From image classification to object detection

Alex Net VGG16 Res Net	U Net	Yolo Faster RCNN	Mask RCNN	OpenPose Keypoints RCNN

Classification

Semantic Segmentation

GRASS, CAT, TREE, SKY

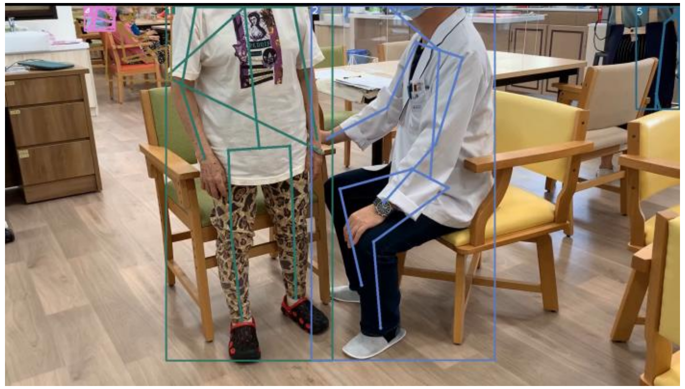
Object Detection

DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

Joint detection



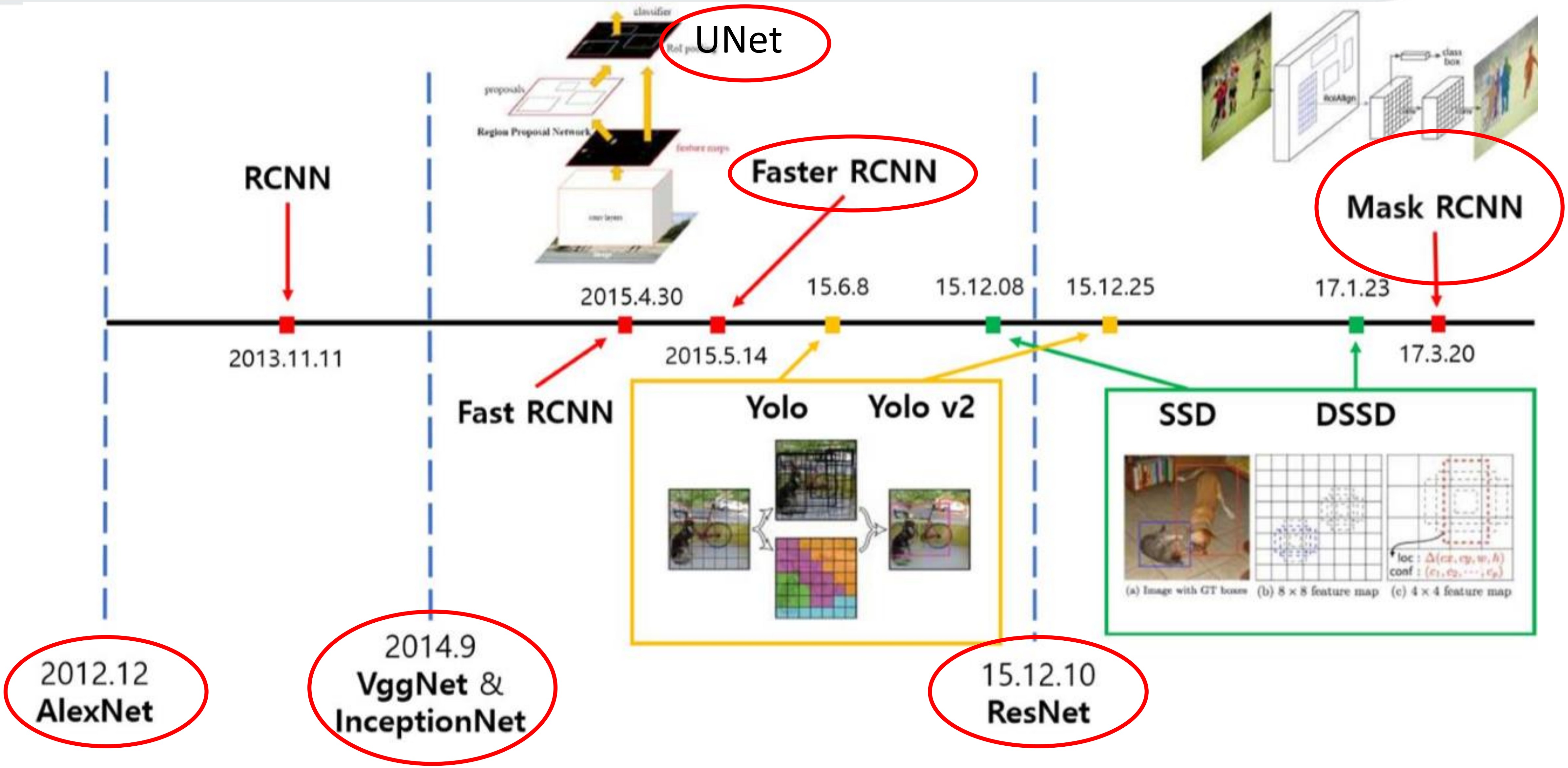
圖片來源: https://kharshit.github.io/blog/2019/08/23/quick-intro-to-instance-segmentation

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Advances in neural information processing systems 2015 (pp. 91-99).

AlexNet, VGG, ResNet and FasterRCNN



圖來源: 李春煌 FasterRCNN講義 https://youtu.be/2i9CcmJp2yI

Class practice

FasterRCNN(1).ipynb

Objects detected by the pre-trained FasterRCNN

```
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N/A',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket',
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]
```

91 objects trained with COCO dataset

Load pre-trained FasterRCNN

```
import torchvision
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
model.to(device)
model.eval()
```

```
Downloading: "https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco
rcnn_resnet50_fpn_coco-258fb6c6.pth
```

```
HBox(children=(FloatProgress(value=0.0, max=167502836.0), HTML(value='')))
```

Use CV2 to read uploaded video

```python
# take a look at the input video
import cv2
import imageio
import matplotlib.pyplot as plt
from IPython import display

cap = cv2.VideoCapture(fname)
total_frames = int(cap.get(7))
vid = imageio.get_reader(fname, 'ffmpeg')
print('No. of frames = ', total_frames)
frame_count = 1
try:
  while(frame_count <= total_frames):
    display.clear_output(wait=True)
    plt.title(str(frame_count)+'/'+str(total_frames))
    frame = vid.get_data(frame_count)  # Capture frame-by-frame
    frame_count += 1
    plt.imshow(frame)
    plt.pause(0.1)
except:
    print("Read video error!")
```

Pass frames to FasterRCNN to detect objects

```
while(frame_count <= total_frames):
  display.clear_output(wait=True)
  plt.title(str(frame_count)+'/'+str(total_frames))
  frame = vid.get_data(frame_count)  # Capture frame-by-frame
  img0 = np.copy(frame)
  transform = transforms.Compose([transforms.ToTensor()]) # Defing PyTorch Transform
  img = transform(frame).to(device) # Apply the transform to the image
  pred = model([img]) # Pass the image to the model
  pred_class = [COCO_INSTANCE_CATEGORY_NAMES[i] for i in list(pred[0]['labels'].cpu().numpy())] # Get
  pred_boxes = [[(i[0], i[1]), (i[2], i[3])] for i in list(pred[0]['boxes'].cpu().detach().numpy())]
  pred_score = list(pred[0]['scores'].cpu().detach().numpy())
  i = 0
  while i < len(pred_score):
```

Class practice

FasterRCNN(2) Step_by_step.ipynb

Reference

Machine-Vision Research Group 503 Followers About Follow Sign in Get started

Guide to build Faster RCNN in PyTorch

Understanding and implementing Faster RCNN from scratch.

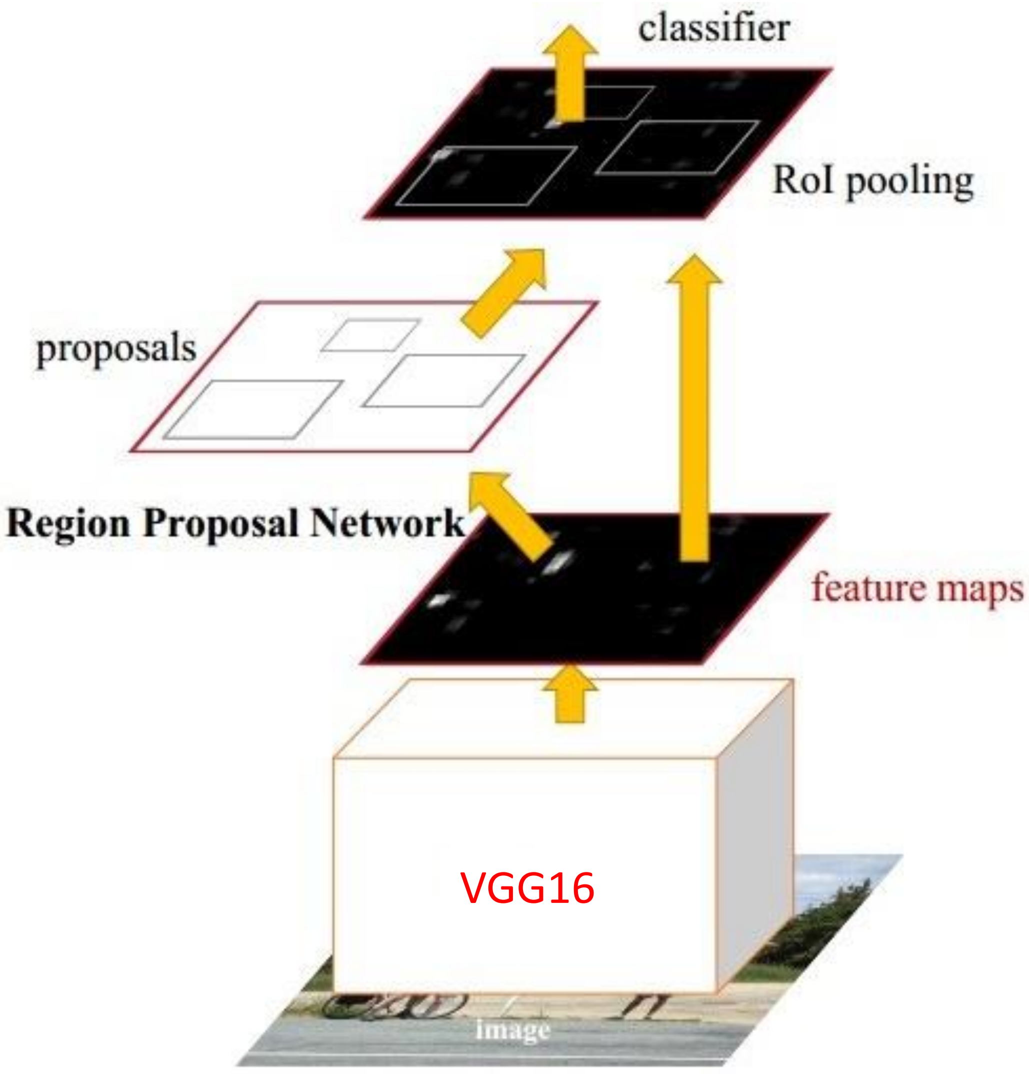
Machine-Vision Research Group Dec 4, 2018 · 31 min read

Introduction

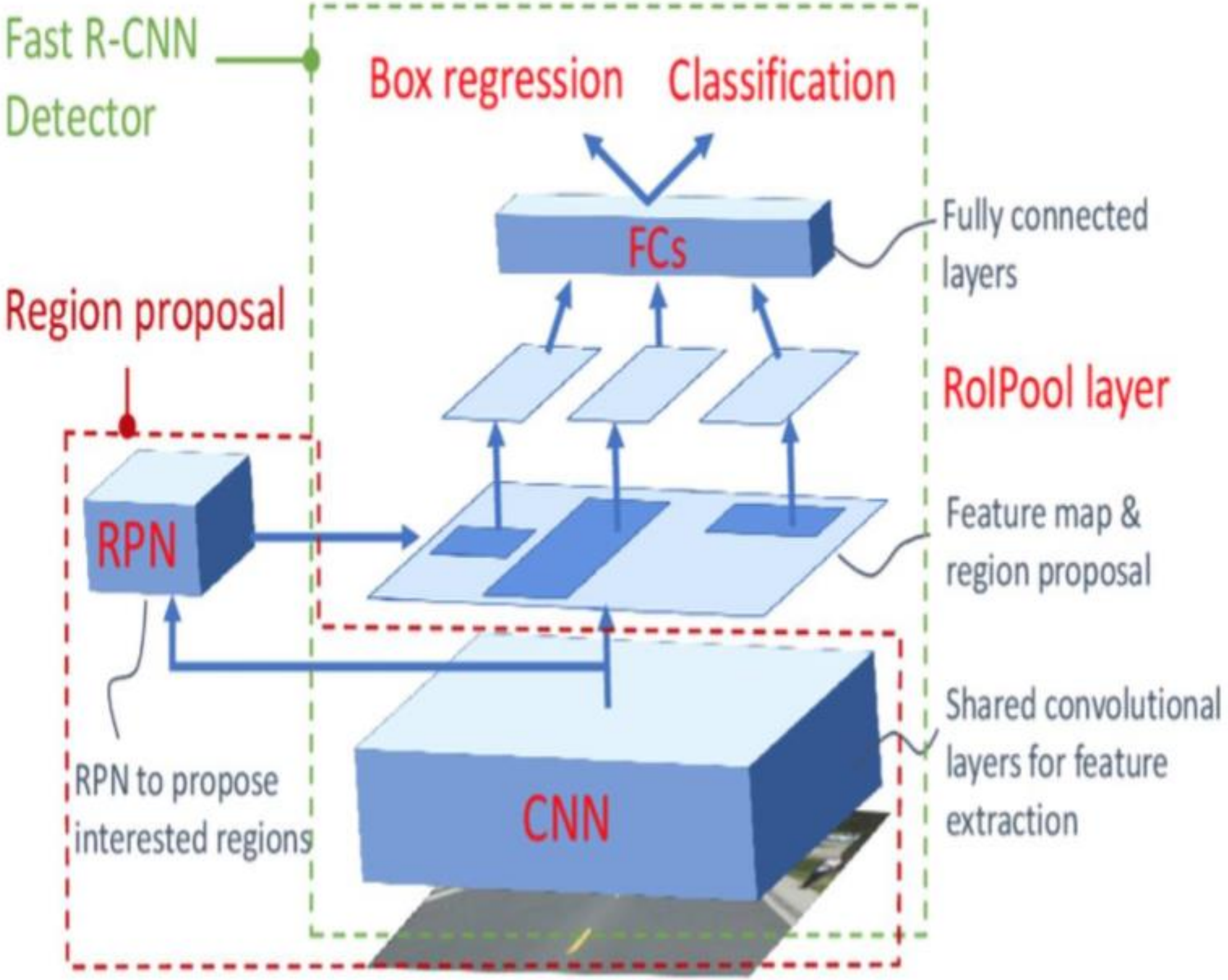
Faster R-CNN is one of the first frameworks which completely works on Deep learning. It is built upon the knowledge of **Fast RCNN** which indeed

https://medium.com/@fractaldle/guide-to-build-faster-rcnn-in-pytorch-95b10c273439

