submission

December 16, 2018

0.1 1. Data Exploration

After downloading the data addressed in README, we can see these files.

```
In [1]: import os
        import sys
        import pandas as pd
        import numpy as np
        import pydicom
        import pylab
```

```
In [2]: data_dir = os.path.join(os.getcwd(), 'darknet/data/rsna')

        !cd $data_dir && ls
```

```
jpg_images                         stage_2_test_images
stage_2_detailed_class_info.csv  stage_2_train_images
stage_2_sample_submission.csv         stage_2_train_labels.csv
```

```
In [3]: train_image_dir = os.path.join(data_dir, 'stage_2_train_images')
        test_image_dir = os.path.join(data_dir, 'stage_2_test_images')

        !cd $train_image_dir && ls | wc -l
        !cd $test_image_dir && ls | wc -l
```

```
26684
3000
```

There are two images directories. One is for training the the other is for test. There are **26684** train images and **3000** test images. First we have a see the detailed files. The image format is dcm and we can use pydicom to see it. The file name is the patientId.

```
In [4]: # print five files without sorting to save time.
        !cd darknet/data/rsna/stage_2_train_images/ && ls -U | head -5
```

```
11a355bd-9544-4211-bfa8-451777c1d3c2.dcm
bf067779-e5d4-444f-b8cc-e5e7ff282fde.dcm
```

```
bda0d7f6-9d6d-43ba-8840-e467933dea68.dcm
6099324c-1fef-47a1-9f1b-b2e8f2a110a1.dcm
2f6ac842-e243-4c5f-a48c-d70862e316d5.dcm
ls: write error: Broken pipe
```

We can see the patient infomation in each image dcm file. But what we care about most is the pixel at the bottom.

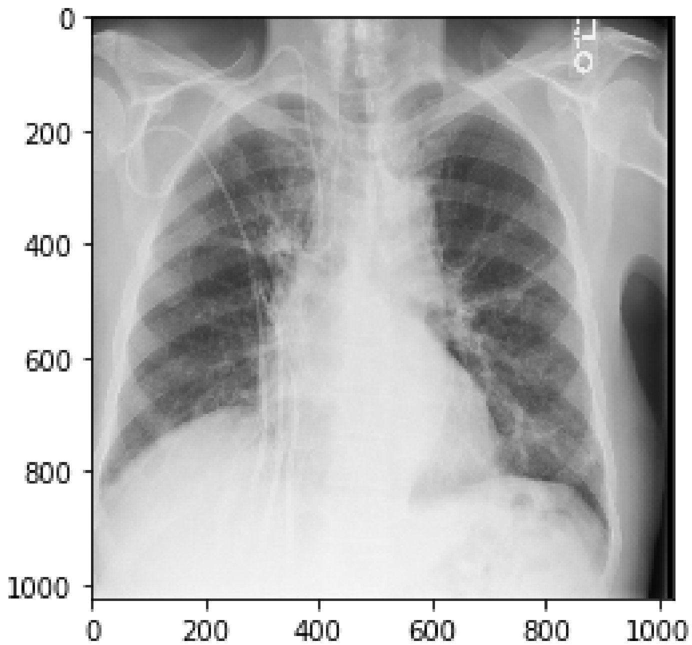
```
In [5]: patientId = '11a355bd-9544-4211-bfa8-451777c1d3c2'
        img_info = pydicom.read_file(os.path.join(train_image_dir, patientId + '.dcm'))
        print(img_info)
```

```
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.23937.15178
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: PA'
(0010, 0010) Patient's Name                      PN: '11a355bd-9544-4211-bfa8-451777c1d3c2'
(0010, 0020) Patient ID                          LO: '11a355bd-9544-4211-bfa8-451777c1d3c2'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'M'
(0010, 1010) Patient's Age                       AS: '68'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.23937.15178
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.23937.15178
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: "1"
(0020, 0013) Instance Number                     IS: "1"
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: ['0.168', '0.168']
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 131722 bytes
```

```
In [6]: def get_image_array(patientId=None):
            image_file = os.path.join(train_image_dir, patientId + '.dcm')
            img_info = pydicom.read_file(image_file)
            img = img_info.pixel_array
            img_arr = np.stack([img] * 3, axis=2)
            return img_arr
```

```
In [7]: def show_image(patientId):
            img_arr = get_image_array(patientId)
            pylab.imshow(img_arr, cmap=pylab.cm.gist_gray)
            pylab.axis('on')
```

```
In [8]: show_image(patientId)
```



