Department of Electrical & Computer Engineering
Faculty of Engineering

Train Component Fault Detection Utilizing Object Detection and Semantic Segmentation

EE5003 Master Project Presentation

Student: *Gu Jiapan*

Supervisor: *Prof. Bharadwaj Veeravalli*

Co-Supervisor: *Dr. Zeng Zeng*

Examiner: *Mr. Rajesh Chandrasekhara Panicker*

CONTENT

1. Background Introduction
2. Two-stage framework
3. Dataset preparation
4. Object detection
5. Fault diagnosis

Industrial Background

- Railway System
  - Convenient and popular transport
  - Long-time operation causes component faults
  - Automatic detection improves efficiency
- TFDS System (Trouble moving Freight car Detection System)
  - Using high-speed line scan cameras to capture images
  - Different views: bottom, roof and two-side
  - Mainly relies on the technicians to inspect and analyze so far

Problem Description

1) Component Displacement

    a. Looseness of locking plate

    b. Closing of valve

    c. dislocation between tube and buckle

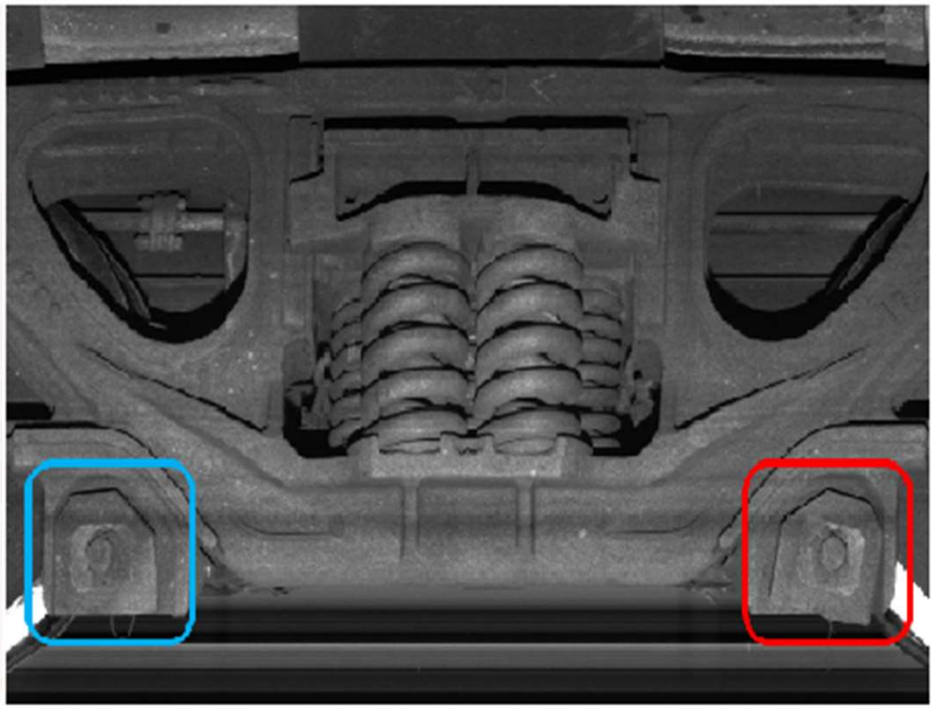
2) Defective Component

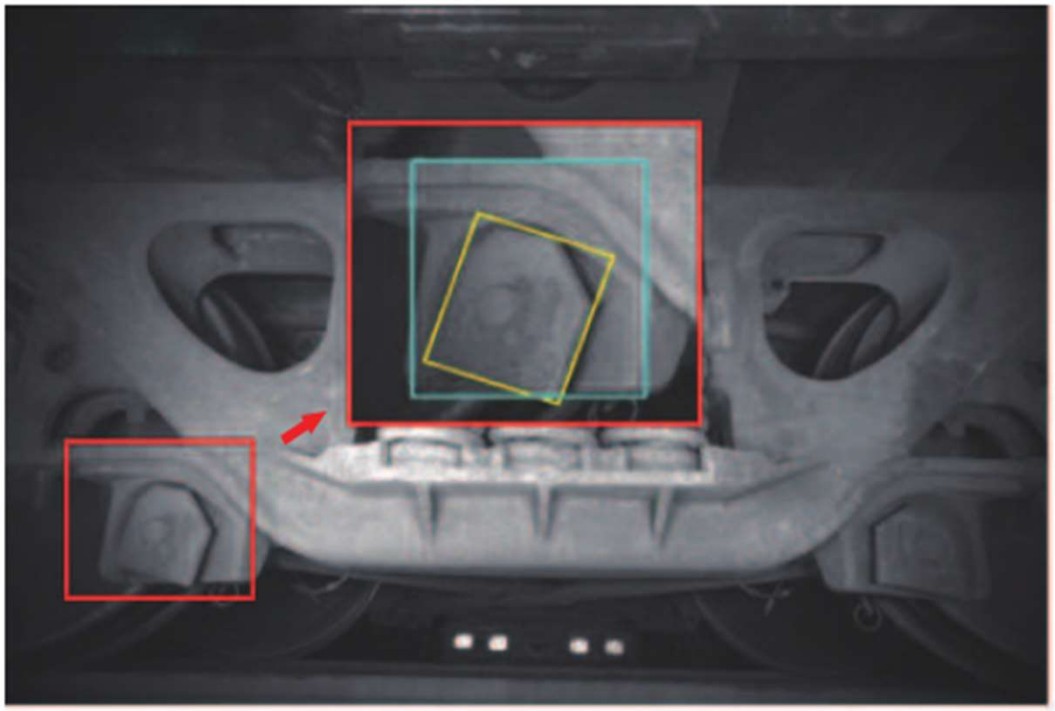
    d. Erosion of bearing and spring

3) Component missing

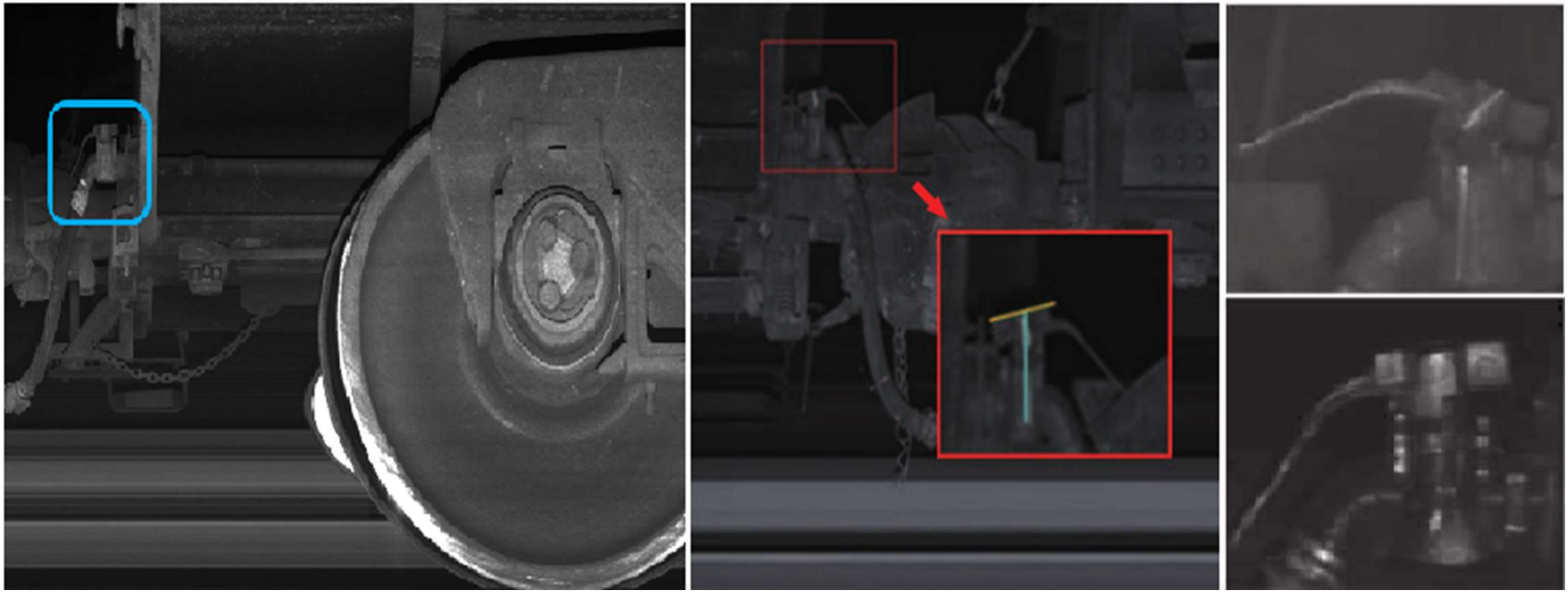
    e. Shedding of screw

a. Looseness of locking plate

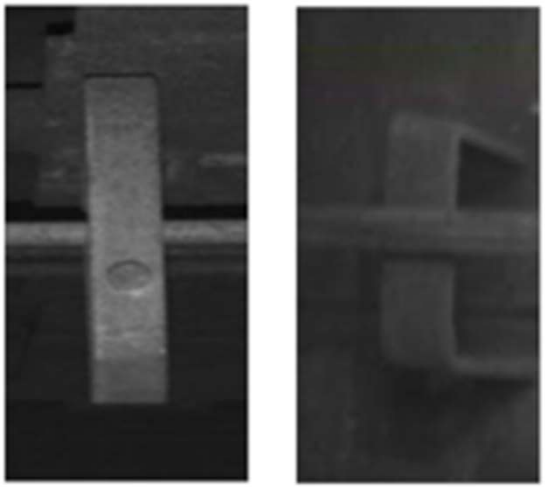




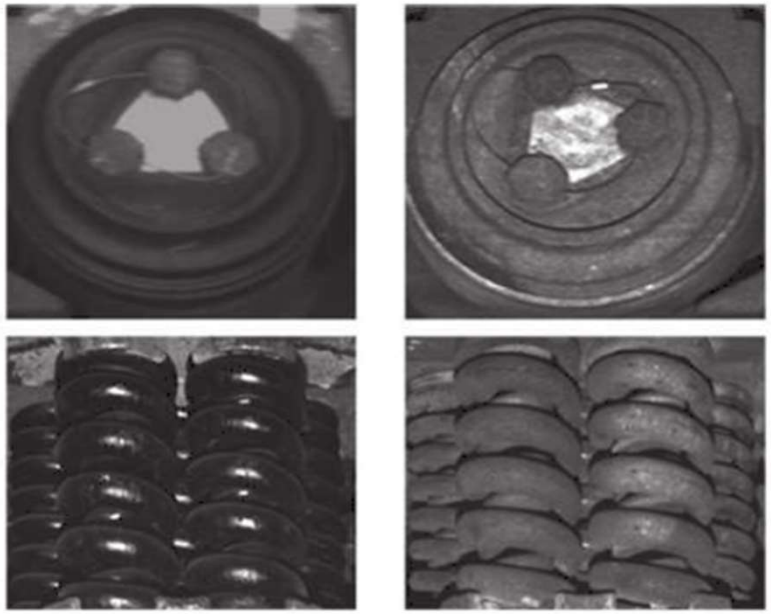
b. Closing of valve



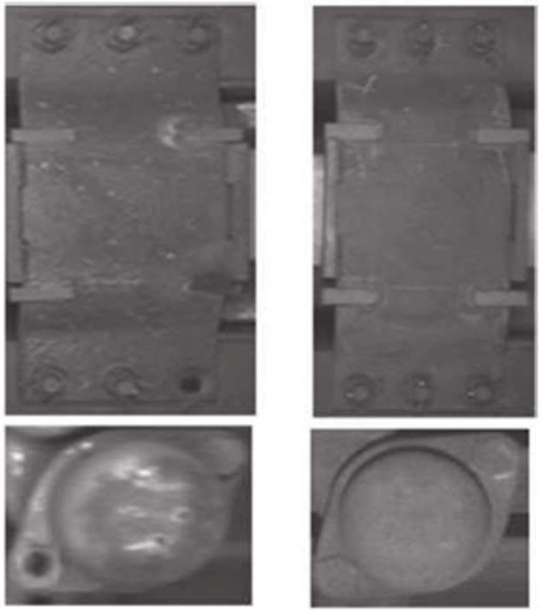
c. Dislocation between tube and buckle



d. Erosion of bearing and spring



e. Shedding of screw



Technical Challenge

1) Very-small component detection

→ Important factor affecting performance

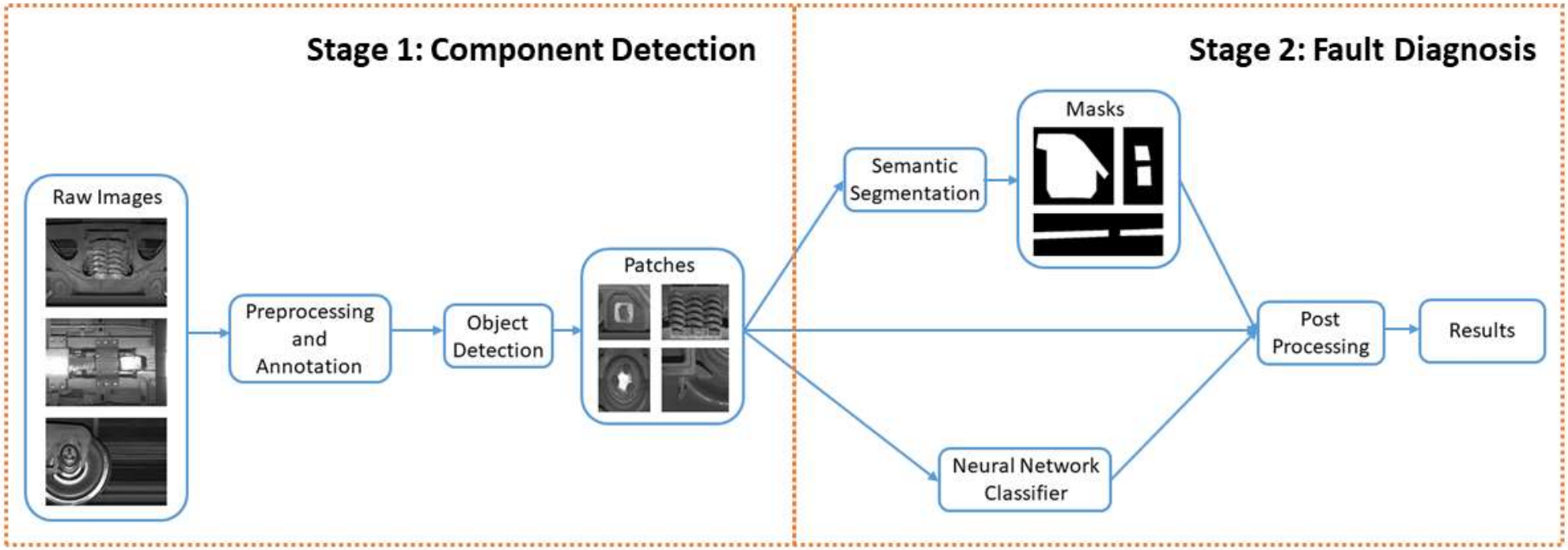
2) Large number of train component fault types

→ Framework consists of multiple methods

3) Limited faulty images

→ Infrequent and imbalanced component faults

Two-stage Framework



Data preparation

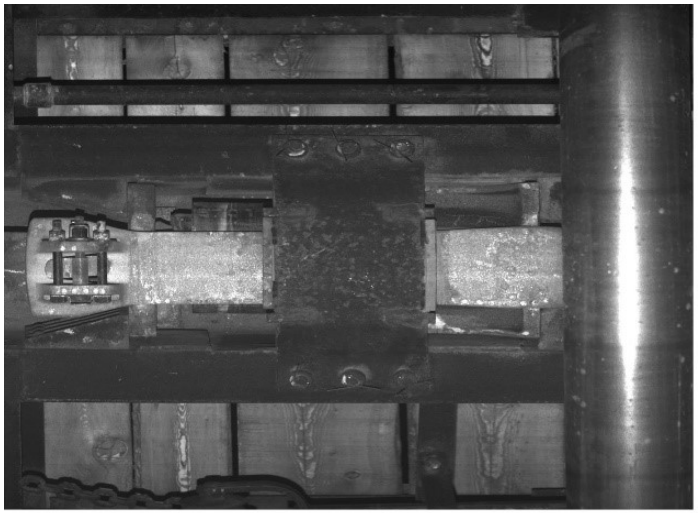
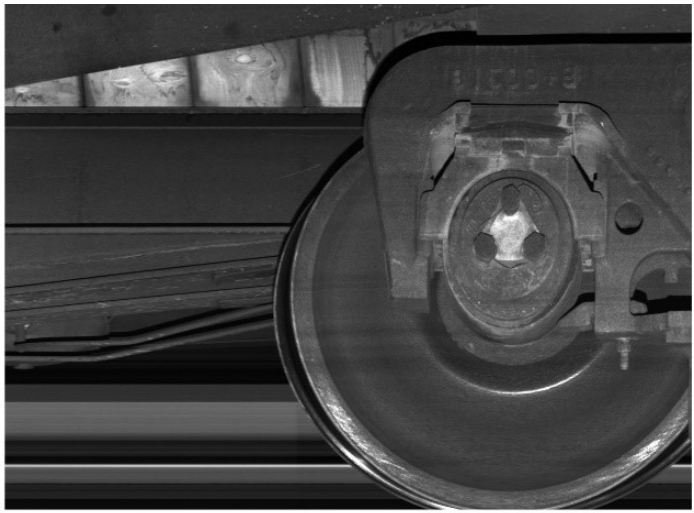
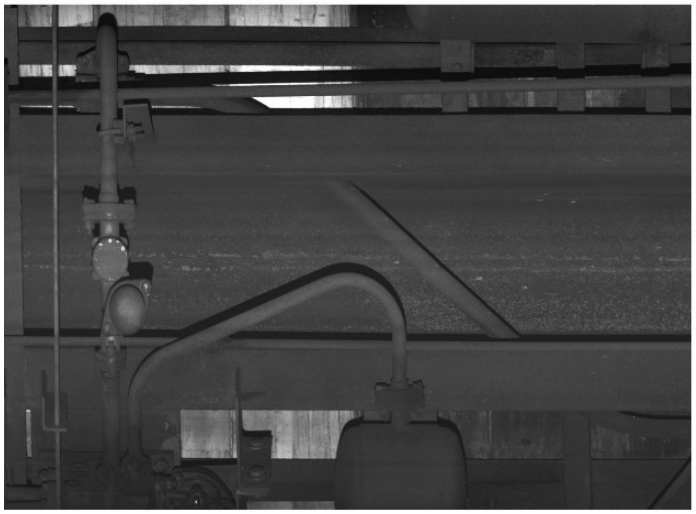
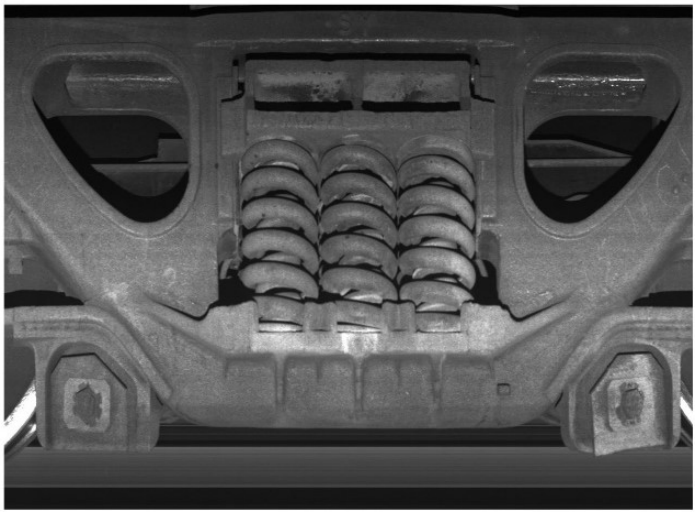





