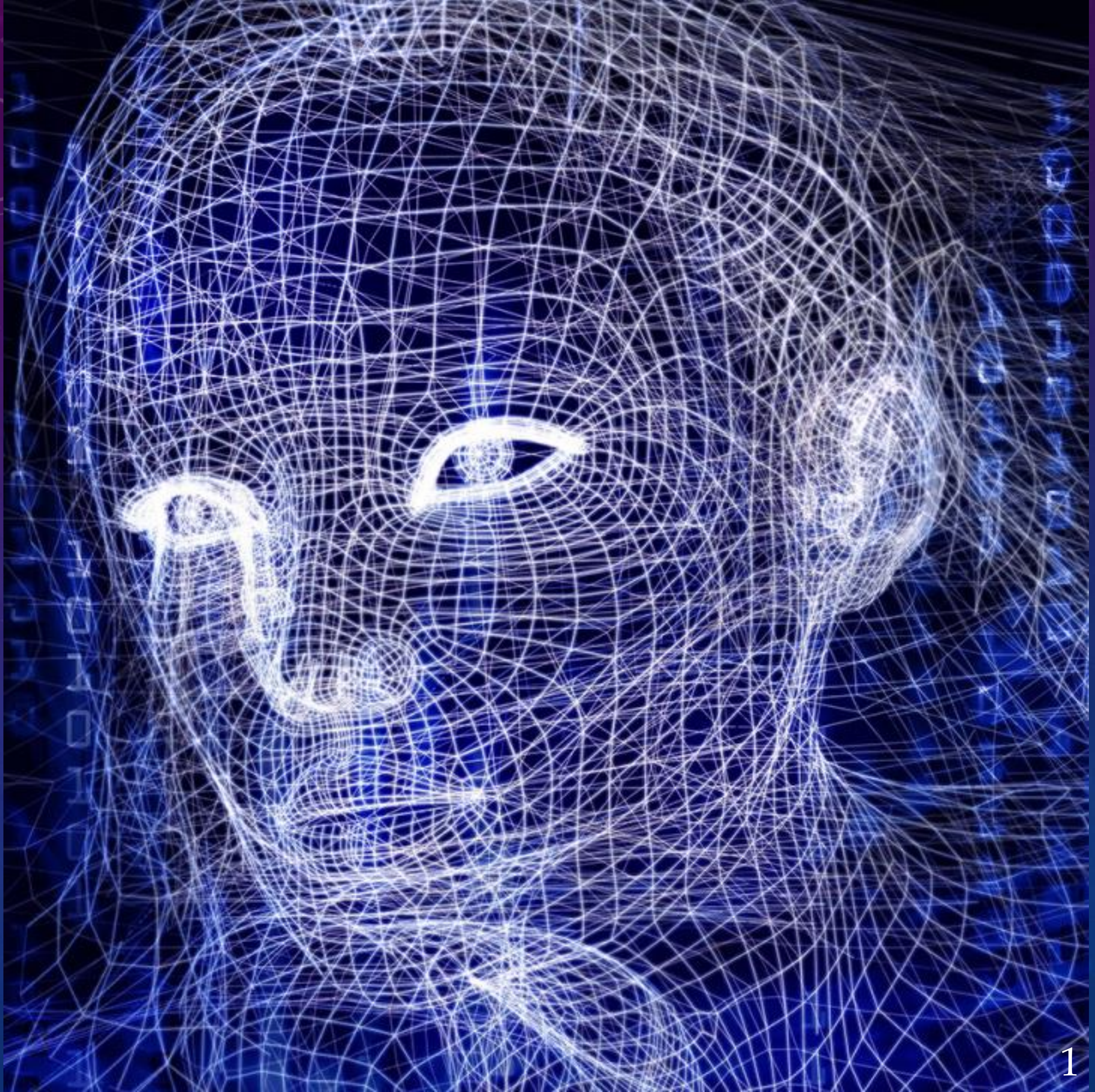
DIGITAL SURVEILLANCE SYSTEMS AND APPLICATION

