

Template Matching

❖ Case study 3: Multi-templates

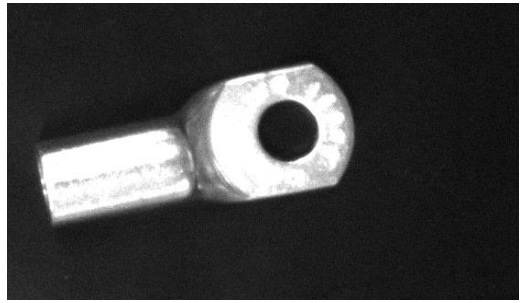
```
sim_data = []
for i in range(height-side+1):
  for j in range(width-side+1):
    patch = image_dim[0,i:i+side,j:j+side,:]
    sim = cosine_similarity(np.reshape(patch, (1, -1)),
                            np.reshape(template_dim, (1,-1)))
    sim_data.append((sim[0][0], i, j))

print(len(sim_data))
```

Template

Image

Template Matching

❖ Future work



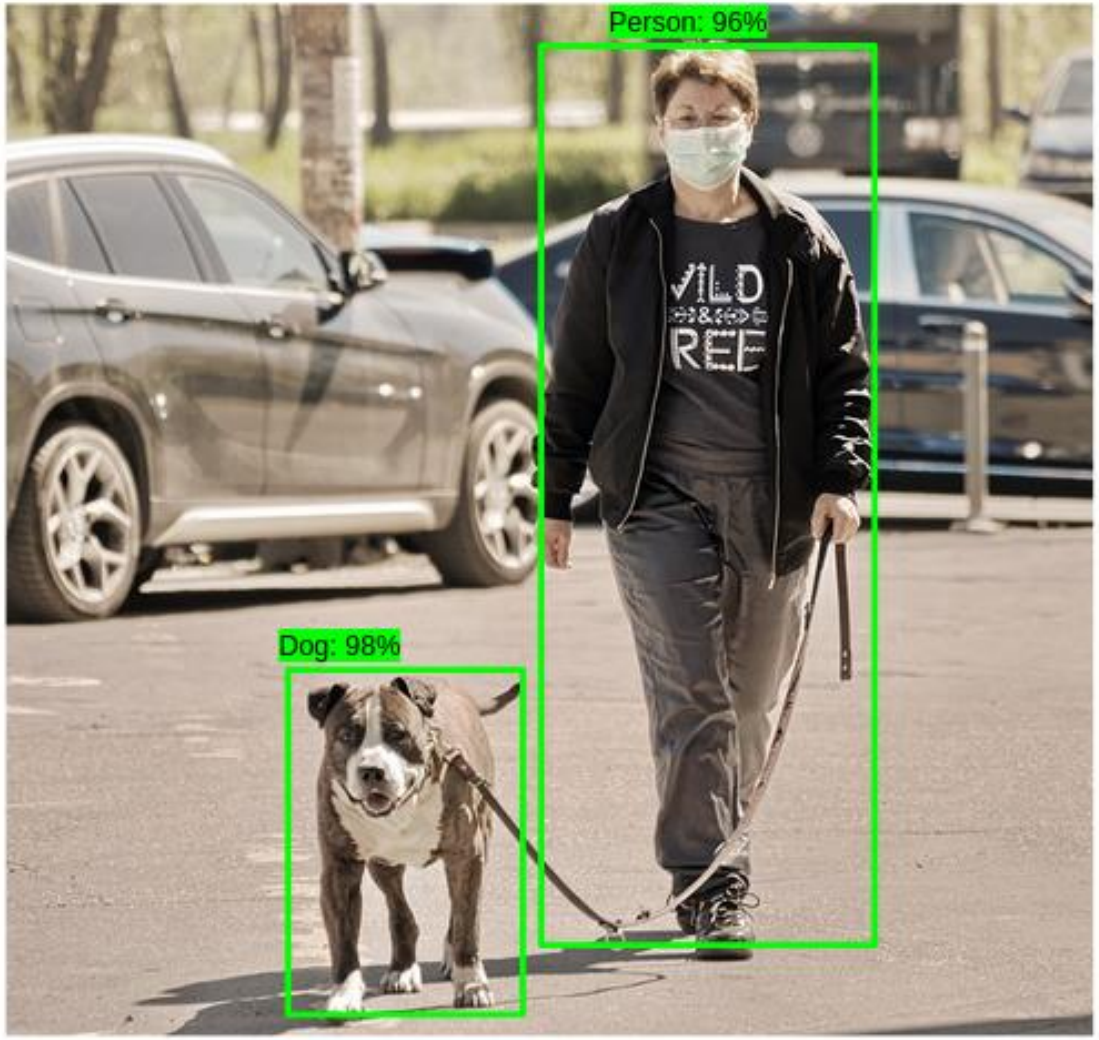
Template

Image

Object Detection Using Pretrained Model

Naïve Object Detection

❖ Idea

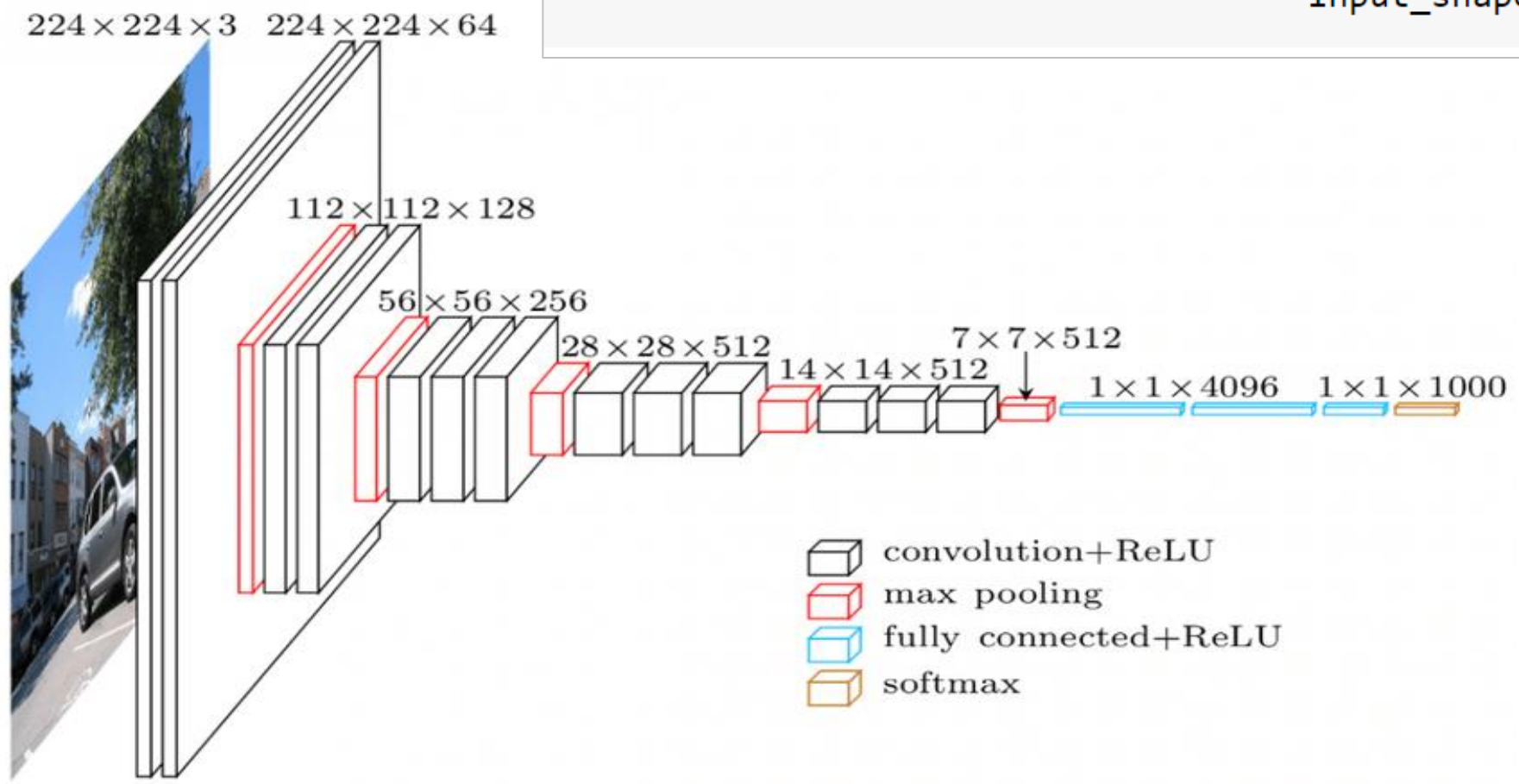


