

**電腦視覺與深度學習
(Computer Vision and Deep Learning)
Homework 2**

TA:

Lydia: lydia2200284@gmail.com

Office Hour: 17:00~19:00, Mon.

10:00~12:00, Wed.

At CSIE 9F Robotics Lab.

Notices (1/2)

- ❑ Copying homework is strictly prohibited!! **Penalty: Grade will be zero for both persons!!**
- ❑ If the code can't run, you can come to our Lab within one week and show that your programming can work. Otherwise you will get zero!!
- ❑ Due date => **Midnight 23:59:59 on 2020/12/30 (Wed.)**
 - No delay. If you submit homework after deadline, you will get 0.
- ❑ Upload to => **140.116.154.1 -> Upload/Homework/Hw2**
 - User ID: cvdl2020 Password: cvdl2020
- ❑ Format
 - Filename: Hw2_StudentID_Name_Version.rar
 - Ex: Hw2_F71234567_林小明_V1.rar
 - If you want to update your file, you should update your version to be V2, ex: Hw2_F71234567_林小明_V2.rar
 - Content: **project folder***(including the pictures)
*note: remove your “Debug” folder to reduce file size

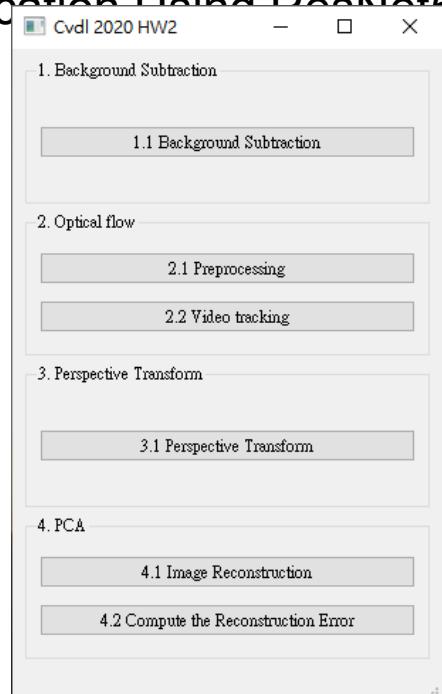
Notices (2/2)

- ❑ Python
 - Python 3.7 (<https://www.python.org/downloads/>)
 - opencv-contrib-python (3.4.2.17)
 - Matplotlib 3.1.1
 - UI framework: pyqt5 (5.15.1)

- ❑ C++ (check MFC guide in ftp)
 - OpenCV 3.3.1 (<https://opencv.org/release.html>)
 - Visual Studio 2015 (download from <http://www.cc.ncku.edu.tw/download/>)
 - UI framework: MFC

Grading

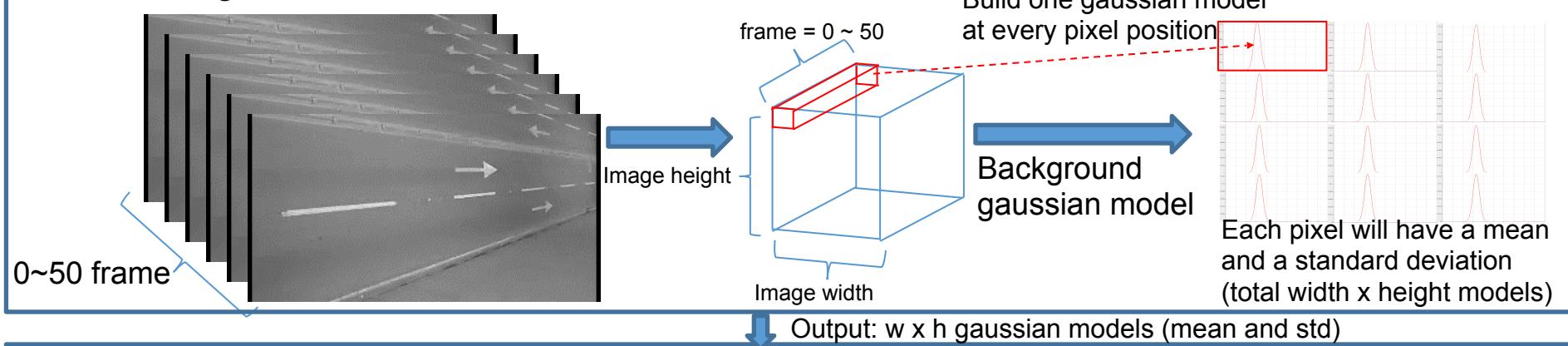
1. (20%) Background Subtraction (出題 : Brian)
2. (20%) Optical Flow (出題 : Kevin)
 - 2.1 Preprocessing (10%)
 - 2.2 Video tracking (10%)
3. (20%) Perspective Transform (出題 : Max)
4. (20%) PCA (出題 : Mark)
 - 4.1 Image Reconstruction (10%)
 - 4.2 Compute the reconstruction error (10%)
5. (20%) Dogs and Cats classification Using DeepNet50 (出題 : Bill)
UI example