CH 1_EXERCISE

徐繼聖
Gee-Sern Jison Hsu
National Taiwan University of Science and Technology



Content

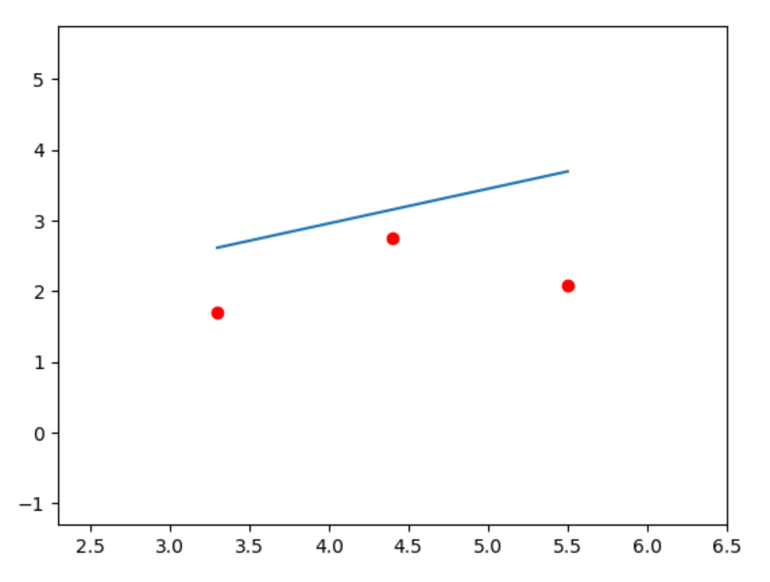
- Linear Regression
- Confusion Matrix

Example 1.1 Linear Regression

Use the Pytorch to build our first linear regression model. We employ the SGD as optimizer and use MSE loss function to train the model. Moreover, we can visualize the training process.

Step:

- Define Model Structure
- Loss Function (Criterion) and Optimizer
- Model Training
- Visualize Linear Regression



Example 1.1 Linear Regression

- Set up the base structure of this model in Pytorch

```
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
plt.ion()

# Hyper-parameters
num_epochs = 50
learning_rate = 0.001
# Set initialize parameter y = ax + b
a = -0.5
b = 1
```

```
# Toy dataset
x_train = np.array([[3.3], [4.4], [5.5]], dtype=np.float32)

y_train = np.array([[1.7], [2.76], [2.09]], dtype=np.float32)
```

Example 1.1 Linear Regression

- Initialize the model type and declare the forward pass

```
# Define Linear regression model
model = nn.Linear(1, 1)
# Initialize parameter
model.weight.data.fill_(a)
model.bias.data.fill_(b)
```

Example 1.1 Linear Regression

Use the Mean Square Error (MSE), which is the most commonly used regression loss function

```
# Define Loss
criterion = nn.MSELoss()
```

Use Stochastic Gradient Descent (SGD) optimizer for the update of hyperparameters

```
# Define optimizer
optimizer = torch.optim.SGD(model.parameters(),
lr=learning_rate)
```