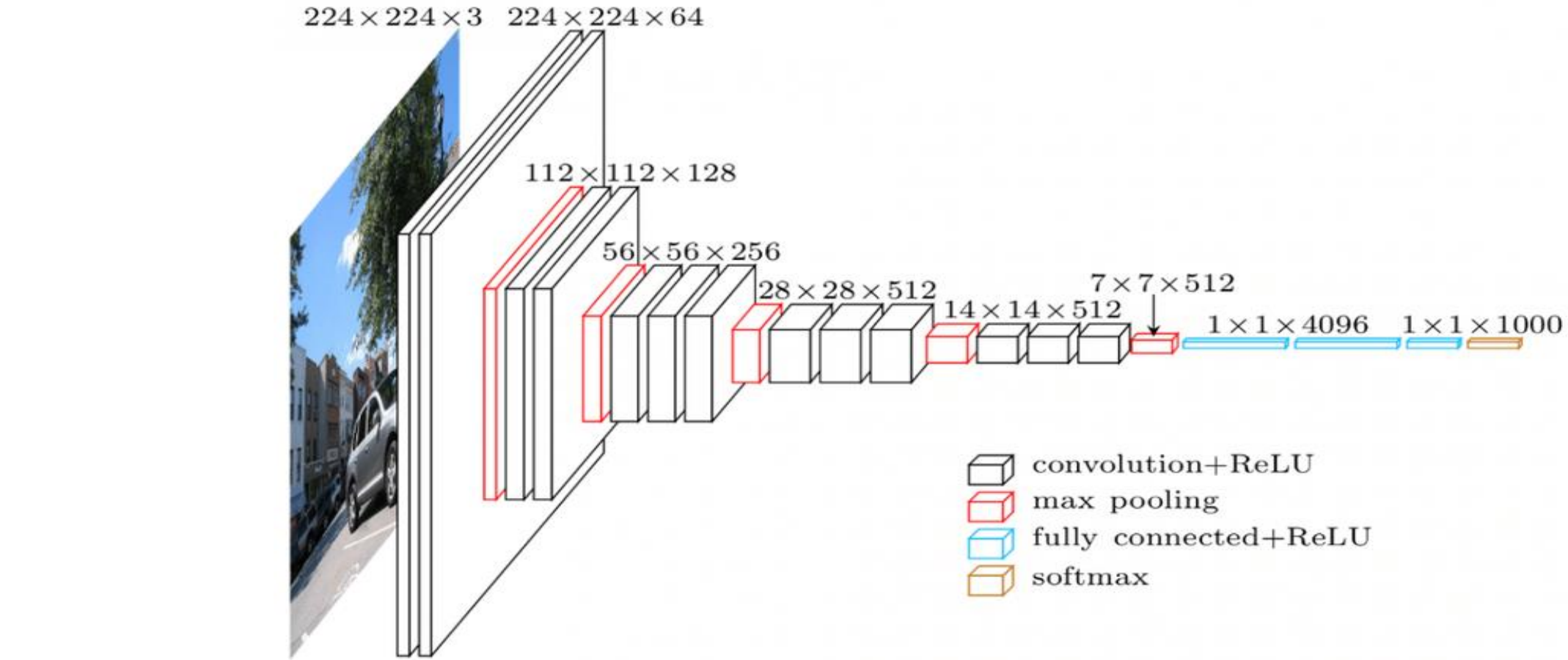
https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/

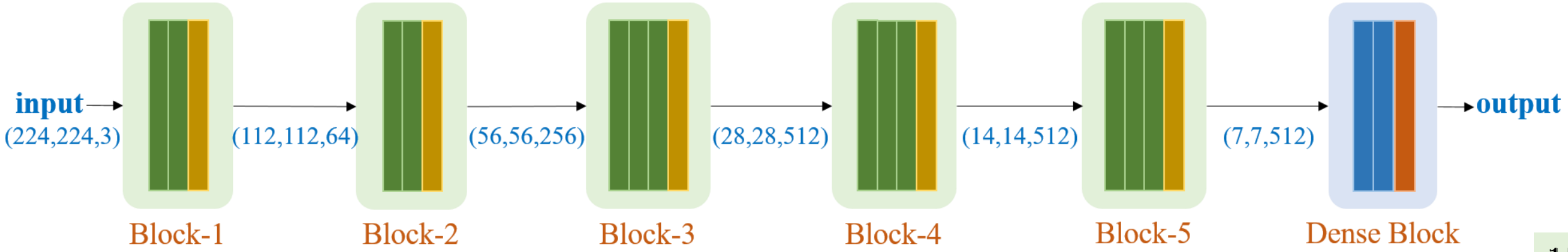
Naïve Object Detection

❖ Get the VGG16 model

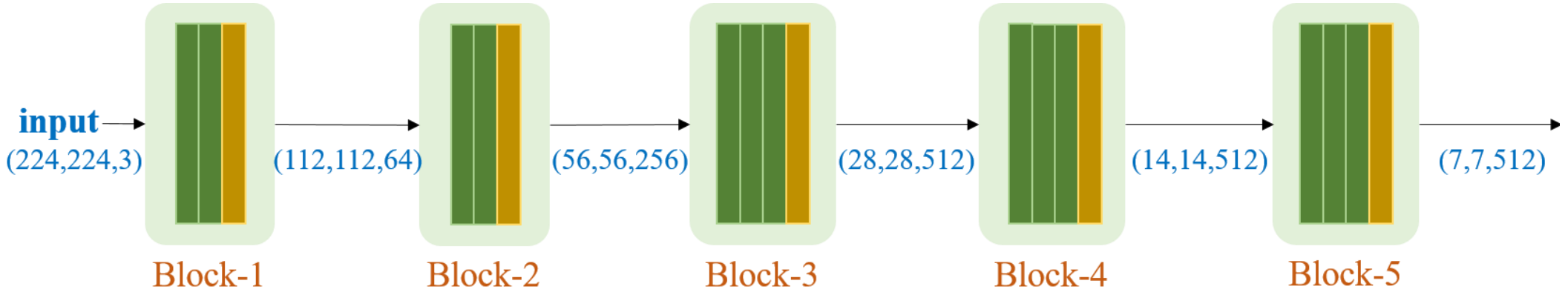
```
# get model
model = tf.keras.applications.VGG16(include_top=True,
                                    weights='imagenet',
                                    input_shape=(224, 224, 3))
```

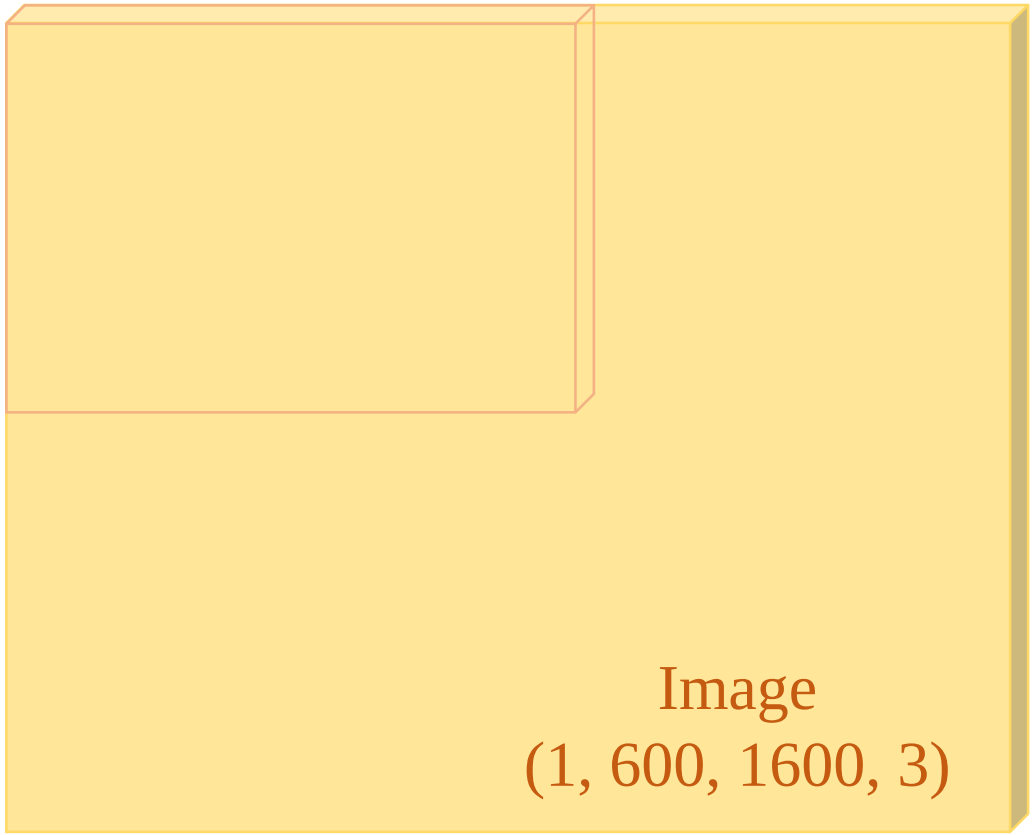



224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax


input
(224,224,3)
(112,112,64)
(56,56,256)
(28,28,512)
(14,14,512)
(7,7,512)
output
Block-1
Block-2
Block-3
Block-4
Block-5
Dense Block