1. (20%) Background Subtraction: Simple Gaussian Model (1/2) (出題 : Brian)

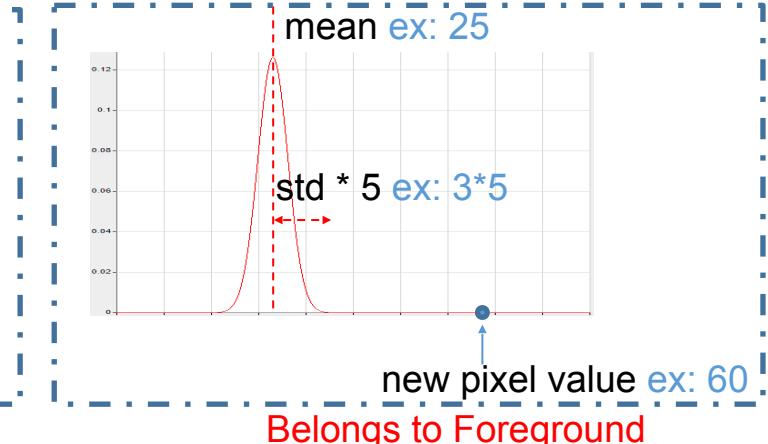
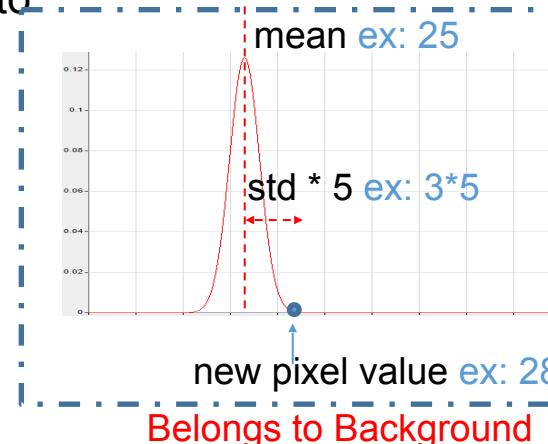
- Given a video: bgSub.mp4
- Q: 1) Click the button “1.1 Background Subtraction” to open two windows:
 - One shows the original video: bgSub.mp4
 - The other is the **foreground video**.
 - Use first **50 frames** to build the background model
 - **DO NOT** use OpenCV function `cv2.createbackgroundSubtractor()`

1. Build background model



2. Test new pixel belongs to background or foreground

Ex:
Mean = 25
Std (standard deviation) = 3



□ Hint : (1) Convert video frames to **gray**

- (2) For every pixels in video from 0~50 frames, build a **gaussian model** with **mean** and **standard deviation** (if standard deviation is less then 5, set to 5)
- (3) For frame > 50, test every frame pixels with respective gaussian model. If **gray value difference** between testing pixel and gaussian mean is larger than **5 times standard deviation**, set testing pixel to 255 (foreground, white), 0 (background, black) otherwise.

□ Demo video:



2. (20%) Optical Flow

(出題 : Kevin)