Example 1.1 Linear Regression

```
for epoch in range(num_epochs):
    # Convert numpy arrays to torch tensors
    inputs = torch.from_numpy(x_train)
    targets = torch.from_numpy(y_train)
    # Forward pass
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    # Get the model parameters (slope, bias)
    [a, b] = model.parameters()
    print ('Epoch [{}/{}], Loss: {:.4f}, New_a:{:.2f}, New_b:{:.2f}, y_predict:{}, a_gradient:{},
b_gradient:{}'.format(epoch+1, num_epochs, loss.item(),a.data[0][0],
b.data[0],outputs.data[0],model.weight.grad,model.bias.grad))
    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

- optimizer.zero_grad():Because every time a variable is back propagated through, the gradient will be accumulated instead of being replaced.
- loss.backward(): Backward
- optimizer.step(): Parameters update based on the current gradient.

Example 1.1 Linear Regression

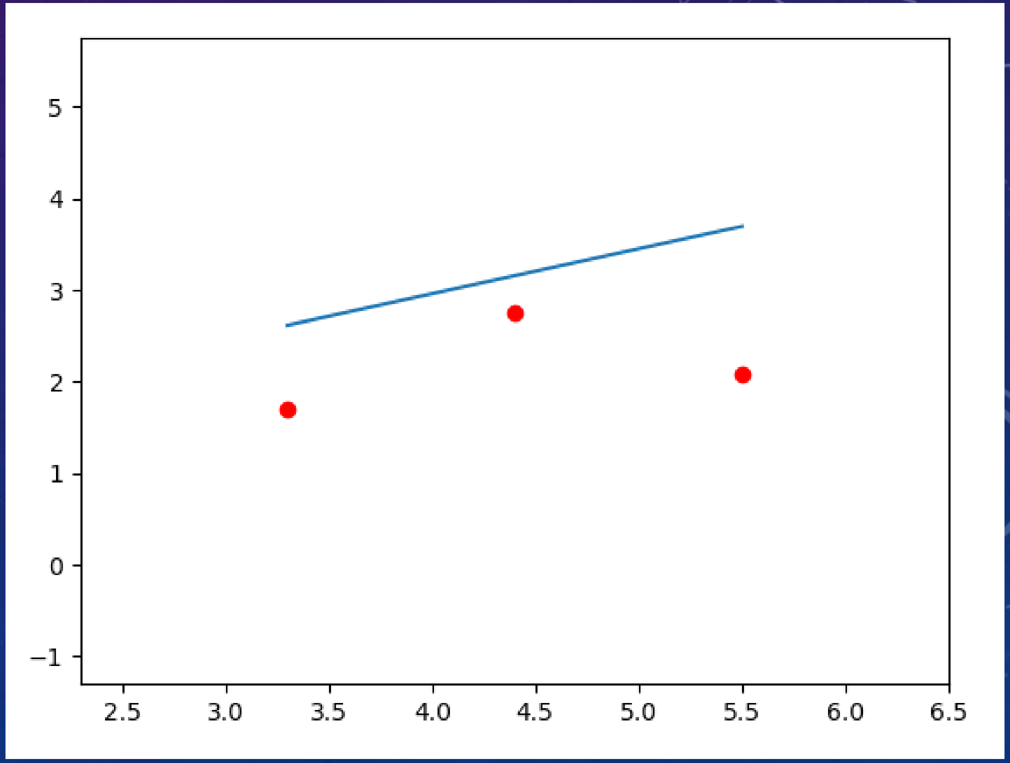
- Visualize the linear regression
- Set initial linear function (In sample code: a = 0.5, b = 1)

$$y = ax + b$$

- Execute the code, we can see:

Red dot : training set
Blue line : linear function

```python
# Hyper-parameters
num_epochs = 1
learning_rate = 0.001
# Set initialize parameter y = ax + b
a = 0.5
b = 1
```



Example 1.1 Linear Regression

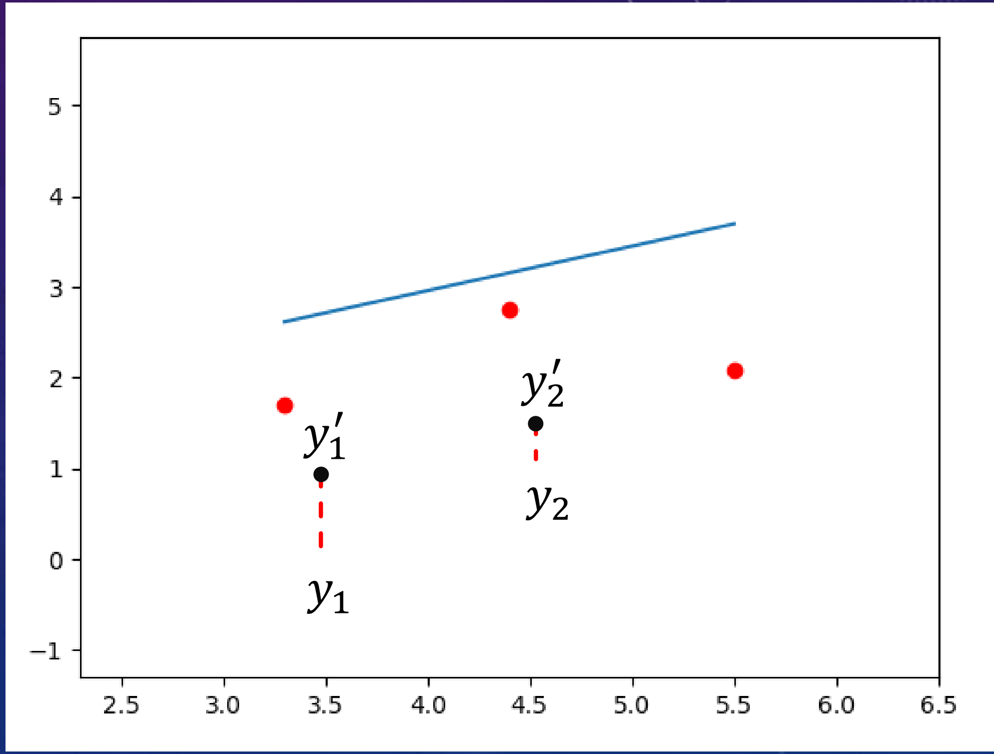
- Loss function: Mean Square Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2$$

- Code in PyTorch:

```
criterion = torch.nn.MSELoss(reduction='mean')
```

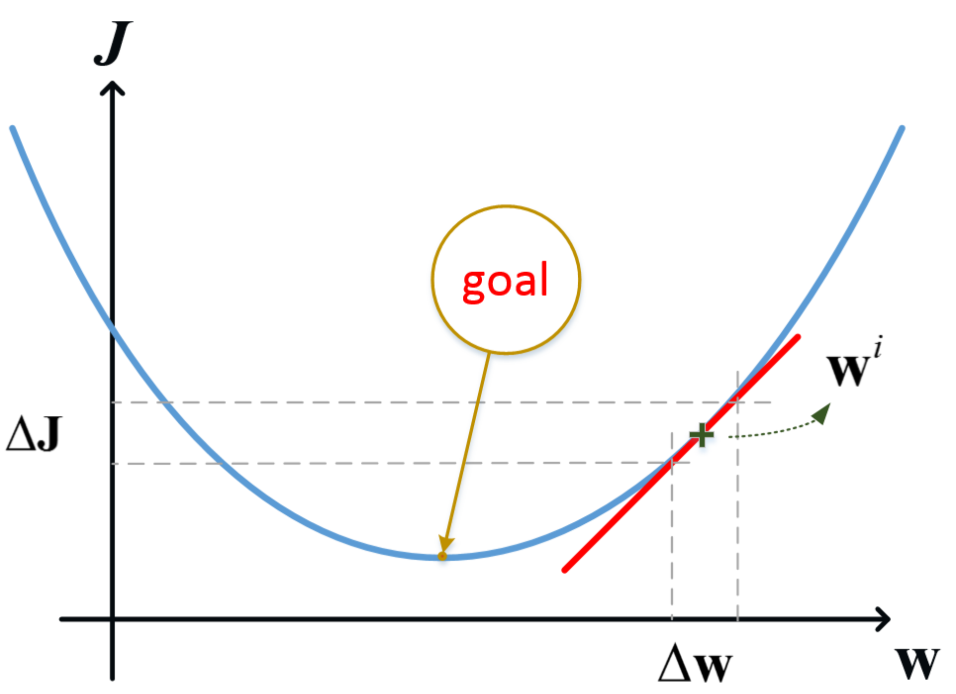
```
Initial Function: y = 0.50x + 1.00
Epoch [1/50], Loss: 1.2839, New_a:0.50, New_b:1.00
Epoch [2/50], Loss: 1.1921, New_a:0.49, New_b:1.00
Epoch [3/50], Loss: 1.1079, New_a:0.48, New_b:1.00
Epoch [4/50], Loss: 1.0307, New_a:0.47, New_b:0.99
Epoch [5/50], Loss: 0.9599, New_a:0.46, New_b:0.99
Epoch [6/50], Loss: 0.8949, New_a:0.46, New_b:0.99
```



How the parameter be updated will talk at later lecture

Example 1.1 Linear Regression

- Gradient descent: update **w** with the data (*t*, *y*).
  - ➢ The update rule :



$$\mathbf{w}^{i+1} = \mathbf{w}^{i} - \alpha\left(\frac{\partial \mathbf{J}}{\partial \mathbf{w}^{i}}\right)$$

where α is the learning rate

$$J = (\vec{t} - \vec{y})^{T}(\vec{t} - \vec{y})$$

$$\Rightarrow \frac{\partial J}{\partial \mathbf{w}} = -(\vec{t} - \vec{y})^{T}\frac{\partial y}{\partial \mathbf{w}}$$

Example 1.1 Linear Regression

Gradient descent for this case:

- $E: \frac{1}{n}\sum_{i=1}^{n}(T_i - Y_i)^2$
- $Y(Predict) = 0.5x+1$, T:target
- Lr=0.001
- Δ:gradient
- $a_1 = a_0 - \Delta_a * lr, b_1 = b_0 - \Delta_b * lr$

$$\Delta_a = \frac{\partial E}{\partial \mathbf{a}} = \frac{1}{3}\sum_{i=1}^{3} 2(T_i - Y_i) * X_i = \frac{6.27 + 3.872 + 18.26}{3} = 9.4673$$

$$\Delta_b = \frac{\partial E}{\partial \boldsymbol{b}} = \frac{1}{3}\sum_{i=1}^{3} 2(T_i - Y_i) = \frac{1.9 + 0.88 + 3.32}{3} = 2.0333$$

