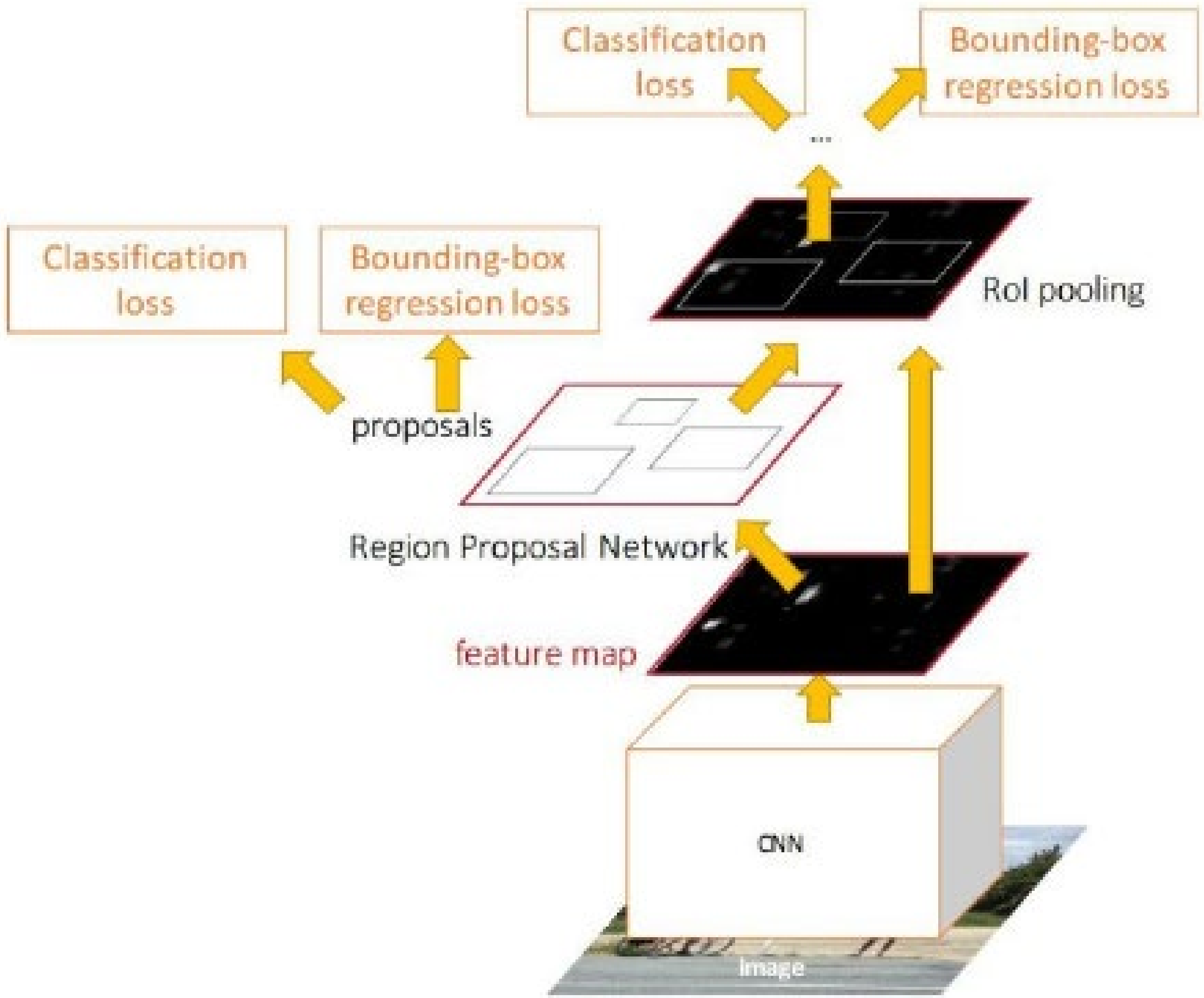
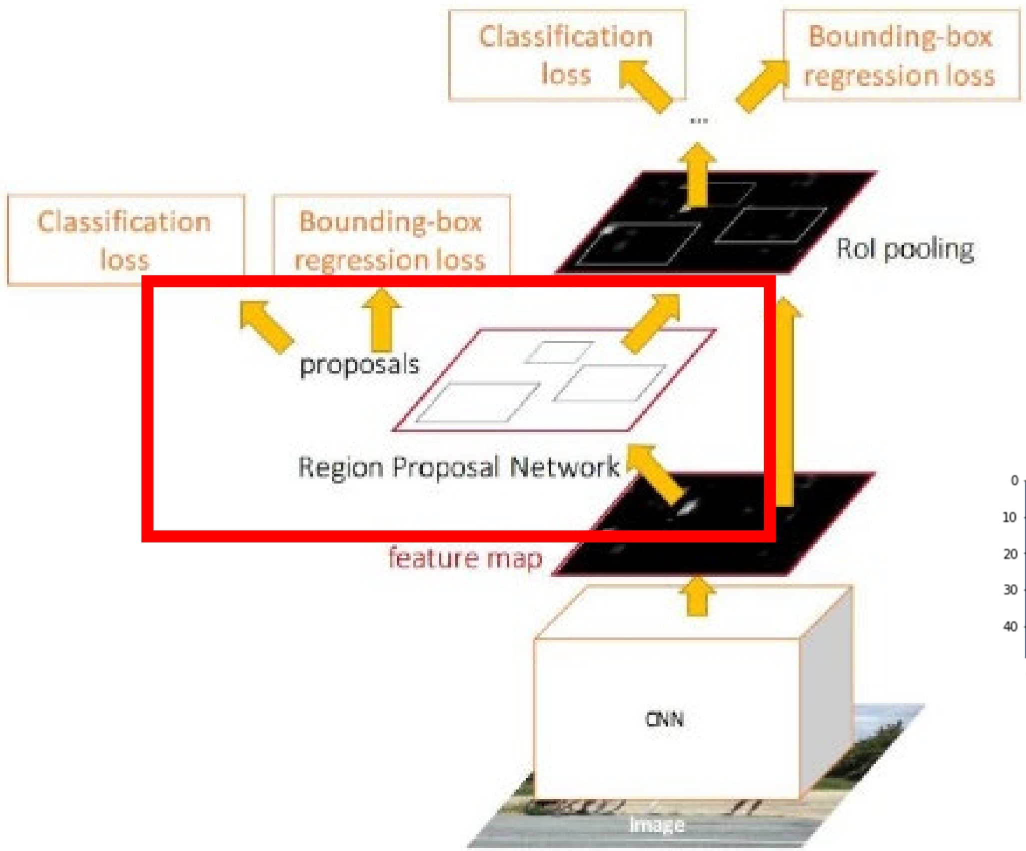
Recap: FasterRCNN

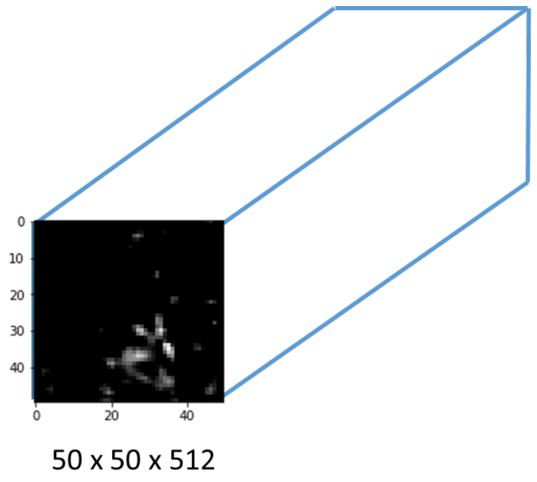


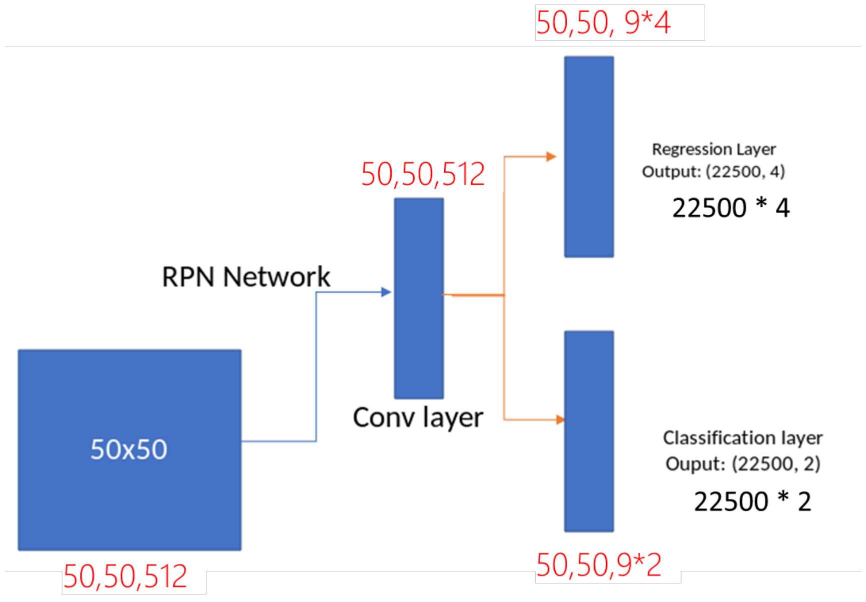
https://kharshit.github.io/blog/2019/08/23/quick-intro-to-instance-segmentation

RPN (region proposal network)

Takes feature map as input and predict 22,500 ROIs (region of interests)

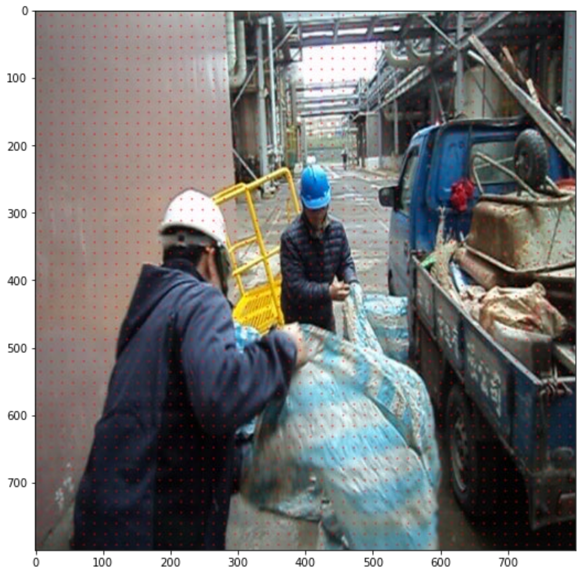




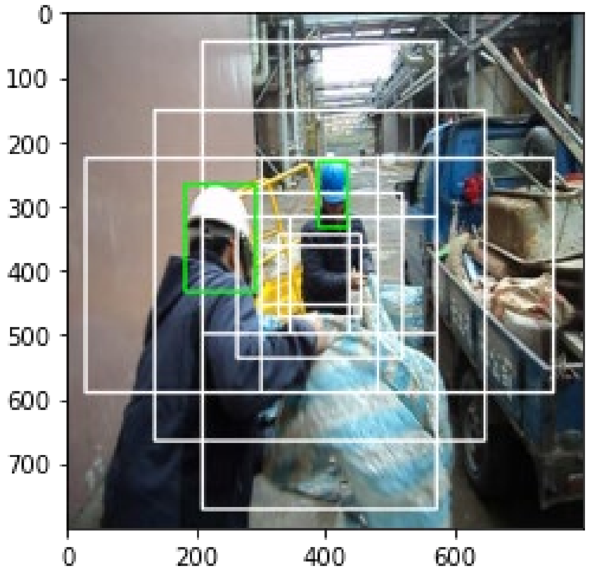


Ground-truth labels to calculate RPN classification loss

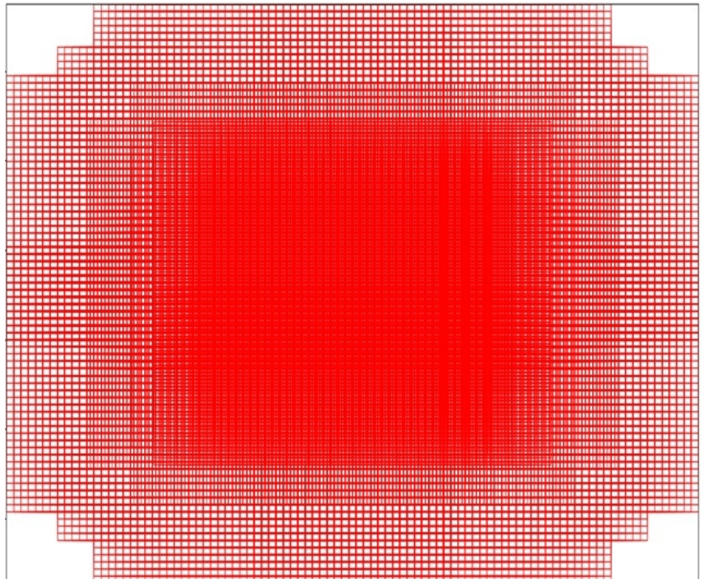
16*16 anchor points



9 anchor boxes for each anchor pt



16*16*9 = 22,500
8940 valid anchor boxes



Label the 8,940 valid anchor boxes

1: IOU > 0.7 (may contain object)
0: IOU < 0.3 (background)
-1: ignore

Sample a batch of anchor boxes to train RPN: 128 positive examples and 128 negative examples (label 0).
Change the labels of all other valid anchor boxes to -1(ignore) at this mini-batch training.