The train labels specifies that every image belong to which target. There are only two targets 1 or 0 that means pneumonia and non-pnemonia. If it is pneumonia, its position is represented by top-left coordinate and the width and height.

```
In [9]: train_labels = pd.read_csv(os.path.join(data_dir, 'stage_2_train_labels.csv'))
        train_labels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30227 entries, 0 to 30226
Data columns (total 6 columns):
patientId    30227 non-null object
x            9555 non-null float64
y            9555 non-null float64
```

```
width        9555 non-null float64
height       9555 non-null float64
Target       30227 non-null int64
dtypes: float64(4), int64(1), object(1)
memory usage: 1.4+ MB
```

There are **30227** 'patientId', **9555** (x, y, width, height) pairs. A patient may have multiple pneumonia regions. So this number is large the the number of images **26684** just now. Let's see the small pit here and take some statistics.

```
In [10]: train_labels[:5]
```

```
Out[10]:                                patientId      x      y  width  height  Target
         0  0004cfab-14fd-4e49-80ba-63a80b6bddd6    NaN    NaN    NaN     NaN       0
         1  00313ee0-9eaa-42f4-b0ab-c148ed3241cd    NaN    NaN    NaN     NaN       0
         2  00322d4d-1c29-4943-afc9-b6754be640eb    NaN    NaN    NaN     NaN       0
         3  003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN    NaN    NaN     NaN       0
         4  00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0       1
```

```
In [11]: display(train_labels.describe()) # to see the pnuemonia region distribution
```

```
                 x            y        width       height        Target
count  9555.000000  9555.000000  9555.000000  9555.000000  30227.000000
mean    394.047724   366.839560   218.471376   329.269702      0.316108
std     204.574172   148.940488    59.289475   157.750755      0.464963
min       2.000000     2.000000    40.000000    45.000000      0.000000
25%     207.000000   249.000000   177.000000   203.000000      0.000000
50%     324.000000   365.000000   217.000000   298.000000      0.000000
75%     594.000000   478.500000   259.000000   438.000000      1.000000
max     835.000000   881.000000   528.000000   942.000000      1.000000
```

We can see the box is really big. The mean width is **218** and mean height is **329**. Let's see the number of boxes distribution in a image.

```
In [12]: non_zero = np.count_nonzero(train_labels['Target'])
         print(non_zero)
```

```
9555
```

```
In [13]: target_dict = {}
         last_patientId = 0
         for i in range(len(train_labels)):
             patientId, _, _, _, _, target = train_labels.loc[i]
             if patientId == last_patientId:
                 target_dict[patientId] += 1
             elif target == 1:
```

```
            target_dict[patientId] = 1
        else:
            target_dict[patientId] = 0
        last_patientId = patientId
```