Image Augmentation

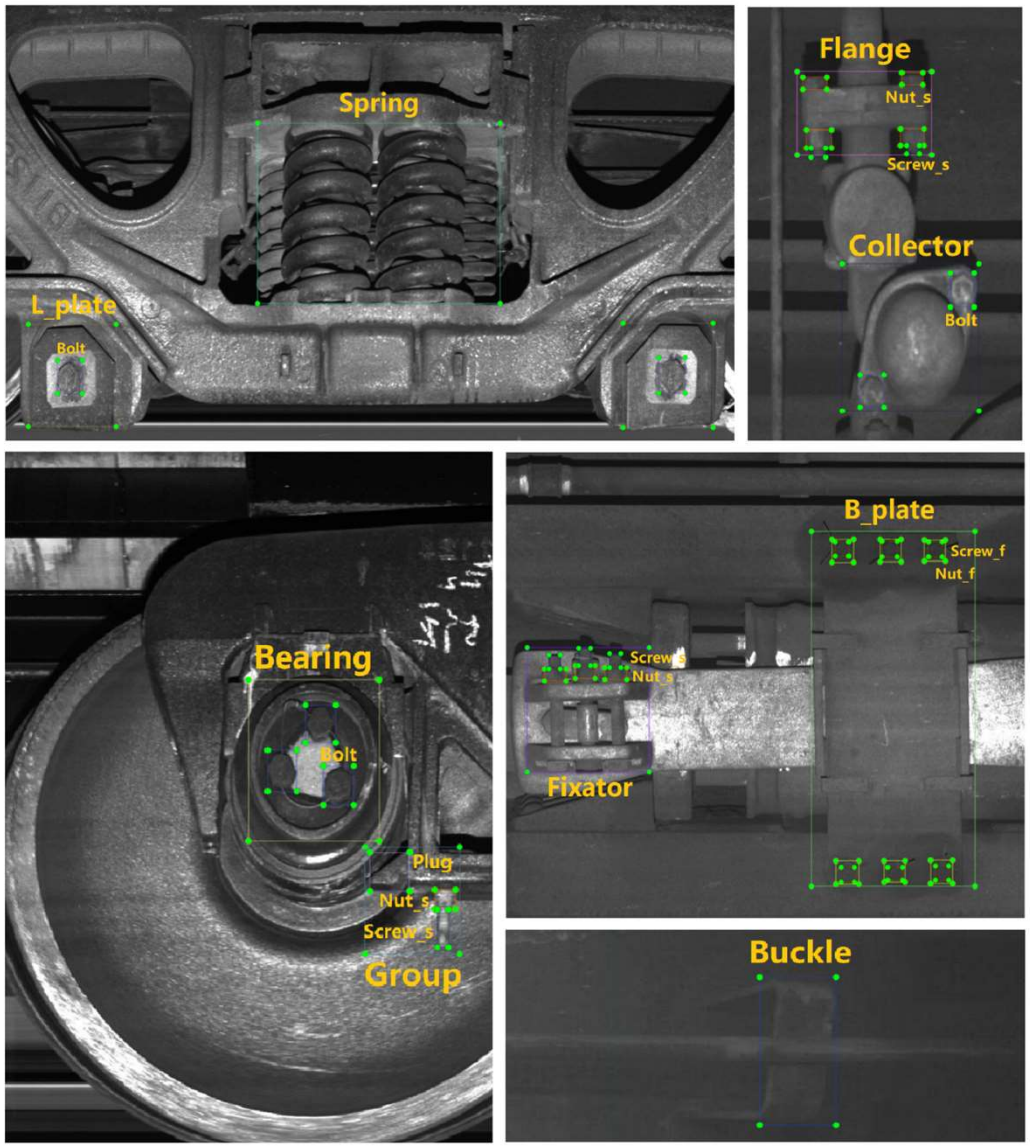
```
from imgaug import augmenters as iaa
seq_img_only = iaa.OneOf(
        [
         ### 1. Add gaussian noise (aka white noise) to images.
         iaa.AdditiveGaussianNoise(scale=0.01*255),
         ### 2. Add random values between -20 and 20 to images
         iaa.AddElementwise((-20, 20)),
         ### 3. Multiply all pixels in an image with a specific value, thereby making the image darker or brighter
         iaa.Multiply((0.5, 1.5)),
         ### 4. Multiply each pixel with a random value between 0.5 and 1.5
         iaa.MultiplyElementwise((0.5, 1.5)),
         ### 4. Sharpen an image, then overlay the results with the original using an alpha between 0.0 and 1.0
         iaa.Sharpen(alpha=(0.0, 1.0), lightness=(0.8, 1.2)),
         ### 5. Augmenter that embosses images and overlays the result with the original image
         iaa.Emboss(alpha=(0.0, 1.0), strength=(0.5, 1.5)),
         ### 6. blur images using gaussiankernels
         iaa.GaussianBlur(sigma=(0.5, 1.5)),
         ### 7. blur images using median kernels
         iaa.MedianBlur(k=(3, 5)),
         ### 8. blur images using average kernals
         iaa.AverageBlur(k=(2, 3)),
         ### 9. Elastic Transformation by moving pixels locally around using displacement fields
         iaa.ElasticTransformation(alpha=(0, 2.0), sigma=0.10),
         ### 10. changes the contrast of images
         iaa.ContrastNormalization((0.8, 1.2))
        ])
```

Label Definition

15 classes of objects:

9 classes of big objects

6 classes of small objects



	Total	Training Set	Testing Set
Images	1051	844	207
Patches	1872	1511	361
Labels	10807	8718	2089
Spring	253	198	55
Buckle	116	93	23
B_plate	315	251	64
Nut_f	2302	1833	469
Screw_f	2302	1833	469
L_plate	504	396	108
Bearing	302	250	52
Collector	178	143	35
Bolt	1755	1421	334
Group	294	246	48
Fixator	107	90	17
Flange	177	142	35
Plug	294	246	48
Nut_s	954	788	166
Screw_s	954	788	166

Big objects without small objects:
Spring, Buckle

Big objects:
B_plate
Small objects included:
Nut_f, Screw_f

Big objects:
L_plate, Bearing, Collector
Small objects included:
Bolt

Big objects:
Group
Small objecs included:
Plug, Nut_s, Screw_s

Big objects:
Fixator, Flange
Small objects included:
Nut_s, Screw_s

Patches generation (for ground truth):

```
for image in train_imgs:
    obj_list = []
    for obj in image['object']:
        if obj['name'] == 'b_plate':
            tree = ET.parse(image['xmlname'])
            root = tree.getroot()
            img = Image.open(image['filename'])
            img4 = img.crop((obj['xmin']-50, obj['ymin']-50, obj['xmax']+50, obj['ymax']+50))
            img4.save(save_pth + str(num) + ".jpg")
            annotation = Annotations.Annotation(root.find('folder').text, root.find('filename').text,root.find('path').text,
                                                obj['xmax'] - obj['xmin']+100, obj['ymax'] - obj['ymin']+100)
            lieche_object = Annotations.liecheObject(obj['name'], 50, 50, obj['xmax']-obj['xmin']+50, obj['ymax']-obj['ymin']+50)
            annotation.add_beer_object(lieche_object)
            for obj1 in image['object']:
                if  obj1['name'] == 'Screw_f' or obj1['name'] == 'Nut_f':
                    if obj1['xmin'] > (obj['xmin']-50) and obj1['ymin'] > (obj['ymin']-50) \
                            and obj1['xmax'] < (obj['xmax']+50) and obj1['ymax'] < (obj['ymax']+50):
                        xmin = obj1['xmin'] - obj['xmin']+50
                        ymin = obj1['ymin'] - obj['ymin']+50
                        xmax = xmin + obj1['xmax'] - obj1['xmin']
                        ymax = ymin + obj1['ymax'] - obj1['ymin']
                        lieche_object = Annotations.liecheObject(obj1['name'], xmin, ymin, xmax, ymax)
                        annotation.add_beer_object(lieche_object)
            annotation.save(annotation_saved_folder  +str(num) + '.xml')
            num += 1
```

Statistics:

```
for xmlFile in files:
    tree=ET.parse(os.path.join(input_path, xmlFile))
    root=tree.getroot()
    f = f + 1
    for i in root:
        for j in i:
            if (j.tag=='name'): # &(j.text=='Spring'):
                sum = sum + 1
                if j.text not in label:
                    label.update({j.text:1})
                else:
                    label[j.text]+=1
print("file:",f)
print("label:",sum)
print(len(label))
for i in label:
    print(i,":",label[i])
```

Modification:

```
for i in root:
    for j in i:
        if (j.tag=='name'):
            if (j.text=='A'):
                j.text = 'B'  # change a label
            elif (j.text == 'C'):
                root.remove(i)  # delete a label
            break
tree.write(os.path.join(output_path,  xmlFile))
```

Object Detection

- Baseline: Faster RCNN
- Backbone: ResNet101
- Improvements:
    a. Two-branch hierarchical scheme
    b. Feature Pyramid Networks (FPN)

Two main categories for Object Detection:

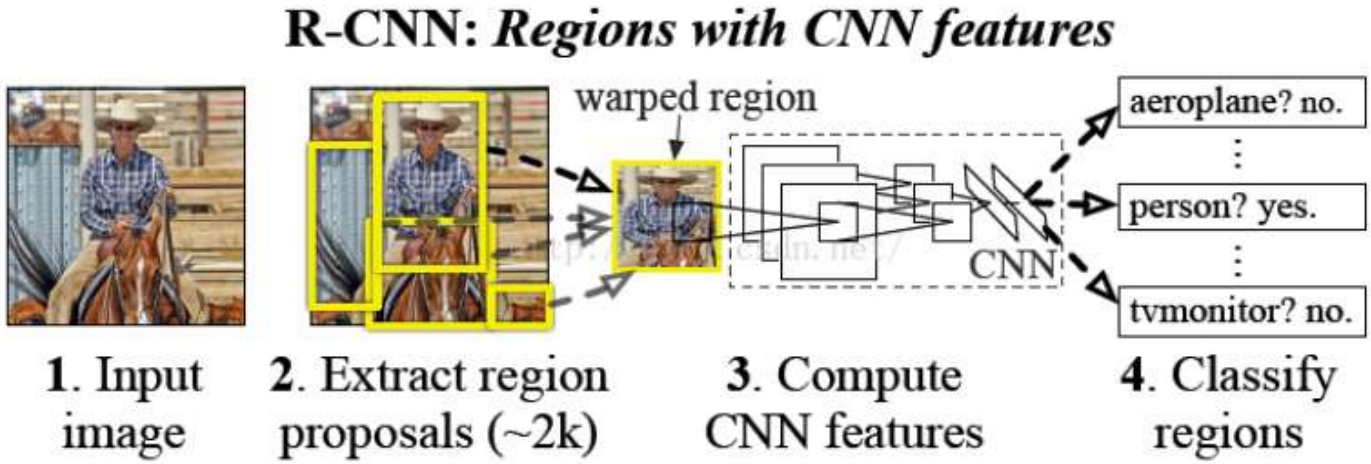
A. Two stage detection framework (Region based)

e.g. R-CNN, Fast RCNN, Faster RCNN

B. One stage detection framework

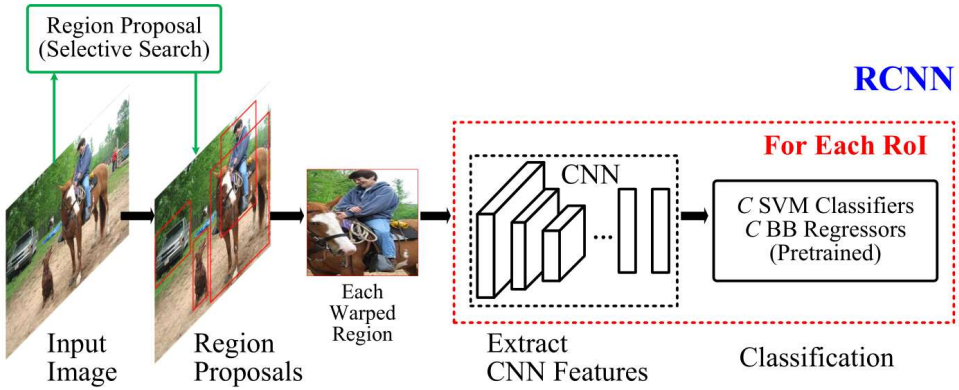
e.g. YOLO, SSD

R-CNN (Region-based Convolutional Neural Network)