RPN classification loss

$$L(p_i, t_i) = \boxed{\frac{1}{N_{cls}} \sum_i L_{cls}(p_i, \hat{p}_i)} + \lambda \frac{1}{N_{reg}} \sum_i p_i L_{reg}(t_i, \hat{t}_i)$$

$N_{cls} = 256$

$\hat{p}_i$ Probability for class 0 and 1 predicted by RPN

$$p_i = \begin{cases} 0, negative\ label \\ 1, positive\ label \end{cases}$$

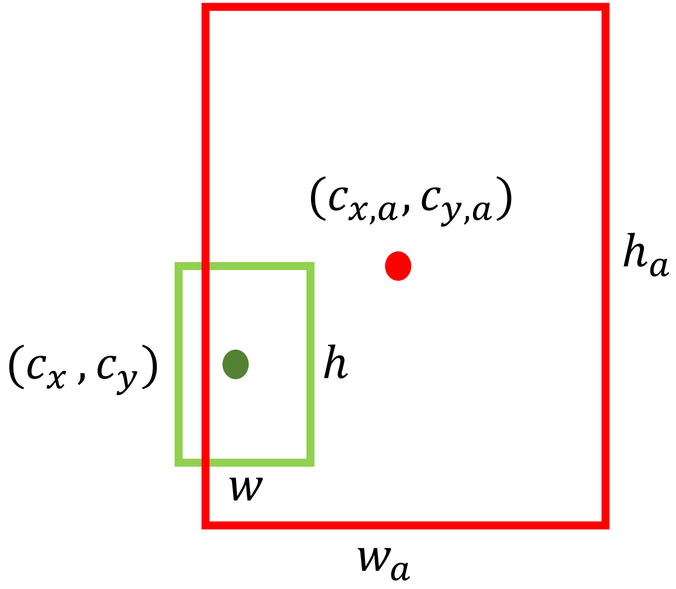
$$L_{cls}(p_i, \hat{p}_i) = CE(p_i, \hat{p}_i) = -\sum_{k=1}^{2} p_k ln(\hat{p}_k)$$

```
# For classification we use cross-entropy loss
rpn_cls_loss = F.cross_entropy(rpn_score, gt_rpn_score.
print(rpn_cls_loss)
```

```
rpn_loc = pred_anchor_locs[0]
rpn_score = pred_cls_scores[0]

gt_rpn_loc = torch.from_numpy(anchor_locations)
gt_rpn_score = torch.from_numpy(anchor_labels)
```

Ground-truth values to calculate RPN bounding box regression loss

For each valid anchor box, use the ground truth bbox with maximum IOU to calculate a normalized location representation



$$d_x = \frac{c_x - c_{x,a}}{w_a} \qquad d_y = \frac{c_y - c_{y,a}}{h_a}$$

$$d_w = \log(\frac{w}{w_a}) \qquad d_h = \log(\frac{h}{h_a})$$

Smooth L1 loss

$$Loss_2 = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}^i)^2$$

$$Loss_1 = \frac{1}{N}\sum_{i=1}^{N}\left|y^i - \hat{y}^i\right|$$

$$smooth\ Loss_1 = \begin{cases} 0.5x^2 \times {}^{1}/_{\sigma^2} & if\ |x| < {}^{1}/_{\sigma^2} \\ |x| - 0.5 & otherwise \end{cases}$$

RPN bounding box regression loss

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, \hat{p}_i) + \boxed{\lambda \frac{1}{N_{reg}} \sum_i p_i L_{reg}(t_i, \hat{t}_i)}$$

$$N_{reg} = 128$$

$$t_i = \left[d_x, d_y, d_w, d_h\right]$$

$$\hat{t}_i = \left[\hat{d}_x, \hat{d}_y, \hat{d}_w, \hat{d}_h\right]$$

$$p_i = \begin{cases} 0, negative\ label \\ 1, positive\ label \end{cases}$$

$$L_{reg} = \begin{cases} 0.5(t_i - \hat{t}_i)^2 \times {}^{1}\!/_{\sigma^2} & if\ |t_i - \hat{t}_i| < {}^{1}\!/_{\sigma^2} \\ |t_i - \hat{t}_i| - 0.5 & otherwise \end{cases}$$

$\sigma = 3$ for RPN training

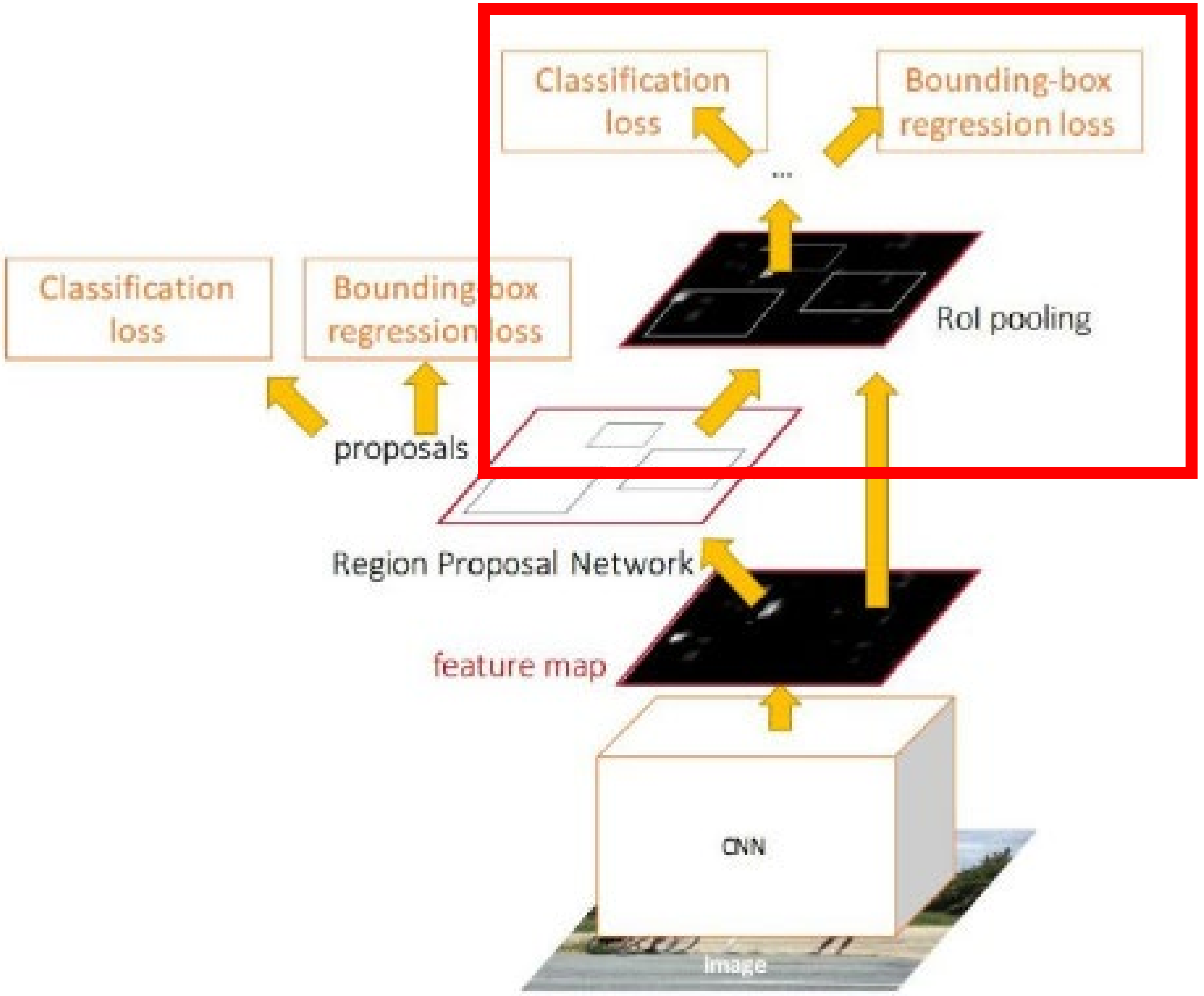
RPN bounding box regression loss

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, \hat{p}_i) + \boxed{\lambda \frac{1}{N_{reg}} \sum_i L_{reg}(t_i, \hat{t}_i)}$$

```
# For Regression we use smooth L1 loss as defined in the Fast RCNN paper
pos = gt_rpn_score > 0
mask = pos.unsqueeze(1).expand_as(rpn_loc)
print(mask.shape)

# take those bounding boxes which have positve labels
mask_loc_preds = rpn_loc[mask].view(-1, 4)
mask_loc_targets = gt_rpn_loc[mask].view(-1, 4)
print(mask_loc_preds.shape, mask_loc_targets.shape)

x = torch.abs(mask_loc_targets.cpu() - mask_loc_preds.cpu())
rpn_loc_loss = ((x < 1).float() * 0.5 * x**2) + ((x >= 1).float() * (x-0.5))
print(rpn_loc_loss.sum())
```

Pass ROIs to FastRCNN detector

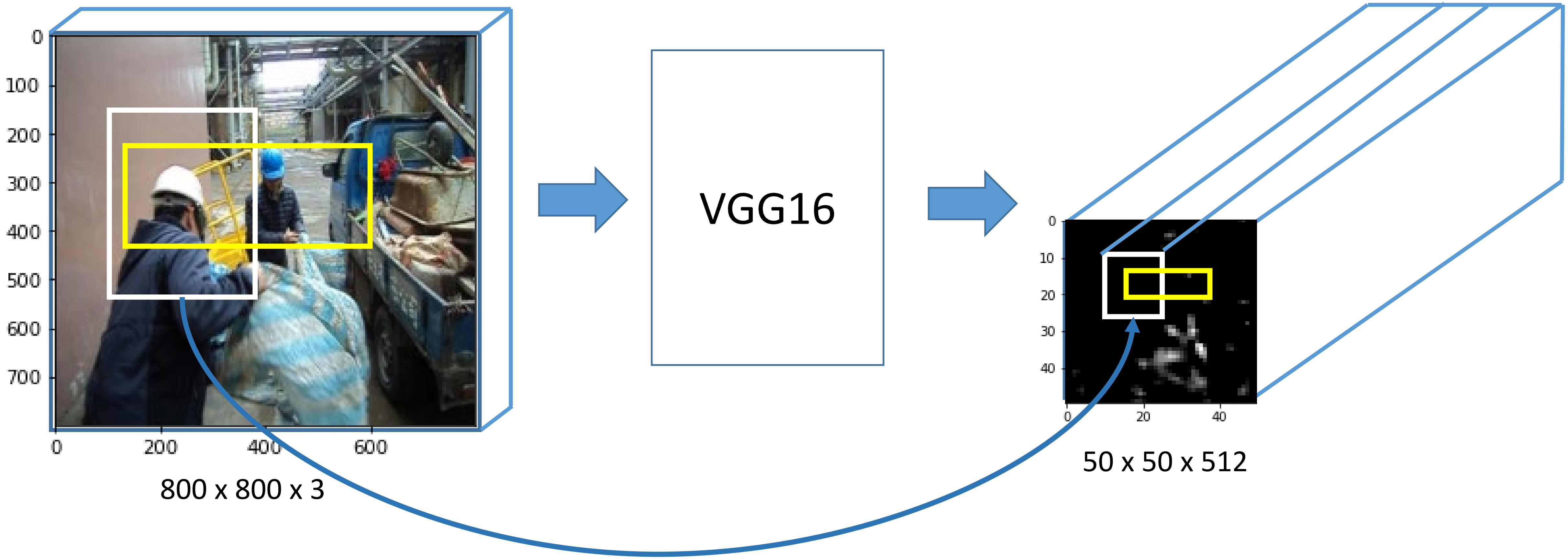
Use NMS, IOU to reduce the number of ROI from 22500 → 2000 → 128



Non-maximum Suppression (NMS)

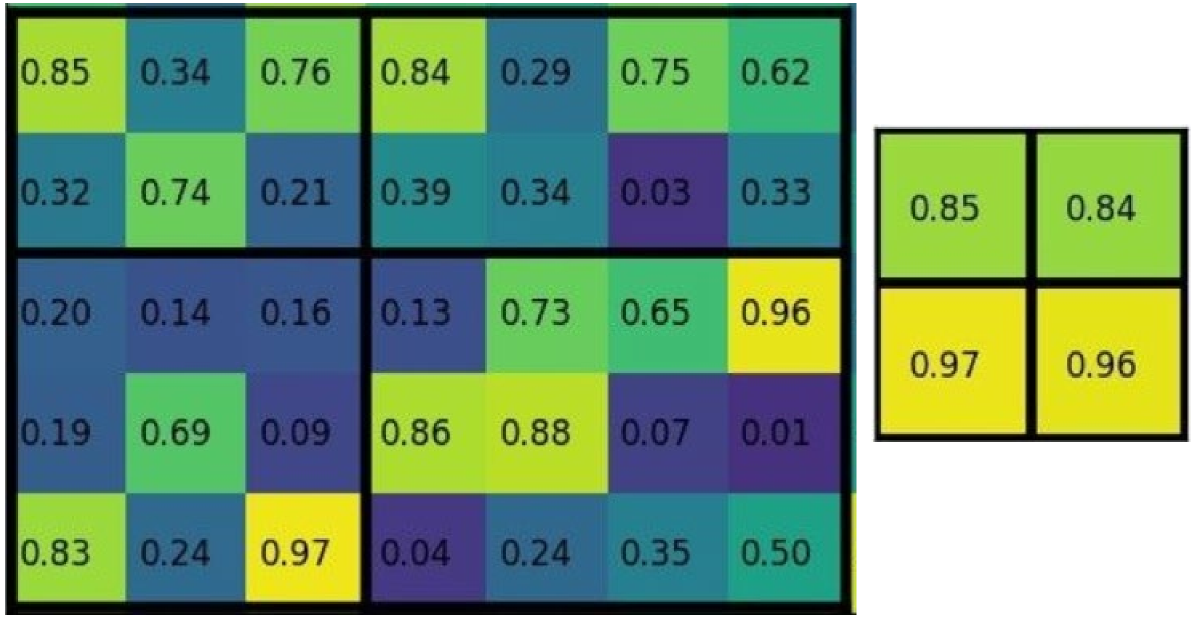
https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c