2.1 Preprocessing (10%)

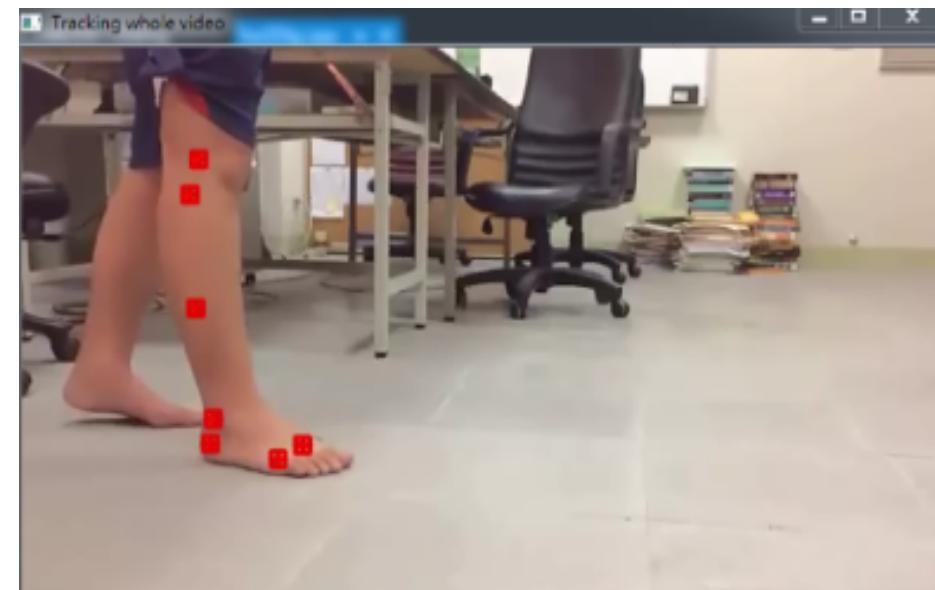
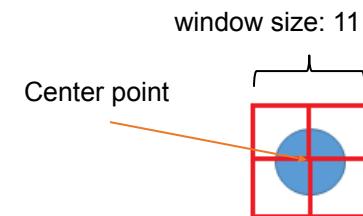
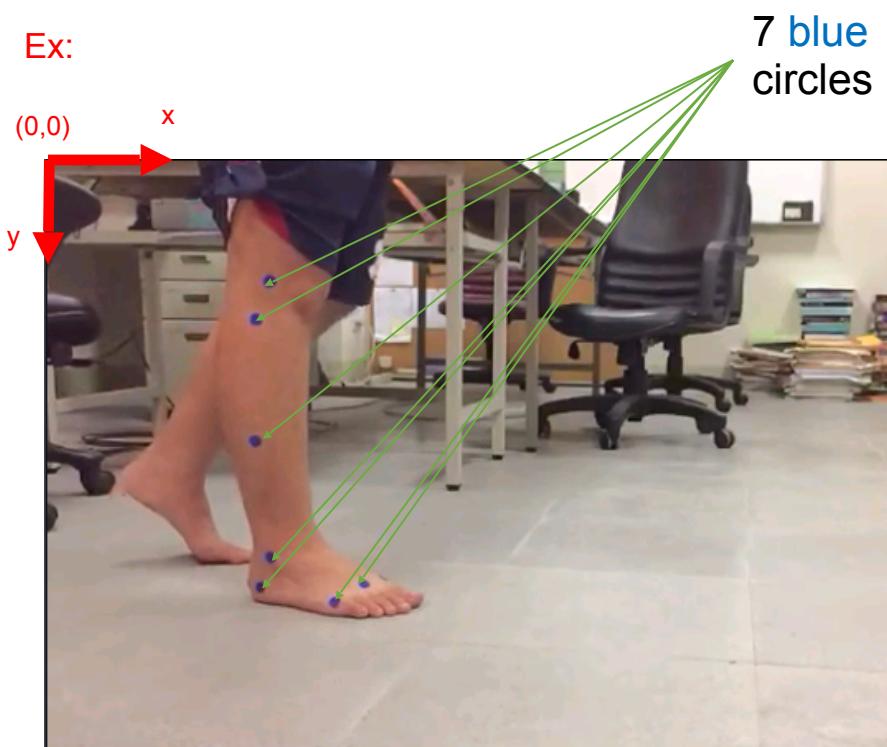
2.2 Video tracking (10%)

2.1 Preprocessing (10%)

(出題 : Kevin)

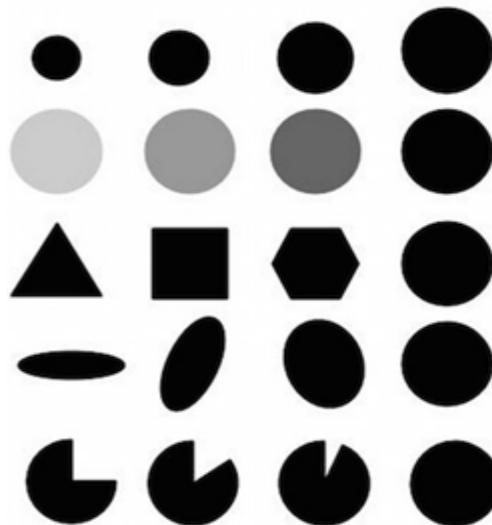
- Given a video: opticalFlow.mp4
- Q: Click the button to show the **square boundary with red cross mark** (not **red filled** like the ex. image).

Ex:

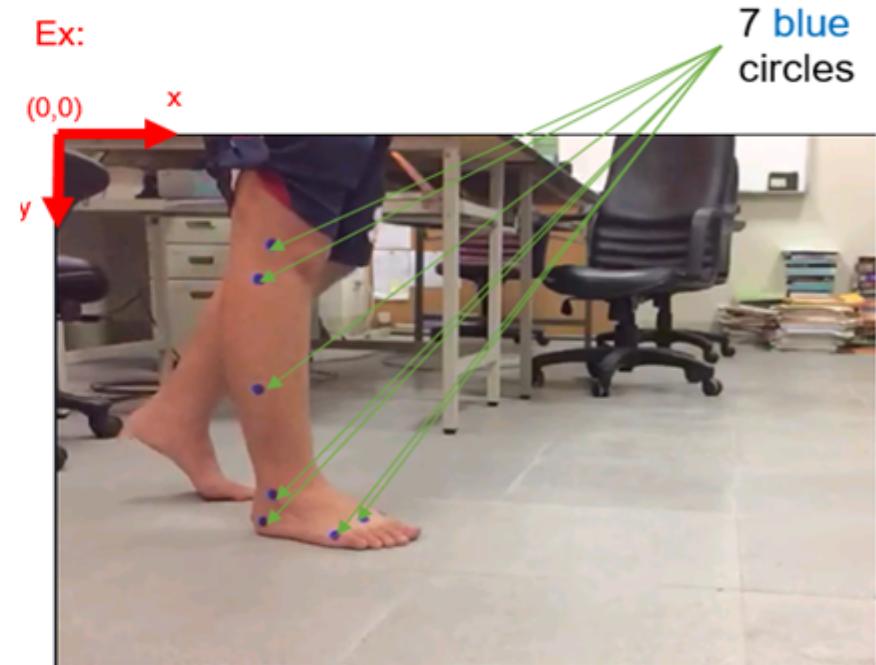


□ How to detect 7 center points of 7 blue circles

- You can use `cv2.SimpleBlobDetector()` to detect blobs (binary large object), it can filter out blobs by examining their:
 - 1) Area (size)
 - 2) Thresholds (gray level)
 - 3) Circularity (if perfect circle, then it is 1.0)
 - 4) Inertia (**ratio of the minor and major axes**)
 - 5) Convexity (if no gap, then it is 1.0)



Area
Thresholds
Circularity
Inertia
Convexity



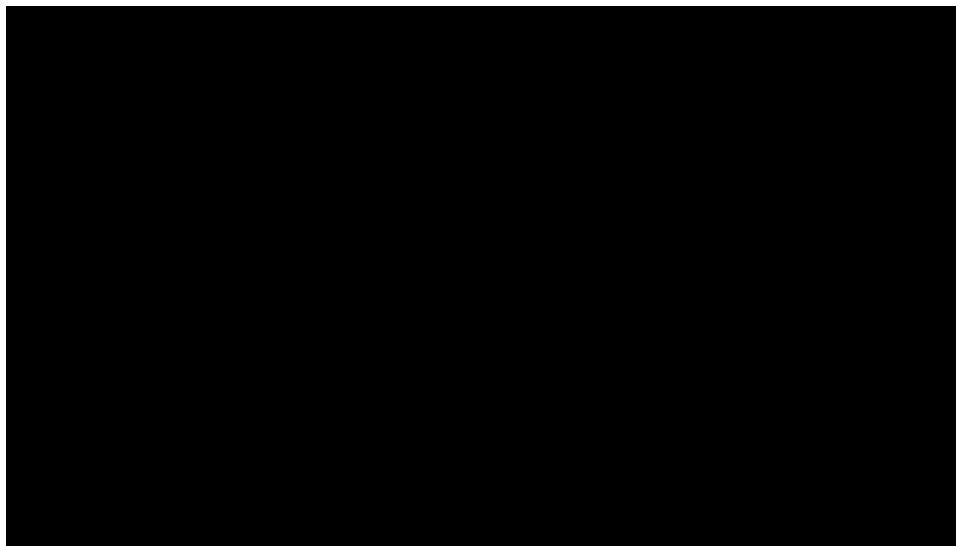
2.2 Video tracking (10%)

(出題 : Kevin)

□ Q: Click button to:

- 1) (5%) Track the 7 center points on the whole video using OpenCv function [cv2.calcOpticalFlowPyrLK](#).
Ex: <https://github.com/opencv/opencv/blob/master/samples/cpp/lkdemo.cpp>
- 2) (5%) Connect 7 corresponding tracking points of all frames (the whole video) to get 7 trajectories (flow). Ex: the demo video.

Demo video:



□ Tool site:

1. [Load video](#)
2. [Circle detect](#)
3. [Optical flow](#)

* Tracking process may be fail, you should make the result as good as possible (at least 4~5 points success)

3. (20%) Perspective Transform:

(出題 : Max)

- Given a video: test4perspective.mp4
- Q: 1) Click the button “ 3.1 Perspective Transform” to show and save result video:
- Hint :
 - (1) ~~Read the test video and process frame by frame~~ ----- Next page will show you how to detect ArUco Marker
 - (2) Detect the ArUco maker inside the frames
 - (3) Get the pixel position (x,y) of four makers in the frame
 - (4) ~~Do perspective transform~~ ----- Please do perspective transform by you own with opencv
 - (5) Show result video and save it
- Demo video:



❑ Hints: How to use ArUco Makers (1/2)

(出題 : Max)