Overall architecture

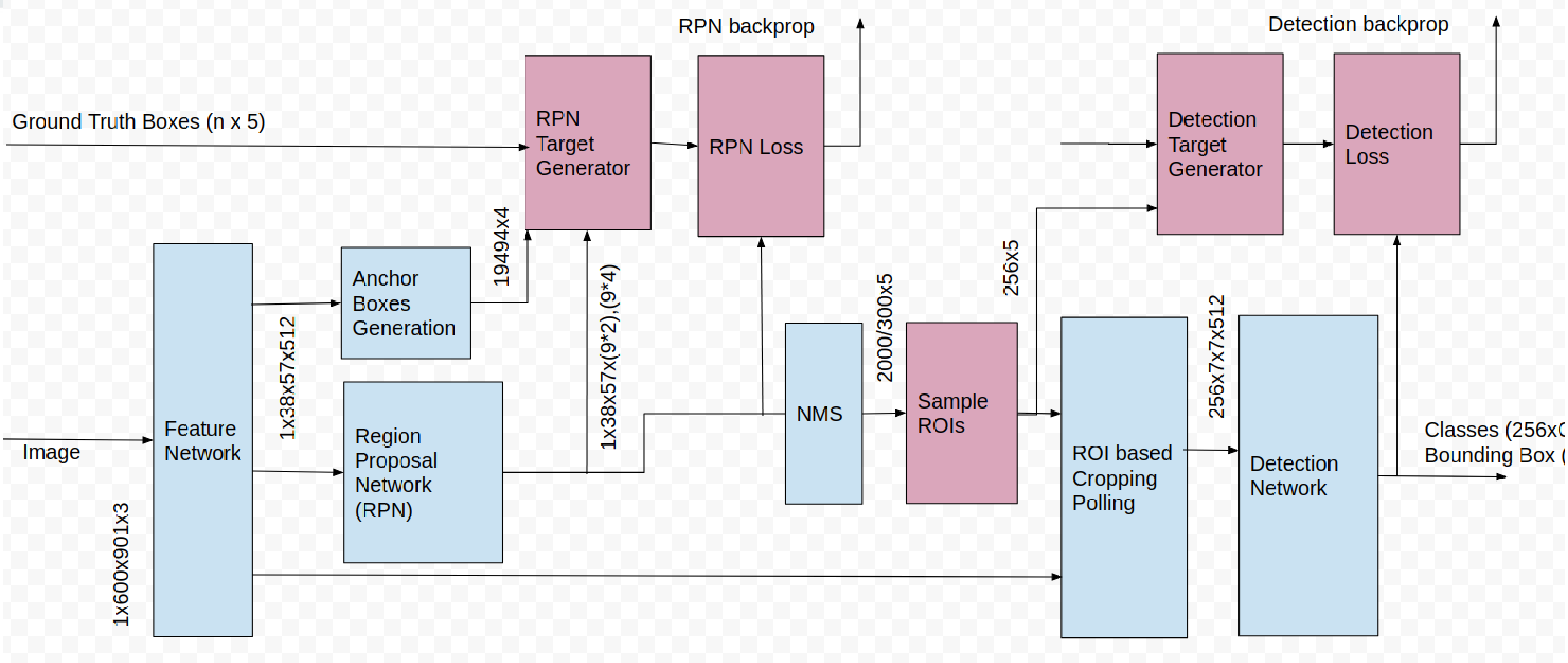


800x800x3



李春煌 FasterRCNN講義 https://youtu.be/2i9CcmJp2yI

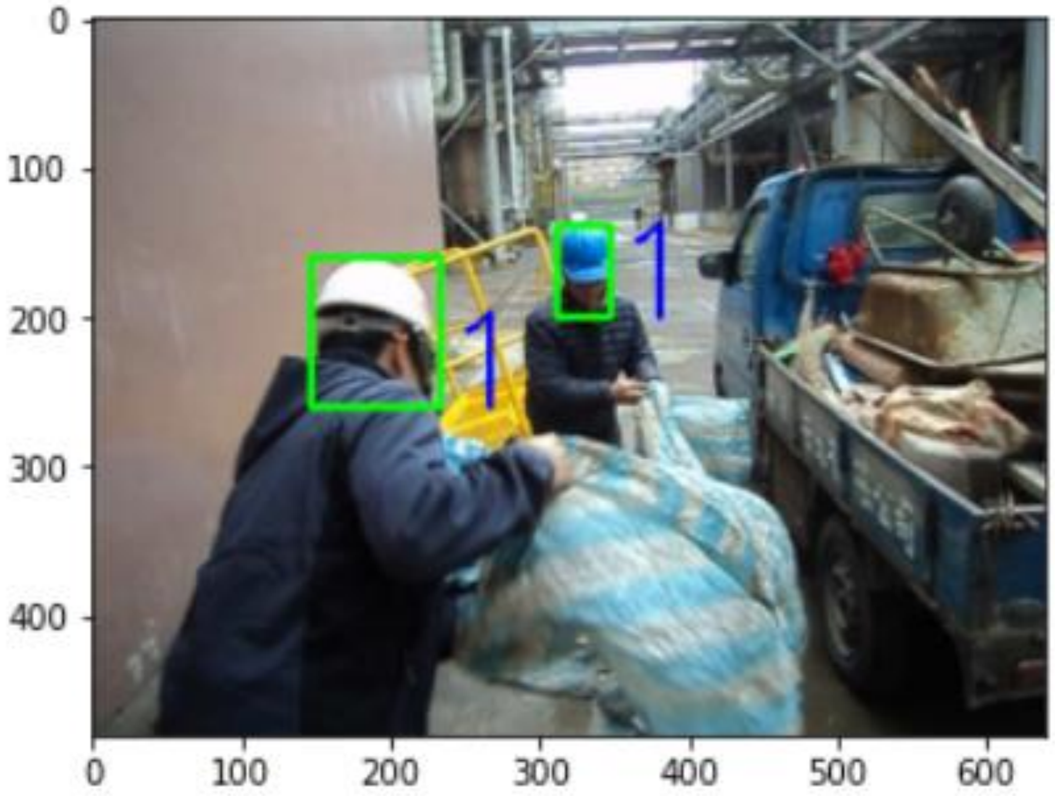
Overall architecture of Faster RCNN



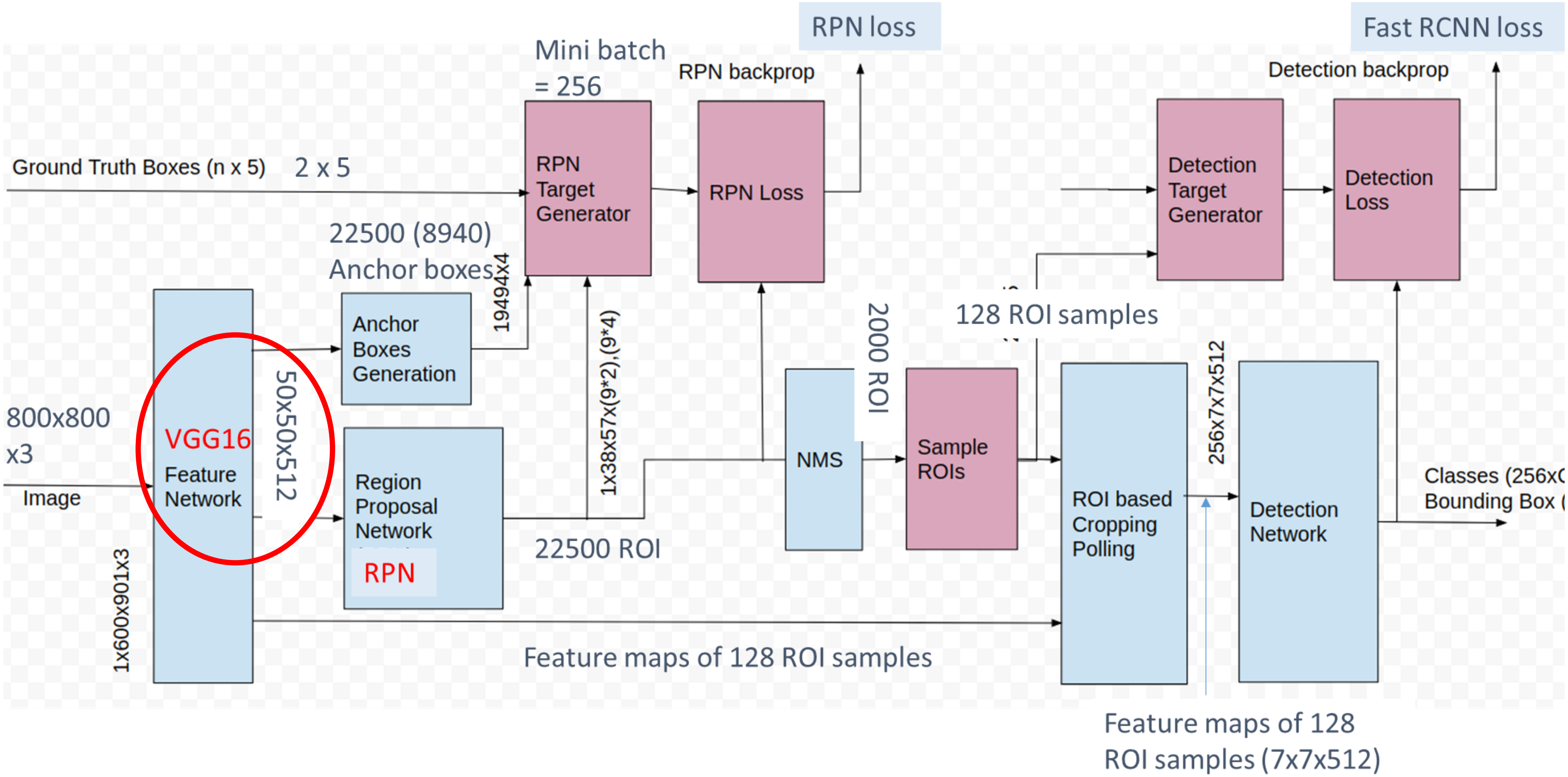
https://medium.com/@whatdhack/a-deeper-look-at-how-faster-rcnn-works-84081284e1cd

Input image and ground truth boxes

[6]:
```python
# Object information: a set of bounding boxes [ymin, xmin, ymax, xmax] and their labels
# If you use your own image, you have to input your own boundary box coordinates
bbox0 = np.array([[160, 147, 260, 234], [139, 312, 200, 348]])
labels = np.array([1, 1]) # 0: background, 1: helmet
```



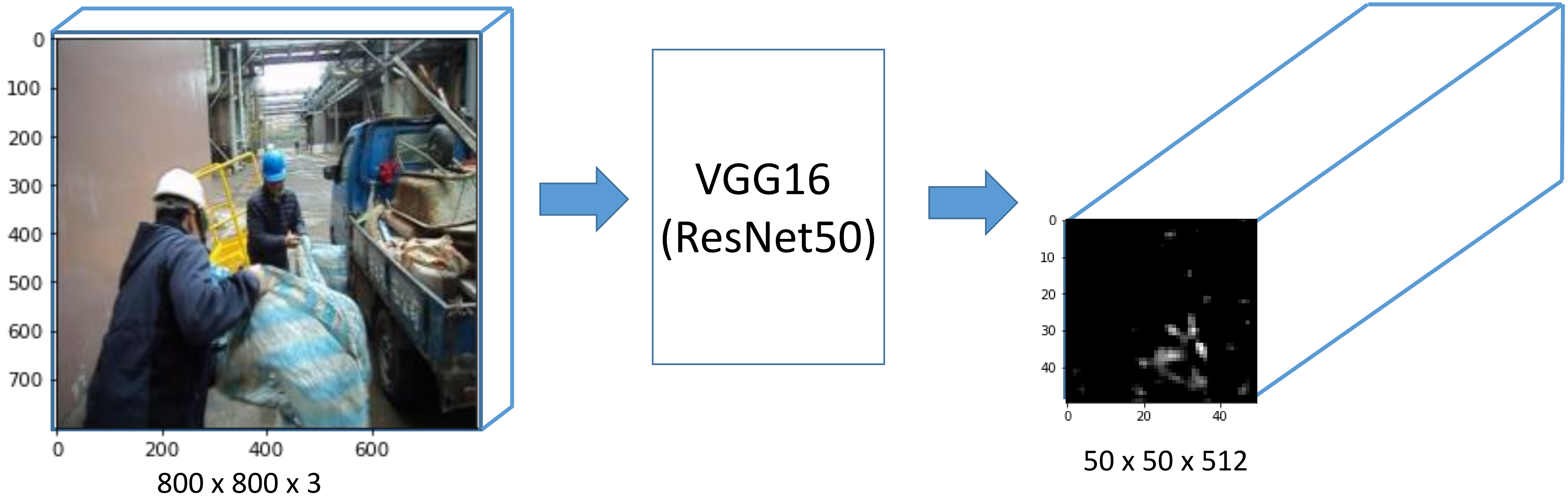
Use a pre-trained CNN to extract image features