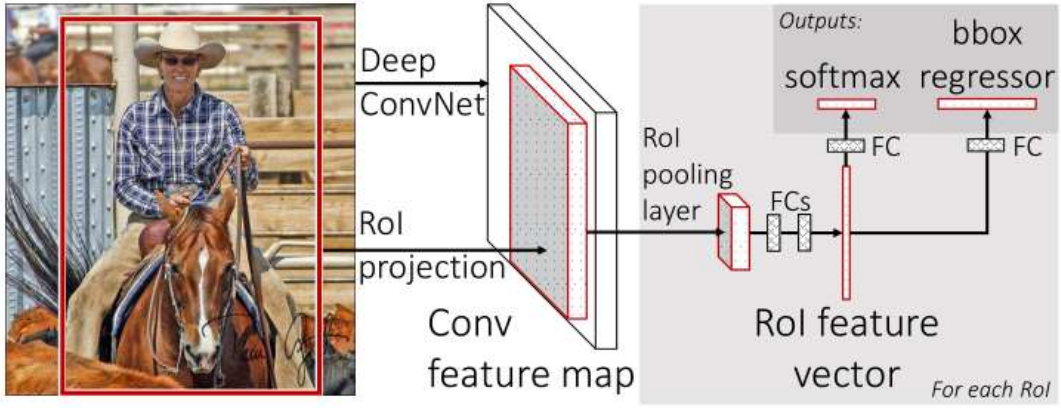
R-CNN: Multistage pipeline:

1. Class-agnostic region proposals
2. Finetuning a CNN model
3. Linear SVM classifiers
4. Bounding box regression



Drawbacks:

1. Training is inelegant, slow and hard to optimize
2. Numerous region proposal only provide rough localization
3. Expansive in both disk space and time
4. Testing is slow



Fast RCNN:

1. Improve detection speed and quality
2. End-to-end detector
3. Joint training simultaneously

   (SVM & BBR)
4. Sharing computation of CONV

   (Region proposal & Feature extraction)
5. Region of Interest (RoI) pooling layer

Faster RCNN:

1. Region Proposal Network (RPN)

   for region proposal

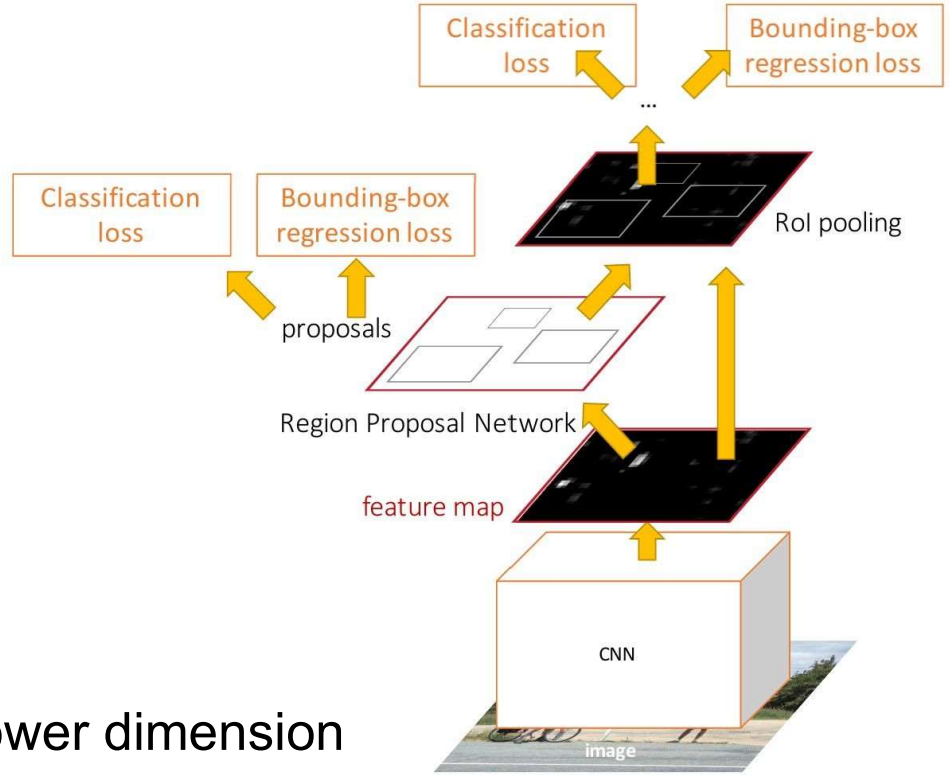
2. Fast RCNN

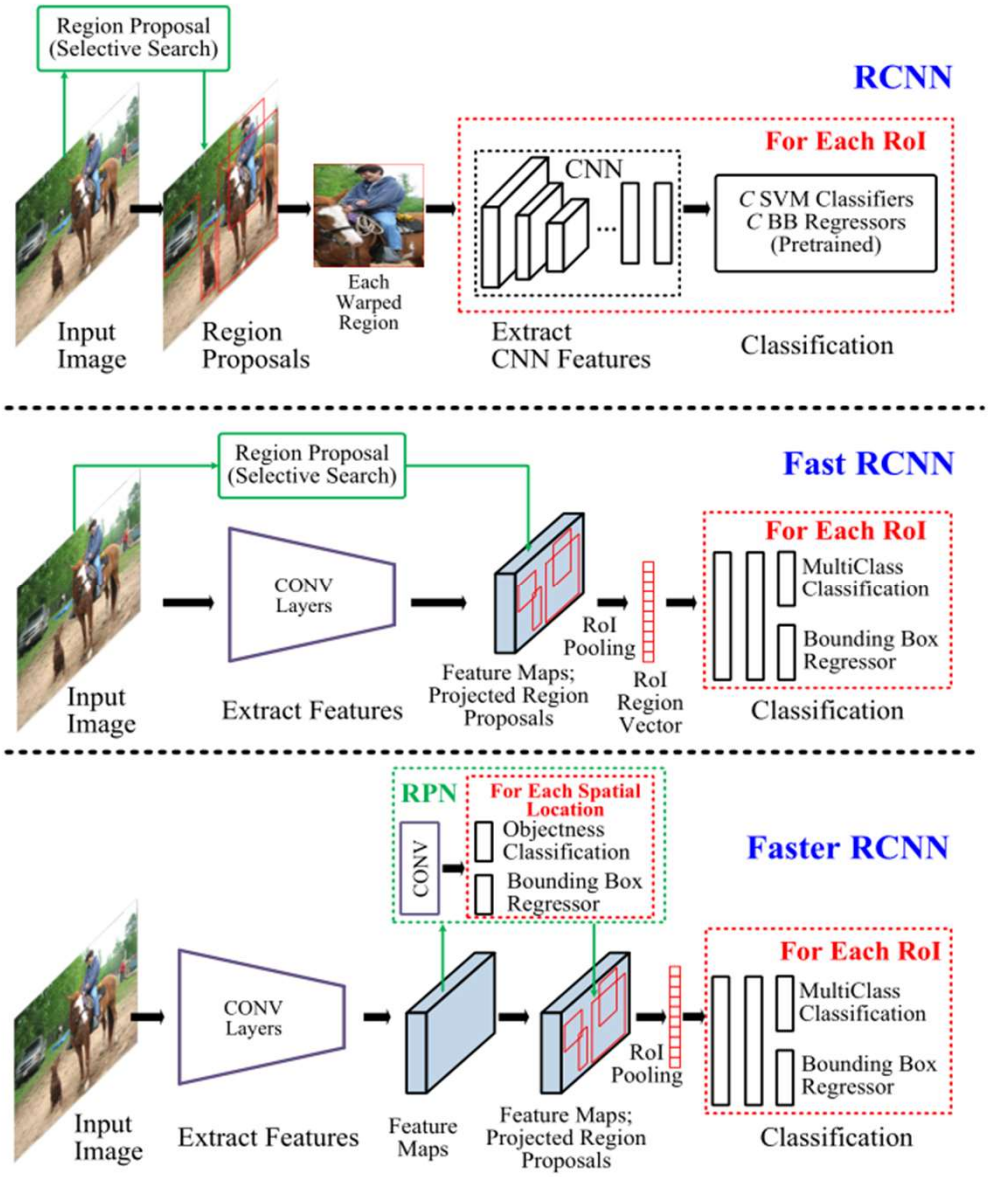
   for region classification

3. RPN: each anchor is mapped to lower dimension

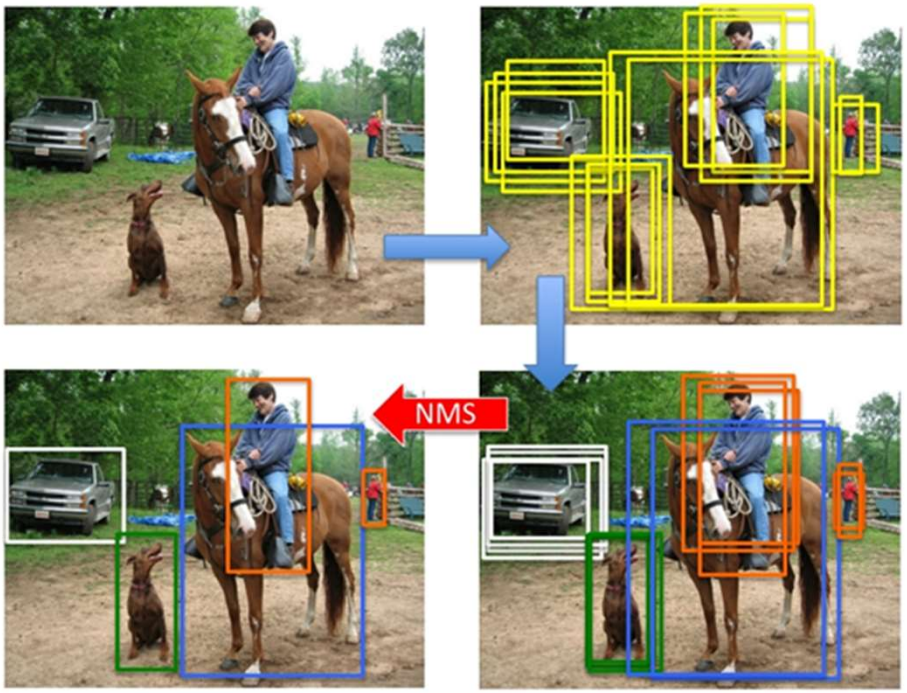
4. RPN shares CONV features with Fast RCNN

   (one step joint training)