❖ Feature extraction





Feature Extraction

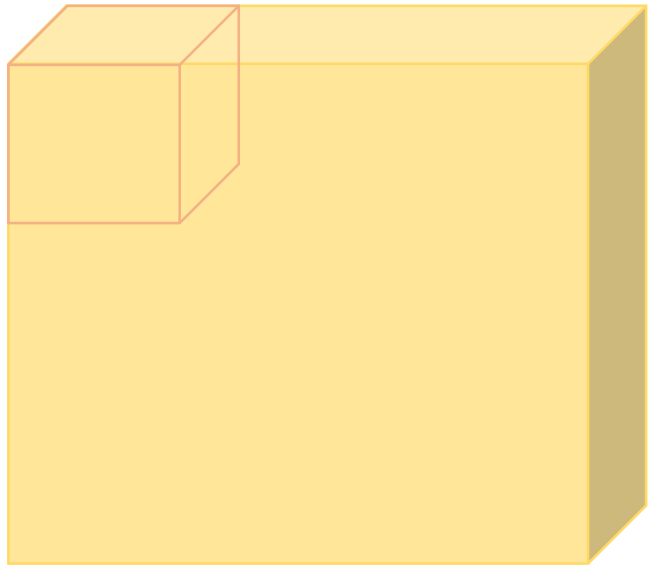
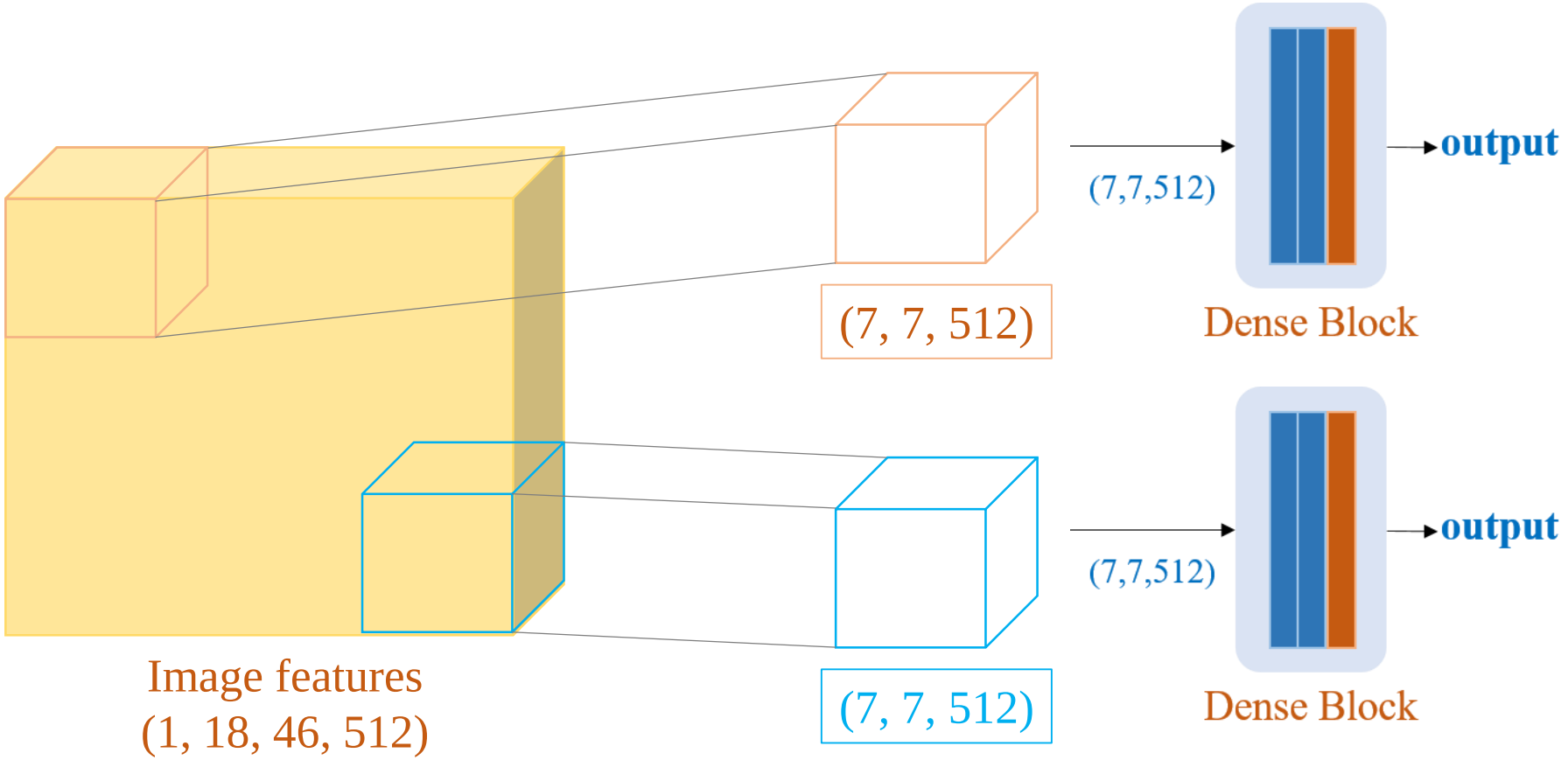


Image features
(1, 18, 46, 512)

Naïve Object Detection

❖ Classification



Naïve Object Detection

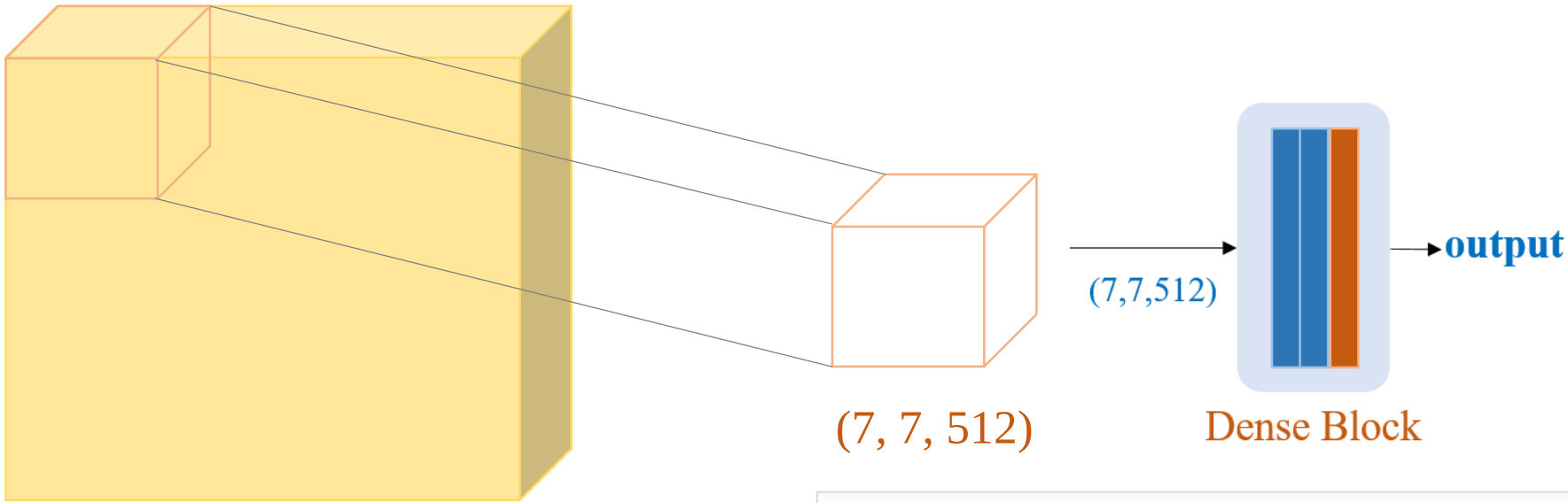


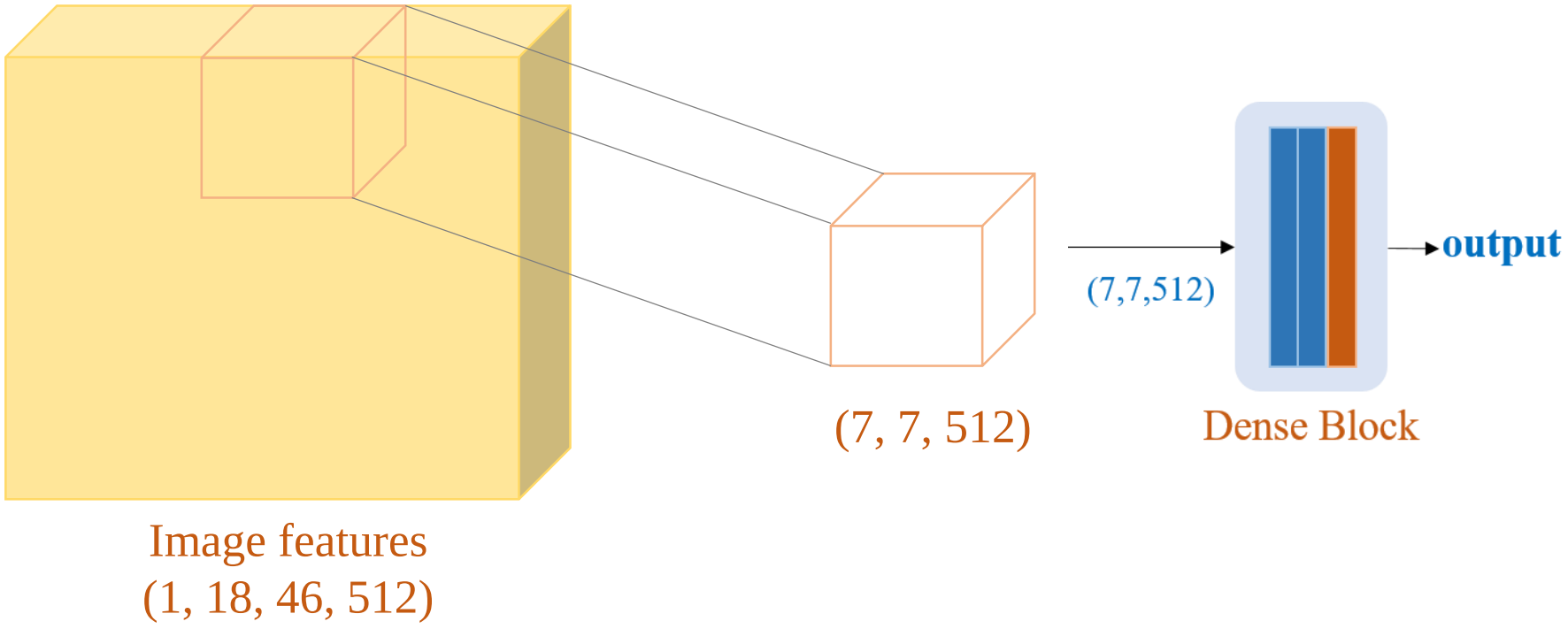
Image features
(1, 18, 46, 512)

```
def compute_features(data):
  for i in range(1,19):
    data = model.layers[i](data)
  return data
```

```
def compute_prediction(data):
  for i in range(19,23):
    data = model.layers[i](data)
  return data
```