```
Target:[[1.7 ]
 [2.76]
 [2.09]]
Predict:tensor([[2.6500],
        [3.2000],
        [3.7500]], grad_fn=<AddmmBackward>)
Epoch [1/50], Loss: 1.2839, New_a:0.50, New_b:1.00, a_gradient:None, b_gradient:None
Target:[[1.7 ]
 [2.76]
 [2.09]]
Predict:tensor([[2.6167],
        [3.1563],
        [3.6959]], grad_fn=<AddmmBackward>)
Epoch [2/50], Loss: 1.1921, New_a:0.49, New_b:1.00, a_gradient:tensor([[9.4673]]), b_gradient:tensor([2.0333])
```

Exercise 1.1 Linear Regression

- Please download 1-1_linear_regression.py and adjust the following parameters:

  Step 1 : Input Data

  x_train= [3.3], [4.4], [5.5],[8.2],[9.4]

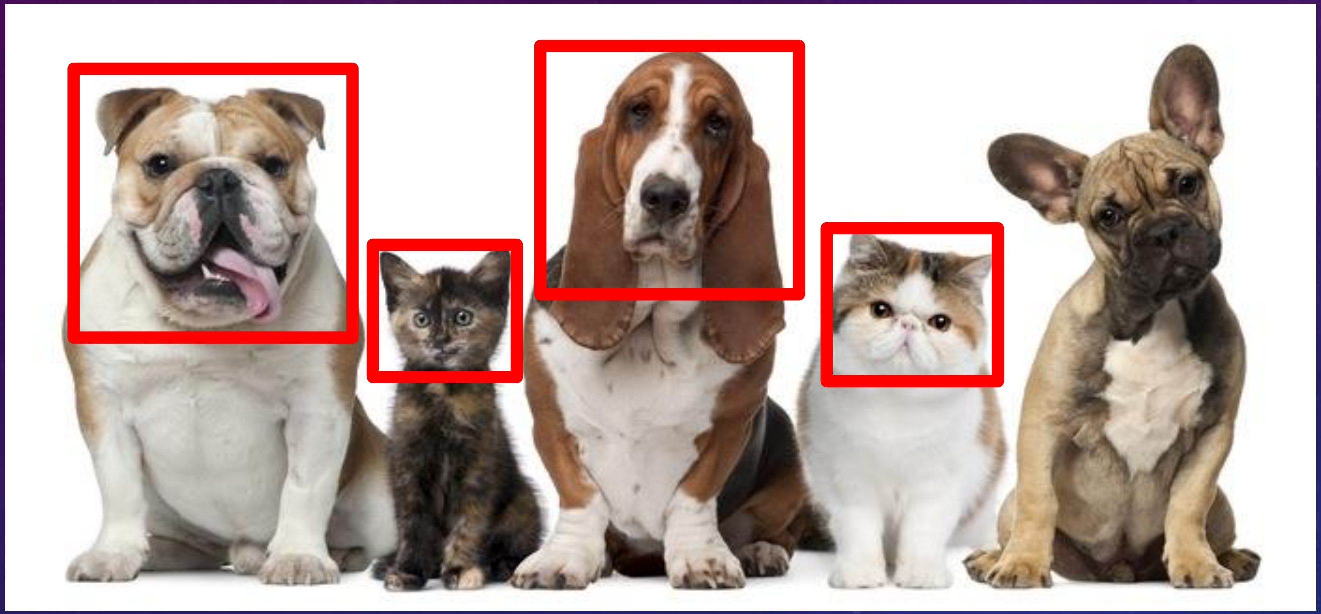
  y_train= [1.7], [2.76], [2.09],[5.48],[4.99]

  Step 2 : Epoch

  Step 3 : Learning Rate

- Please upload your code and your comments in MS Word to Clouds.

Example 1.2 Samples of Dog Detection



TP = 2 FP = 2 FN = 1

Three dogs in the image.

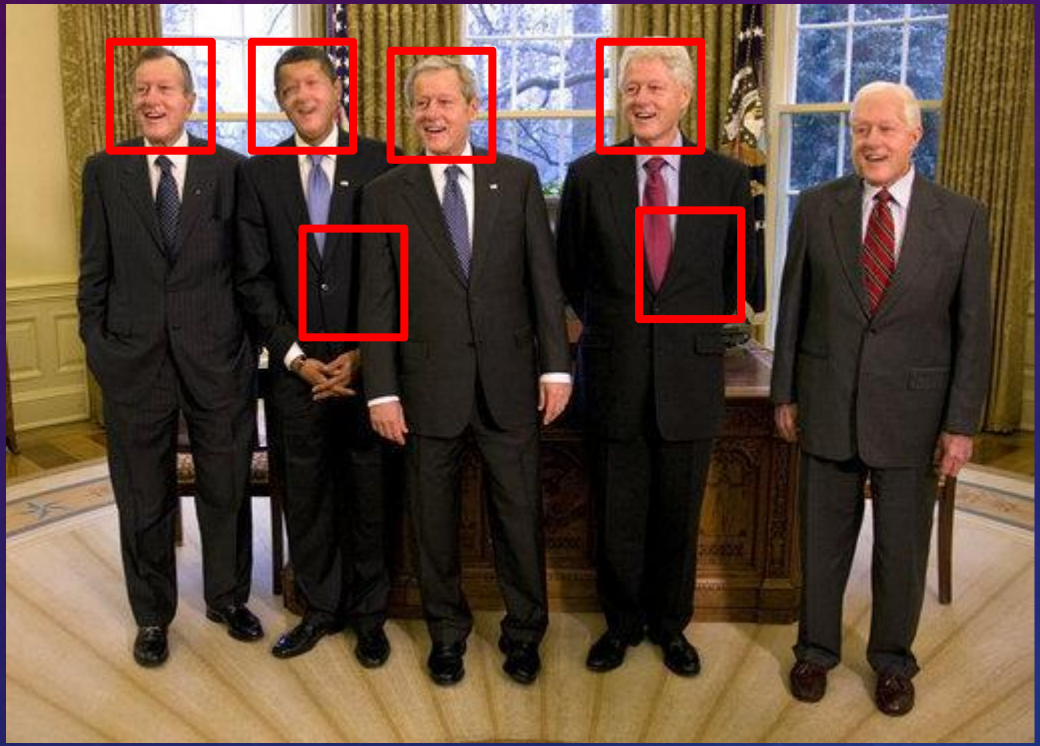
$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{2}{2+2} = 0.5$$