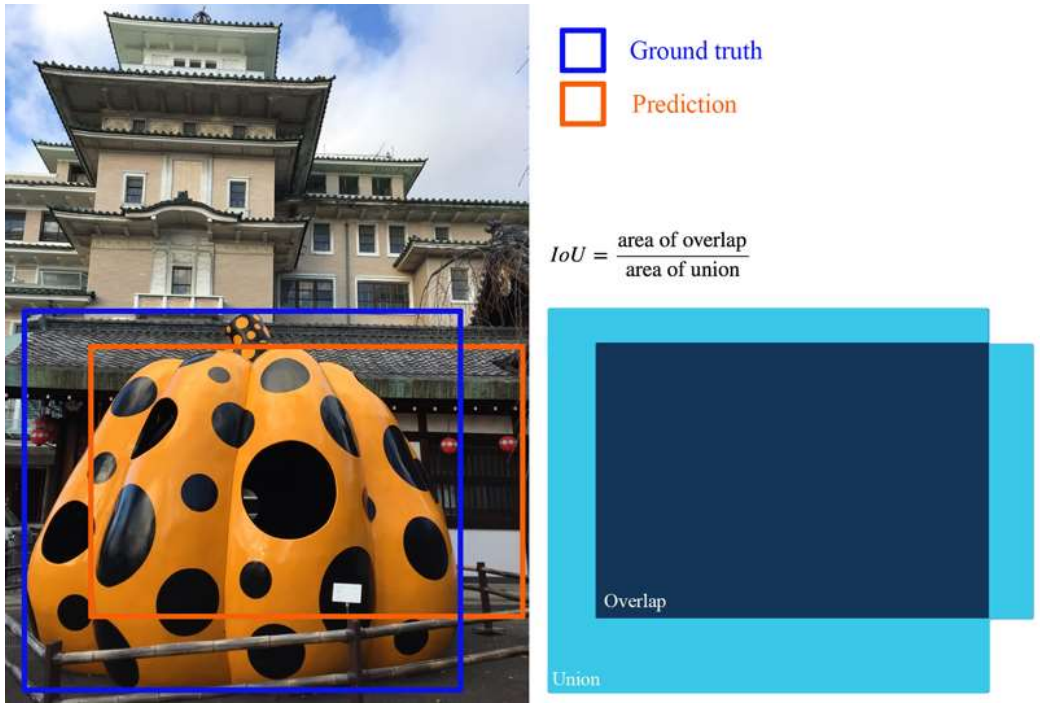



RCNN
Region Proposal (Selective Search)
Input Image
Region Proposals
Each Warped Region
For Each RoI
CNN
C SVM Classifiers
C BB Regressors
(Pretrained)
Extract CNN Features
Classification
Fast RCNN
Region Proposal (Selective Search)
Input Image
CONV Layers
Extract Features
Feature Maps; Projected Region Proposals
RoI Pooling
RoI Region Vector
For Each RoI
MultiClass Classification
Bounding Box Regressor
Classification
Faster RCNN
RPN
CONV
For Each Spatial Location
Objectness Classification
Bounding Box Regressor
Input Image
CONV Layers
Extract Features
Feature Maps
Feature Maps; Projected Region Proposals
RoI Pooling
For Each RoI
MultiClass Classification
Bounding Box Regressor
Classification

NMS (Non-maximum Suppression)



IoU (Intersection over union)



Threshold = **0.7** for each bounding box

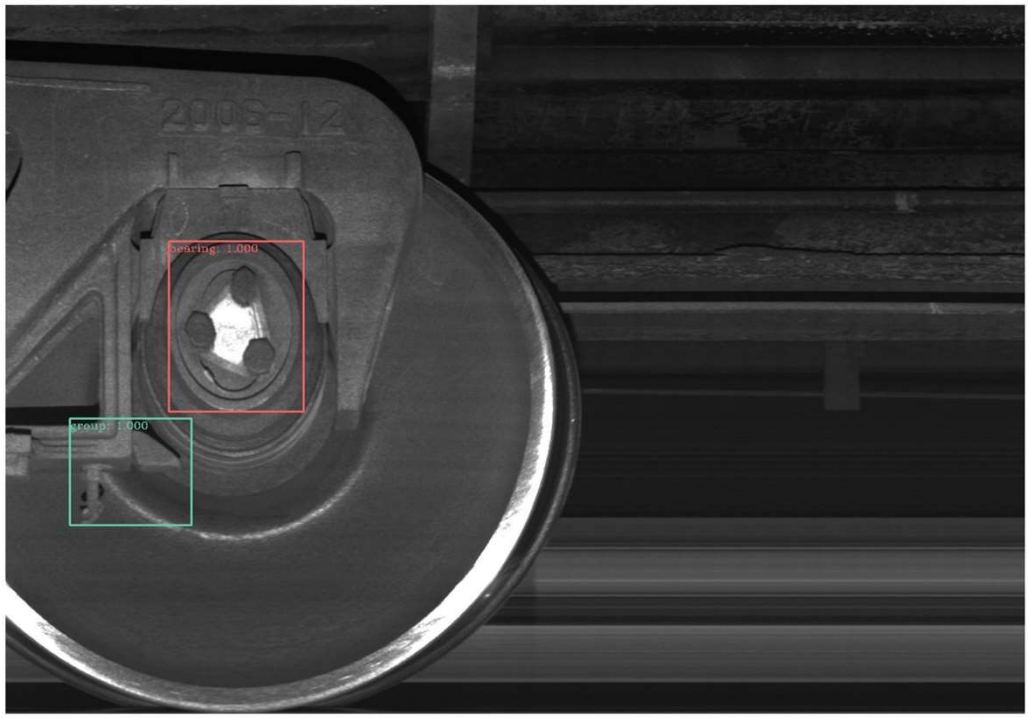
Two-branch Hierarchical Scheme

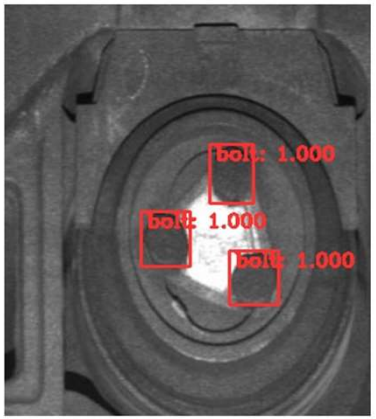
Branch 1: detection of big objects based on images and crop raw images into patches.

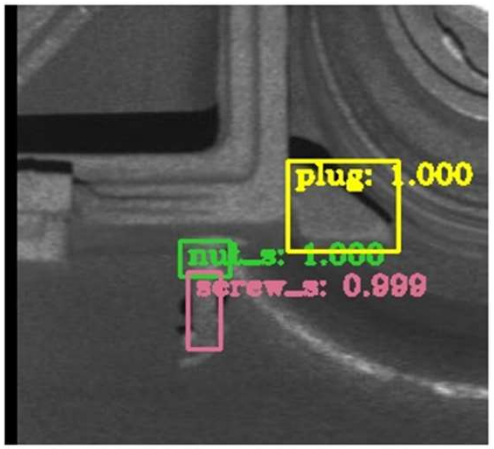
Branch 2: detection of small objects based on patches.

Patches generation (for prediction):

```
im_crop = im_crop_[bbox[1]-50:bbox[3]+50, bbox[0]-50:bbox[2]+50, :]
results.append([img_name, class_name, score, bbox[0], bbox[1], bbox[2], bbox[3]])
if not os.path.exists(path):
    os.makedirs(path)
cv2.imwrite(os.path.join(path,"{}_{}_{}.jpg".format(img_name,class_name,k)), im_crop)
cv2.rectangle(im, bbox[0:2], bbox[2:4], color, 2)
cv2.putText(im, '%s: %.3f' % (class_name, score), (bbox[0], bbox[1] + 15), cv2.FONT_HERSHEY_COMPLEX,
            0.5, color, thickness=1)
```