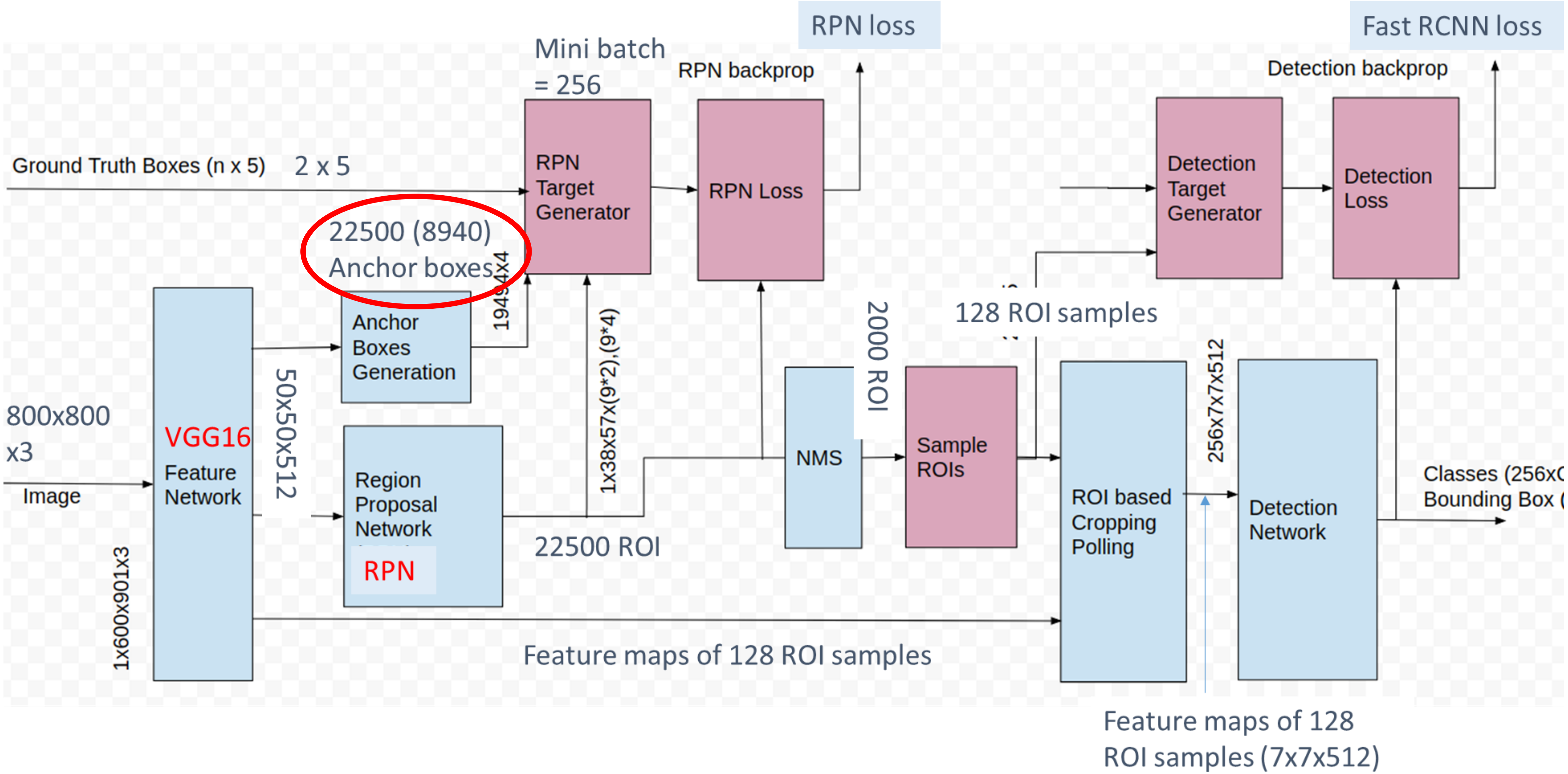
Use a pre-trained CNN to extract image features

[14]:
```python
# Try to pass input image through feature extractor
transform = transforms.Compose([transforms.ToTensor()])
imgTensor = transform(img).to(device)
imgTensor = imgTensor.unsqueeze(0)
out_map = faster_rcnn_fe_extractor(imgTensor)
print(out_map.size())
```

```
torch.Size([1, 512, 50, 50])
```

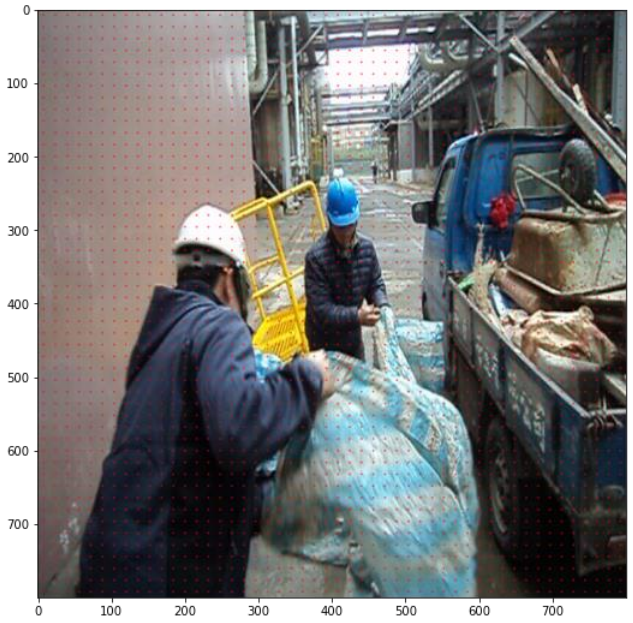


Generate anchors and anchor boxes

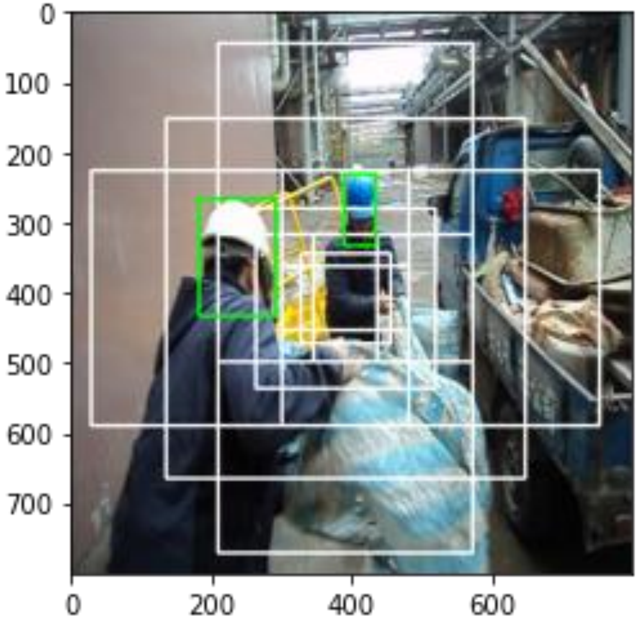


Generate anchors and anchor boxes

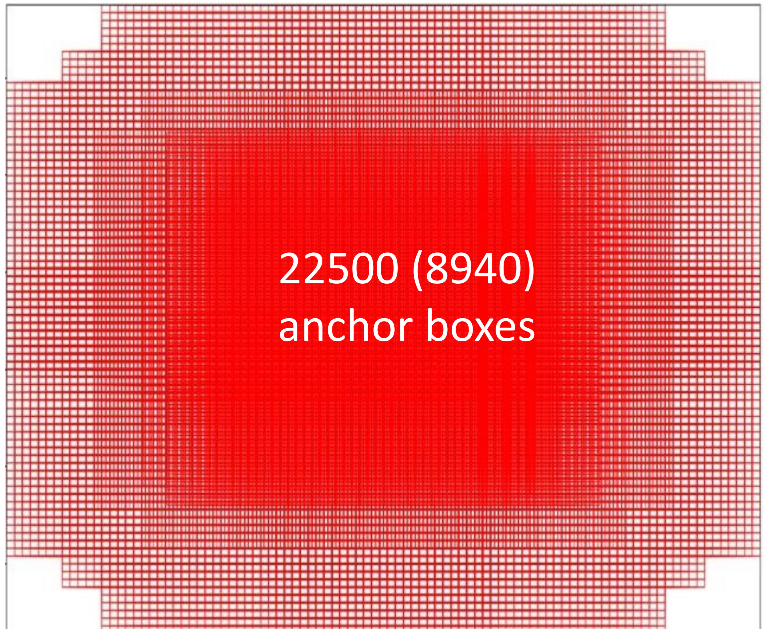
Total number of anchors = 16*16=2500



9 anchor boxes are generated at an anchor point



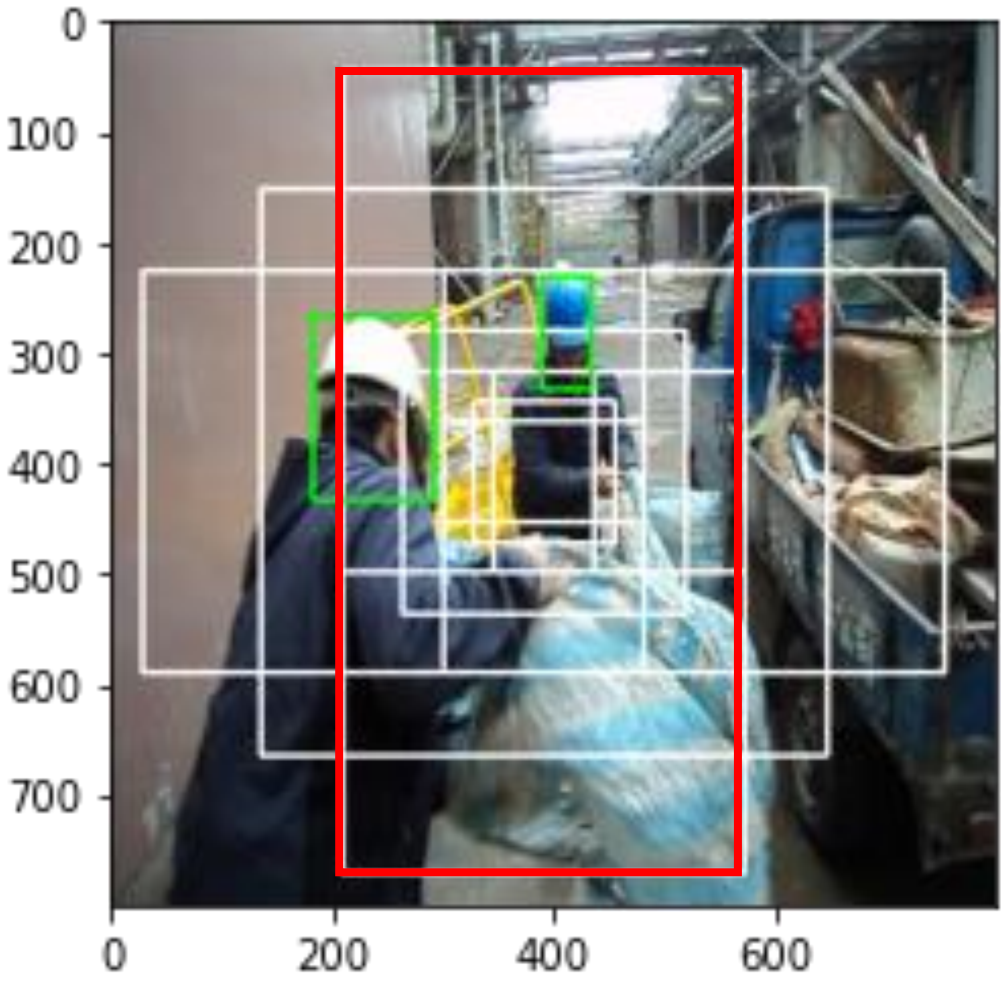
A total of 16*16*9 = 22,500 anchor boxes are generated. After removing cross-boundary ones, 8940 valid anchor boxes are left.