- Step 1: You need to detect the aruco makers inside each frame as below

```
while cv.waitKey(1) < 0:  
    try:  
        # get frame from the video  
        hasFrame, frame = cap.read()  
  
        # Load the dictionary that was used to generate the markers.  
        dictionary = cv.aruco.Dictionary_get(cv.aruco.DICT_6X6_250)  
  
        # Initialize the detector parameters using default values  
        parameters = cv.aruco.DetectorParameters_create()  
  
        # Detect the markers in the image  
        markerCorners, markerIds, rejectedCandidates = cv.aruco.detectMarkers(  
            frame, dictionary, parameters)
```

- ↓
Step 2: Then you can get the pixel value of each makers, for example
you can get pixel value of makers which's ID is 25

```
index = np.squeeze(np.where(markerIds == 25))  
refPt1 = np.squeeze(markerCorners[index[0]])[1]
```

□ Hints: How to use ArUco Makers (2/2)

(出題 : Max)

Step 3: Then calculate the four edges of the region you want to perspective,

pts_dst are four points

```
index = np.squeeze(np.where(markerIds == 25))
refPt1 = np.squeeze(markerCorners[index[0]])[1]

index = np.squeeze(np.where(markerIds == 33))
refPt2 = np.squeeze(markerCorners[index[0]])[2]

distance = np.linalg.norm(refPt1-refPt2)
```

Find makers 25,33



```
scalingFac = 0.02
pts_dst = [
    [refPt1[0] - round(scalingFac*distance), refPt1[1] - round(scalingFac*distance)],
    [refPt2[0] + round(scalingFac*distance),
     refPt2[1] - round(scalingFac*distance)]]
```

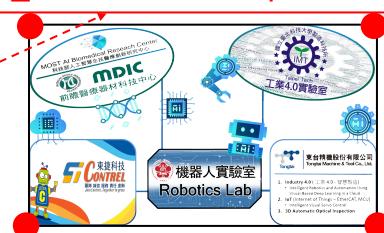
Calculate the edge points for perspective

```
index = np.squeeze(np.where(markerIds == 30))
refPt3 = np.squeeze(markerCorners[index[0]])[0]
pts_dst = pts_dst + [
    [refPt3[0] + round(scalingFac*distance),
     refPt3[1] + round(scalingFac*distance)]]
```

```
index = np.squeeze(np.where(markerIds == 23))
refPt4 = np.squeeze(markerCorners[index[0]])[0]
pts_dst = pts_dst + [
    [refPt4[0] - round(scalingFac*distance),
     refPt4[1] + round(scalingFac*distance)]]
```

```
pts_src = [[0, 0], [im_src.shape[1], 0],
           [im_src.shape[1], im_src.shape[0]], [0, im_src.shape[0]]]
```

pts_src are also four points



3. (20%) Perspective Transform:

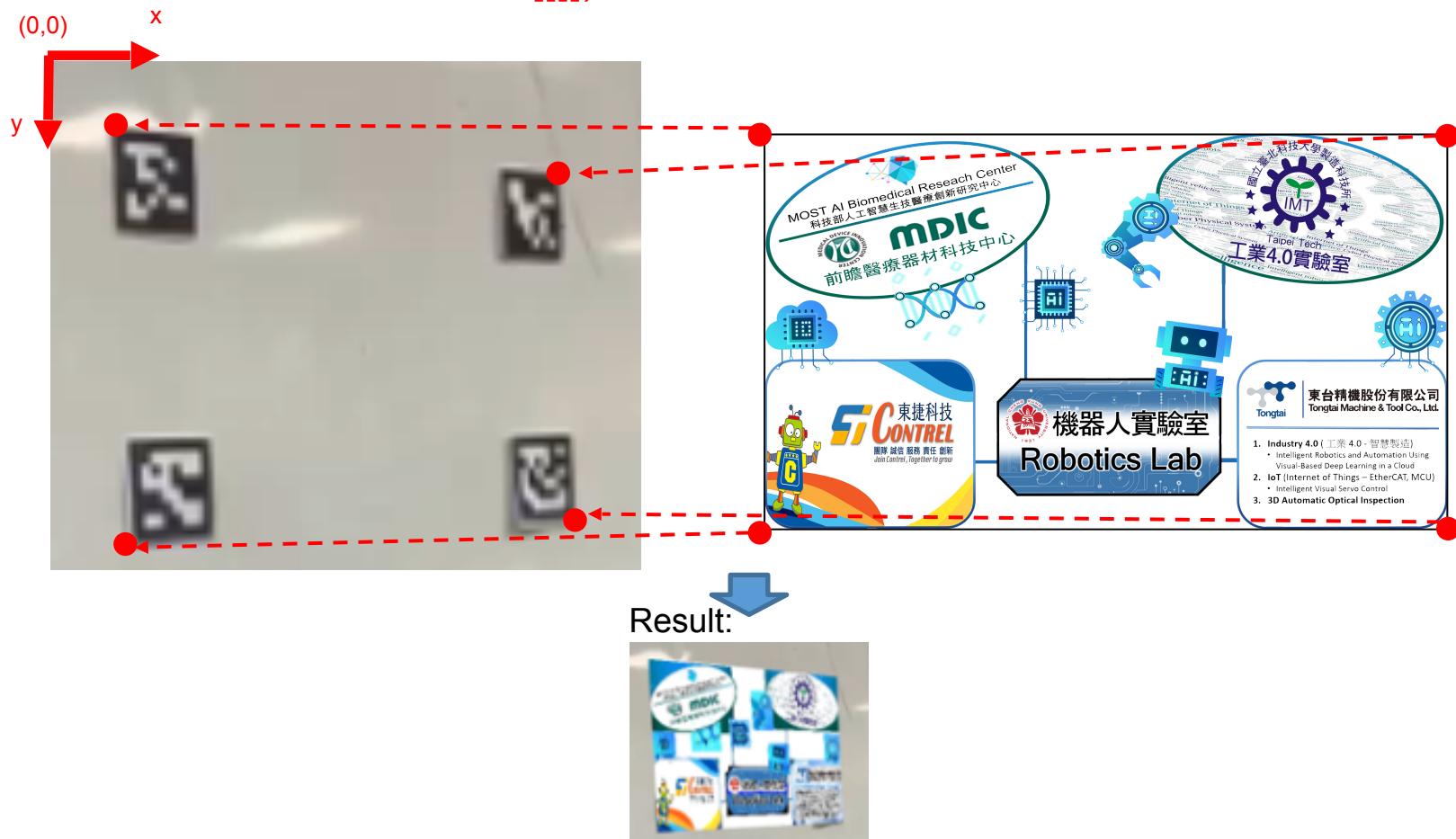
(出題 : Max)