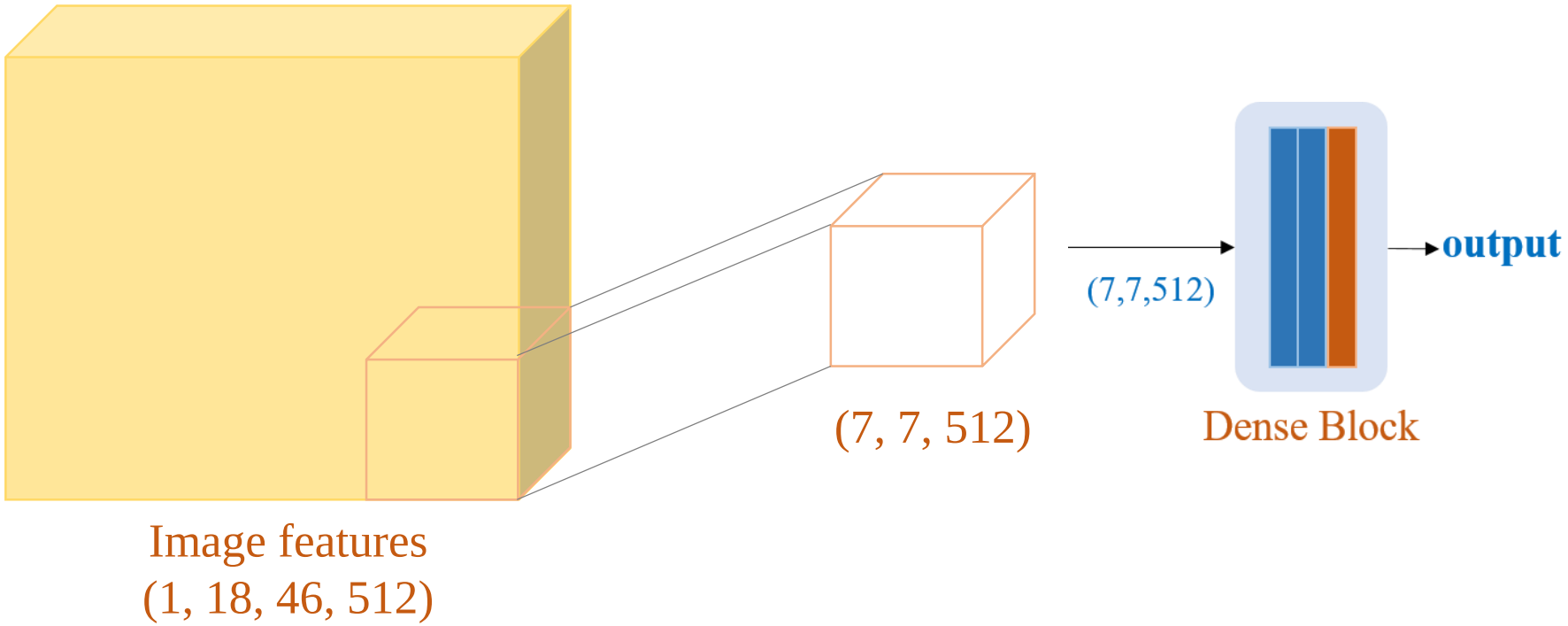
Naïve Object Detection

❖ Classification

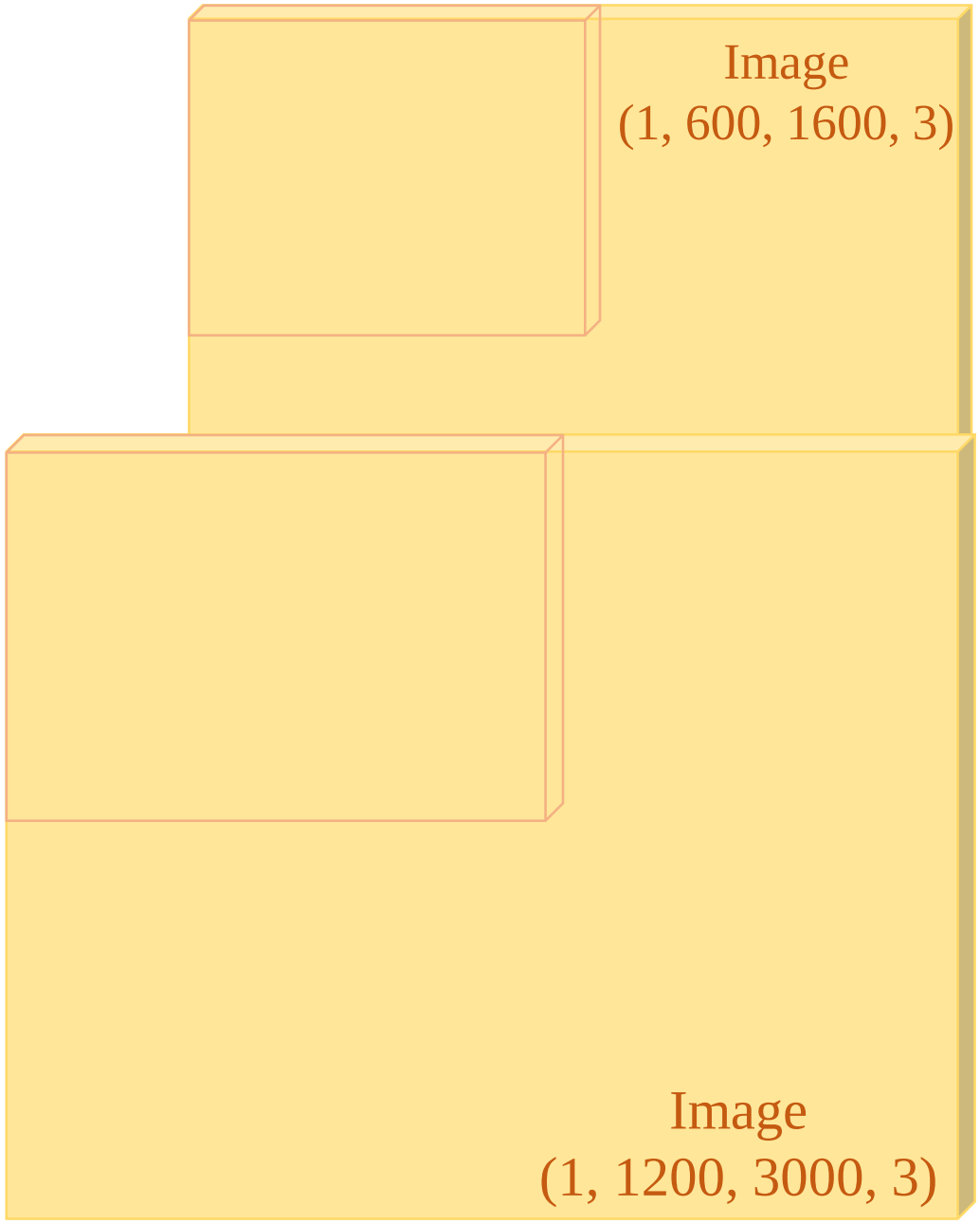


Naïve Object Detection

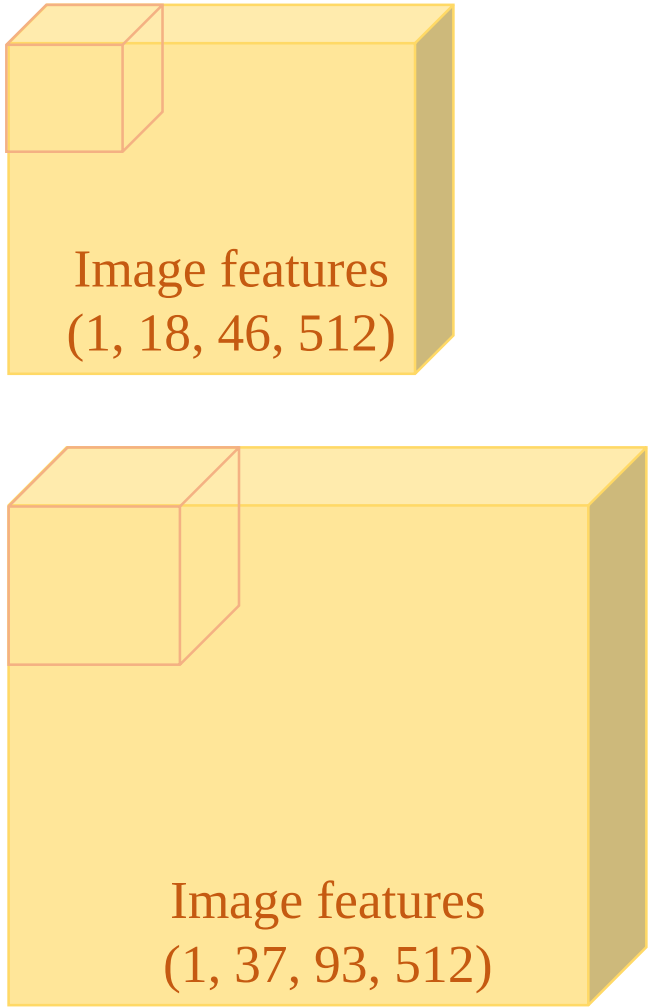
❖ Classification



❖ Feature extraction

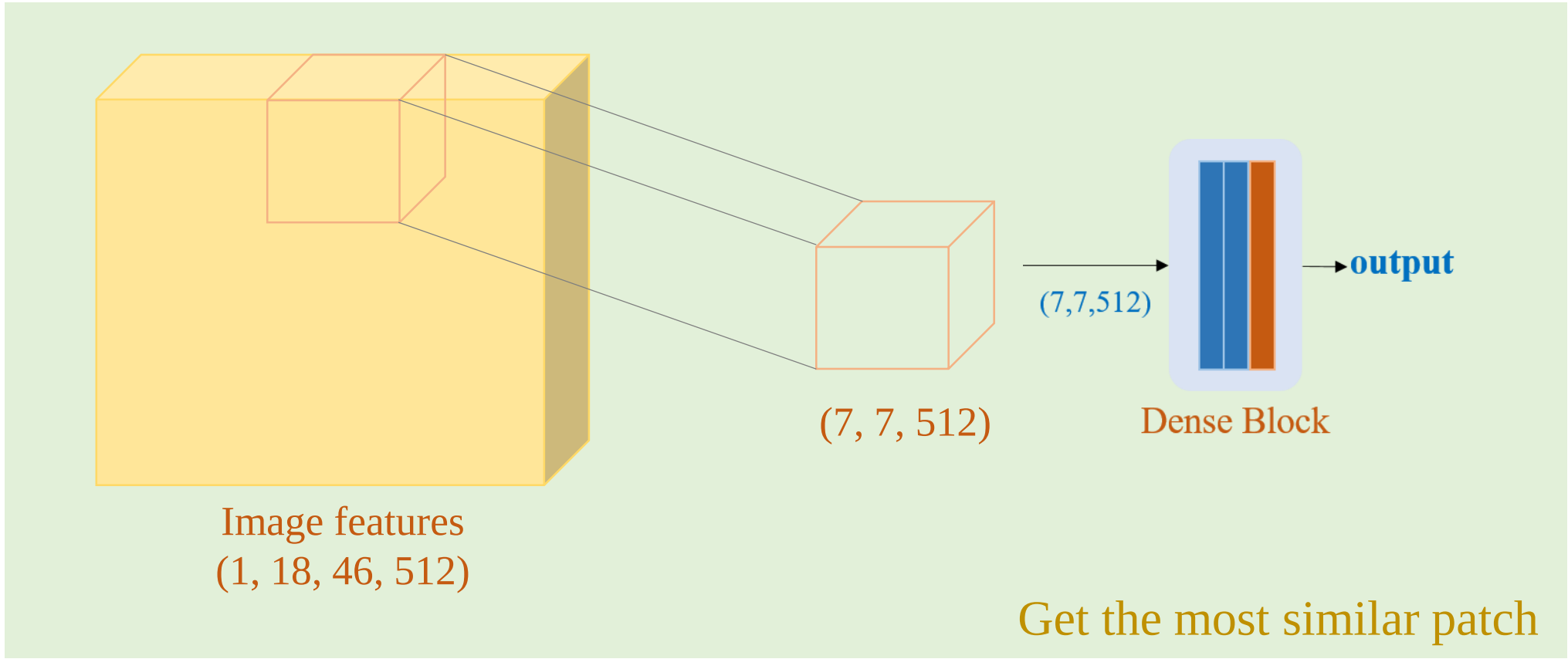


Feature Extraction



Naïve Object Detection

❖ Case study 1: Single Object Detection



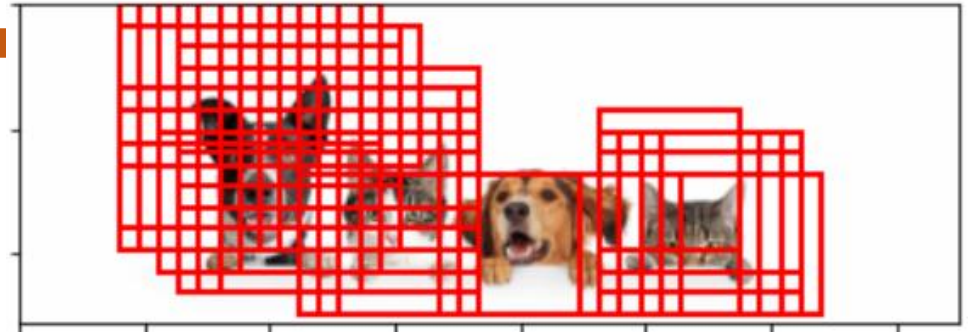
Naïve Object Detection

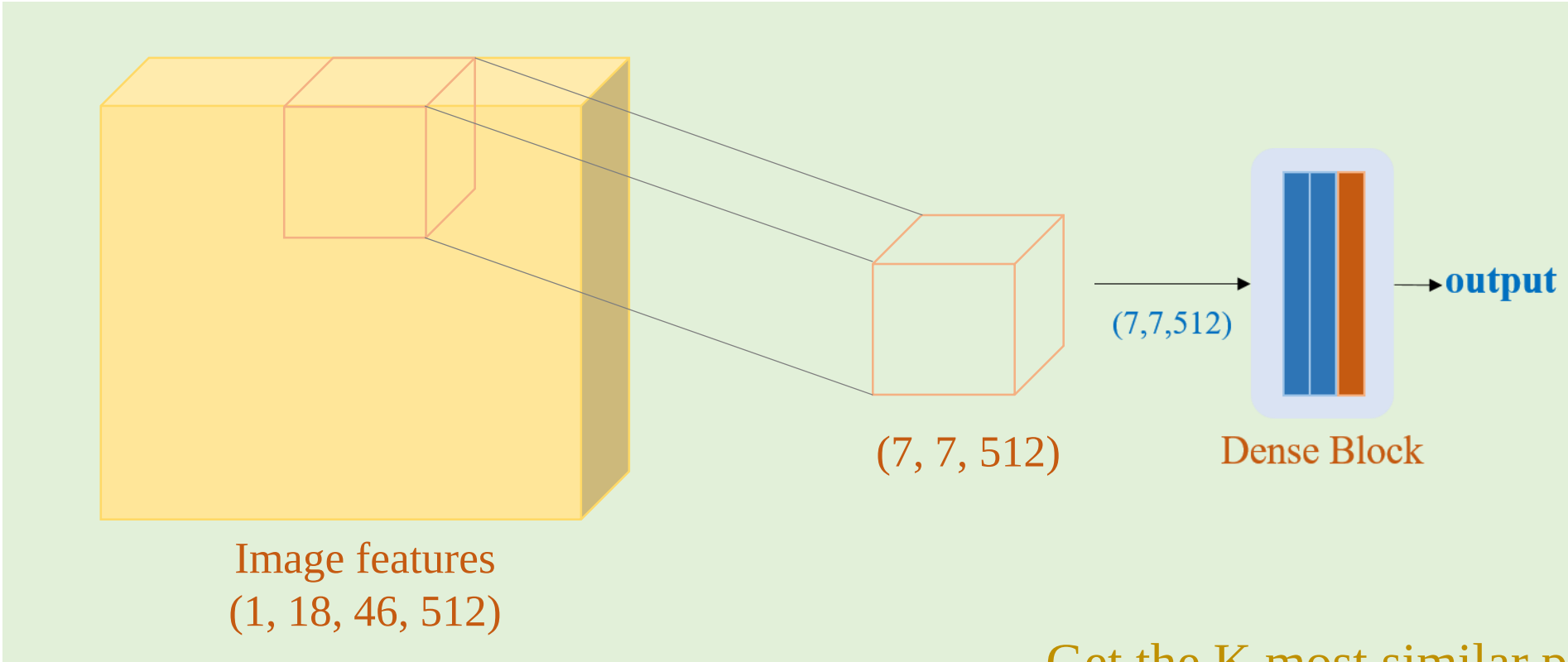
❖ Case study 1: Single Object Detection


```
# compute predictions
height = pred_query.shape[1]
width  = pred_query.shape[2]
depth  = pred_query.shape[3]

side = 7
prediction_data = []
for i in range(height-side+1):
  for j in range(width-side+1):
    patch = pred_query[:,i:i+side,j:j+side,:]
    patch = compute_prediction(patch)
    prediction_data.append( (tf.math.reduce_max(patch[0]).numpy(),
                             i, j, tf.math.argmax(patch[0]).numpy()) )
```