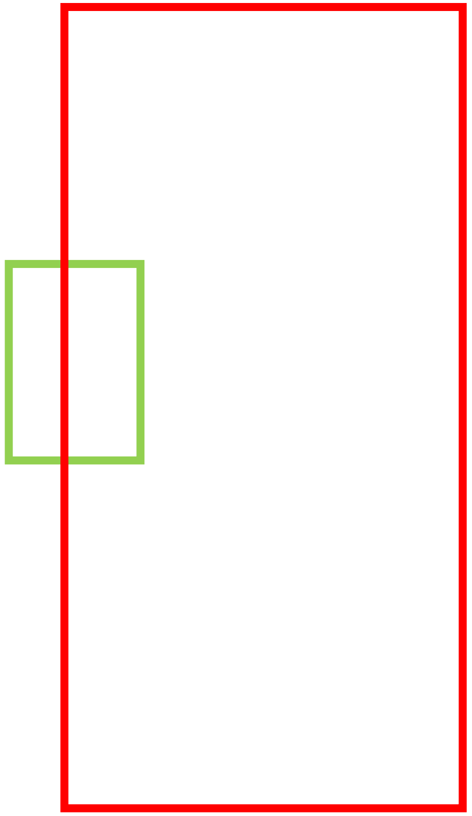


https://zhuanlan.zhihu.com/p/31426458

Calculate IOU

For each valid anchor box, calculate its Intersection over union (IOU) with the given ground truth bounding boxes





$$\text{iou} = \frac{intersection}{union}$$

For each ground truth bbox, record the anchor box with maximum IOU

```
[23]: # What anchor box has max iou with the ground truth bbox
      gt_argmax_ious = ious.argmax(axis=0)
      print(gt_argmax_ious)

      gt_max_ious = ious[gt_argmax_ious, np.arange(ious.shape[1])]
      print(gt_max_ious)

      gt_argmax_ious = np.where(ious == gt_max_ious)[0]
      print(gt_argmax_ious)
```

For each anchor box, record the ground truth bbox with maximum IOU

```
[24]: # What ground truth bbox is associated with each anchor box
      argmax_ious = ious.argmax(axis=1)
      print(argmax_ious.shape)
      print(argmax_ious)
      max_ious = ious[np.arange(len(index_inside)), argmax_ious]
      print(max_ious)
```

RPN target generation

Label the 8,940 valid anchor boxes

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

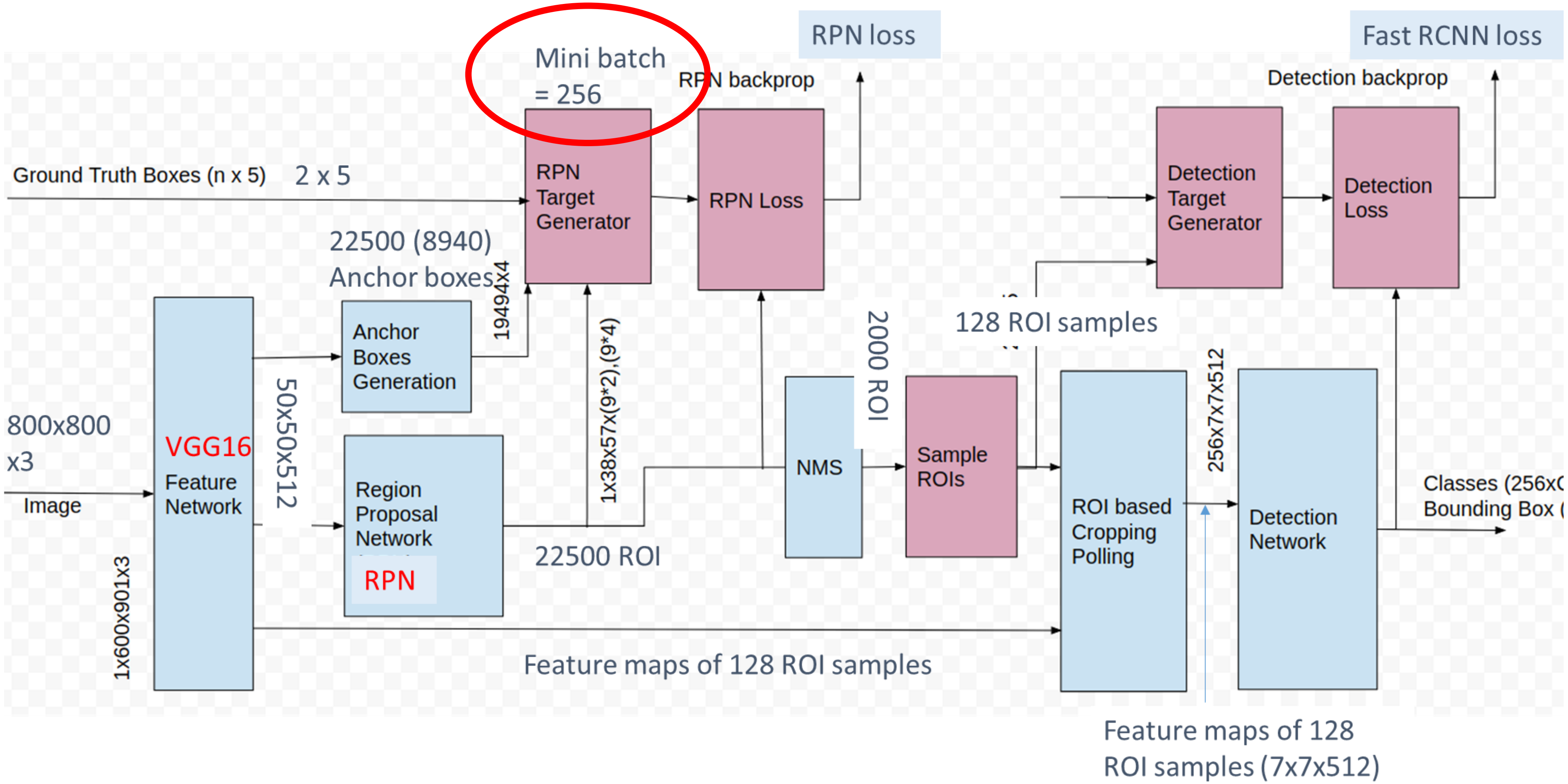
[25]:
```
# Set the labels of all 8940 valid anchor boxes to -1 (ignore) first
label = np.empty((len(index_inside), ), dtype=np.int32)
label.fill(-1)
print(label.shape)
```

(8940,)

[26]:
```
# Use iou to assign 1 (objects) to two kind of anchors
# a) The anchors with the highest iou overlap with a ground-truth-box
# b) An anchor that has an IoU overlap higher than 0.7 with ground-truth box

# Assign 0 (background) to an anchor if its IoU ratio is lower than 0.3 for a
pos_iou_threshold  = 0.7
neg_iou_threshold = 0.3
label[gt_argmax_ious] = 1
label[max_ious >= pos_iou_threshold] = 1
label[max_ious < neg_iou_threshold] = 0
```

RPN target generation

Sample a batch of anchor boxes to generate y for RPN. y= regression value for translating and scaling (no rotation) the anchor box + the classes



RPN target generation

Sample a batch of anchor boxes to generate y for RPN. y= regression value for translating and scaling (no rotation) the anchor box + the classes

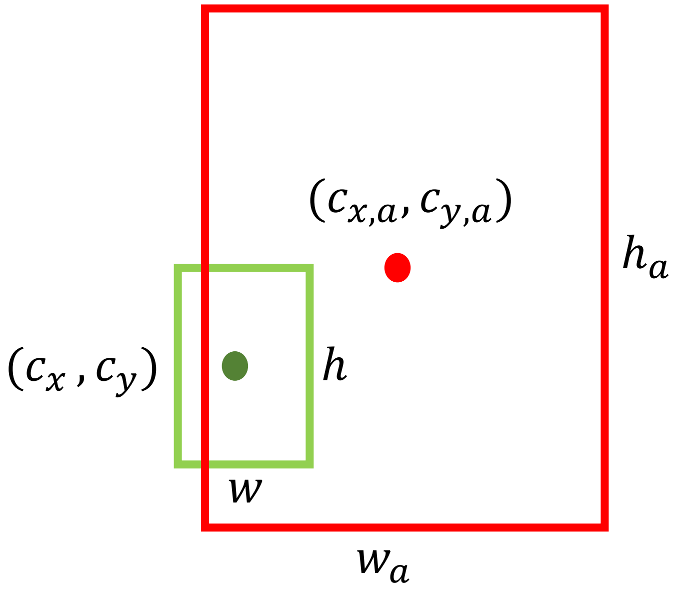
128 positive (label = 1)
128 negative (label = 0)

[27]:

```
n_sample = 256
pos_ratio = 0.5
n_pos = pos_ratio * n_sample

pos_index = np.where(label == 1)[0]
if len(pos_index) > n_pos:
    disable_index = np.random.choice(pos_index, size=(len(pos_index) - n_pos),
    label[disable_index] = -1

n_neg = n_sample * np.sum(label == 1)
neg_index = np.where(label == 0)[0]
if len(neg_index) > n_neg:
    disable_index = np.random.choice(neg_index, size=(len(neg_index) - n_neg),
    label[disable_index] = -1
```

Normalized location representation of anchor box

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



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

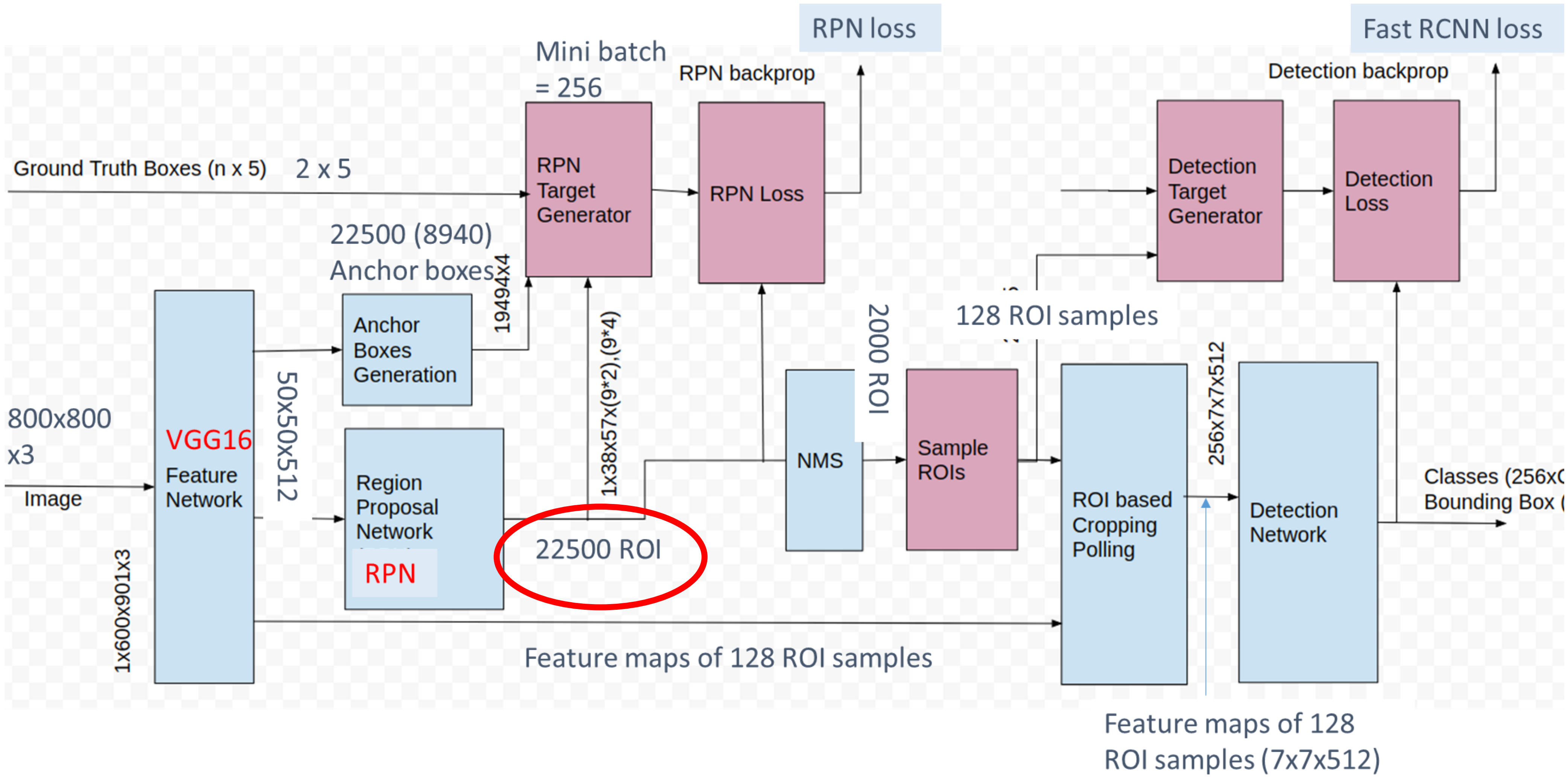
$$d_w = \log(\frac{w}{w_a}) \qquad d_h = \log(\frac{h}{h_a})$$

```
# valid anchor boxes 的 h, w, cx, cy
height = valid_anchor_boxes[:, 2] - valid_anchor_boxes[:, 0]
width = valid_anchor_boxes[:, 3] - valid_anchor_boxes[:, 1]
ctr_y = valid_anchor_boxes[:, 0] + 0.5 * height
ctr_x = valid_anchor_boxes[:, 1] + 0.5 * width

# valid anchor box 的 max iou 的 bbox 的 h, w, cx, cy
base_height = max_iou_bbox[:, 2] - max_iou_bbox[:, 0]
base_width = max_iou_bbox[:, 3] - max_iou_bbox[:, 1]
base_ctr_y = max_iou_bbox[:, 0] + 0.5 * base_height
base_ctr_x = max_iou_bbox[:, 1] + 0.5 * base_width

# valid anchor boxes 的 loc = (y-ya/ha), (x-xa/wa), log(h/ha), log(w/wa)
eps = np.finfo(height.dtype).eps
height = np.maximum(height, eps) #讓 height !=0, 最小值為 eps
width = np.maximum(width, eps)
dy = (base_ctr_y - ctr_y) / height
dx = (base_ctr_x - ctr_x) / width
dh = np.log(base_height / height)
dw = np.log(base_width / width)
anchor_locs = np.vstack((dy, dx, dh, dw)).transpose()
print(anchor_locs.shape)
```