Extract the feature maps of the 128 ROI samples

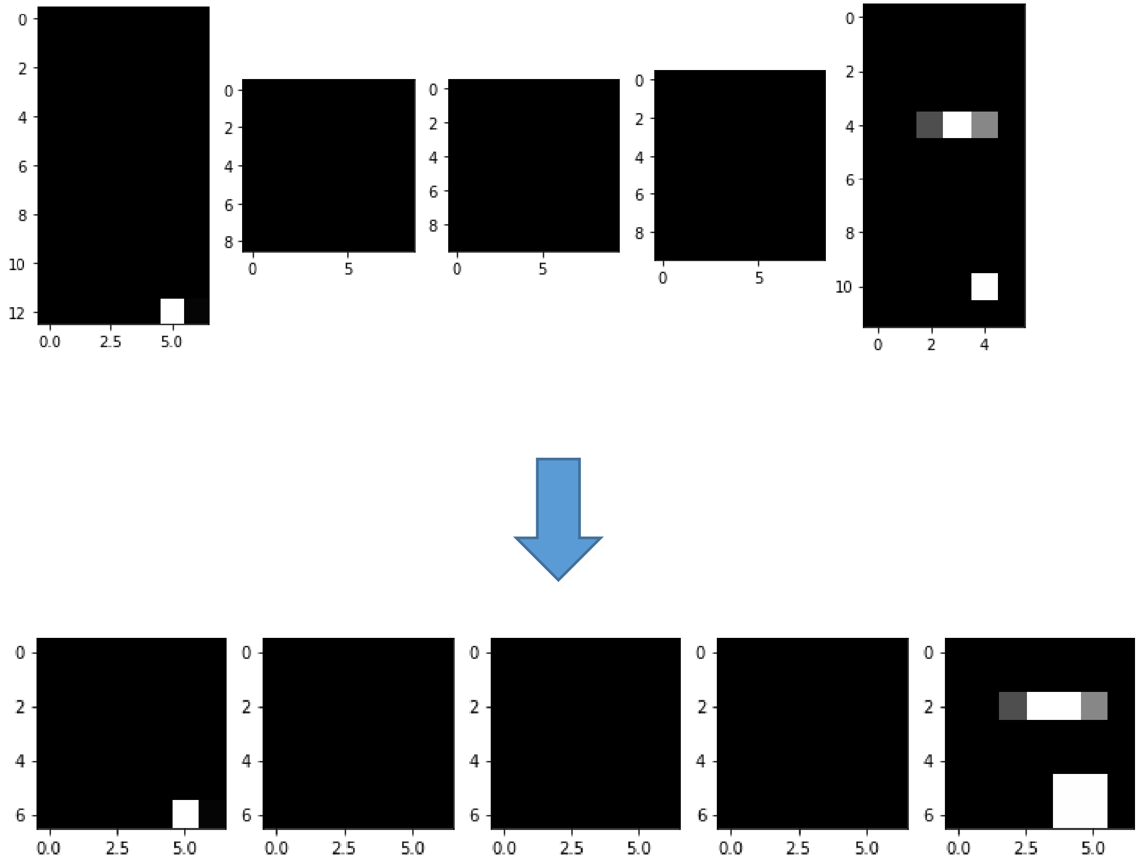


ROI Pooling

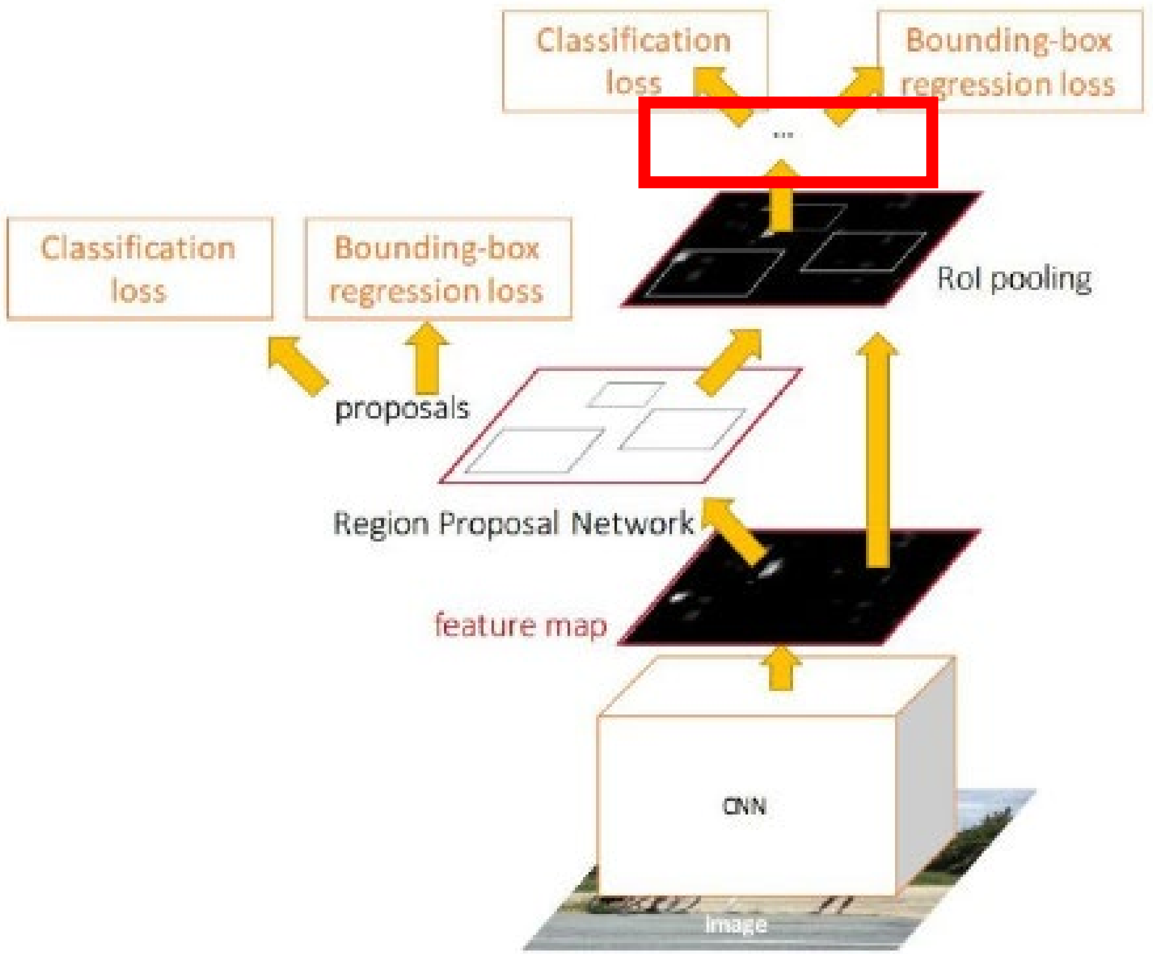
Extract the feature maps of the 128 ROI samples, adjust to the same size H=7, W=7 using max pooling (ROI Pooling)

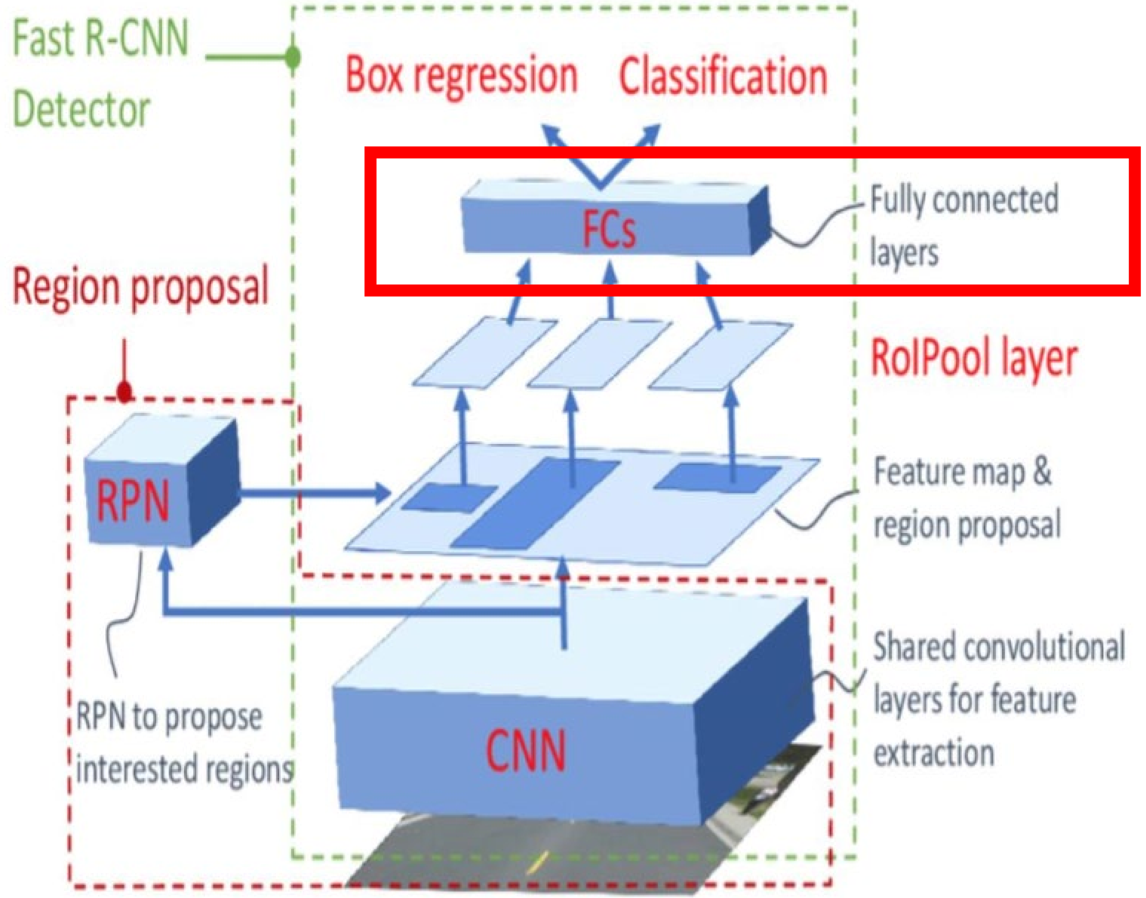


https://blog.csdn.net/qq_35586657/article/details/97885290



FastRCNN detector





FastRCNN detector

```
# Reshape the tensor so that we can p
k = output.view(output.size(0), -1)
print(k.shape) # 25088 = 7*7*512
```

```
roi_head_classifier = nn.Sequential(*[nn.Linear(25088, 4096), nn.Linear(4096, 4096)])
cls_loc = nn.Linear(4096, 2 * 4).to(device) # (1 classes 安全帽 + 1 background. Each 
cls_loc.weight.data.normal_(0, 0.01)
cls_loc.bias.data.zero_()

score = nn.Linear(4096, 2).to(device) # (1 classes, 安全帽 + 1 background)
```

No of object classes you want to predict + 1 (background)

```
# passing the output of roi-pooling to ROI head
k = roi_head_classifier(k.to(device))
roi_cls_loc = cls_loc(k)
roi_cls_score = score(k)
print(roi_cls_loc.shape, roi_cls_score.shape)
```

```
torch.Size([128, 8]) torch.Size([128, 2])
```

Class practice

- Prepare a training image that has at least two classes of objects to be recognized. Mark the 2 bounding boxes that represent 2 classes of objects. Pass the image + bbox through FasterRCNN to calculate training loss.