Naïve Object Detection

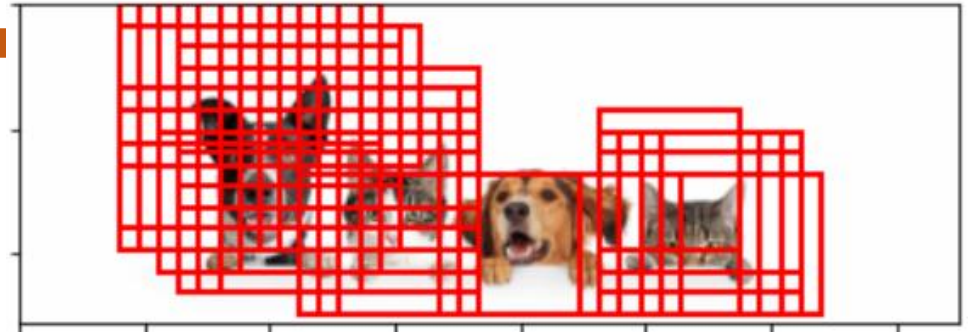
❖ Case study 2: Multi-Object Detection





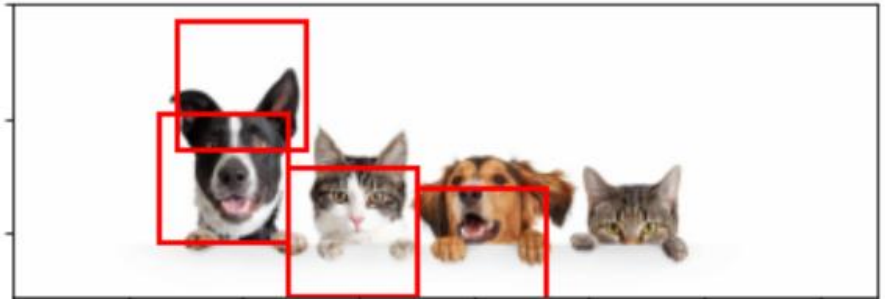
Naïve Object Detection

❖ Case study 2: Multi-Object Detection

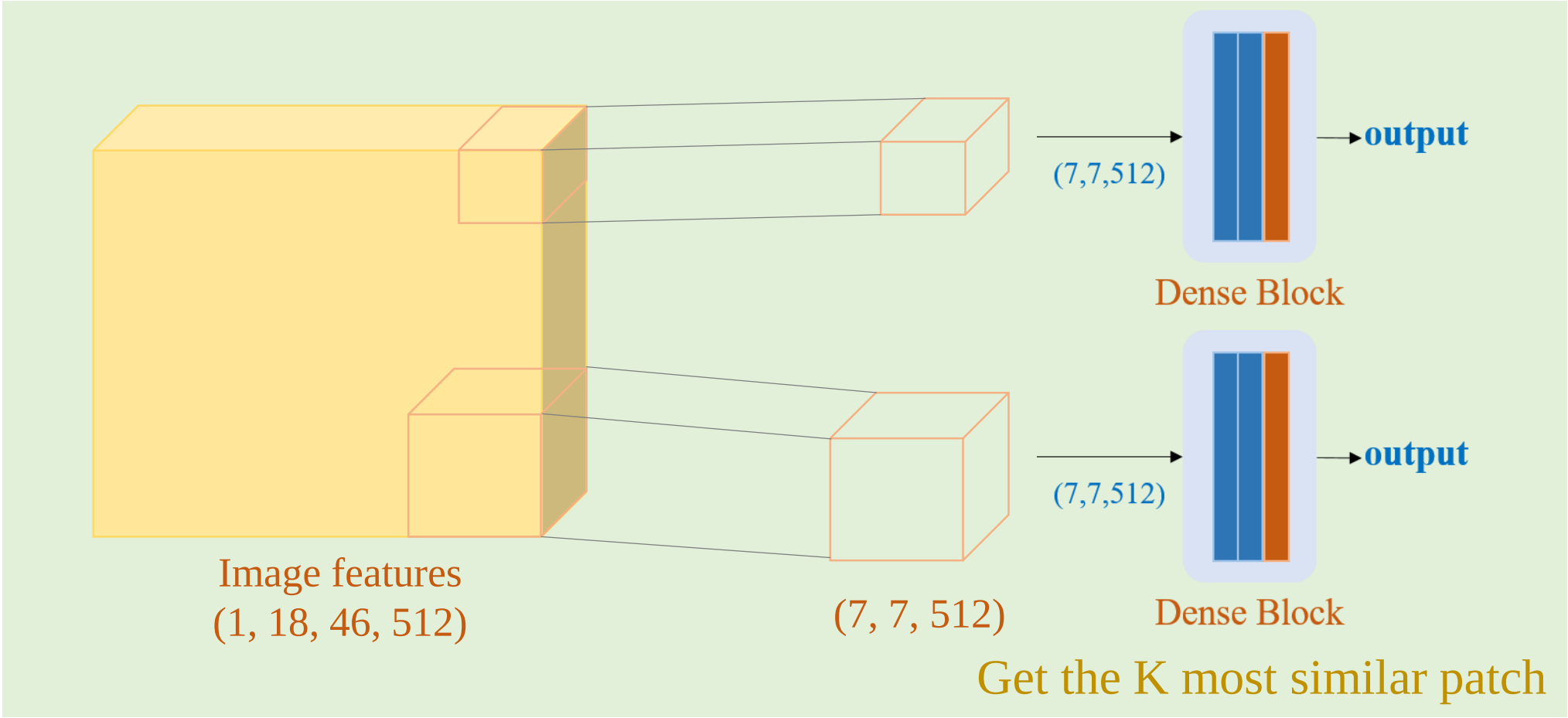


```
from scipy.spatial import distance

# remove duplication
def remove_duplication(data):
  result = []

  length = len(data)
  for k in range(length-1):
    duplicated = check_duplication(data[k][1], data[k][2], result)

    if (duplicated==False and data[k][0]>0.5):
      result.append( data[k] )


  return result
```