Normalized anchor location representation = the relative distance from the gt-bbox with max. IOU: (dx, dy, dw, dh)

Predict ROIs

Pass the feature map to RPN to predict 22500 region of interests (ROIs)



Predict ROIs

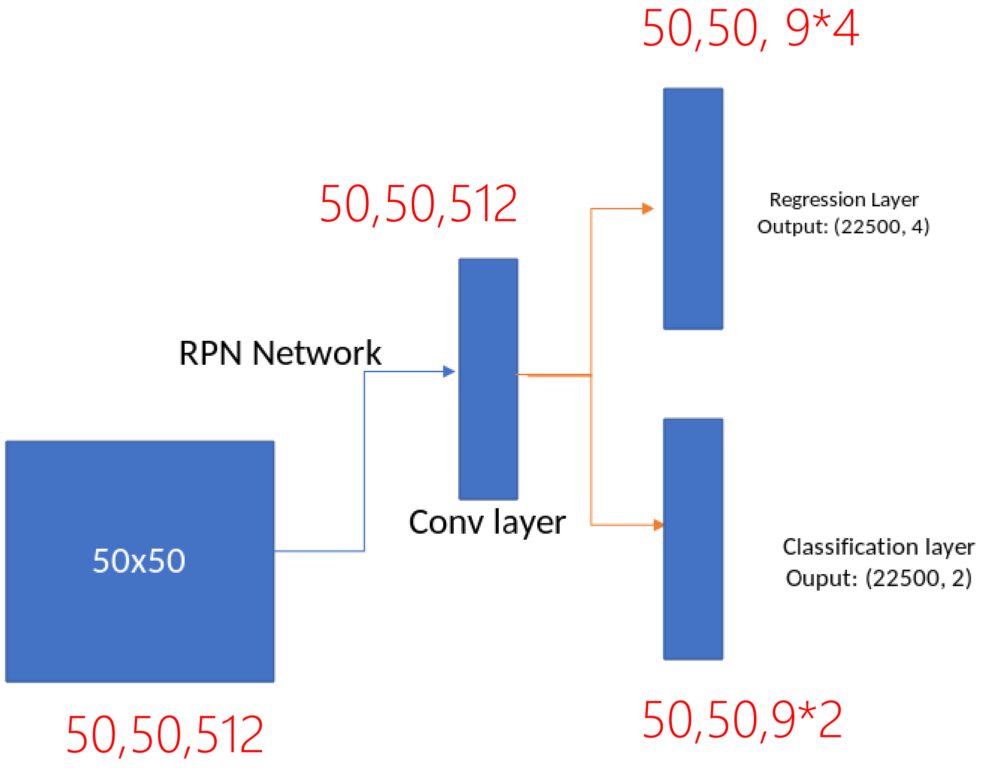
Pass the feature map to RPN to predict 22500 region of interests (ROIs)

[30]:
```
in_channels = 512 # depends on the output feature map. in vgg 16 it is eq
mid_channels = 512
n_anchor = 9  # Number of anchors at each location

conv1 = nn.Conv2d(in_channels, mid_channels, 3, 1, 1).to(device)
conv1.weight.data.normal_(0, 0.01)
conv1.bias.data.zero_()

reg_layer = nn.Conv2d(mid_channels, n_anchor *4, 1, 1, 0).to(device)
reg_layer.weight.data.normal_(0, 0.01)
reg_layer.bias.data.zero_()

cls_layer = nn.Conv2d(mid_channels, n_anchor *2, 1, 1, 0).to(device) ## I
sigmoid if u replace 2 with 1.
cls_layer.weight.data.normal_(0, 0.01)
cls_layer.bias.data.zero_()
```

dx, dy, dw, dh

2 classes

[31]:
```
x = conv1(out_map.to(device)) # out_map = faster_rcnn_
pred_anchor_locs = reg_layer(x)
pred_cls_scores = cls_layer(x)
print(pred_anchor_locs.shape, pred_cls_scores.shape)
```



https://medium.com/@fractaldle/guide-to-build-faster-rcnn-in-pytorch-95b10c273439

Change RPN output format (VERY NICE DESIGN!!)

[32]:
```
# 轉換 RPN 預測 anchor box 的位置與分類之 format
# 位置: [1, 36(9*4), 50, 50] => [1, 22500(50*50*9), 4] (dy, dx, dh, dw)
# 分類: [1, 18(9*2), 50, 50] => [1, 22500, 2]  (1, 0)
pred_anchor_locs = pred_anchor_locs.permute(0, 2, 3, 1).contiguous().view(1, -1, 4)
print(pred_anchor_locs.shape)

pred_cls_scores = pred_cls_scores.permute(0, 2, 3, 1).contiguous()
print(pred_cls_scores.shape)

objectness_score = pred_cls_scores.view(1, 50, 50, 9, 2)[:, :, :, :, 1].contiguous().view(1, -1)
print(objectness_score.shape)

pred_cls_scores  = pred_cls_scores.view(1, -1, 2)
print(pred_cls_scores.shape)
```

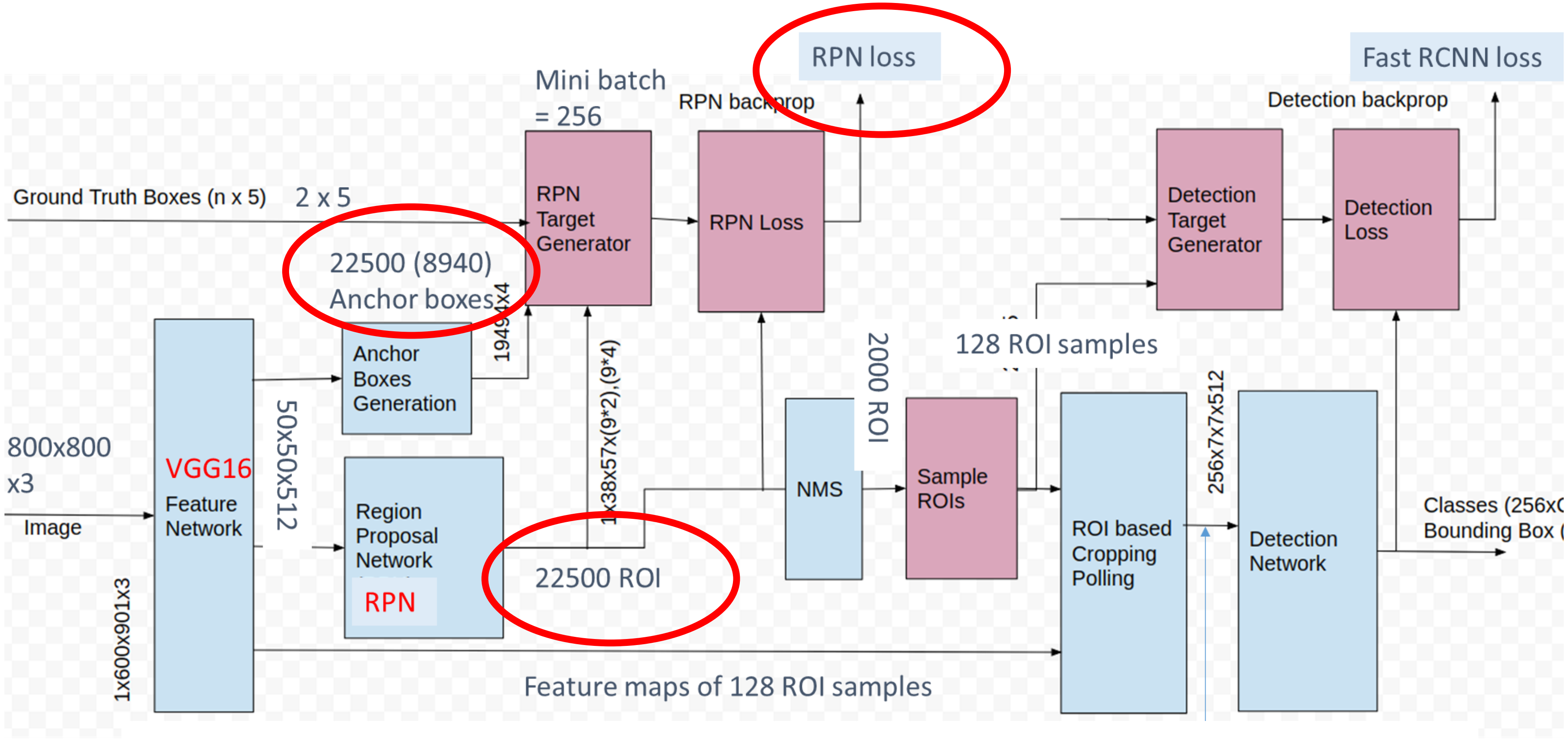
```
torch.Size([1, 22500, 4])
torch.Size([1, 50, 50, 18])
torch.Size([1, 22500])
torch.Size([1, 22500, 2])
```

dx, dy, dw, dh

2 classes, p(c1|x), p(c2|x)

Train RPN

Calculate RPN loss based on the 22,500 ROIs predicted by RPN and the 22,500 true answers calculated from anchor boxes



$$L(p_i, t_i) = (1 / N_{cls}) * \sum_i L_{cls}(p_i, p_i^*) + \lambda * (1 / N_{reg}) * \sum_i p_i^* L_{reg}(t_i, t_{i*})$$

Train RPN

Calculate RPN loss based on the 22,500 ROIs predicted by RPN and the 22,500 true answers calculated from anchor boxes

[35]:
```
# For classification we use cross-entropy loss
rpn_cls_loss = F.cross_entropy(rpn_score, gt_rpn_score.long().to(device), ign
print(rpn_cls_loss)
```

```
tensor(0.6896, grad_fn=<NllLossBackward>)
```

[36]:
```
# 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())
```

```
torch.Size([22500, 4])
torch.Size([4, 4]) torch.Size([4, 4])
tensor(0.1542, dtype=torch.float64, grad_fn=<SumBackward0>)
```

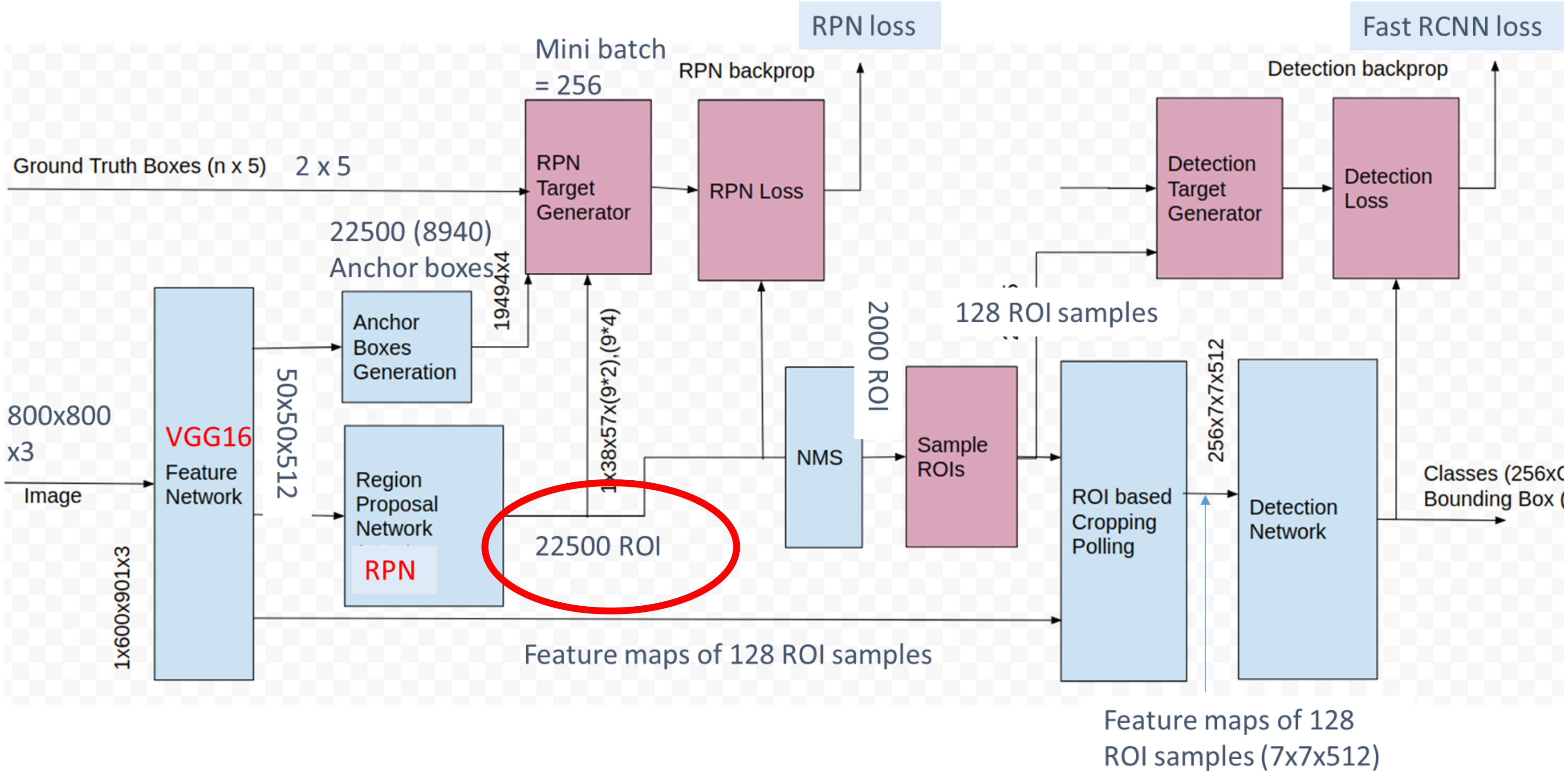
[37]:
```
# Combining both the rpn_cls_loss and rpn_reg_loss
rpn_lambda = 10.
N_reg = (gt_rpn_score >0).float().sum()
rpn_loc_loss = rpn_loc_loss.sum() / N_reg
rpn_loss = rpn_cls_loss + (rpn_lambda * rpn_loc_loss)
print(rpn_loss)
```

```
tensor(1.0751, dtype=torch.float64, grad_fn=<AddBackward0>)
```

$$L(p_i, t_i) = (1 / N_{cls}) * \sum_i L_{cls}(p_i, p_i^*) + \lambda * (1 / N_{reg}) * \sum_i p_i^* L_{reg}(t_i, t_{i*})$$

https://medium.com/@fractaldle/guide-to-build-faster-rcnn-in-pytorch-95b10c273439