FastRCNN detector classification loss

$$L(p_i, t_i) = \boxed{\frac{1}{N_{cls}} \sum_i L_{cls}(p_i, \hat{p}_i)} + \lambda \frac{1}{N_{reg}} \sum_i p_i L_{reg}(t_i, \hat{t}_i)$$

$N_{cls} = 128$

$\hat{p}_i$ Probabilities for each class

p_i 0 or 1

$$L_{cls}(p_i, \hat{p}_i) = CE(p_i, \hat{p}_i) = -\sum_{k=1}^{C} p_k ln(\hat{p}_k)$$

FastRCNN detector bounding box regression loss

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, \hat{p}_i) + \boxed{\lambda \frac{1}{N_{reg}} \sum_i p_i L_{reg}(t_i, \hat{t}_i)}$$

$$N_{reg} = 128$$

$$t_i = [x_1, y_1, x_2, y_2]$$

$$\hat{t}_i = [\hat{x}_1, \hat{y}_1, \hat{x}_2, \hat{y}_2]$$

$$p_i = \begin{cases} 0, negative\ label \\ 1, positive\ label \end{cases}$$

$$L_{reg} = \begin{cases} 0.5(t_i - \hat{t}_i)^2 \times {}^1/_{\sigma^2} & if\ |t_i - \hat{t}_i| < {}^1/_{\sigma^2} \\ |t_i - \hat{t}_i| - 0.5 & otherwise \end{cases}$$

$\sigma = 1$ for FastRCNN training

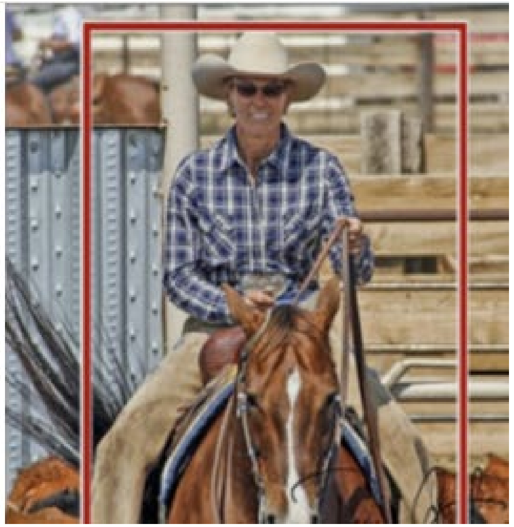
Fine tune FasterRCNN to detect our own objects

PyTorch tutorial

TorchVision Object Detection Finetuning Tutorial

Finetune a pre-trained Mask R-CNN model.

Image/Video



https://pytorch.org/tutorials/

PyTorch tutorial

TORCHVISION OBJECT DETECTION FINETUNING TUTORIAL

• TIP

To get the most of this tutorial, we suggest using this Colab Version. This will allow you to experiment with the information presented below.

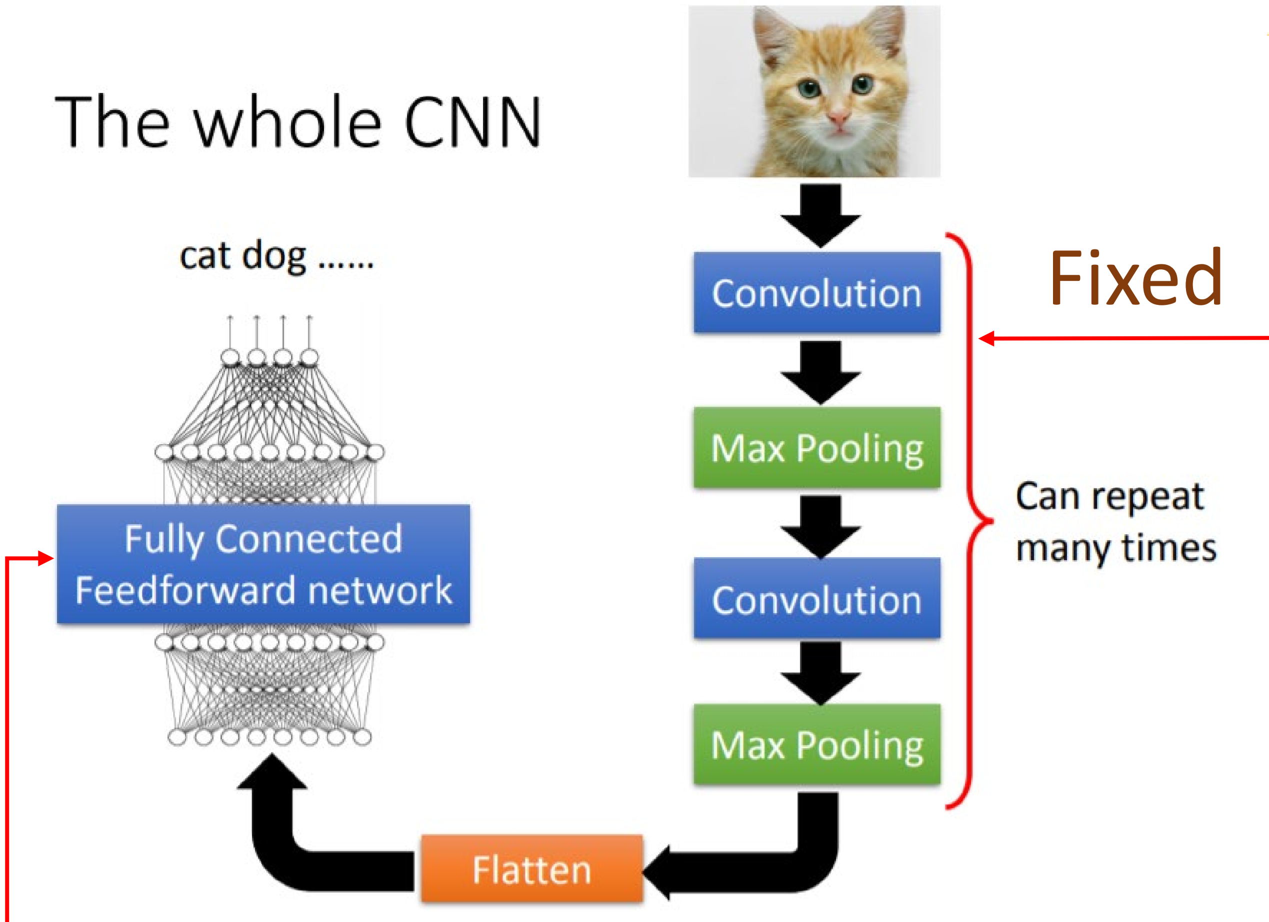
For this tutorial, we will be finetuning a pre-trained Mask R-CNN model in the Penn-Fudan Database for Pedestrian Detection and Segmentation. It contains 170 images with 345 instances of pedestrians, and we will use it to illustrate how to use the new features in torchvision in order to train an instance segmentation model on a custom dataset.

https://pytorch.org/tutorials/

Class practice

FasterRCNN(3) Fine_tune.ipynb

Recap – Fine-tune a pre-trained image classifier



Reference: 李弘毅 ML Lecture 10
https://youtu.be/FrKWiRv254g

Train

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), paddi
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), paddi
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padd
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padd
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pad
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Recap – Fine-tune a pre-trained image classifier

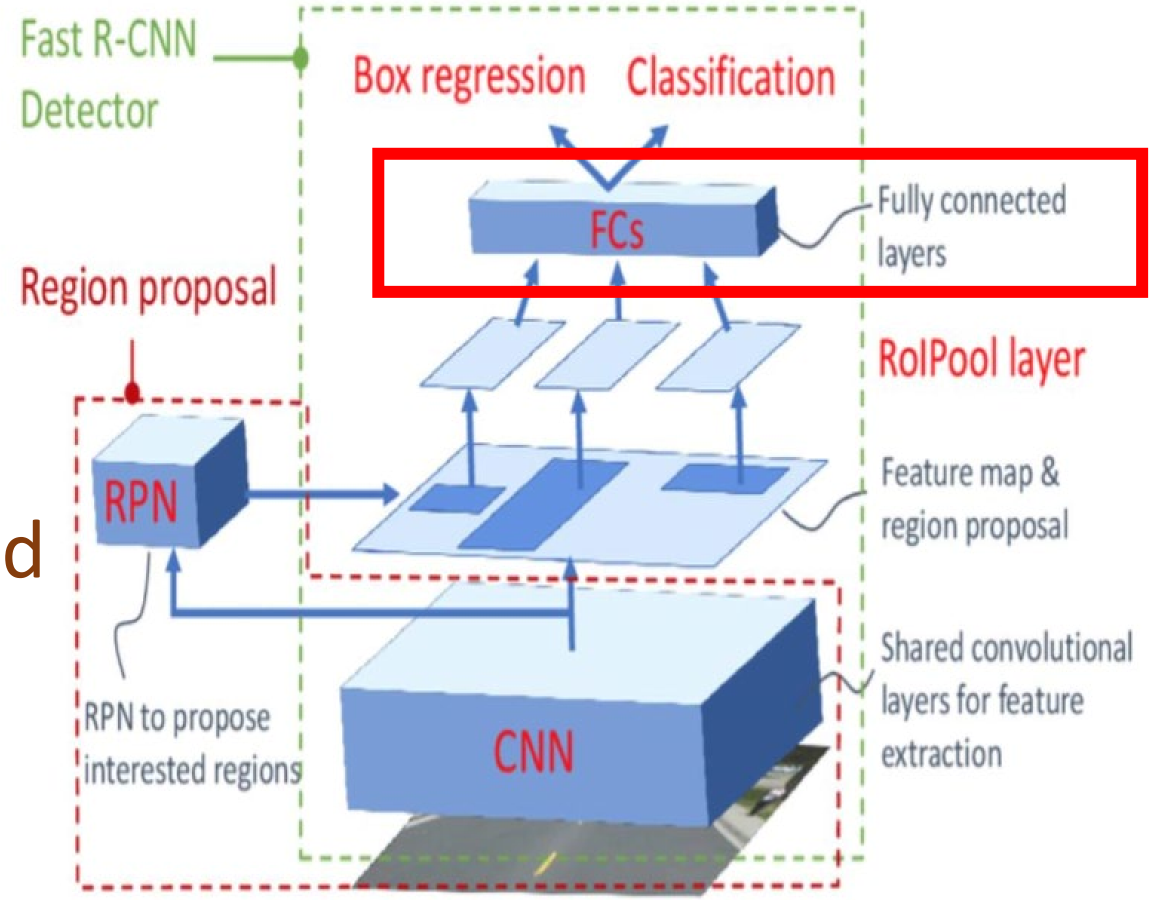
In [3]:

```
import torch.nn as nn
# fix the weight of convolution layers
model.features.eval()

# modify classifier
model.classifier = torch.nn.Sequential(
  nn.Linear(25088, 4096),
  nn.ReLU(inplace=True),
  nn.Dropout(p=0.5, inplace=False),
  nn.Linear(4096, 4096),
  nn.ReLU(inplace=True),
  nn.Dropout(p=0.5, inplace=False),
  torch.nn.Linear(4096, 5))
```

Fixed

Train

Fine-tune FasterRCNN



Train

Fixed

Fine-tune FasterRCNN

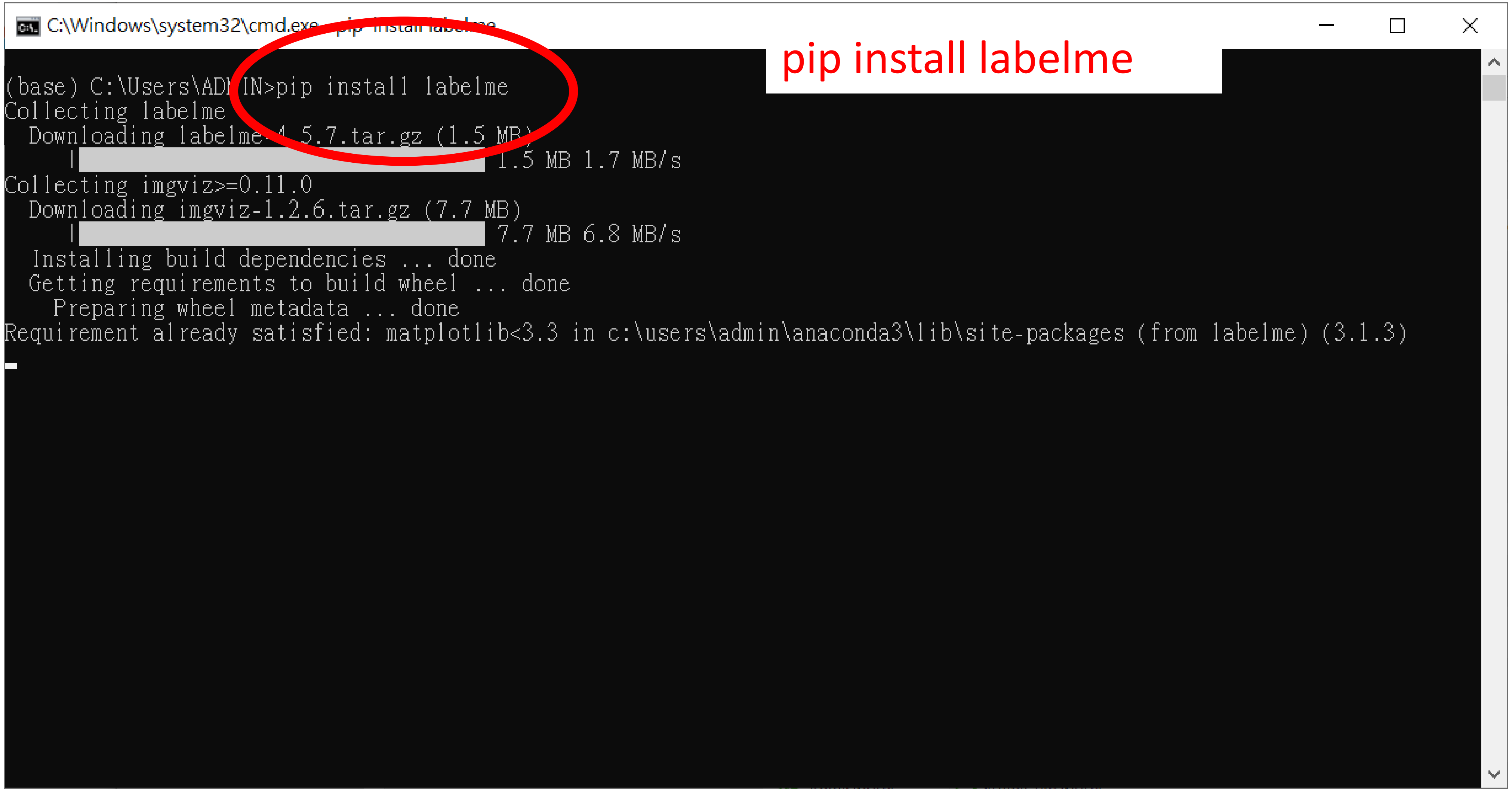
```
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor

# load a model pre-trained pre-trained on COCO
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

# replace the classifier
num_classes = 2  # 1 class (person) + background

# get number of input features for the classifier
in_features = model.roi_heads.box_predictor.cls_score.in_features

# replace the pre-trained head with a new one
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
```