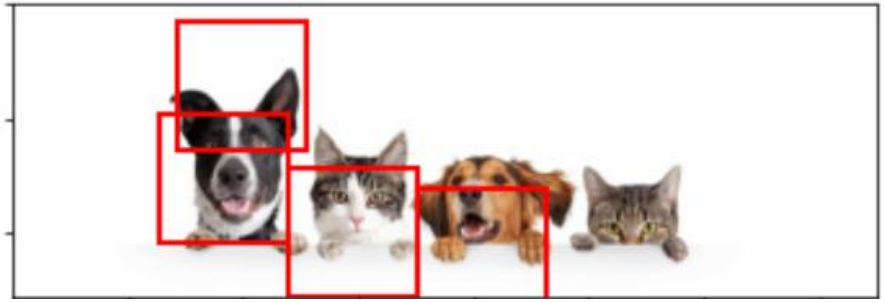
Naïve Object Detection



❖ Case study 3: Multi-scale Object Detection



Naïve Object Detection



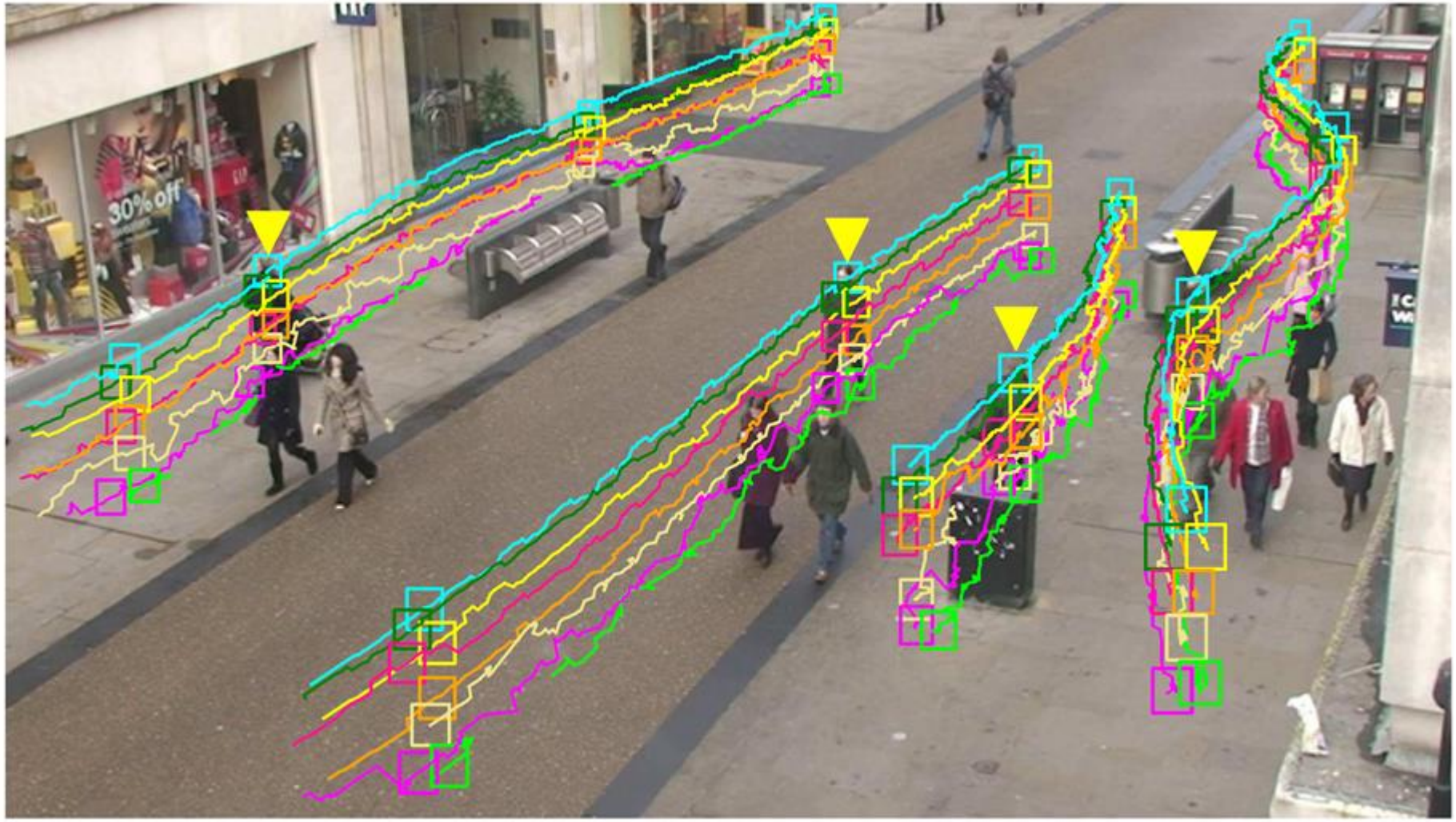
❖ Case study 3: Multi-scale Object Detection

```
# compute predictions
side = 7
prediction_data = []

for scale in range(scale_level):
  height_fm = list_of_features[scale].shape[1]
  width_fm  = list_of_features[scale].shape[2]
  depth_fm  = list_of_features[scale].shape[3]

  for i in range(height_fm-side+1):
    for j in range(width_fm-side+1):
      patch = list_of_features[scale][:,i:i+side,j:j+side,:]
      patch = compute_prediction(patch)
      prediction_data.append( (tf.math.reduce_max(patch[0]).numpy(),
                               i, j, tf.math.argmax(patch[0]).numpy(), scale) )

print(len(prediction_data))
```