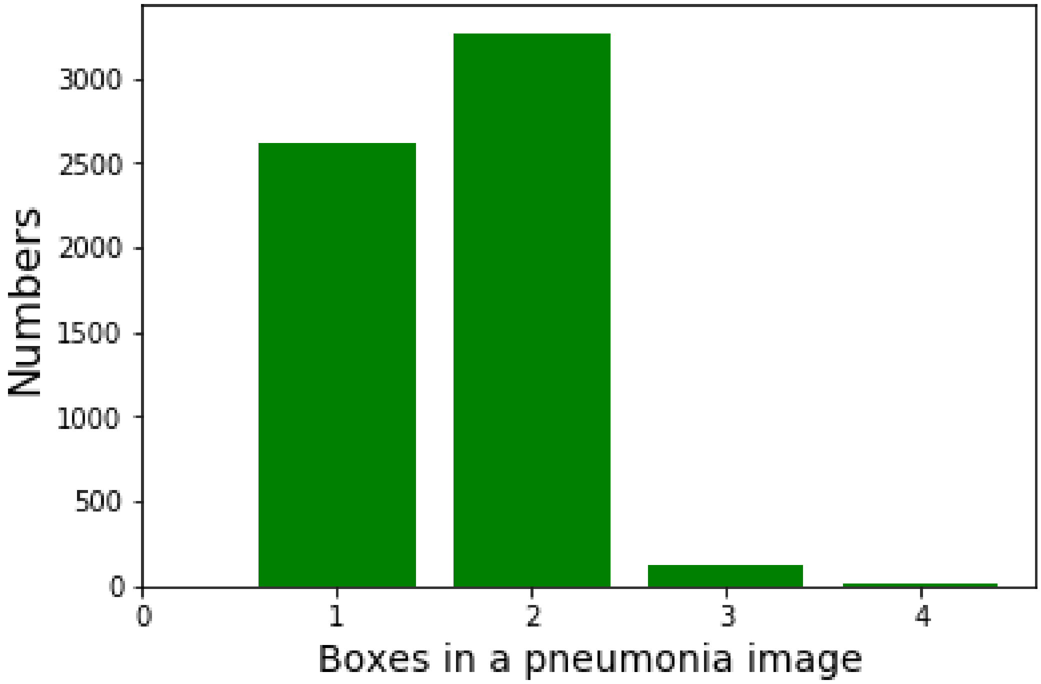
In [14]:
```
set(target_dict.values())
```

Out[14]:
```
{0, 1, 2, 3, 4}
```

In [15]:
```
num_dict = {0: 0, 1: 0, 2: 0, 3: 0, 4: 0}
for num in target_dict.values():
    num_dict[num] += 1
```

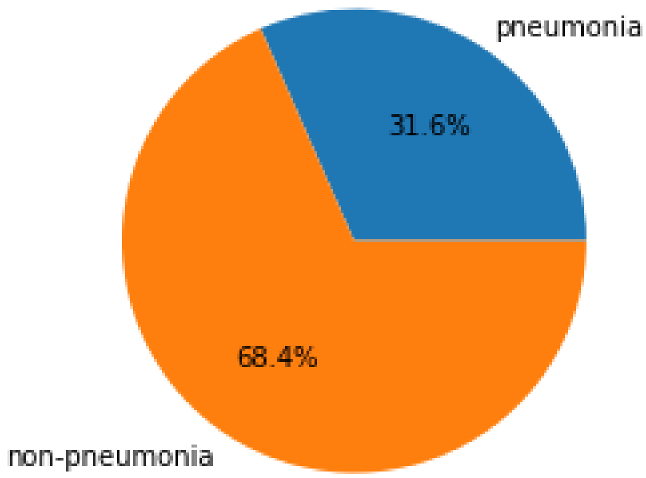
In [16]:
```
import matplotlib.pyplot as plt

plt.bar(range(len(num_dict))[1:], list(num_dict.values())[1:], align='center', color='g
plt.xlabel('Boxes in a pneumonia image', fontsize=14)
plt.ylabel('Numbers', fontsize=16)
plt.xticks(range(len(num_dict)), num_dict.keys())
# plt.savefig("Number_boxes")
```