Train

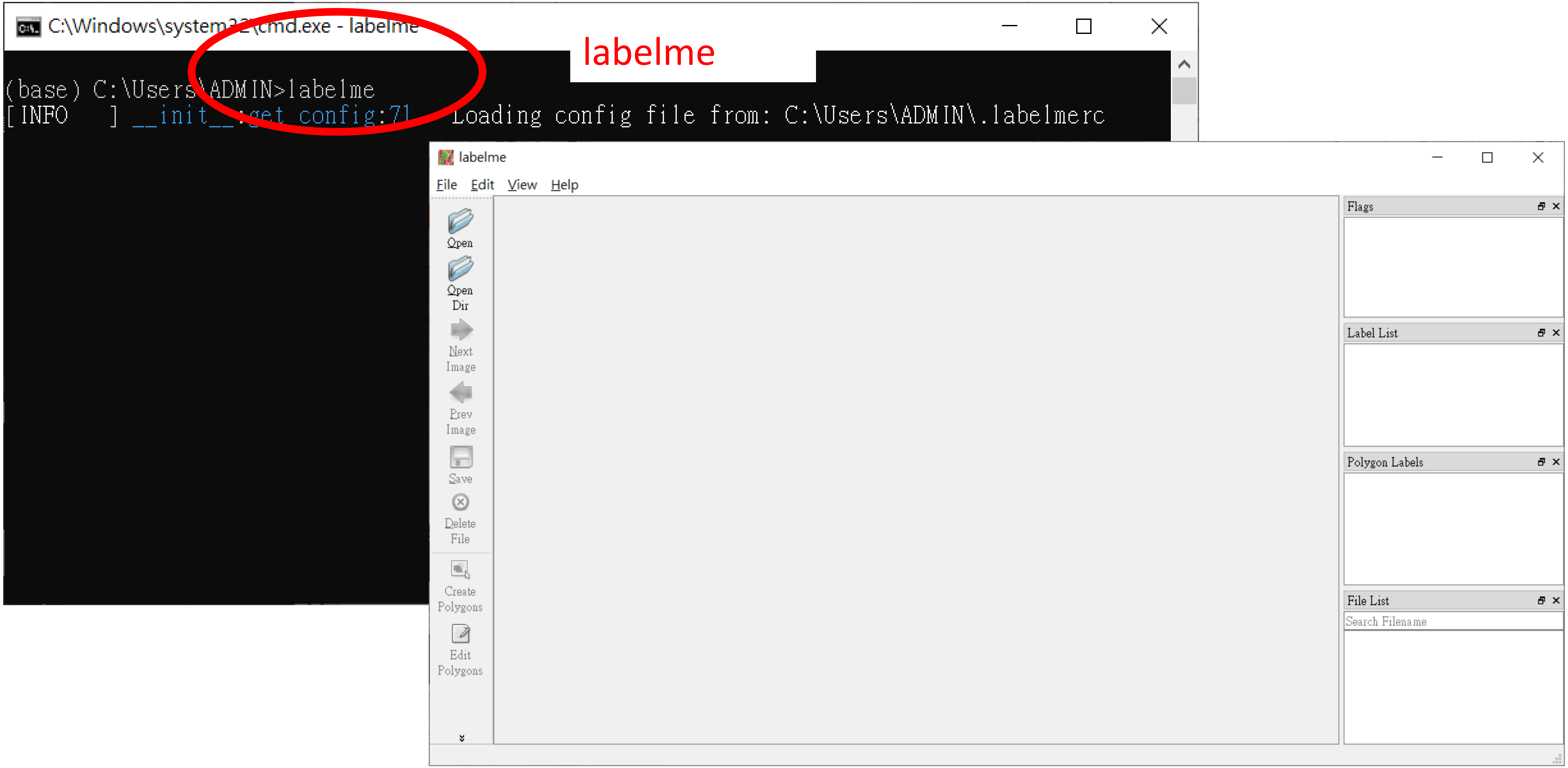
Class practice – Train your own object detector

1. Select an object that is not in the COCO dataset and collect 10 pictures that contain this object.
2. Use LabelMe to label the object and save to json. Then convert json files to mask images.
3. Run "FasterRCNN(3) Fine_tune.ipynb" to fine tune the pre-trained FasterRCNN to train a customized NN that can recognize your own object.

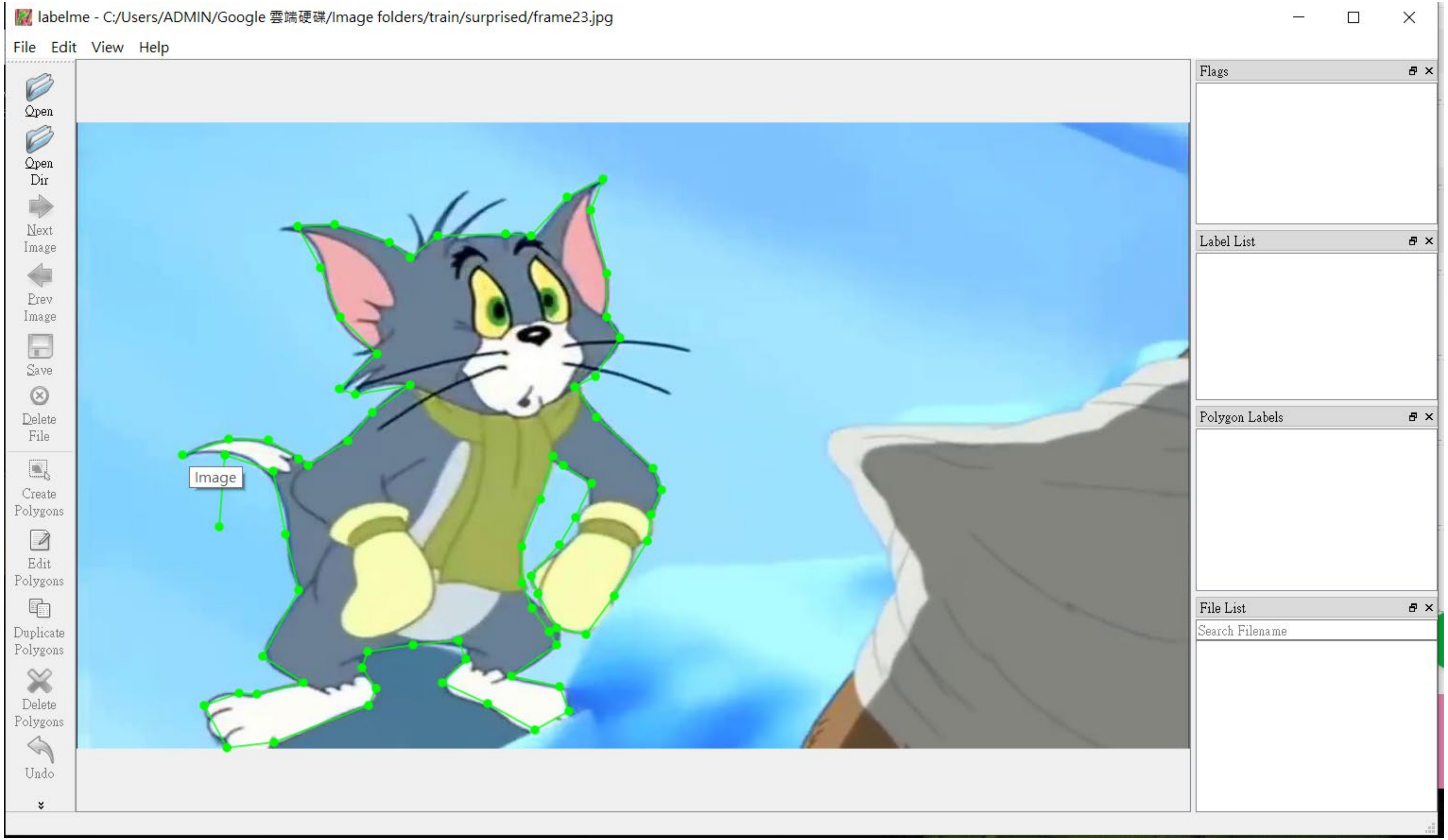
pip install labelme in your Anaconda environment



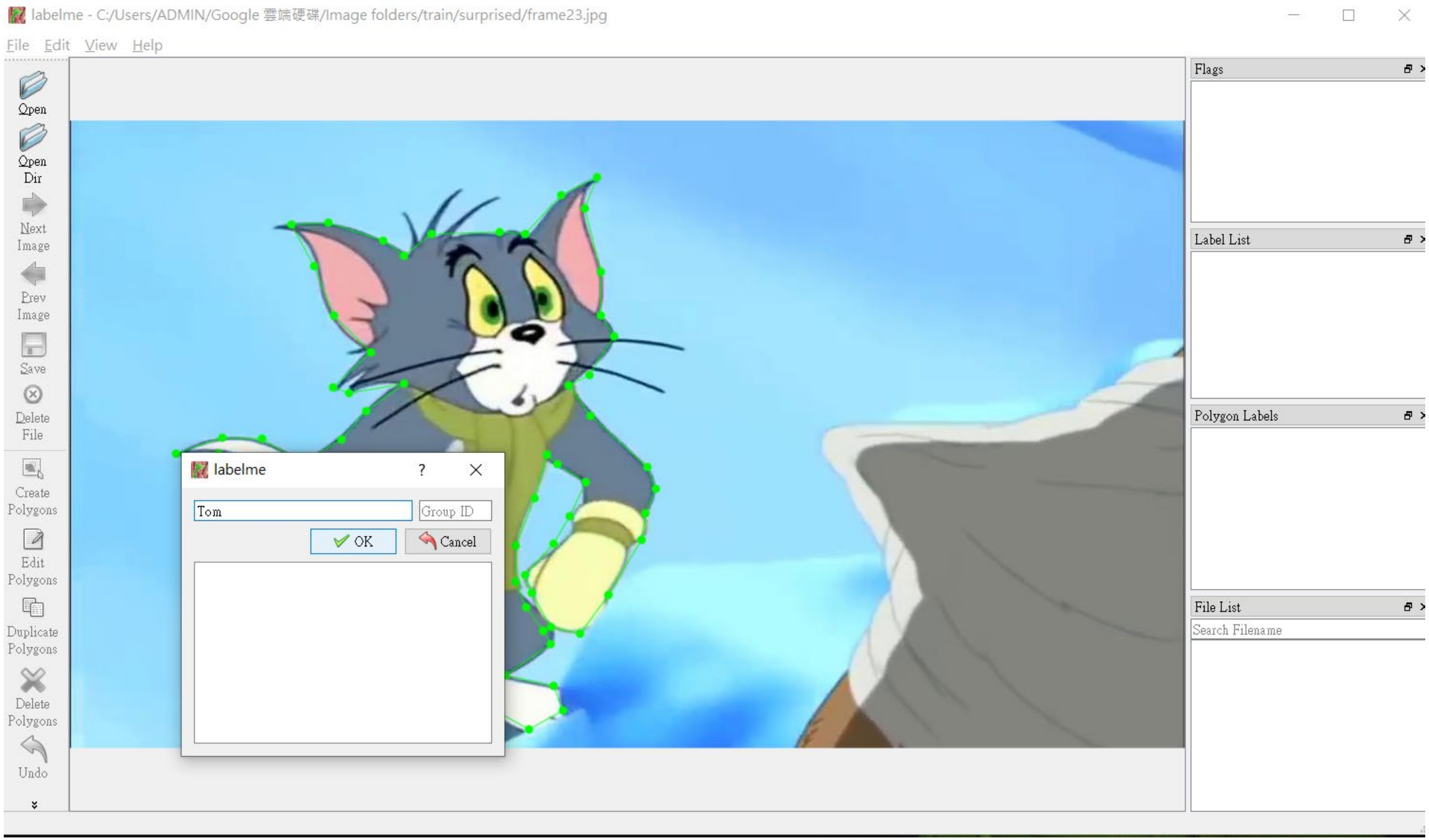
Run labelme


C:\Windows\system32\cmd.exe - labelme
labelme
(base) C:\Users\ADMIN>labelme
[INFO] __init__:get_config:71 - Loading config file from: C:\Users\ADMIN\.labelmerc
labelme
File Edit View Help
Open
Open Dir
Next Image
Prev Image
Save
Delete File
Create Polygons
Edit Polygons
Flags
Label List
Polygon Labels
File List
Search Filename