Send ROI to Detection Network



Change the ROI format

The 22500 anchor boxes location and labels predicted by RPN format (d_x, d_y, d_h, d_w)

We change the format to: (y_1, x_1, y_2, x_2)

$$d_h = \log(\frac{h}{h_a}) \qquad h = e^{d_h} \times h_a$$

$$y_1 = c_y - 0.5 * h$$

$$x_1 = c_x - 0.5 * w$$

$$y_2 = c_y + 0.5 * h$$

$$x_2 = c_x + 0.5 * w$$

```
ctr_y = dy * anc_height[:, np.newaxis] + anc_ctr_y[:, np.newaxis]
ctr_x = dx * anc_width[:, np.newaxis] + anc_ctr_x[:, np.newaxis]
h = np.exp(dh) * anc_height[:, np.newaxis]
w = np.exp(dw) * anc_width[:, np.newaxis]
print(w.shape)
```

```
(22500,)
(22500, 1)
(22500, 1)
```

[40]:

```
# 用 labelled 的 anchor boxes 與 RPN 預測的 anchor boxes來計算 ROI =
roi = np.zeros(pred_anchor_locs_numpy.shape, dtype=anchor_locs.dtype
roi[:, 0::4] = ctr_y - 0.5 * h
roi[:, 1::4] = ctr_x - 0.5 * w
roi[:, 2::4] = ctr_y + 0.5 * h
roi[:, 3::4] = ctr_x + 0.5 * w
print(roi.shape)
```

Remove ROIs with h, w <16

```
[41]: # Remove predicted boxes with either height or width < threshold.
      hs = roi[:, 2] - roi[:, 0]
      ws = roi[:, 3] - roi[:, 1]
      keep = np.where((hs >= min_size) & (ws >= min_size))[0] #min_size=16
      roi = roi[keep, :]
      score = objectness_score_numpy[keep]
      print(keep.shape, roi.shape, score.shape)

      # Sort all (proposal, score) pairs by score from highest to lowest
      order = score.ravel().argsort()[::-1]
      print(order.shape)

      #Take top pre_nms_topN (e.g. 12000 while training and 300 while testing)
      order = order[:n_train_pre_nms]
      roi = roi[order, :]
      print(order.shape, roi.shape, roi.shape)
```

Sort remaining ROIs according RPN classification scores

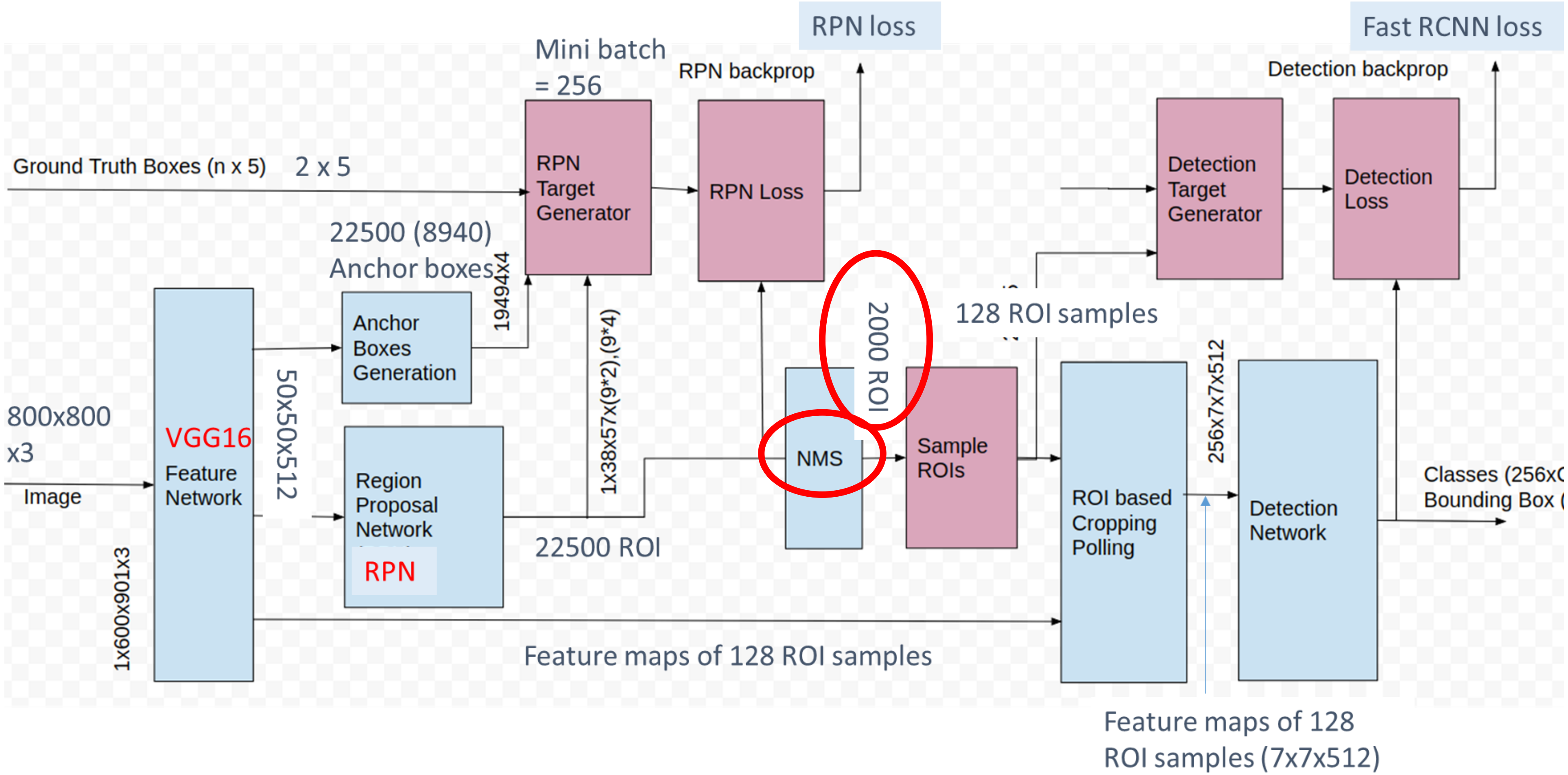
[41]:

```
# Remove predicted boxes with either height or width < threshold.
hs = roi[:, 2] - roi[:, 0]
ws = roi[:, 3] - roi[:, 1]
keep = np.where((hs >= min_size) & (ws >= min_size))[0] #min_size=16
roi = roi[keep, :]
score = objectness_score_numpy[keep]
print(keep.shape, roi.shape, score.shape)

# Sort all (proposal, score) pairs by score from highest to lowest
order = score.ravel().argsort()[::-1]
print(order.shape)

#Take top pre_nms_topN (e.g. 12000 while training and 300 while testing)
order = order[:n_train_pre_nms]
roi = roi[order, :]
print(order.shape, roi.shape, roi.shape)
```

```
(22500,) (22500, 4) (22500,)
(22500,)
(12000,) (12000, 4) (12000, 4)
```

Non-maximum Suppression (NMS)

Use NMS (Non-maximum suppression) to reduce 22,500 ROI to 2,000



Non-maximum Suppression (NMS)

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

Non-maximum Suppression (NMS)

1. Select the proposal with highest confidence score, remove it from B and add it to the final proposal list D. (Initially D is empty).
2. Now compare this proposal with all the proposals — calculate the IOU (Intersection over Union) of this proposal with every other proposal. If the IOU is greater than the threshold N, remove that proposal from B.
3. Again take the proposal with the highest confidence from the remaining proposals in B and remove it from B and add it to D.
4. Once again calculate the IOU of this proposal with all the proposals in B and eliminate the boxes which have high IOU than threshold.
5. This process is repeated until there are no more proposals left in B.

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

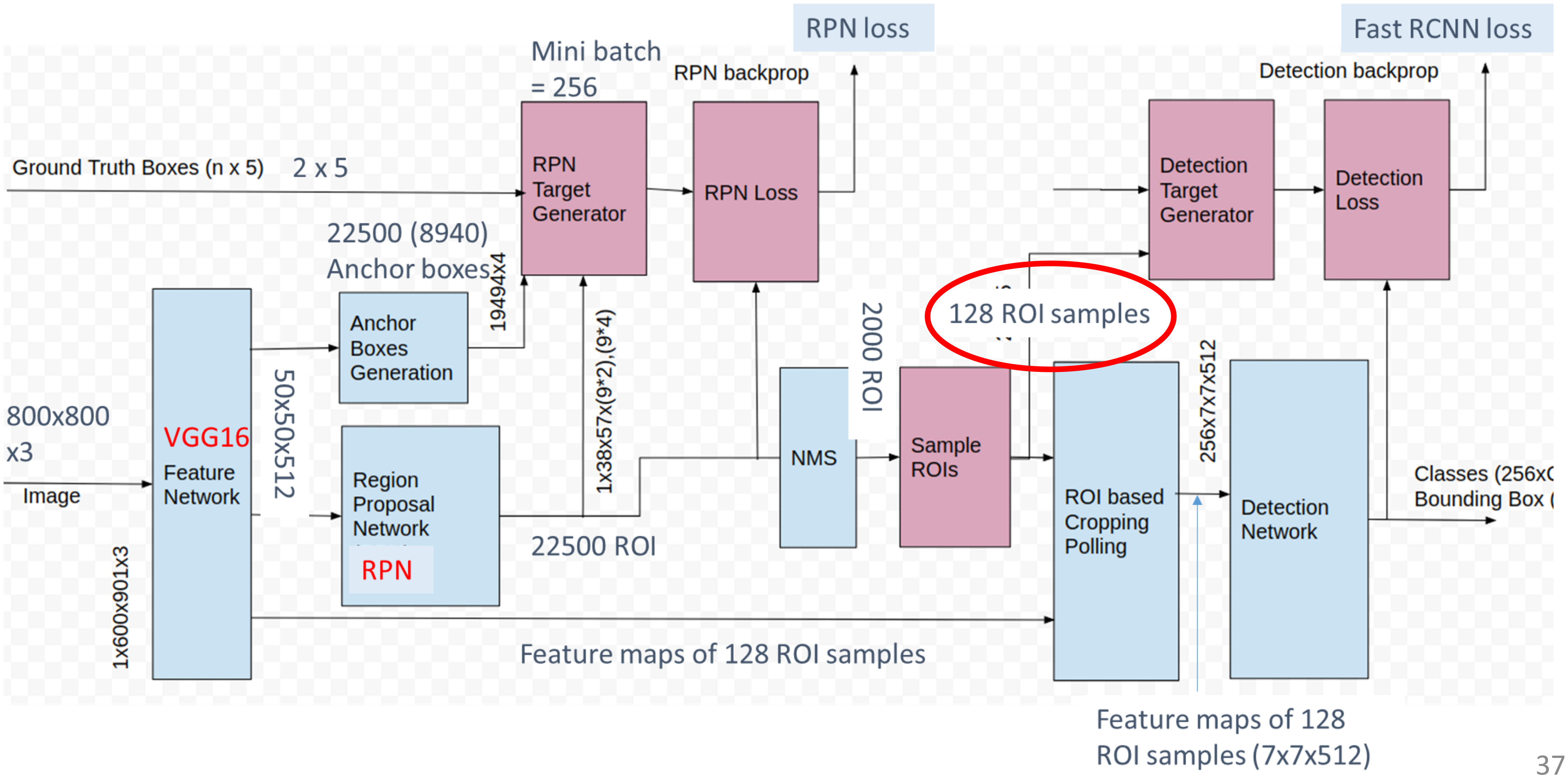
Non-maximum Suppression (NMS)