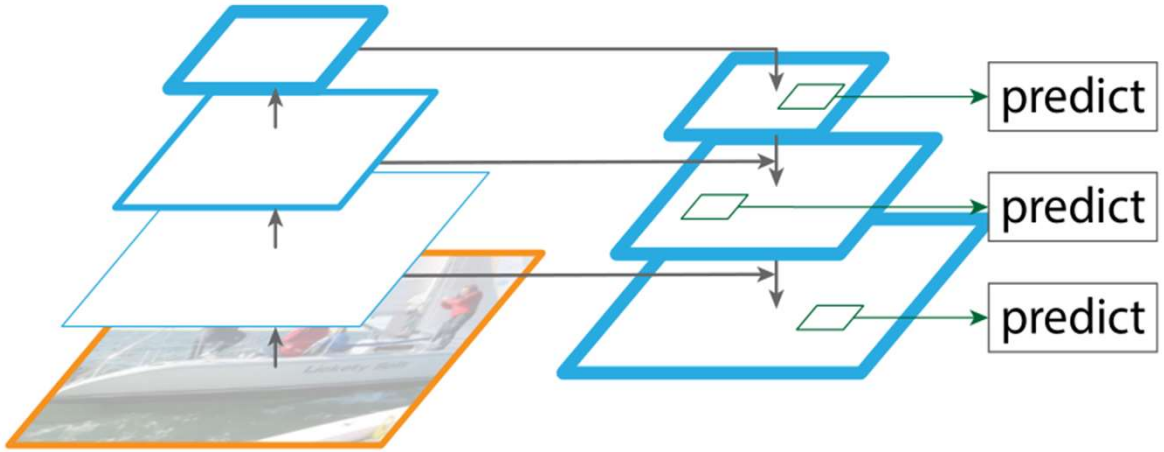

bearing: 1.000
group: 1.000

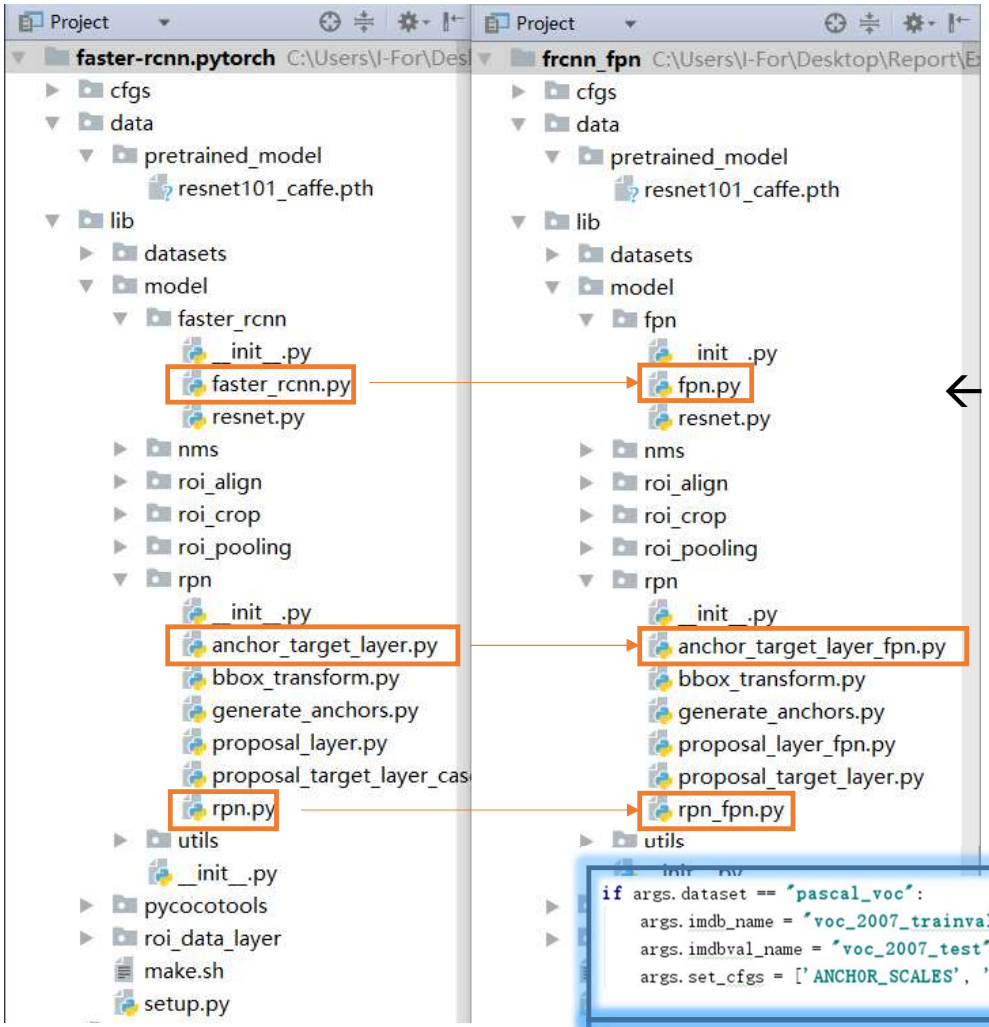

bolt: 1.000
bolt: 1.000
bolt: 1.000


plug: 1.000
nut_s: 1.000
screw_s: 0.999

Feature Pyramid Networks (FPN)

- Replacing the single-scale feature map of RPN with the FPN
- A head is attached to each level on feature pyramid
- The size and number of anchors change





← Directory structure

fpn.py → (forward)

```python
def forward(self, im_data, im_info, gt_boxes, num_boxes):
    batch_size = im_data.size(0)
    im_info = im_info.data
    gt_boxes = gt_boxes.data
    num_boxes = num_boxes.data
    # feed image data to base model to obtain base feature map
    # Bottom-up
    c1 = self.RCNN_layer0(im_data)
    c2 = self.RCNN_layer1(c1)
    c3 = self.RCNN_layer2(c2)
    c4 = self.RCNN_layer3(c3)
    c5 = self.RCNN_layer4(c4)
    # Top-down
    p5 = self.RCNN_toplayer(c5)
    p4 = self._upsample_add(p5, self.RCNN_latlayer1(c4))
    p4 = self.RCNN_smooth1(p4)
    p3 = self._upsample_add(p4, self.RCNN_latlayer2(c3))
    p3 = self.RCNN_smooth2(p3)
    p2 = self._upsample_add(p3, self.RCNN_latlayer3(c2))
    p2 = self.RCNN_smooth3(p2)
    p6 = self.maxpool2d(p5)
    rpn_feature_maps = [p2, p3, p4, p5, p6] # for RPN
    mrcnn_feature_maps = [p2, p3, p4, p5] # for RoI pooling
    rois, rpn_loss_cls, rpn_loss_bbox = self.RCNN_rpn(rpn_feature_maps,
```

Anchors: ↓

```python
if args.dataset == "pascal_voc":
    args.imdb_name = "voc_2007_trainval"
    args.imdbval_name = "voc_2007_test"
    args.set_cfgs = ['ANCHOR_SCALES', '[8, 16, 32]', 'ANCHOR_RATIOS', '[0.5,1,2]', 'MAX_NUM_GT_BOXES', '20']
```

```python
if args.dataset == "pascal_voc":
    args.imdb_name = "voc_2007_trainval"
    args.imdbval_name = "voc_2007_test"
    args.set_cfgs = ['FPN_ANCHOR_SCALES', '[32, 64, 128, 256, 512]', 'FPN_FEAT_STRIDES', '[4, 8, 16, 32, 64]', 'MAX_NUM_GT_BOXES', '20']
```

$$Precision = \frac{TP}{\text{total positive results}}$$

$$Recall = \frac{TP}{\text{total cancer cases}}$$

Precision measures how accurate is your prediction.

Recall measures how good you find all the positives.

$$\mathrm{AP} = \int_0^1 p(r)dr$$

AP (Average Precision)

MAP (Mean AP)

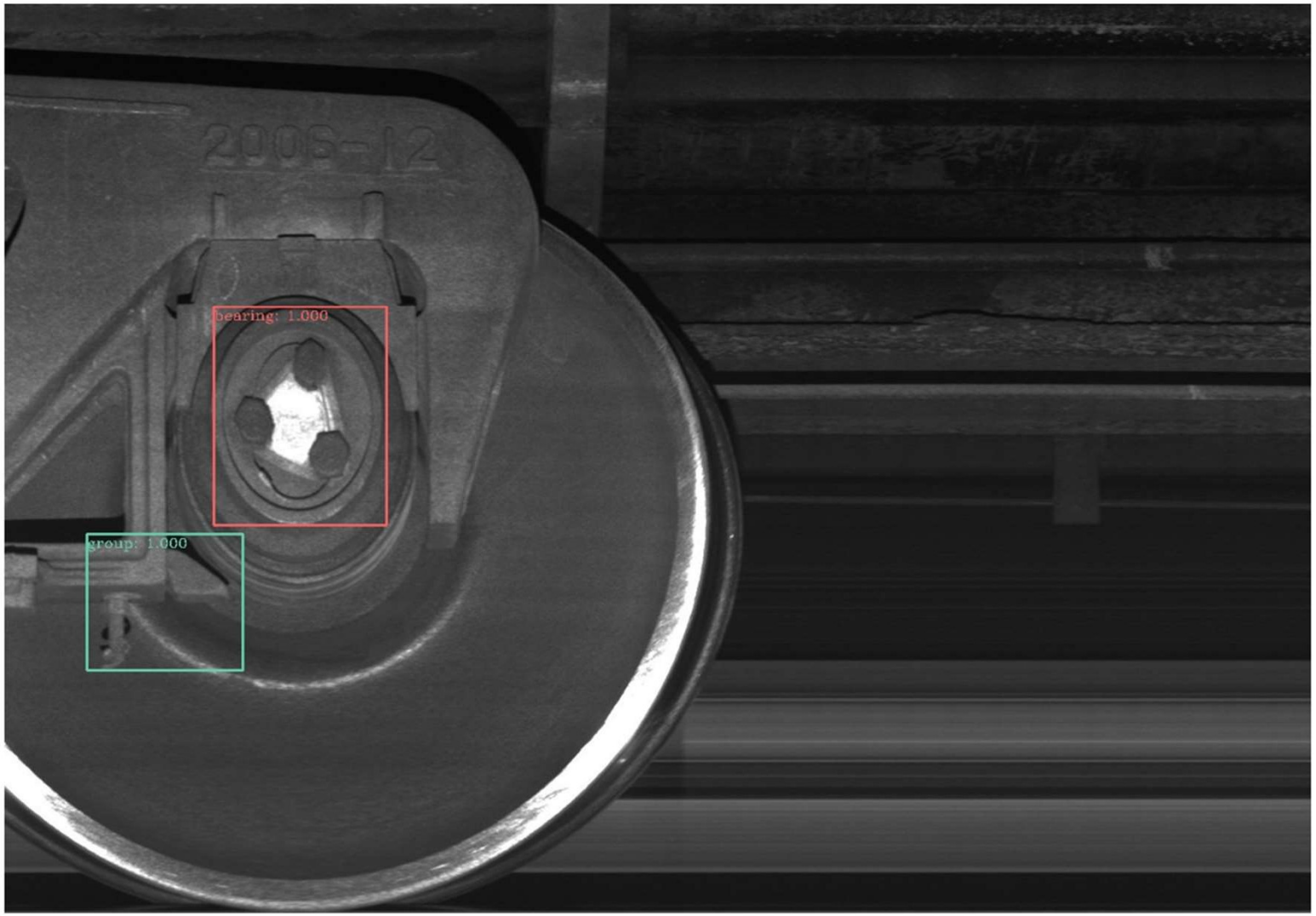
Rank	Correct?	Precision	Recall
1	True	1.0 ↑	0.2 ↑
2	True	1.0 –	0.4 ↑
3	False	0.67 ↓	0.4 –
4	False	0.5 ↓	0.4 –
5	False	0.4 ↓	0.4 –
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑

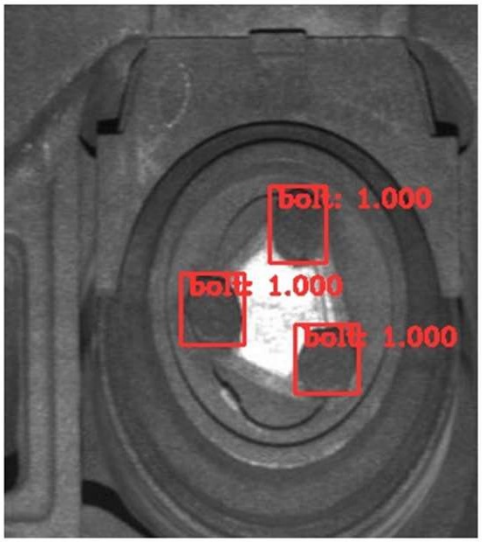
Performance of Object Detection

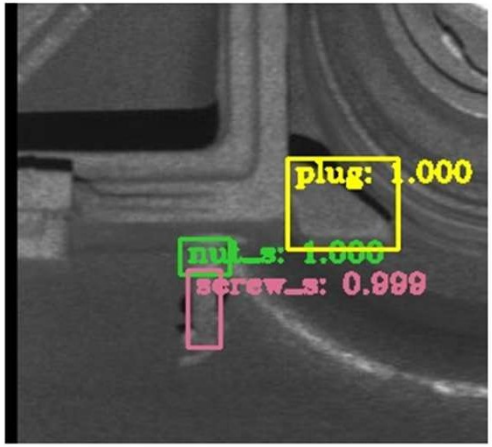
- The mark * means this class belongs to small objects.
- Others are big objects.