- Do the perspective transform

- The function you will use for perspective transform

`retval, mask=cv2.findHomography(srcPoints, dstPoints[, method[, ransacReprojThreshold[, mask[, maxIters[, confidence]]]]])`

`dst = cv2.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])`



4. (20%) PCA

(出題 : Mark)

4.1 Image Reconstruction (10%)

4.2 Compute the reconstruction error (10%)

4.1 (10%) PCA - Image Reconstruction (1/2) (出題 : Mark)

- Given: Images of school badge
- Q1: Using **PCA** to do dimension reduction then reconstruct, click button “4.1 Image Reconstruction” and show original and reconstructed images.
- Hint: Use PCA from python library `sklearn.decomposition`



4.2 (10%) PCA - Compute the Reconstruction Error (2/2)

(出題 : Mark)

- Given: Images of school badge
- Q2: Computing the **reconstruction error (RE)** for each school badge, and click button “4.2 Compute the reconstruction error” and print out on the console.
- EX:

[493174, 482253, 524416, 487007, 502523, 942532, 521524, 433513, 758603, 620116, 652831, 574274, 592094, 878298, 537152, 558690, 490437, 828428, 910308, 666765, 66006]

Using the reference formula as shown below:

$$RE = \sum_{i=1}^{n_{pixels}} \left\| \vec{x}_i - \vec{y}_i \right\|, \text{ where}$$

\vec{x}_i is the gray value of pixel in original image

\vec{y}_i is the gray value of pixel in reconstruction image

- Hint:

\vec{y}_i is the modified result after the reconstruction

(which is

$T^{-1}T(\vec{x}_i)$, where T is the dimension reduction transformation,)

You should do post processing such as normalization to make the gray value fall in the range [0, 255]

5.0 (20%) Dogs and Cats classification Using ResNet50 (出題 : Bill)

1) Dataset introduction:

- (1) ASIRRA (Animal Species Image Recognition for Restricting Access) is a HIP(Human Interactive Proof) that works by asking users to identify photographs of cats and dogs, that's supposed to be easy for people to solve, but difficult for computers. Now we can use artificial intelligence technology to achieve this goal.
- (2) The dataset includes 12501 photos of cats and 12501 photos of dogs. You need to download them in Reference below(R2), and split the training set, validation set and test set by yourself.

2) Objective:

- You need to use python to write the ResNet network and complete the questions on the next few pages.

3) Environment Requirement

- (1) Python 3.6
- (2) Tensorflow 1.14
- (3) Opencv-contrib-python 4.1.1
(for image show and write)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2 3×3 max pool, stride 2		
conv2_x	56×56	$\left[\begin{matrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{matrix} \right] \times 2$	$\left[\begin{matrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{matrix} \right] \times 3$	$\left[\begin{matrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{matrix} \right] \times 3$	$\left[\begin{matrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{matrix} \right] \times 3$	$\left[\begin{matrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{matrix} \right] \times 3$
conv3_x	28×28	$\left[\begin{matrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{matrix} \right] \times 2$	$\left[\begin{matrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{matrix} \right] \times 4$	$\left[\begin{matrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{matrix} \right] \times 4$	$\left[\begin{matrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{matrix} \right] \times 4$	$\left[\begin{matrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{matrix} \right] \times 8$
conv4_x	14×14	$\left[\begin{matrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{matrix} \right] \times 2$	$\left[\begin{matrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{matrix} \right] \times 6$	$\left[\begin{matrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{matrix} \right] \times 6$	$\left[\begin{matrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{matrix} \right] \times 23$	$\left[\begin{matrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{matrix} \right] \times 36$
conv5_x	7×7	$\left[\begin{matrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{matrix} \right] \times 2$	$\left[\begin{matrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{matrix} \right] \times 3$	$\left[\begin{matrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{matrix} \right] \times 3$	$\left[\begin{matrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{matrix} \right] \times 3$	$\left[\begin{matrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{matrix} \right] \times 3$
	1×1			average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig. ResNet's Network Architecture