Use NMS (Non-maximum suppression) to reduce 22,500 ROI to 2,000

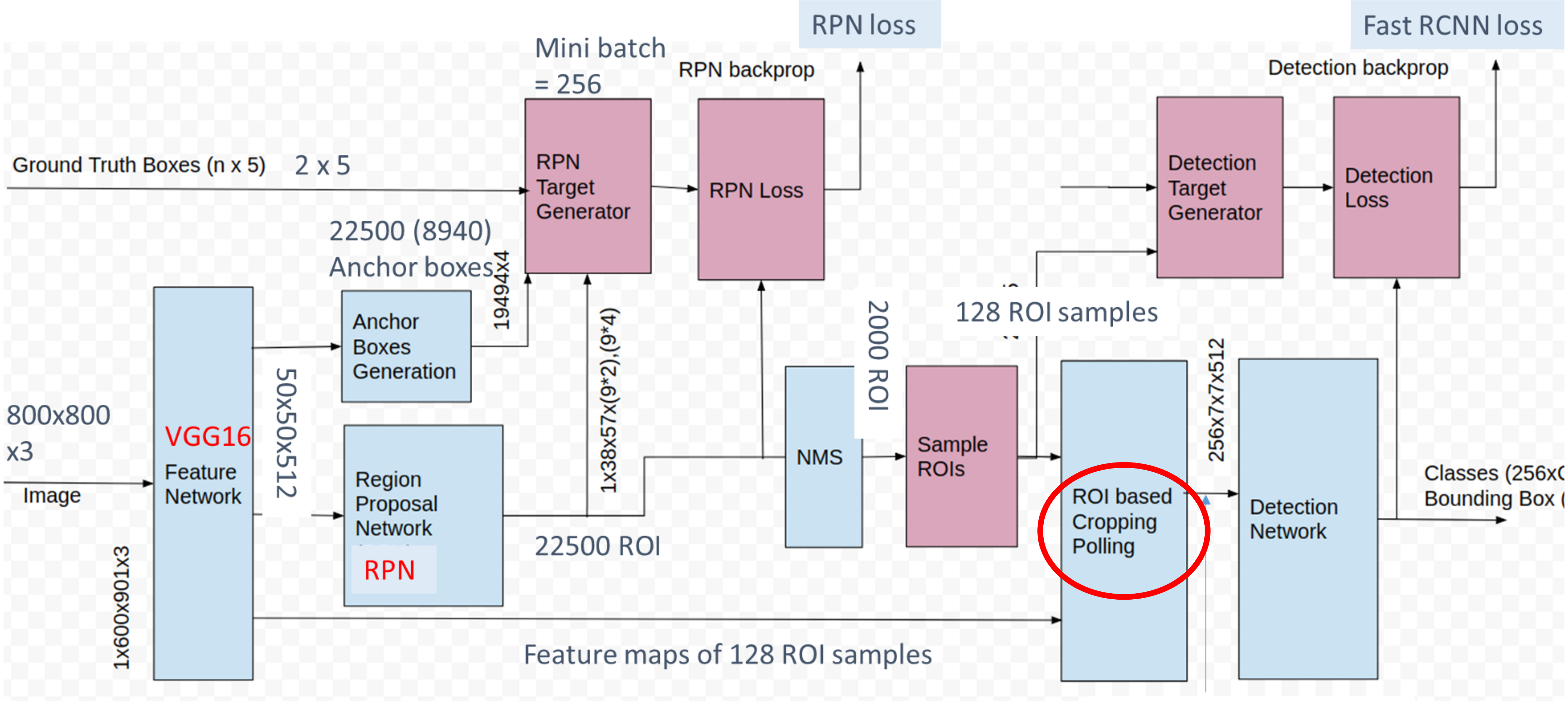
[43]:
```
#Take the indexes of order the probability score in descending order
order = order.argsort()[::-1]
keep = []
while (order.size > 0):
    i = order[0] #take the 1st elt in order and append to keep
    keep.append(i)
    xx1 = np.maximum(x1[i], x1[order[1:]])
    yy1 = np.maximum(y1[i], y1[order[1:]])
    xx2 = np.minimum(x2[i], x2[order[1:]])
    yy2 = np.minimum(y2[i], y2[order[1:]])
    w = np.maximum(0.0, xx2 - xx1 + 1)
    h = np.maximum(0.0, yy2 - yy1 + 1)
    inter = w * h
    ovr = inter / (areas[i] + areas[order[1:]] - inter)
    inds = np.where(ovr <= nms_thresh)[0]
    order = order[inds + 1]
keep = keep[:n_train_post_nms] # while training/testing , use accordir
roi = roi[keep] # the final region proposals
print(len(keep), roi.shape)
```

```
2000 (2000, 4)
```

Based on the 2,000 ROIs filtered by NMS, calculate their IOU with the ground truth boxes to select 128 ROI samples, of which at most 128x0.25=32 positive samples

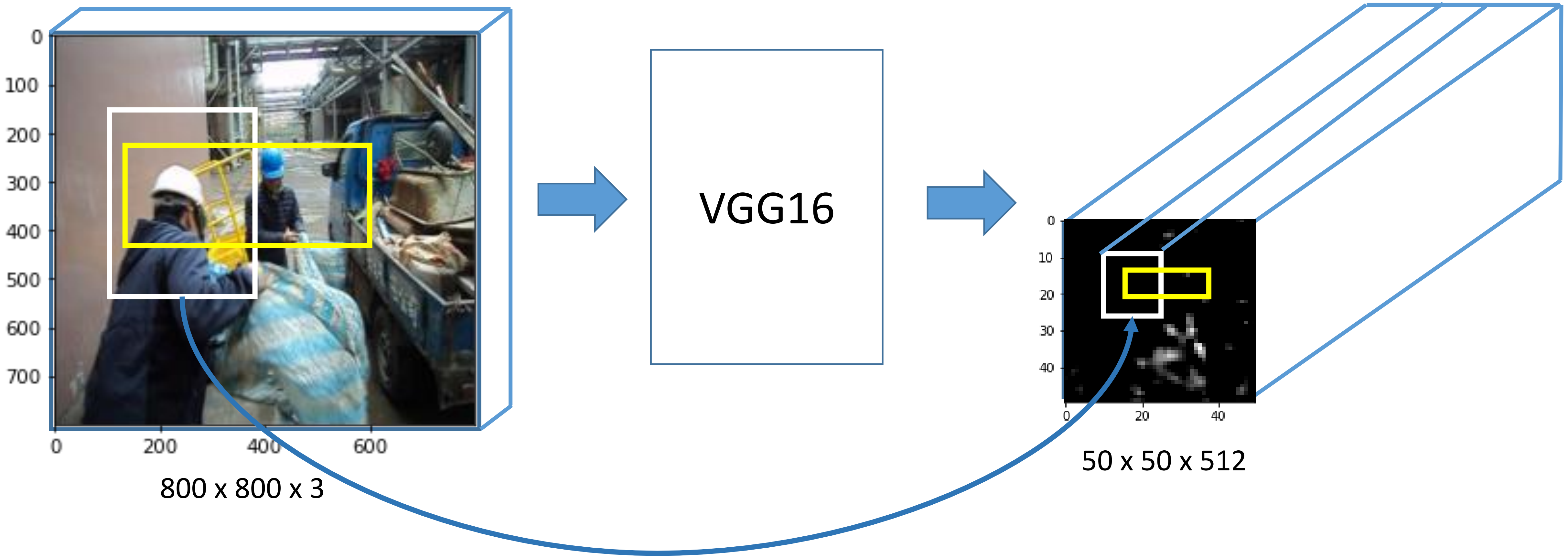


Extract the feature maps of the 128 ROI samples, adjust to the same size H=7, W=7 using max pooling (ROI Pooling)



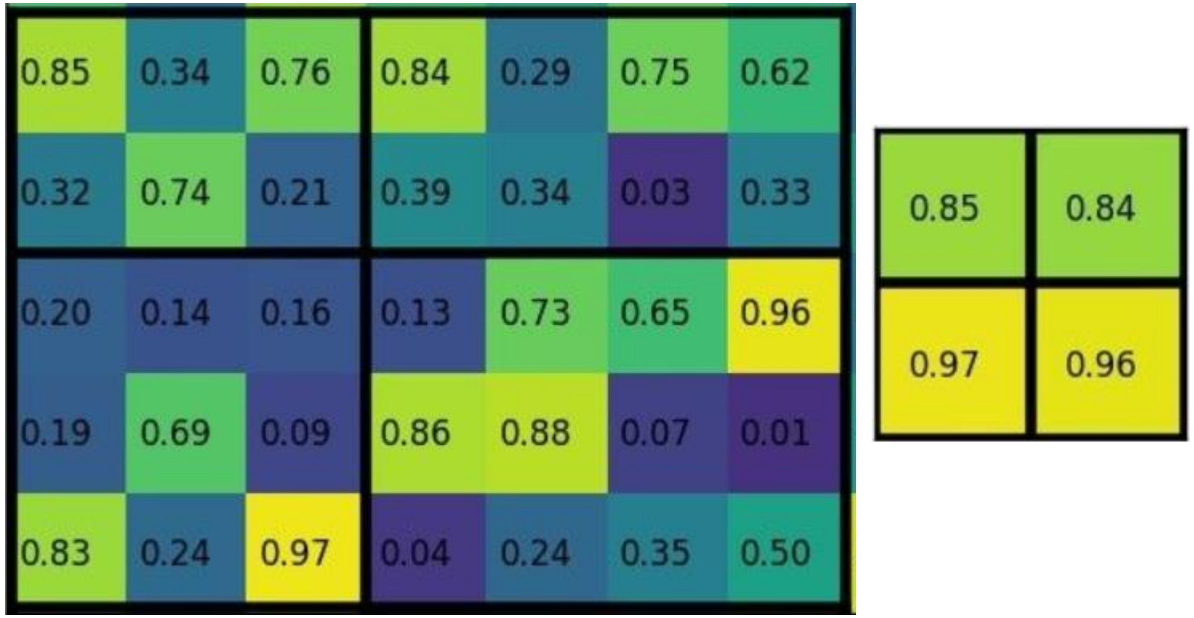
Feature maps of 128 ROI samples (7x7x512)

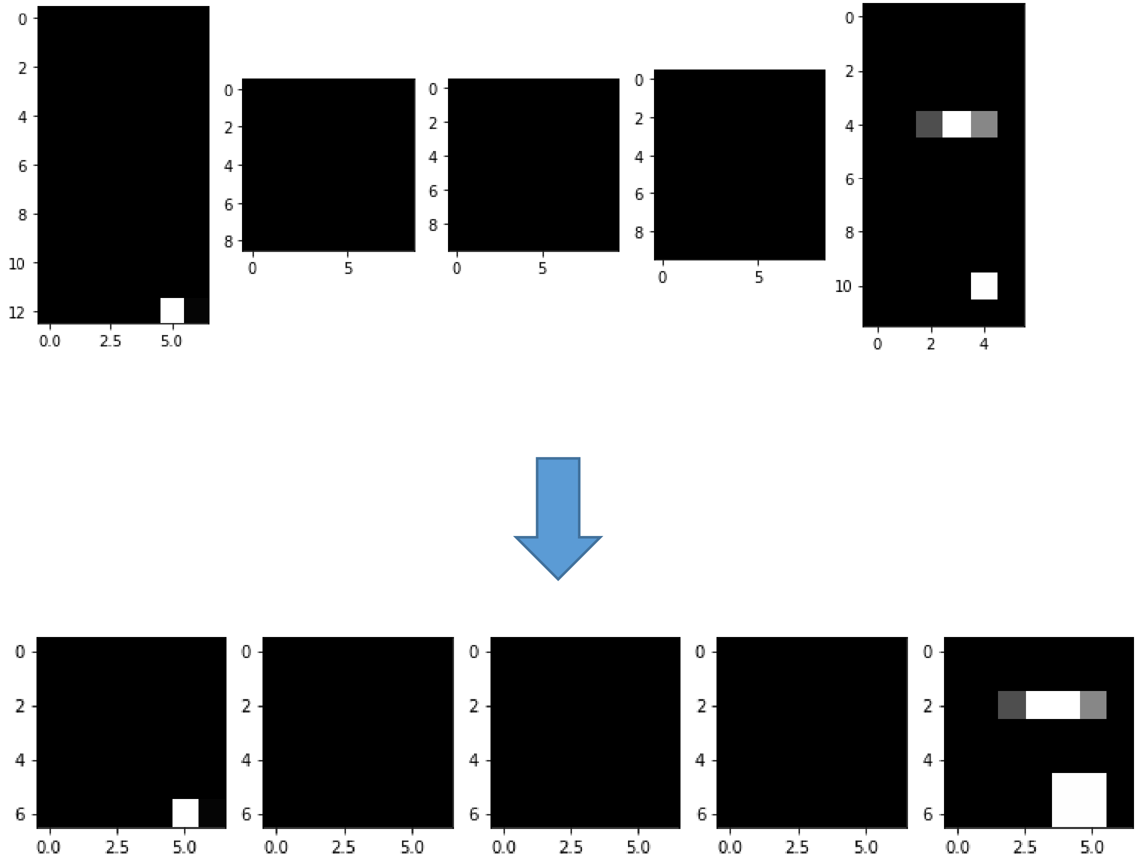
Extract the feature maps of the 128 ROI samples



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)