Out[16]:
```
([<matplotlib.axis.XTick at 0x7f55c1069fd0>,
  <matplotlib.axis.XTick at 0x7f55c105c3c8>,
  <matplotlib.axis.XTick at 0x7f55c1069860>,
  <matplotlib.axis.XTick at 0x7f55bd10ba58>,
  <matplotlib.axis.XTick at 0x7f55bd10bf28>],
 <a list of 5 Text xticklabel objects>)
```



```
In [17]: a = int(non_zero), int(len(train_labels) - non_zero)
         b = 'pneumonia', 'non-pneumonia'
         non_zero = np.count_nonzero(train_labels['Target'])
         plt.pie(a, labels=b, autopct = '%3.1f%%')
         # plt.savefig("Proportion")
```

```
Out[17]: ([<matplotlib.patches.Wedge at 0x7f55bd0ed208>,
           <matplotlib.patches.Wedge at 0x7f55bd0ed908>],
          [Text(0.6007208521938164, 0.921484919974025, 'pneumonia'),
           Text(-0.6007208521938167, -0.9214849199740248, 'non-pneumonia')],
          [Text(0.3276659193784453, 0.50262813816765, '31.6%'),
           Text(-0.3276659193784454, -0.5026281381676498, '68.4%')])
```

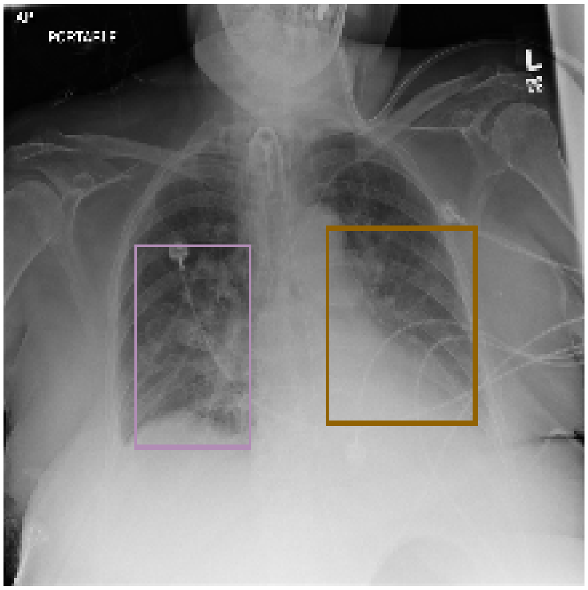


We have shown the image above. Let's see the it again with the box.

```
In [18]: import random
         import cv2

         def draw_image(patientId):
             img_arr = get_image_array(patientId)
             labels = train_labels.loc[train_labels['patientId'] == patientId]
             for i in range(len(labels)):
                 label = labels.iloc[i]
                 if label['Target'] == 1:
                     x = int(label['x'])
```

```
                y = int(label['y'])
                w = int(label['width'])
                h = int(label['height'])
                color = random.sample(range(255), 3)
                cv2.rectangle(img_arr, pt1=(x, y), pt2=(x+w, y+h), color=color, thickness=5
        pylab.imshow(img_arr, cmap='gray')
        pylab.axis('off')
        pylab.savefig(patientId+'.jpg')
        pylab.show()
```

In [19]:
```
draw_image(patientId)
```



The class_info file specifies each patient's class. There are three classes. Both 'Normal' and 'No Lung Opacity / Not Normal' are non-pneumonia. 'Lung Opacity' are pneumonia. As Guy Zahavi explained, 'No Lung Opacity / Not Normal' refers to no opacity for pneumonia. Here we don't consider this. We only use the target.

In [20]:
```
detail_class_info = pd.read_csv(os.path.join(data_dir, 'stage_2_detailed_class_info.csv
detail_class_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30227 entries, 0 to 30226
Data columns (total 2 columns):
patientId    30227 non-null object
class        30227 non-null object
dtypes: object(2)
memory usage: 472.4+ KB
```

```
In [21]: detail_class_info['class'].unique()
```

```
Out[21]: array(['No Lung Opacity / Not Normal', 'Normal', 'Lung Opacity'],
               dtype=object)
```

0.2 2. Preprocess the Data

Now we have known about the playground. I choose the yolov3 model. However in order to satisfy the format, I need to convert the dcm images to jpg format, and convert the labels to relative r_x, r_y, r_w, r_h. The meaning of these parameters are declared in Report. Here I process the data format.