Load an image and draw boundary

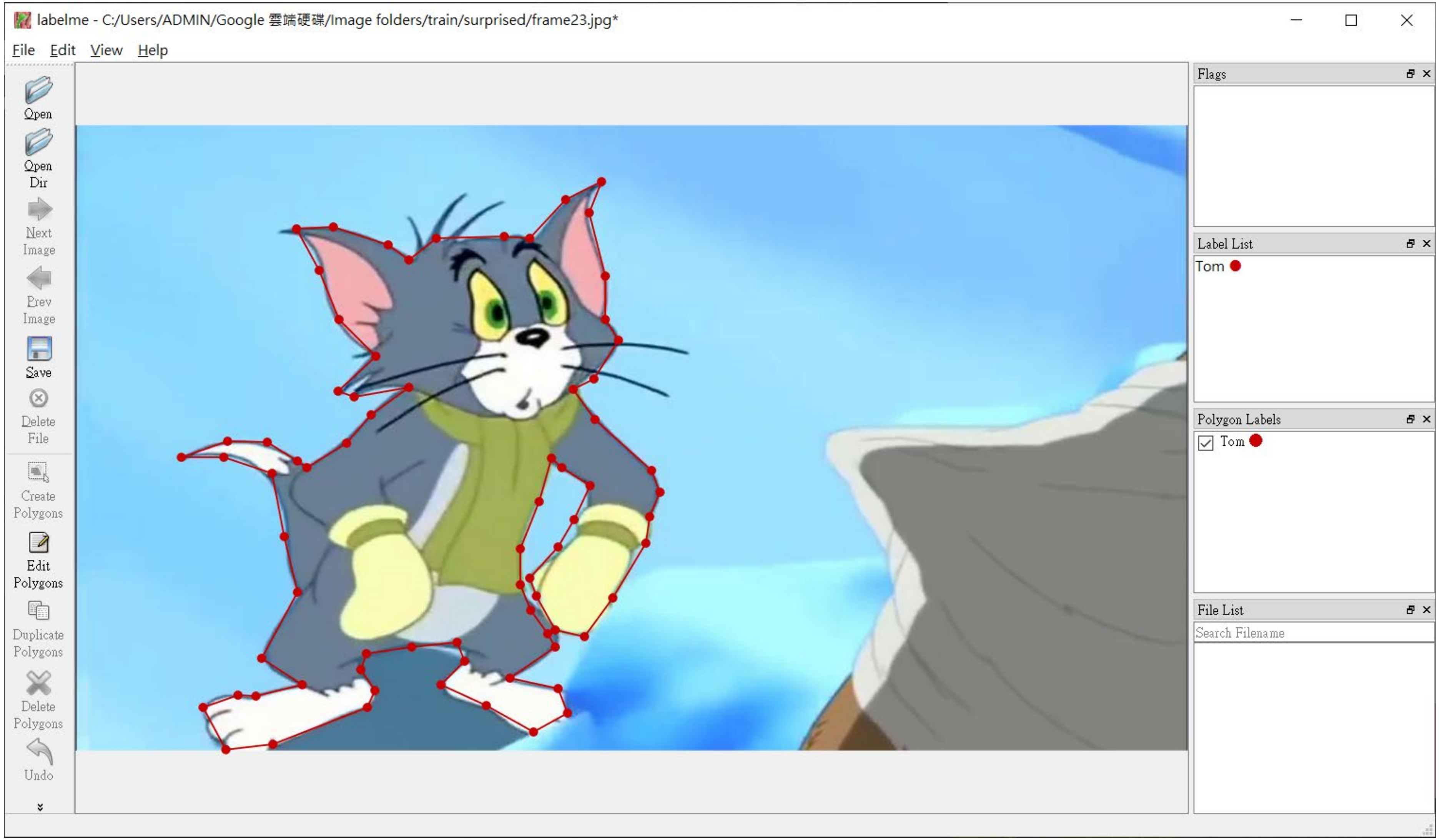


Save label



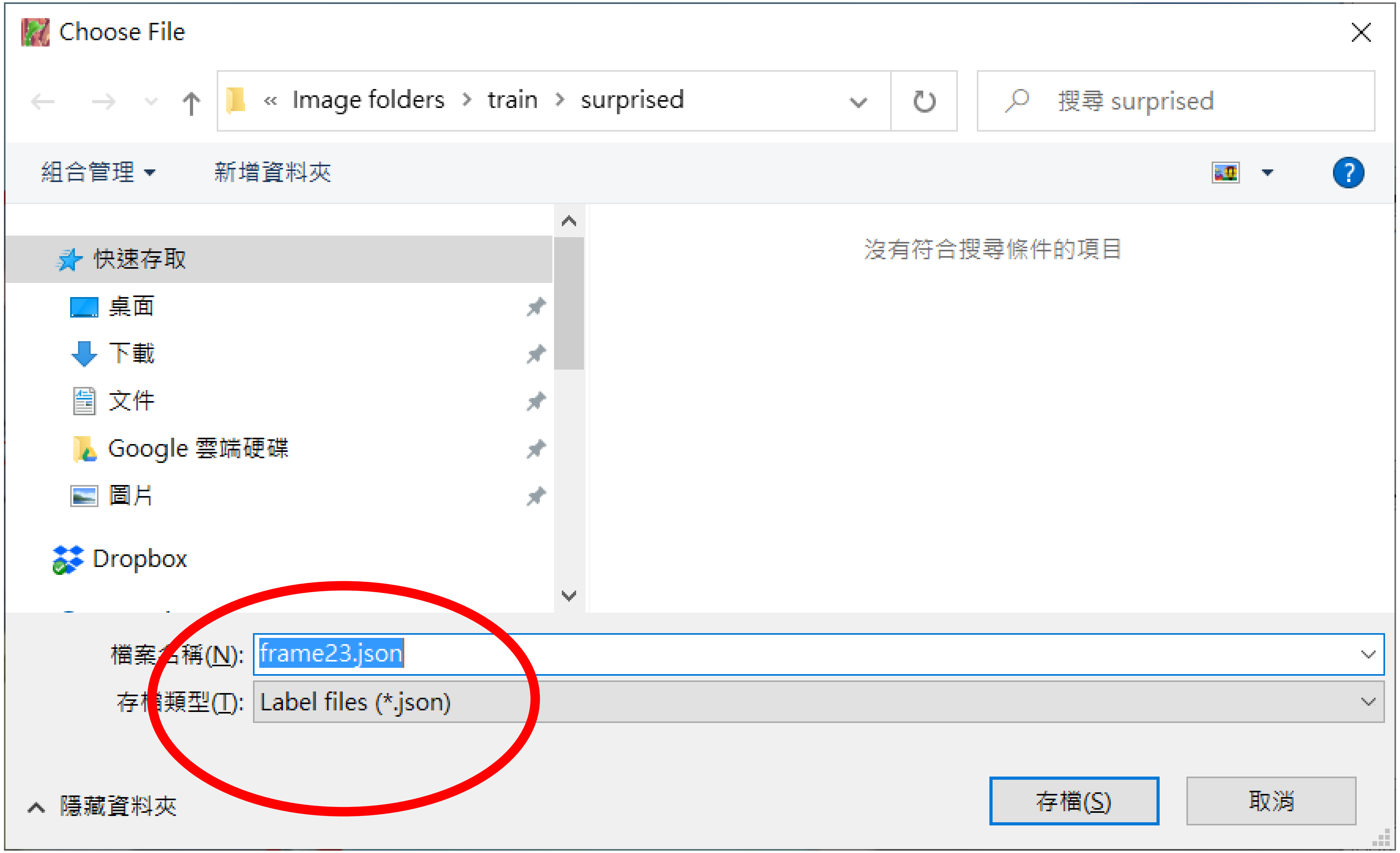
labelme - C:/Users/ADMIN/Google 雲端硬碟/Image folders/train/surprised/frame23.jpg
File
Edit
View
Help
Open
Open Dir
Next Image
Prev Image
Save
Delete File
Create Polygons
Edit Polygons
Duplicate Polygons
Delete Polygons
Undo
labelme
Tom
Group ID
OK
Cancel
Flags
Label List
Polygon Labels
File List
Search Filename

Saved label

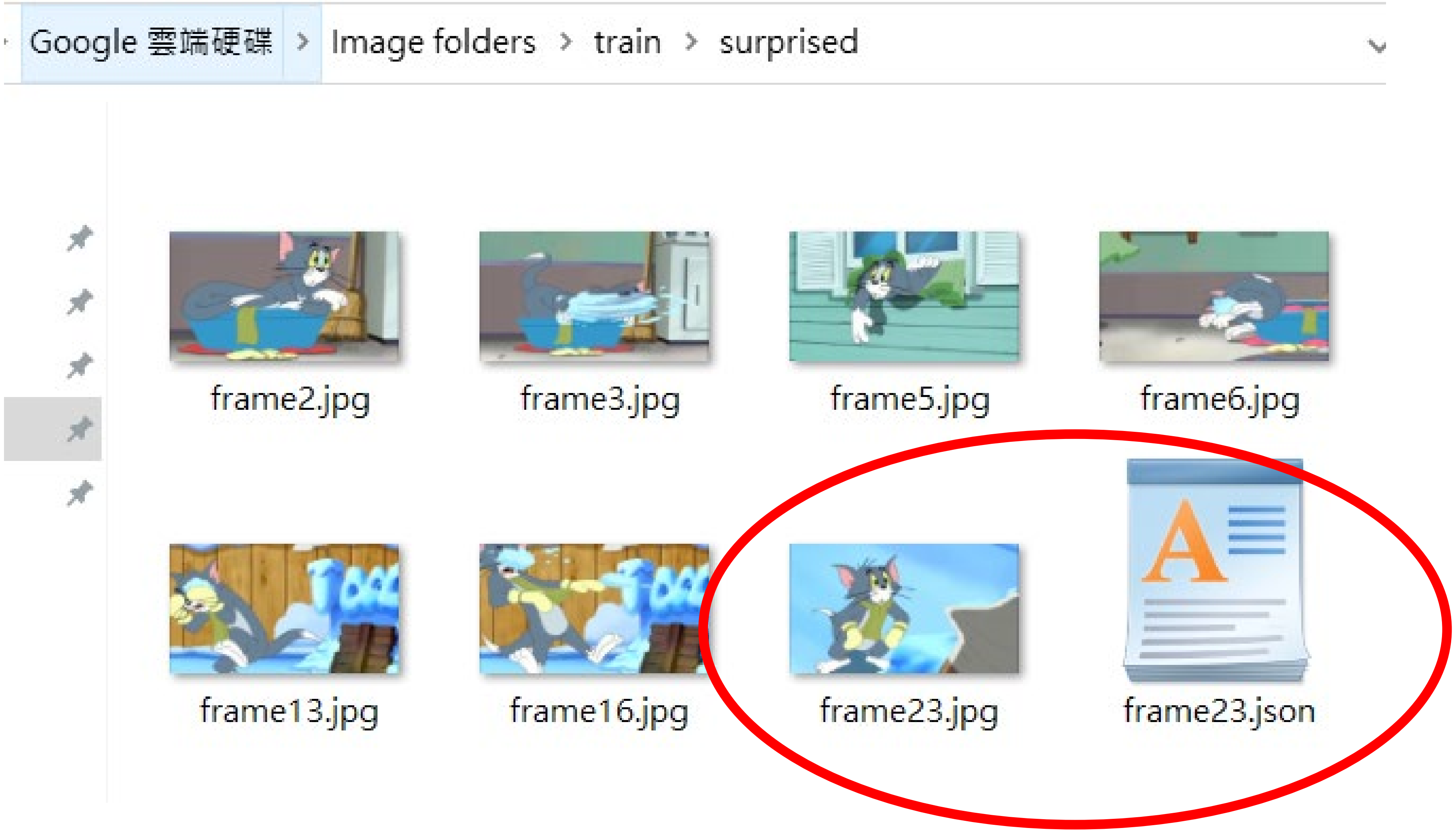


labelme - C:/Users/ADMIN/Google 雲端硬碟/Image folders/train/surprised/frame23.jpg*
File Edit View Help
Open
Open Dir
Next Image
Prev Image
Save
Delete File
Create Polygons
Edit Polygons
Duplicate Polygons
Delete Polygons
Undo
Flags
Label List
Tom
Polygon Labels
Tom
File List
Search Filename