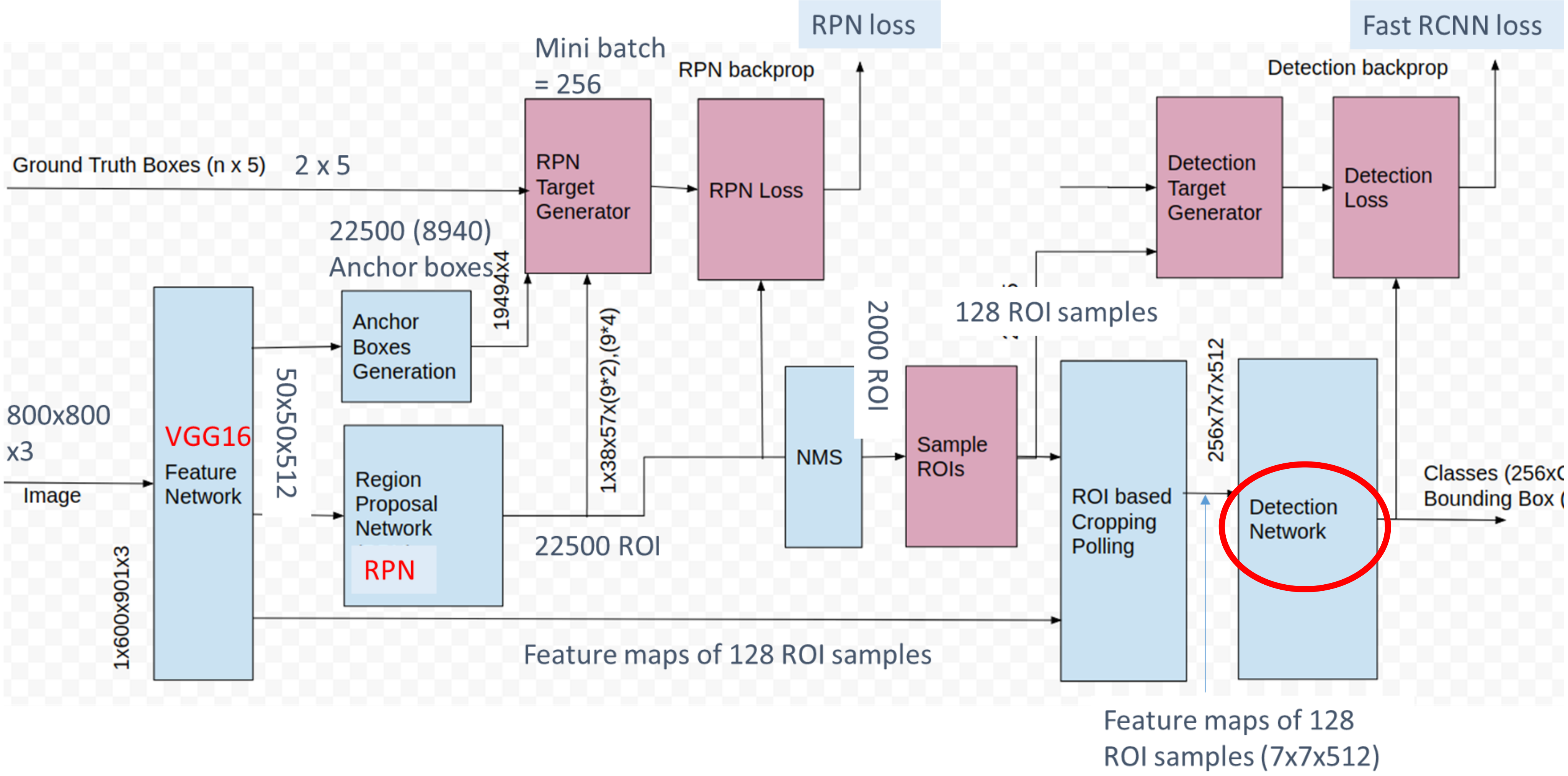




https://blog.csdn.net/qq_35586657/article/details/97885290

Send ROI samples to detection network

Send the boxes + features (7x7x512) of 128 ROI samples to Detection network to predict bounding box and class



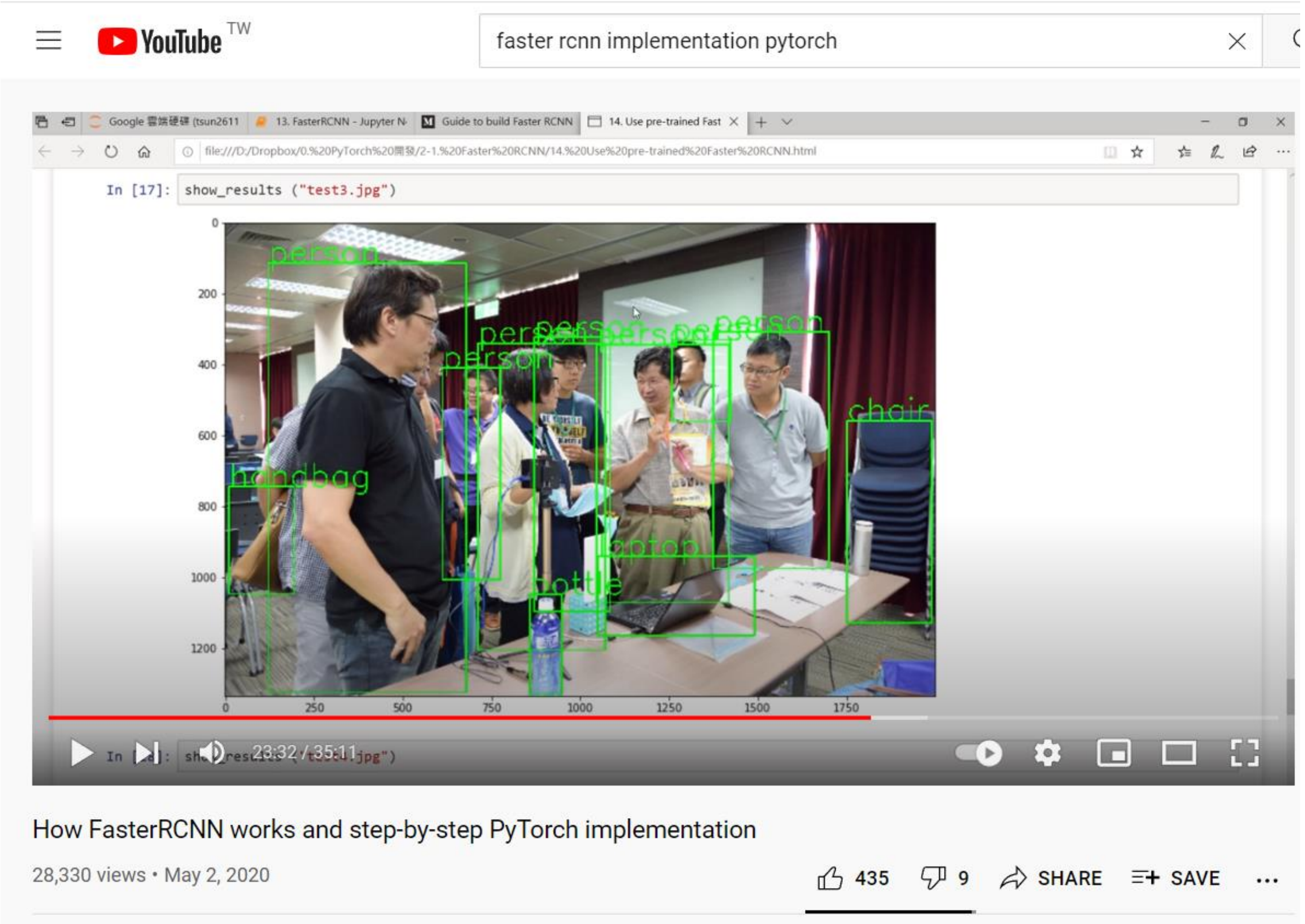
Send the boxes + features (7x7x512) of 128 ROI samples to Detection network to predict bounding box and class

```
[58]: 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 w
      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)
```

```
[59]: # 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])
```

For more details, watch my Youtube video

How Faster RCNN works and step-by-step PyTorch implementation

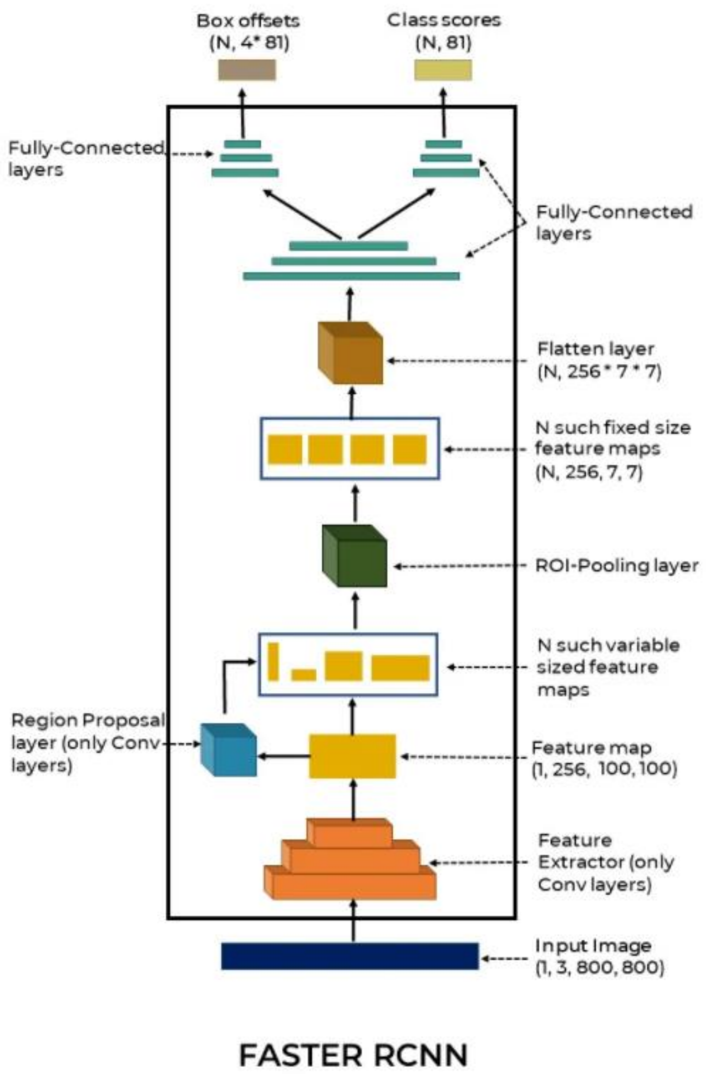


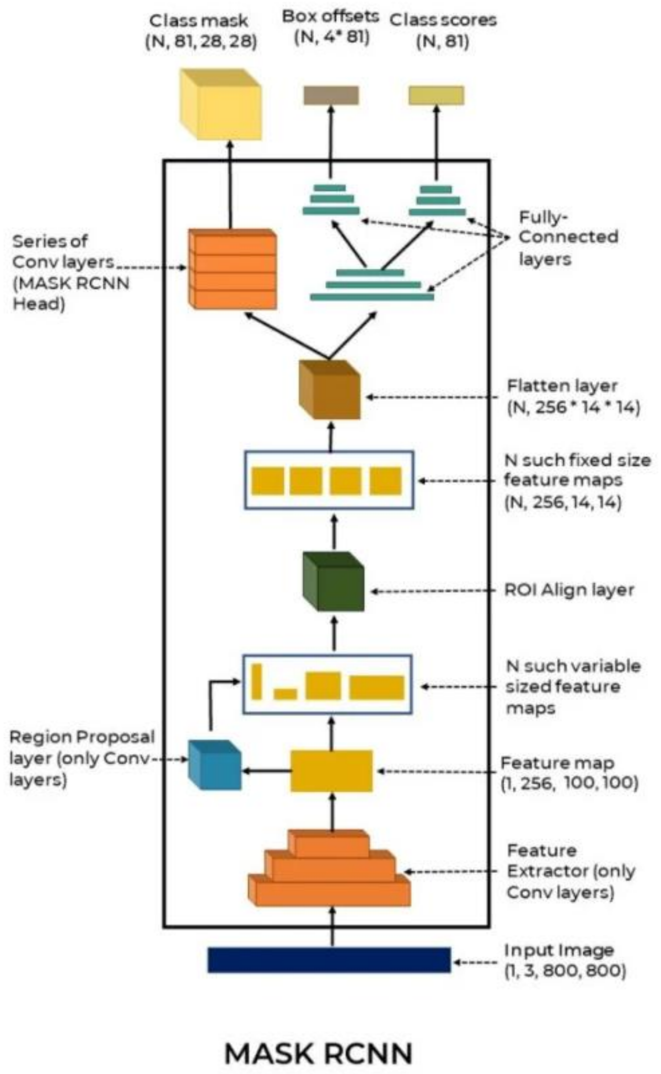
https://youtu.be/4yOcsWg-7g8

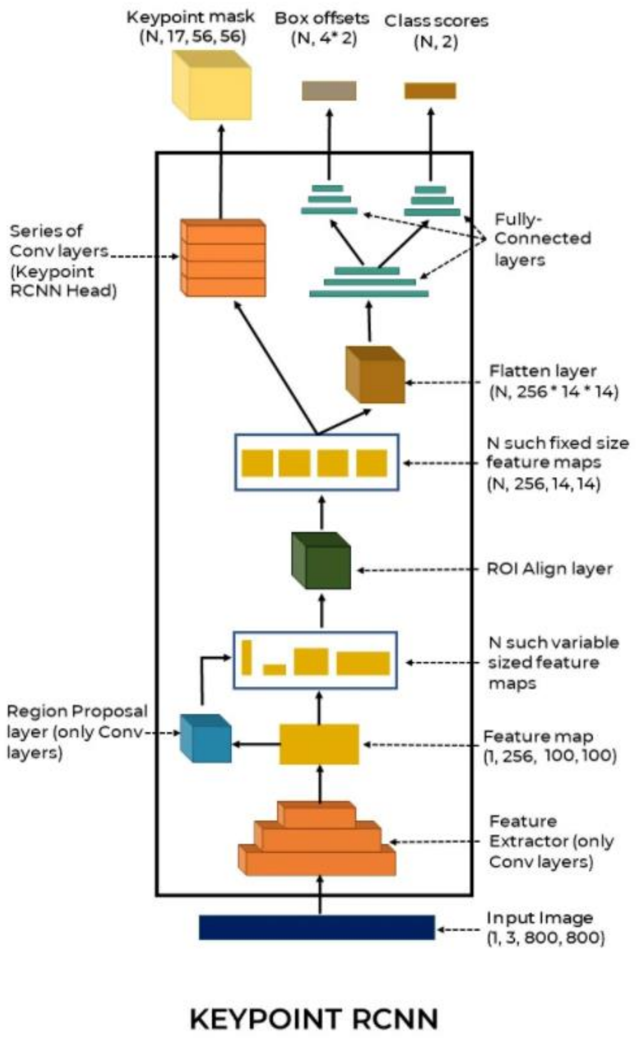
Class practice

- 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.

From Faster RCNN to Mask RCNN and Keypoint RCNN







https://learnopencv.com/human-pose-estimation-using-keypoint-rcnn-in-pytorch/?ck_subscriber_id=297191382