Object Tracking Using Pretrained Model

Object Tracking

❖ **Objective**



http://deepmachinelearningai.com/object-tracking-in-deep-learning/

Object Tracking

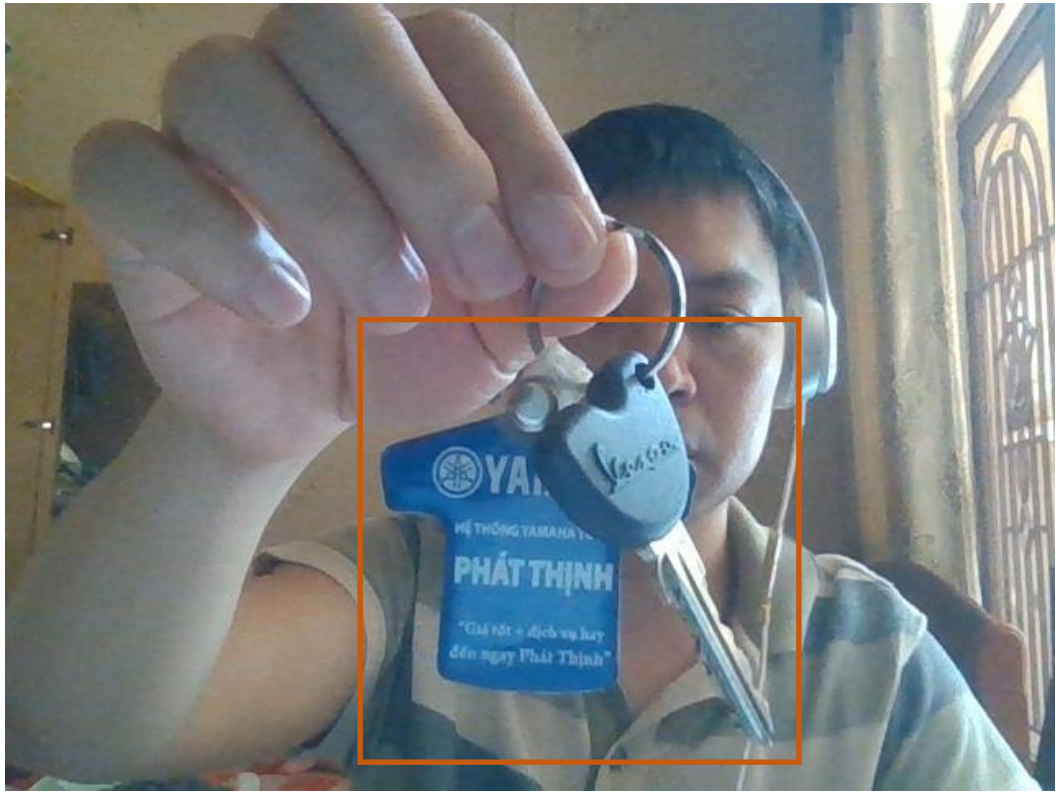
❖ Objective

Frame at time t

Frame at time k

Object Tracking

❖ Objective

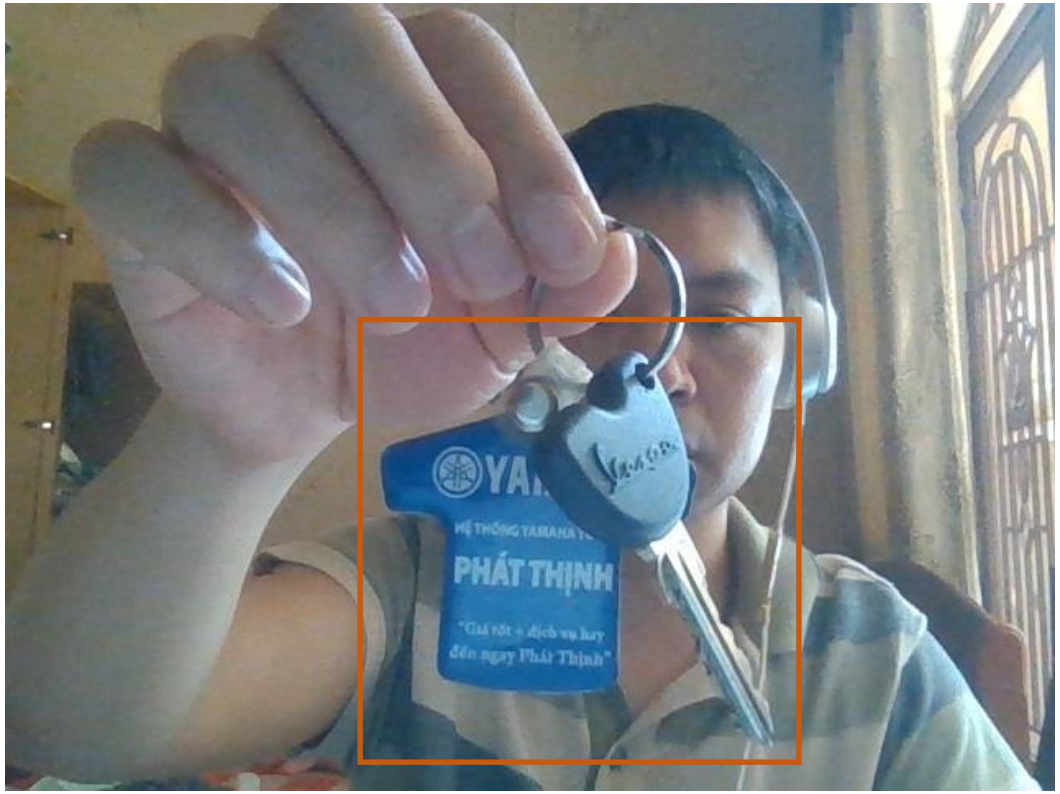


Frame at time t

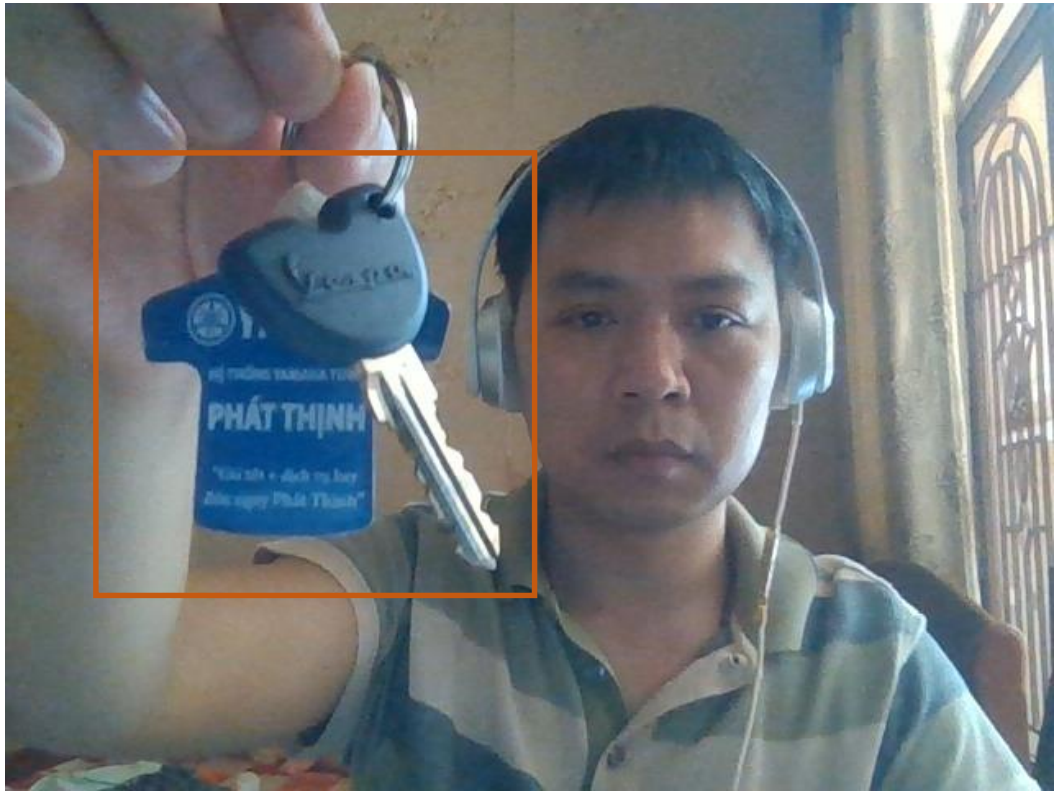
Frame at time k

Object Tracking

❖ Objective



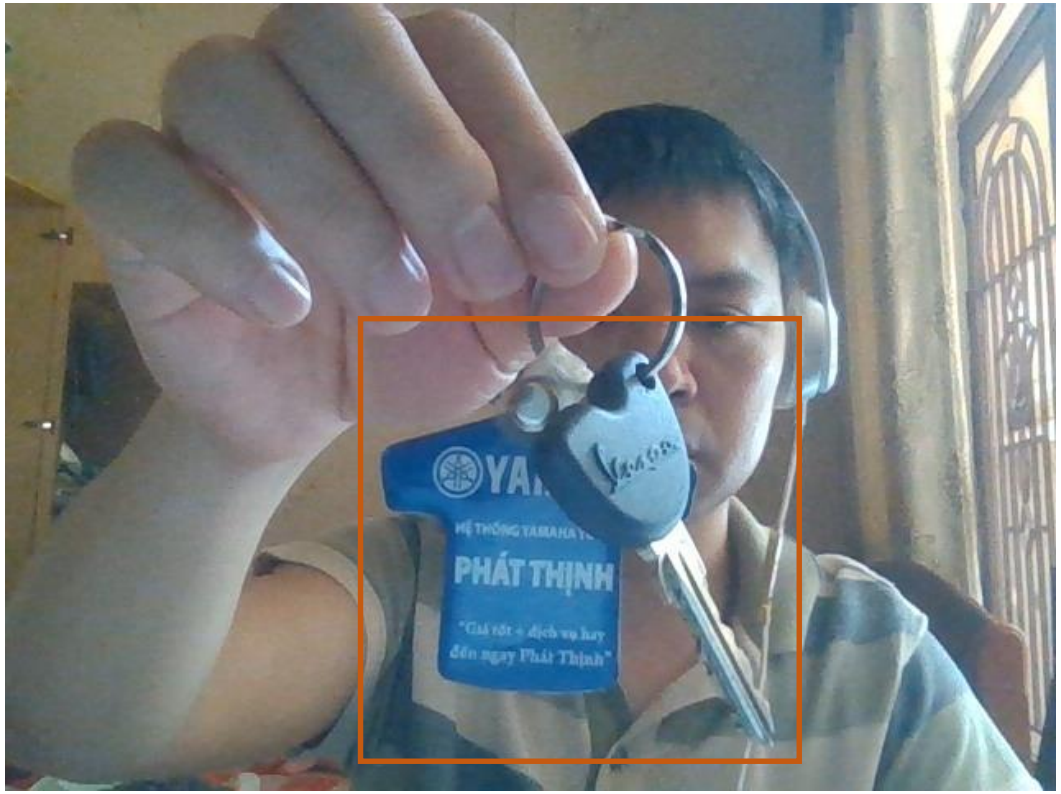
Frame at time t



Frame at time k

Object Tracking

❖ Idea



Frame at time t

Object Tracking

❖ Idea

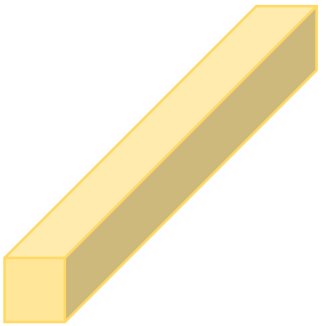
Frame at time k

Object Tracking

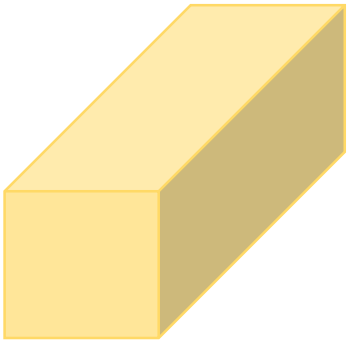
Template

Frame at time k

Feature Extraction

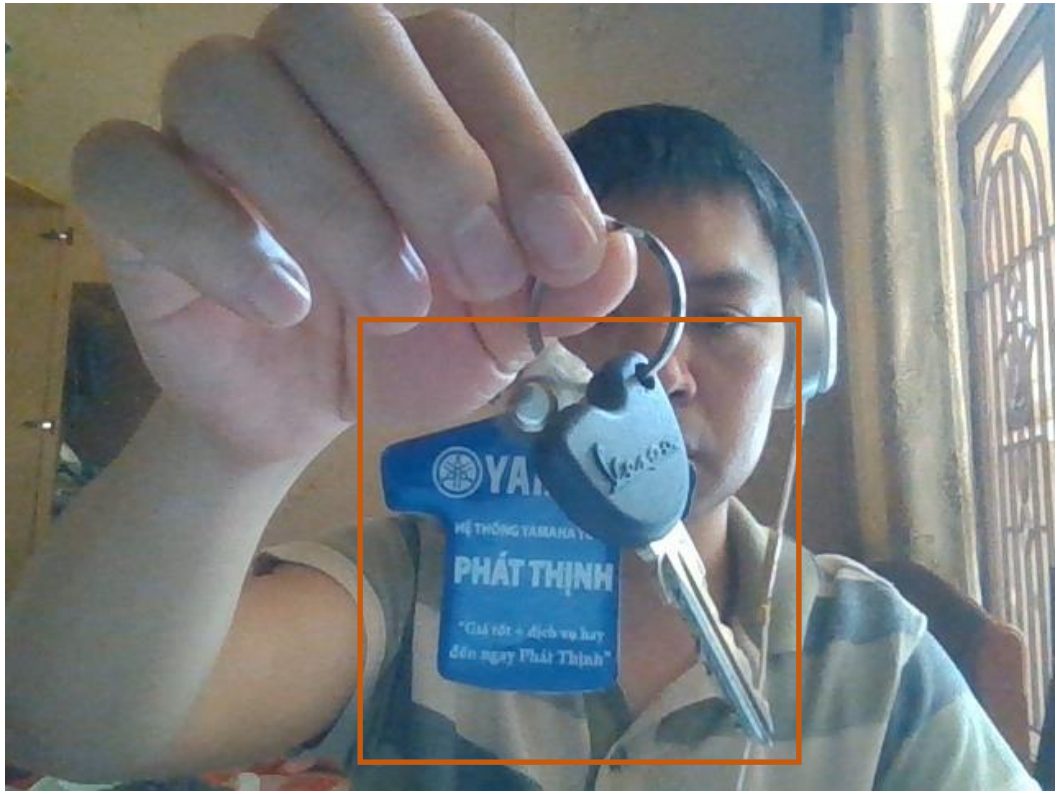


Feature Extraction



Object Tracking

❖ Case Study



Frame at time t

Frame at time k


Thank you!