Save boundary to json file



Saved json file



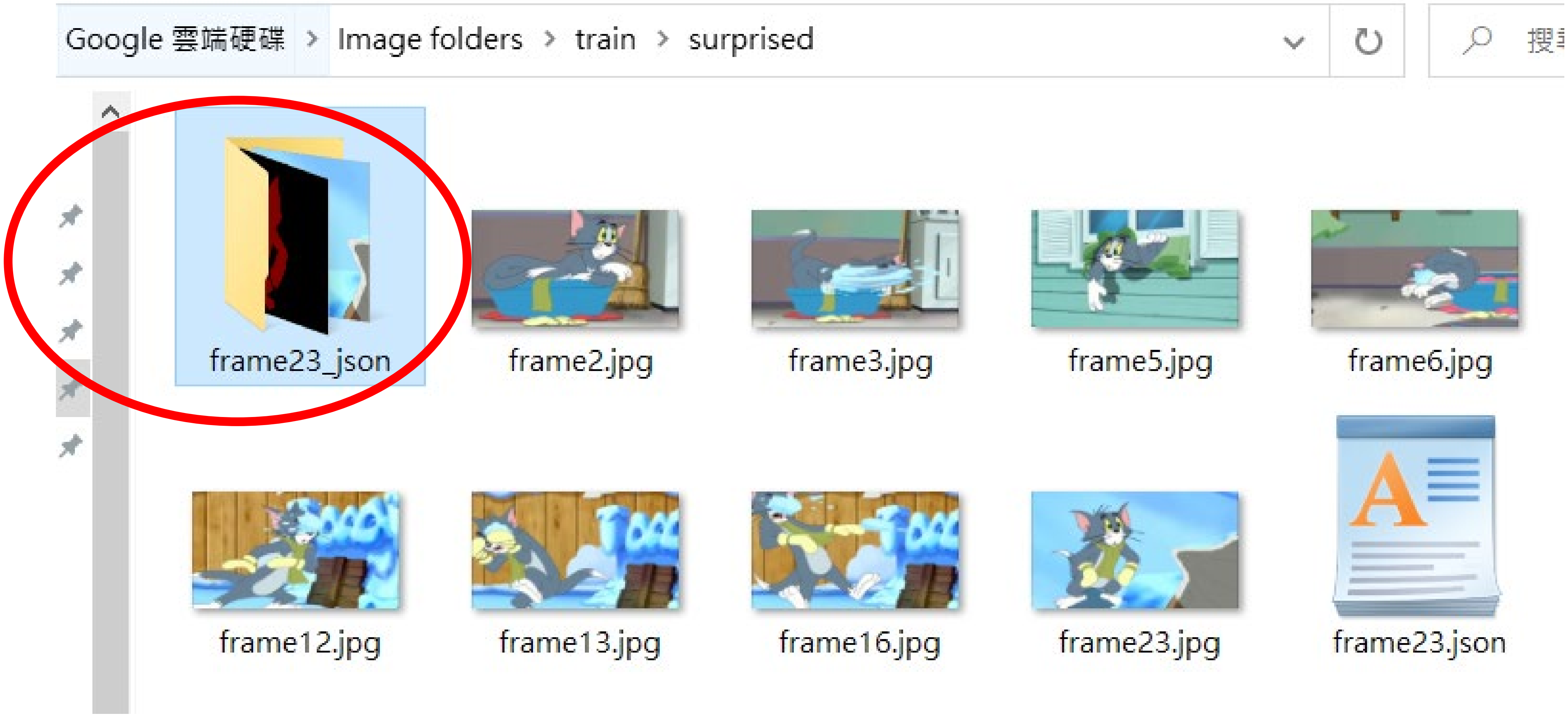
Convert json file to mask image

cd to the folder where you save the *.json file

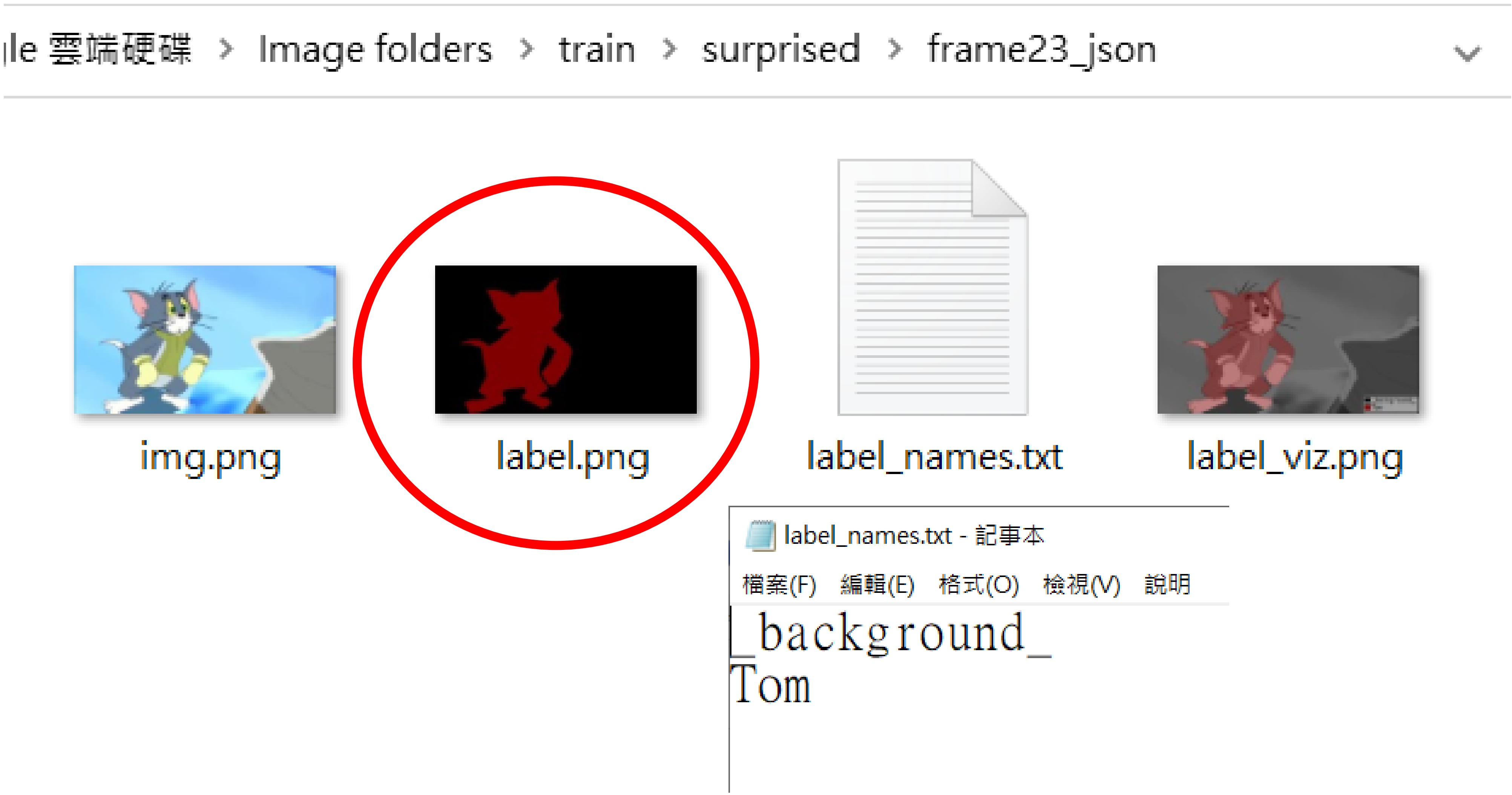
Labelme_json_to_dataset *.json

```
(base) C:\Users\ADMIN>cd C:\Users\ADMIN\Google 雲端硬碟\Image folders\train\surprised

(base) C:\Users\ADMIN\Google 雲端硬碟\Image folders\train\surprised>labelme_json_to_dataset frame23.json
[WARNING] json_to_dataset:main:16 - This script is aimed to demonstrate how to convert the JSON file to a sin
gle image dataset.
[WARNING] json_to_dataset:main:20 - It won't handle multiple JSON files to generate a real-use dataset.
[ INFO   ] json_to_dataset:main:77 - Saved to: frame23_json

(base) C:\Users\ADMIN\Google 雲端硬碟\Image folders\train\surprised>
```