R. Reference

[R1\) Deep Residual Learning for Image Recognition](#)

[R2\) https://www.microsoft.com/en-us/download/details.aspx?id=54765](https://www.microsoft.com/en-us/download/details.aspx?id=54765) (ASIRRA)

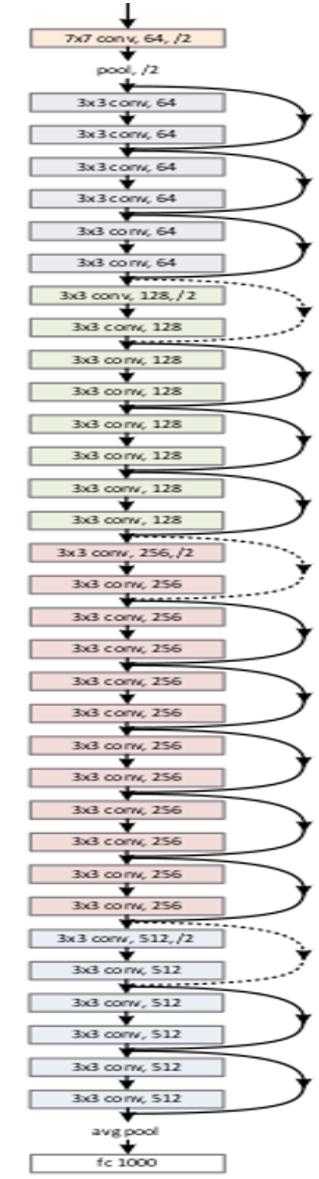


Fig. ResNet50's Schematic Diagram

- 5.1 Training by your computer **at least 5 epochs** and show(or upload) your source code of ResNet. (5%)

```
train loss = 0.67, train accuracy = 62.50%
train loss = 0.67, train accuracy = 62.50%
train loss = 0.70, train accuracy = 50.00%
train loss = 0.62, train accuracy = 87.50%
```

Fig. Accuracy display during training

- 5.2 Training at home and use TensorBoard to monitor, Save the final **screenshot of TensorBoard** (5%, Use other tools can get 3%).

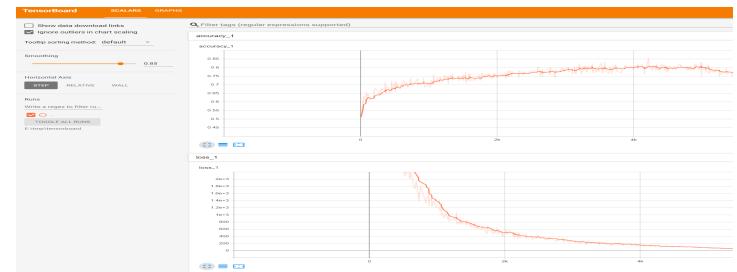


Fig. Example of training with TensorBoard

- 5.3 Randomly select a picture from the test set and mark its category.(5%)

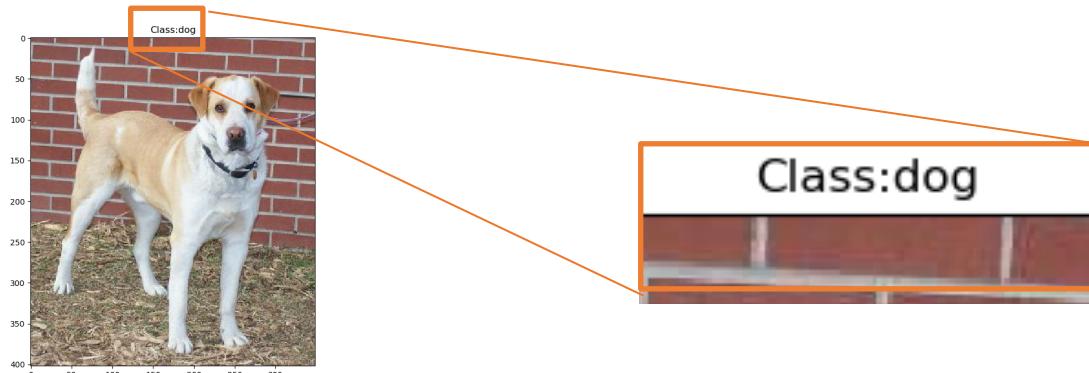


Fig. Classification display example

5.4 Improve ResNet50 with the method in yolov4:

Due to the early appearance of ResNet50, many augmentation methods have not implemented. In the latest ResNet50, Random-Erasing have been used to greatly improve the accuracy, so why don't we use this method on ResNet50? **Write the code of Random-Erasing (3%) and draw the comparison table of accuracy, save it as a picture.(2%)**