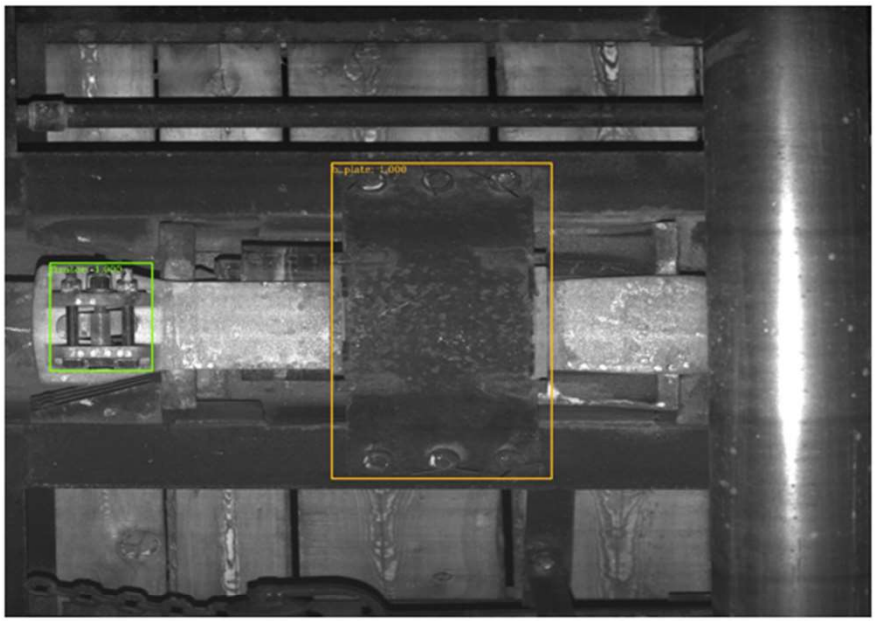
- For demo, confidence of candidate boxes increases from 0 to 0.5, less targets are determined.

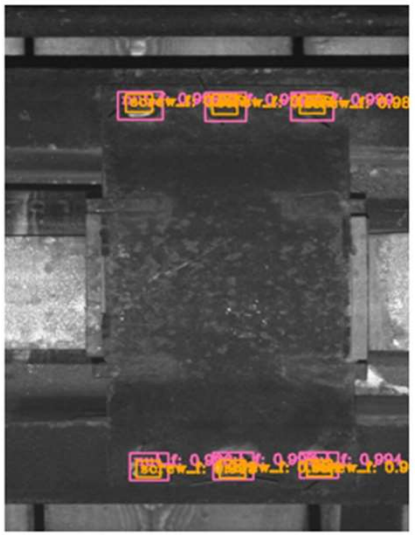
	Faster RCNN -- Branch1	Faster RCNN -- Branch2	Faster RCNN + FPN	Faster RCNN + FPN (for demo)
Spring	1.0000	1.0000	1.0000	1.0000
Buckle	1.0000	1.0000	1.0000	1.0000
B_plate	1.0000	1.0000	0.9986	0.9091
Nut_f *	0.5182	0.9057	0.9055	0.9055
Screw_f *	0.0909	0.6005	0.9074	0.9080
L_plate	0.9958	1.0000	1.0000	1.0000
Bearing	1.0000	1.0000	1.0000	1.0000
Collector	1.0000	1.0000	1.0000	0.9091
Bolt *	0.0000	0.9050	0.9052	0.9074
Group	1.0000	1.0000	1.0000	1.0000
Fixator	1.0000	1.0000	1.0000	1.0000
Flange	0.9063	0.9117	0.9091	0.9091
Plug *	0.0000	1.0000	1.0000	1.0000
Nut_s *	0.0000	0.6355	0.9091	0.9091
Screw_s *	0.0000	0.5444	0.7955	0.7961
MAP	**0.6341**	**0.9002**	**0.9554**	**0.9436**