Mask images are saved in a folder

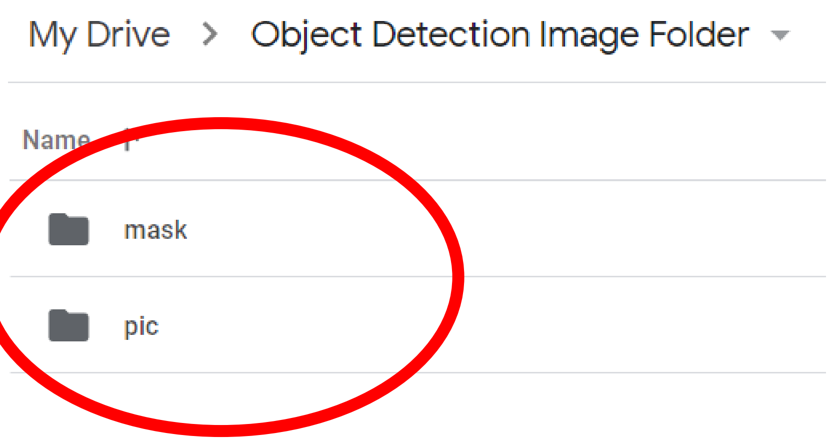


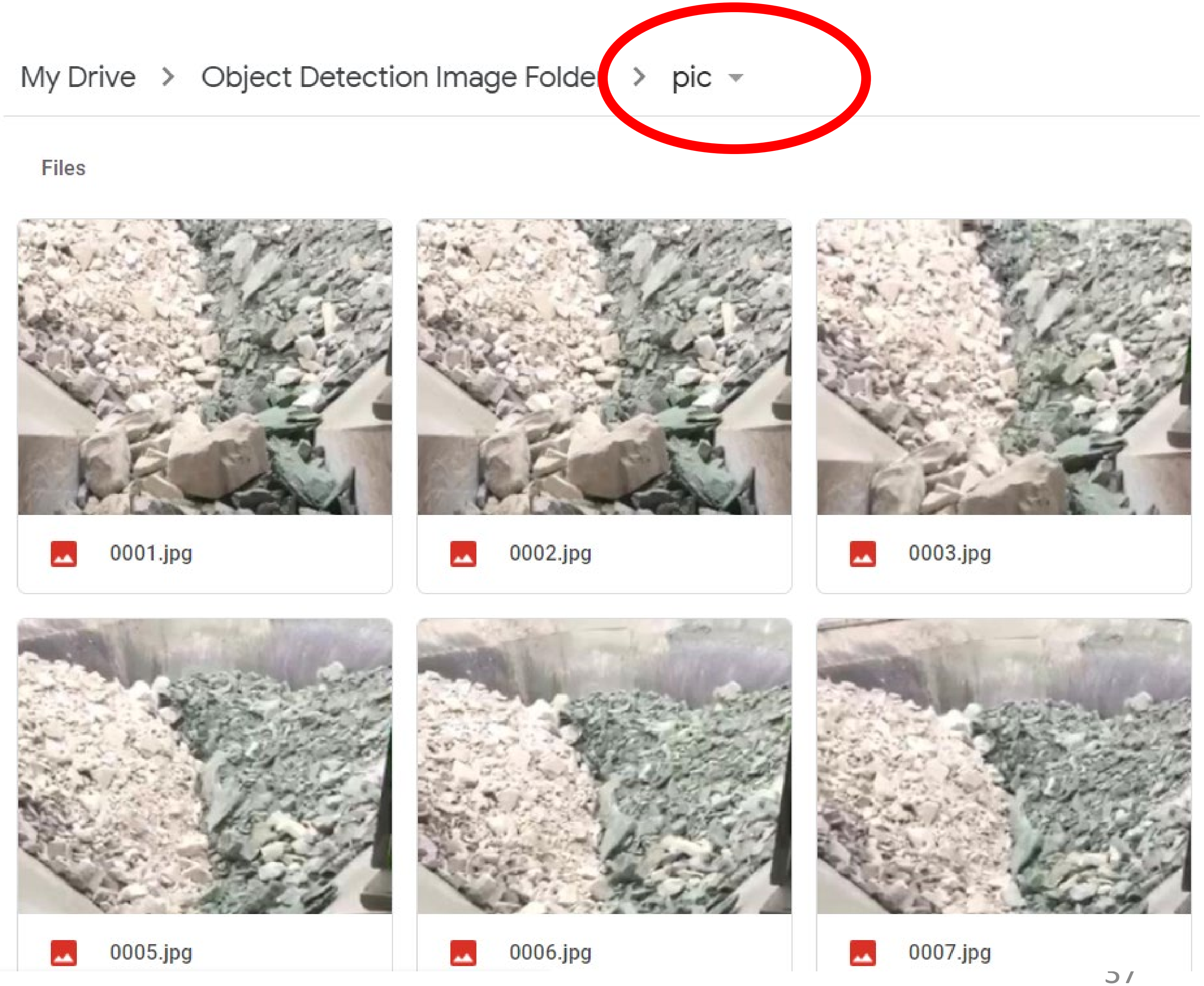
Mask image



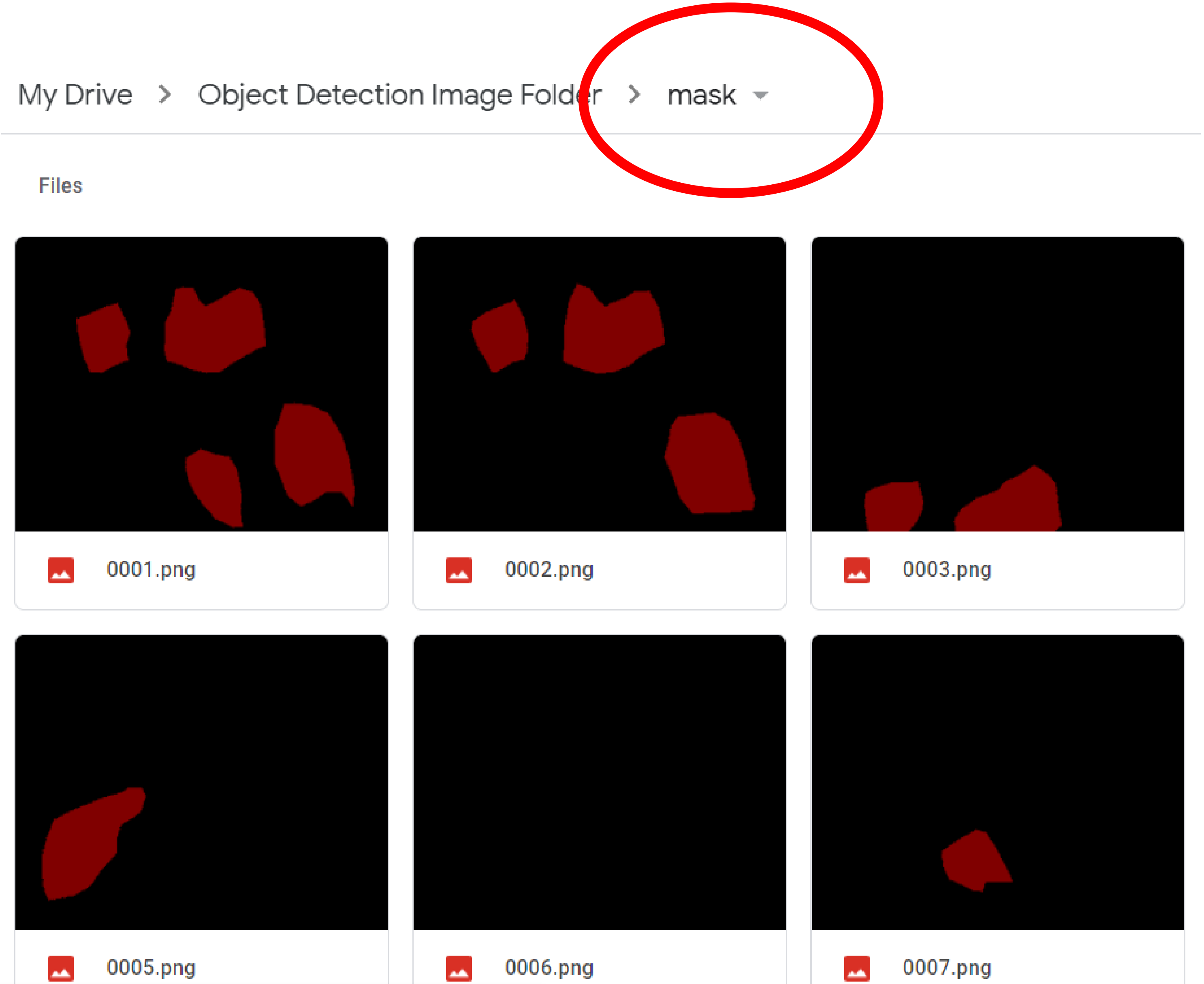
le 雲端硬碟 › Image folders › train › surprised › frame23_json
img.png
label.png
label_names.txt
label_viz.png
label_names.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
background
Tom

Save RGB and mask images on your Google drive





Save RGB and mask images on your Google drive



Split training and test set

I have 89 labelled images, of which 50 were used as training images and the remaining 39 used as test.

```
# split the dataset in train and test set
torch.manual_seed(1)
indices = torch.randperm(len(dataset)).tolist()
dataset = torch.utils.data.Subset(dataset, indices[:-50])
dataset_test = torch.utils.data.Subset(dataset_test, indices[-50:])
```

Batch size


```
# define training and validation data loaders
data_loader = torch.utils.data.DataLoader(
    dataset,
    batch_size=2,
    shuffle=True,
    num_workers=4,
    collate_fn=utils.collate_fn)

data_loader_test = torch.utils.data.DataLoader(
    dataset_test,
    batch_size=1,
    shuffle=False,
    num_workers=4,
    collate_fn=utils.collate_fn)
```

Try larger batch size

Training performance evaluation

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, \hat{p}_i) + \lambda \frac{1}{N_{reg}} \sum_i p_i L_{reg}(t_i, \hat{t}_i)$$

```
Epoch: [0]  [ 0/20]  eta: 0:01:04  lr: 0.000268  loss: 3.3367 (3.3367)  loss_classifier: 0.7079 (0.7079)
loss_box_reg: 0.1177 (0.1177)  loss_mask: 2.4933 (2.4933)  loss_objectness: 0.0119 (0.0119)  loss_rpn_bo
x_reg: 0.0060 (0.0060)  time: 3.2353  data: 2.5175  max mem: 2483
Epoch: [0]  [10/20]  eta: 0:00:07  lr: 0.002897  loss: 1.3361 (1.7297)  loss_classifier: 0.1779 (0.2886)
loss_box_reg: 0.0578 (0.0657)  loss_mask: 1.0733 (1.2952)  loss_objectness: 0.0762 (0.0733)  loss_rpn_bo
x_reg: 0.0060 (0.0068)  time: 0.7238  data: 0.3247  max mem: 2759
Epoch: [0]  [19/20]  eta: 0:00:00  lr: 0.005000  loss: 0.8906 (1.2921)  loss_classifier: 0.1090 (0.2024)
loss_box_reg: 0.0579 (0.0652)  loss_mask: 0.6534 (0.9624)  loss_objectness: 0.0416 (0.0560)  loss_rpn_bo
x_reg: 0.0045 (0.0061)  time: 0.6082  data: 0.2312  max mem: 2759
Epoch: [0] Total time: 0:00:12 (0.6111 s / it)
```

Testing performance evaluation

- To evaluate object detection models like FasterRCNN and YOLO, the mean average precision (mAP) is used. The mAP compares the ground-truth bounding box to the detected box and returns a score. The higher the score, the more accurate the model is in its detections.

https://blog.paperspace.com/mean-average-precision/

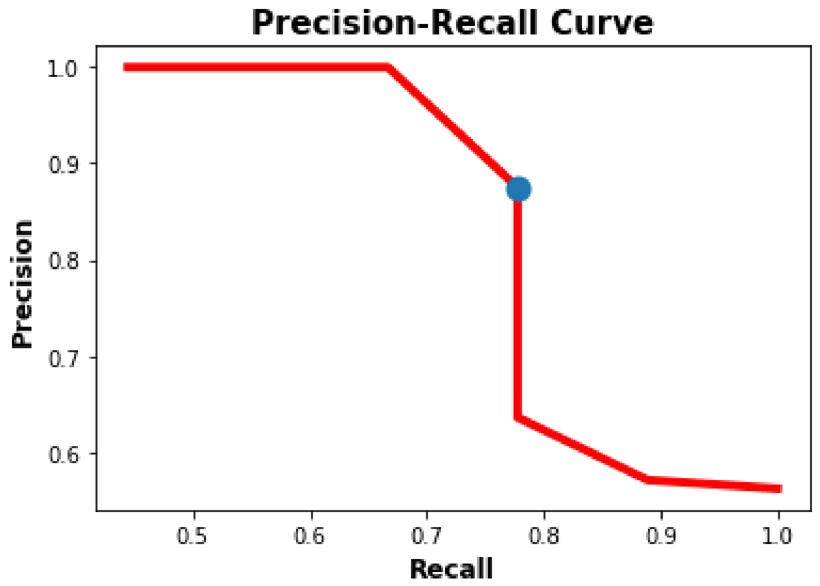
Precision and recall

Table 3 Confusion matrix

	Reference (high-risk)	Reference (low-risk)
Predicted (high-risk)	True positive (TP)	False positive (FP)
Predicted (low-risk)	False negative (FN)	True negative (TN)

$$Recall = \frac{TP}{TP + FP}$$

> **When a model has high recall but low precision, then the model classifies most of the positive samples correctly but it has many false positives (i.e. classifies many Negative samples as Positive). When a model has high precision but low recall, then the model is accurate when it classifies a sample as Positive but it may classify only some of the positive samples.**



https://blog.paperspace.com/mean-average-precision/

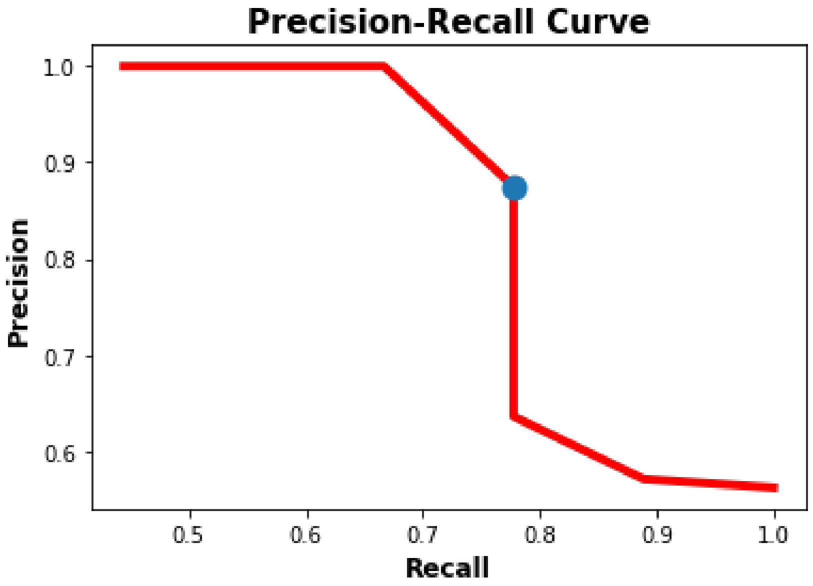
Average precision

The **average precision (AP)** is a way to summarize the precision-recall curve into a single value representing the average of all precisions.

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k)$$

$$Recalls(n) = 0, Precisions(n) = 1$$

$$n = Number\ of\ thresholds.$$



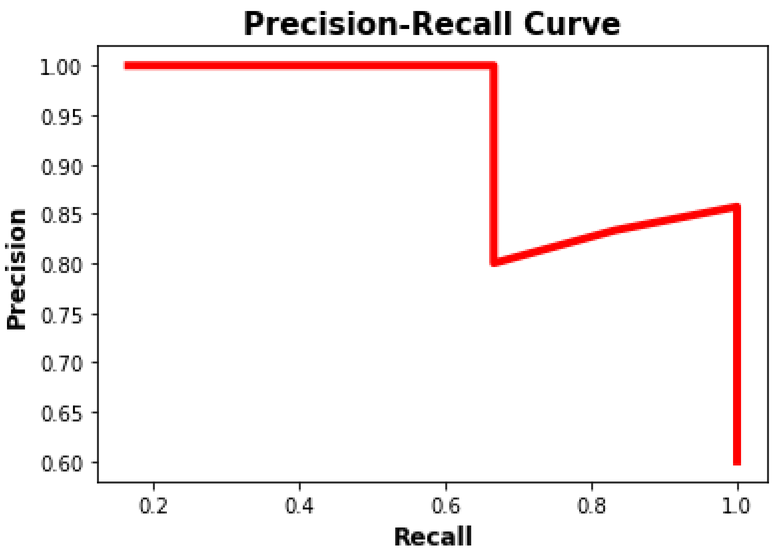
AP is the weighted sum of precisions at each threshold where the weight is the increase in recall.

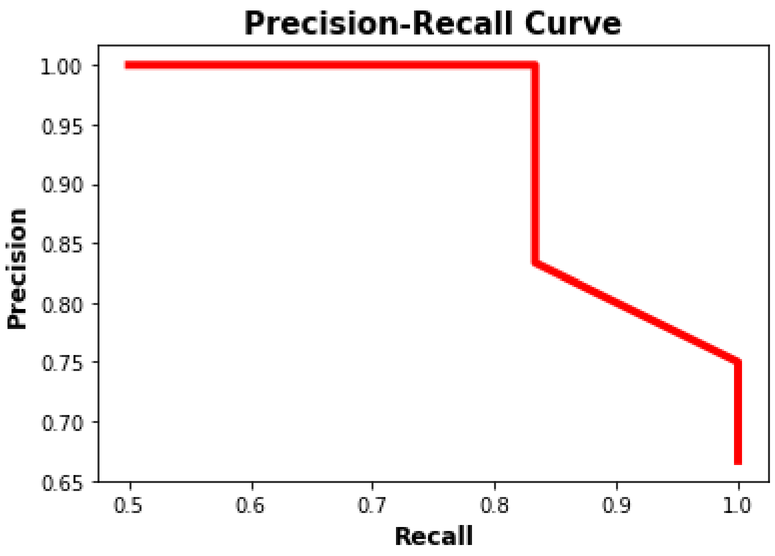
https://blog.paperspace.com/mean-average-precision/

mAP (mean average precision)

$$mAP = \frac{1}{n}\sum_{k=1}^{n} AP_k$$

n = number of classes





https://blog.paperspace.com/mean-average-precision/

HW4 – Object detector

- Fine-tune pre-trained FasterRCNN to detect your own objects.
- Show training loss plot.
- Show detection results.

Automatically labelled photo-realistic images

Accelerate computer vision model training with the synthetic image data generated using Unity's perception package

https://unity.com/products/computer-vision

Unity perception package

https://github.com/Unity-Technologies/com.unity.perception