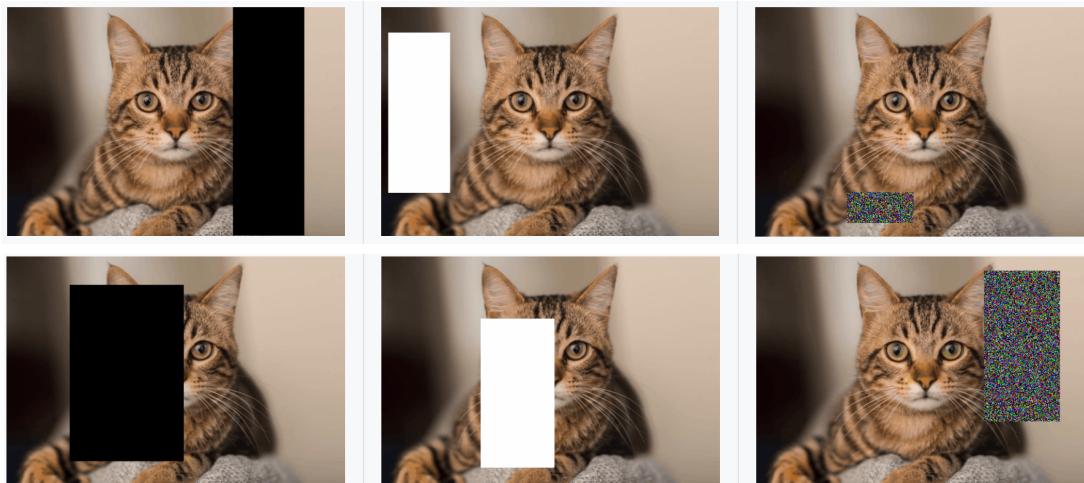


Fig. Examples of the use of Random-Erasing

R. Reference

[Random Erasing Data Augmentation](#)

Algorithm 1: Random Erasing Procedure

```
Input : Input image  $I$ ;  
       Image size  $W$  and  $H$ ;  
       Area of image  $S$ ;  
       Erasing probability  $p$ ;  
       Erasing area ratio range  $s_l$  and  $s_h$ ;  
       Erasing aspect ratio range  $r_1$  and  $r_2$ .  
Output: Erased image  $I^*$ .  
Initialization:  $p_1 \leftarrow \text{Rand}(0, 1)$ .  
1 if  $p_1 \geq p$  then  
2    $I^* \leftarrow I$ ;  
3   return  $I^*$ .  
4 else  
5   while True do  
6      $S_e \leftarrow \text{Rand}(s_l, s_h) \times S$ ;  
7      $r_e \leftarrow \text{Rand}(r_1, r_2)$ ;  
8      $H_e \leftarrow \sqrt{S_e \times r_e}$ ,  $W_e \leftarrow \sqrt{\frac{S_e}{r_e}}$ ;  
9      $x_e \leftarrow \text{Rand}(0, W)$ ,  $y_e \leftarrow \text{Rand}(0, H)$ ;  
10    if  $x_e + W_e \leq W$  and  $y_e + H_e \leq H$  then  
11       $I_e \leftarrow (x_e, y_e, x_e + W_e, y_e + H_e)$ ;  
12       $I(I_e) \leftarrow \text{Rand}(0, 255)$ ;  
13       $I^* \leftarrow I$ ;  
14      return  $I^*$ .  
15    end  
16  end  
17 end
```

Fig. Random-Erasing algorithm

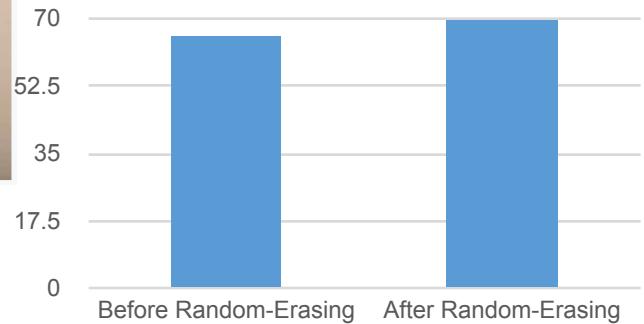


Fig. Random-Erasing effect comparison example