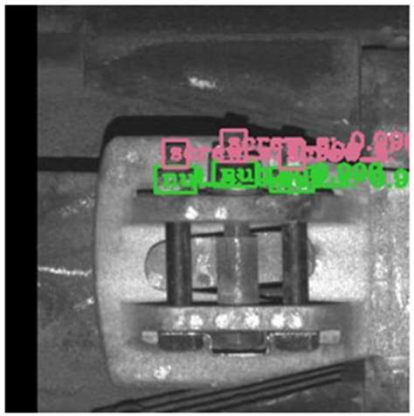

2006-12
bearing: 1.000
group: 1.000

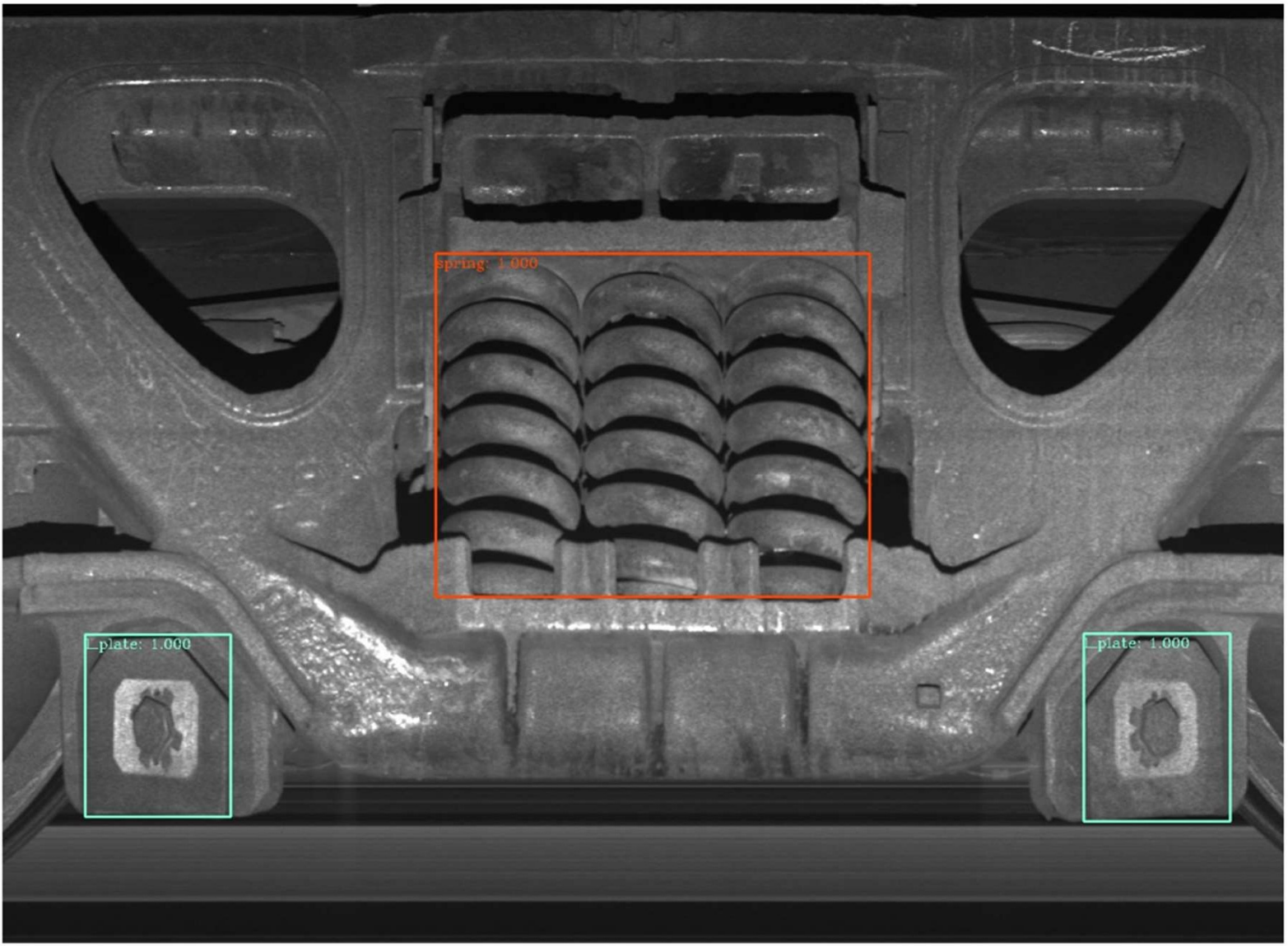

bolt: 1.000
bolt: 1.000
bolt: 1.000

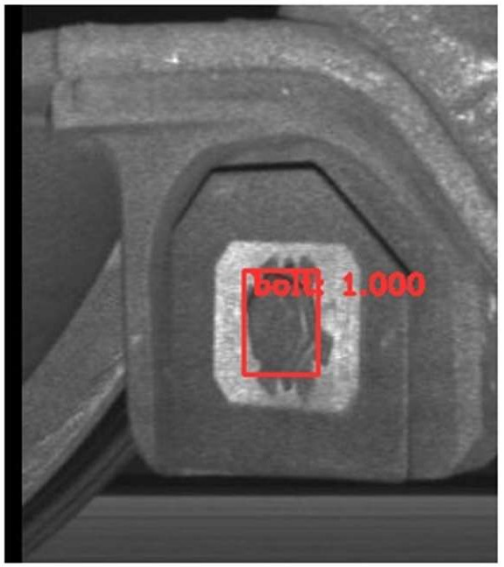

plug: 1.000
nut_s: 1.000
screw_s: 0.999

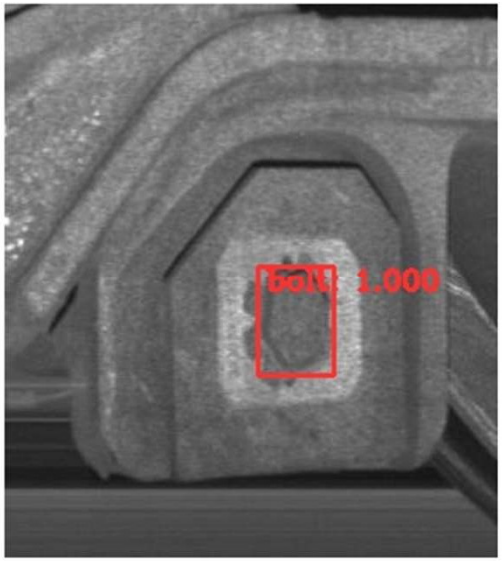

s_plate: 1.000






spring: 1.000
plate: 1.000
plate: 1.000

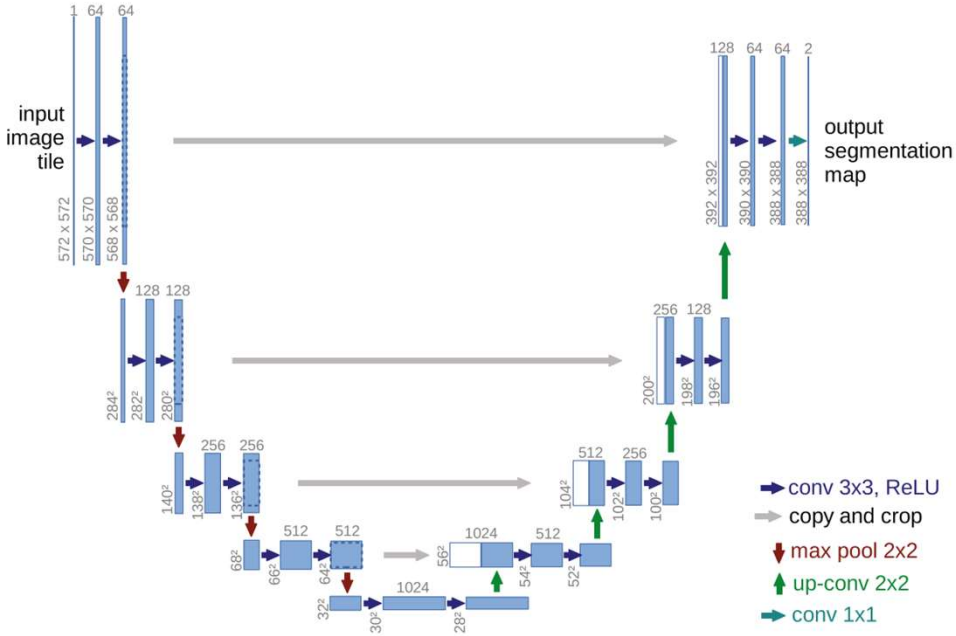

bolt: 1.000


bolt: 1.000

Fault Diagnosis

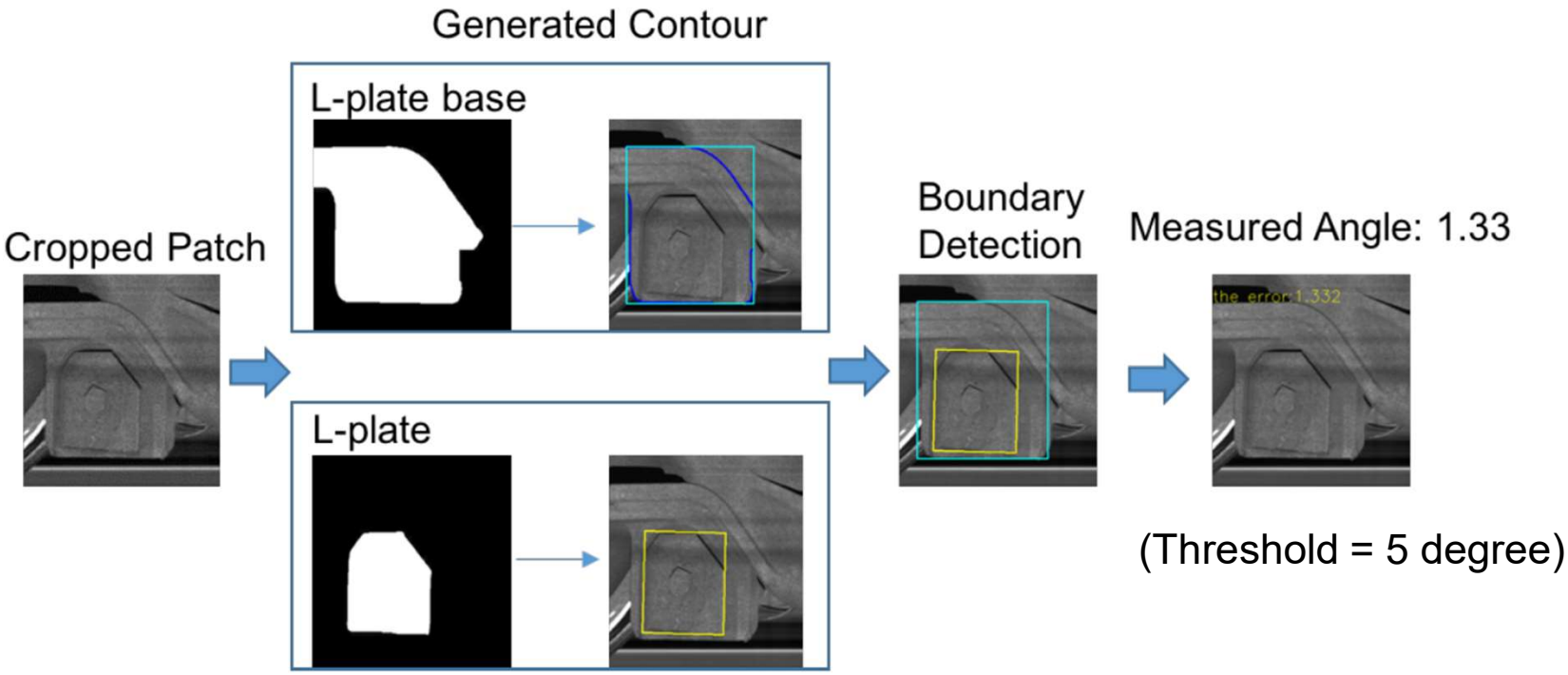
Semantic Segmentation based on U-Net

- Contracting path: convolution & downsampling (Feature Extraction)
- Expansive path: “up-convolution” & upsampling (Feature Visualization)
- Concatenation with the correspondingly cropped feature map.



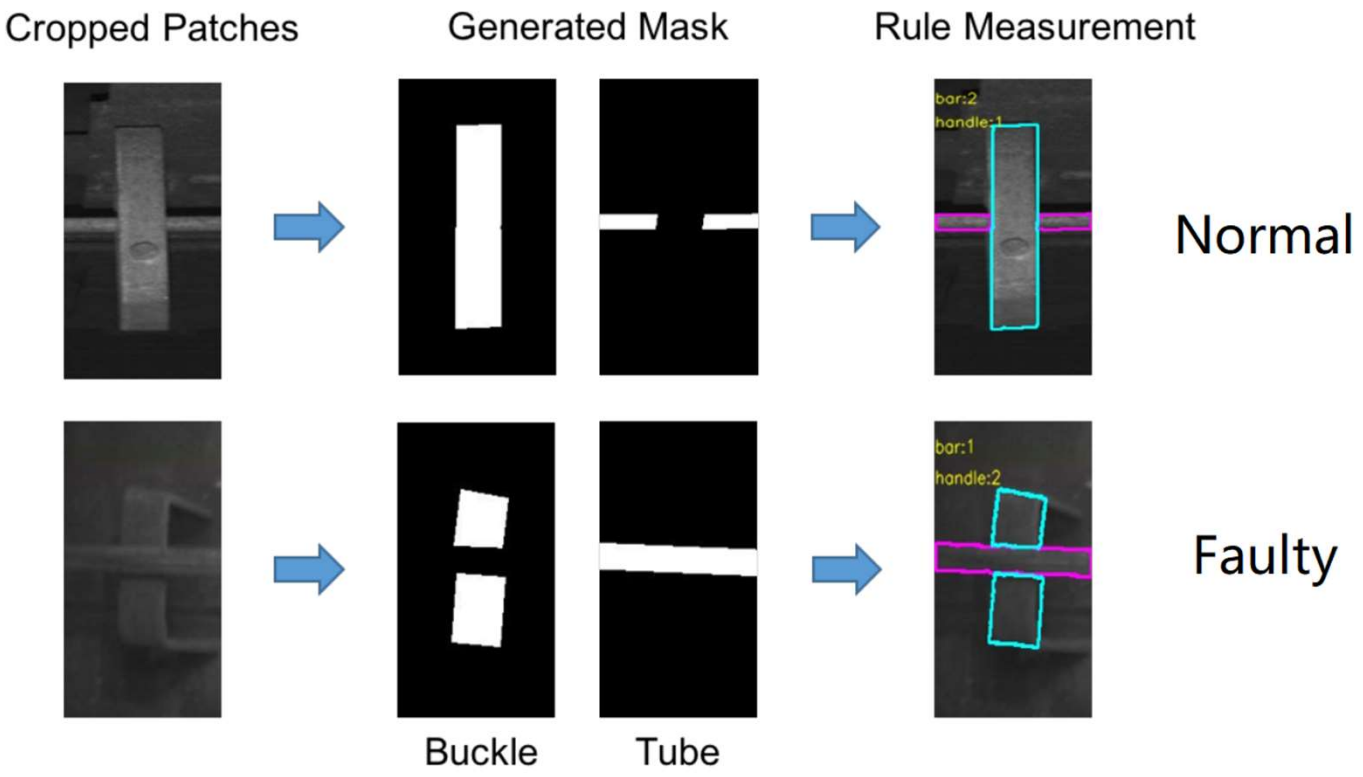
- Post processing: traditional image processing

  → Angle measurement between bottom edges



- Post processing: traditional image processing

  → Counting how many parts each object is divided into



Generating contours and counting the number

```
mask1 = cv2.blur(mask1, (2,2))
cv2.rectangle(mask1, (0,0), (mask1.shape[1],mask1.shape[0]), (0,0,0), 1)
imgray = cv2.cvtColor(mask1,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(imgray,127,255,0)
im2, contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
num = 0
cnt_ls = []
if len(contours) > 0:
    for cnt in contours:
        if cv2.contourArea(cnt) > 100 and cv2.contourArea(cnt) < (mask1.shape[1] - 1) * (mask1.shape[0] - 1) :
            num = num + 1
            cnt_ls.append(cnt)
return num,cnt_ls
```

Angle measurement

```
angel1 = rect[2]
angel2 = rect2[2]
if angel1 <0:
    angel1 = 90 + angel1
if angel2 < 0:
    angel2 = 90 + angel2
angle = angel1- angel2
image = cv2.imread(os.path.join('../data/Images',file))
cv2.putText(image, 'the error:' + str(abs(round(angle,3))), (0, 50), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 255), lineType=cv2.LINE_AA)
cv2.imwrite(os.path.join(sample_folder,'result_' + file), image)
```

Performance of Fault Diagnosis

	Precision (L_plate)	Recall (L_plate)	Precision (Tube)	Recall (Tube)
ResNet50	0.923	0.894	0.933	0.903
Faster RCNN	0.967	0.923	0.950	0.919
Detection + ResNet50	0.967	0.935	0.966	0.935
Detection + U-Net	**0.973**	**1.000**	**0.968**	**1.000**

Contrast experiments:

1. ResNet50 classifier (image level)
2. Faster RCNN object detection (giving different labels)
3. After object detection, using ResNet50 classifier (patches level)
4. After object detection, using U-Net to get masks and post processing

Conclusion and Future work

- Problem description: three main categories and several small kinds
- Dataset preparation: Image augmentation & Label annotation
- Two-stage framework: Component detection & Fault diagnosis
- Object detection:
  a. Baseline: Faster RCNN
  b. Backbone: ResNet101
  c. Two-branch hierarchical scheme
  d. Feature Pyramid Networks
- Fault diagnosis:
  a. Semantic segmentation: U-Net
  b. Post processing

Thank you!