$$\text{Recall} = \frac{2}{2+1} = 0.666$$

Example 1.2 Samples of Face Detection



TP = 4 FP = 2 FN = 1

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

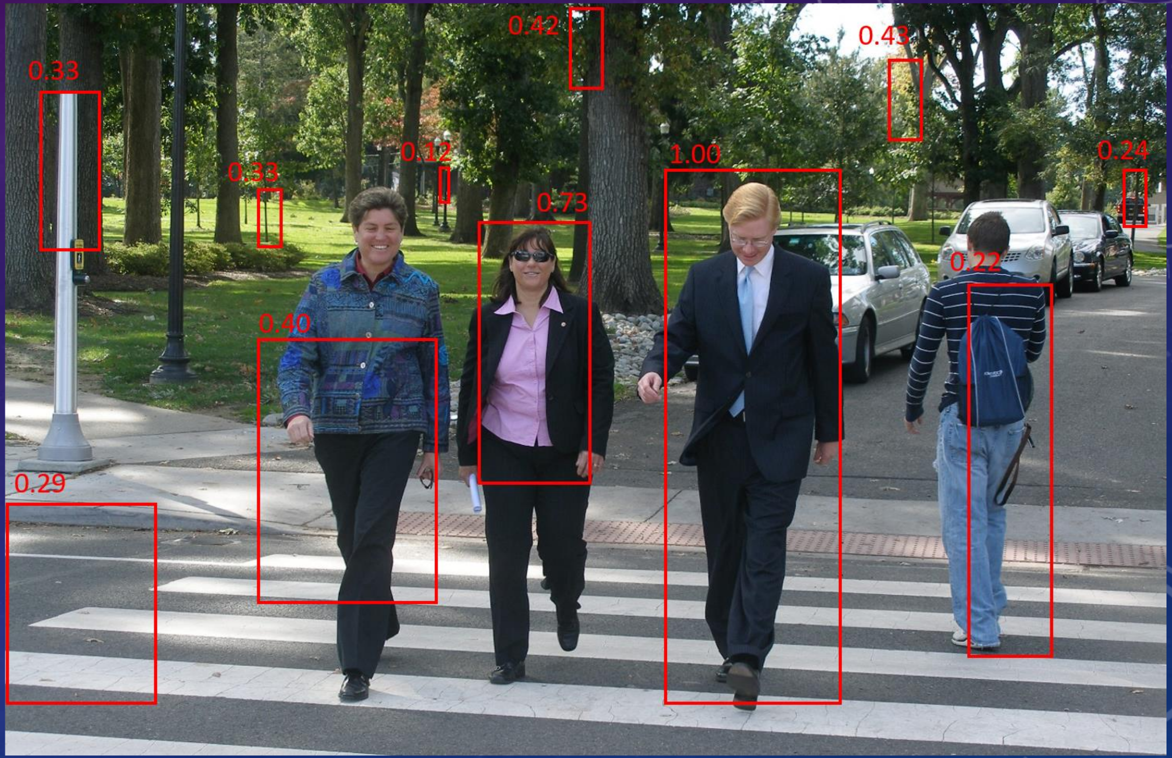
$$\text{Precision} = \frac{4}{4+2} = 0.667$$

$$\text{Recall} = \frac{4}{4+1} = 0.8$$

Exercise 1.2 Confusion Matrix

Consider using a pedestrian detector to detect the image, please check out the detection boxes with the confident scores below and compute the "Precision and Recall rate" with the following thresholds:

- Threshold=0.2
- Threshold=0.4
- Threshold=0.7
- Threshold=1.0



Draw the Precision-Recall curve using different probability thresholds mentioned above, and upload your code and your comments in MS Word to Moodle.

Hint-1：If the confident score is greater or equal to the threshold, the pedestrian is correctly detected (True Positive).

Hint-2：A precision-recall curve is a plot of the **precision** (y-axis) and the **recall** (x-axis) for different thresholds.