0.2.1 2.1 Process the image

The image is (1024, 1024, 3) RGB format. I check each one to see if there is any exception when converting.

```
In [22]: jpg_dir = os.path.join(data_dir, 'jpg_images')

         def convert_images(patientId):
             img_arr = get_image_array(patientId)
             shape = img_arr.shape
             if shape != (1024, 1024, 3):
                 print('There is an exception:', shape)
                 print('patientId')
             cv2.imwrite(jpg_dir + "/" + patientId + '.jpg', img_arr)     # this is much faster t
```

```
In [24]: from multiprocessing import Pool

         p = Pool(11)    # set a thread aside to do others
         p.map(convert_images, train_labels['patientId'])
         p.terminate()
         print("Finished")
```

```
Finished
```

Let's check the images. The number is right and it also shows well.

```
In [25]: !cd $jpg_dir && ls |wc -l
```

```
26684
```

```
In [26]: from glob import glob
         from PIL import Image
         jpg_file = glob(jpg_dir + "/*.jpg")
         image = Image.open(jpg_file[0])
         image.show()
```

0.2.2 2.2 Process the label

Read the labels files and save as text. If the target is 0, the text is empty. If there are multiple boxes, they are in the same text file.

```
In [28]: !mkdir labels
```

```
In [31]: for patientId in train_labels['patientId']:
             label_file = os.path.join(os.getcwd(), 'labels/' + patientId + '.txt')
             with open(label_file, 'w') as f:
                 labels = train_labels.loc[train_labels['patientId'] == patientId]
                 for i in range(len(labels)):
                     label = labels.iloc[i]
                     target = label['Target']
                     img_size = 1024
                     if target == 0:
                         break
                     x = label['x']
                     y = label['y']
                     w = label['width']
                     h = label['height']

                     rx = x / img_size
                     ry = y / img_size
                     rw = w / img_size
                     rh = h/ img_size
                     rcx = rx + rw / 2
                     rcy = ry + rh / 2
                     line = "{} {} {} {} {}".format(0, rcx, rcy, rw, rh)
                     f.write(line)
                     if i != len(labels) - 1:
                         f.write('\n')
```

Now we split the images and then save path.

```
In [32]: positive_patients = train_labels.loc[train_labels['Target'] == 1]['patientId']
         negative_patients = train_labels.loc[train_labels['Target'] == 0]['patientId'].iloc[:le
```

```
In [33]: def get_images_paths(patients):
             image_paths = []
             for id in patients:
                 image_paths.append(os.path.join(jpg_dir, id + '.jpg'))
             return image_paths
```

```
In [34]: label_dir = os.path.join(data_dir, 'labels')
         def get_label_paths(patients):
             label_paths = []
             for id in patients:
                 label_paths.append(os.path.join(label_dir, id + '.txt'))
             return label_paths
```

In [35]:
```python
positive_image_paths = get_images_paths(positive_patients)
negative_image_paths = get_images_paths(negative_patients)

positive_label_paths = get_label_paths(positive_patients)
negative_label_paths = get_label_paths(negative_patients)
```

In [36]:
```python
from sklearn.model_selection import train_test_split

pos_train, pos_valid = train_test_split(positive_patients, test_size=0.1, random_state=
neg_train, neg_valid = train_test_split(negative_patients, test_size=0.1, random_state=

train_data = pd.concat([pos_train, neg_train])
valid_data = pd.concat([pos_valid, neg_valid])
```

In [37]:
```python
train_image_paths = get_images_paths(train_data)
train_label_paths = get_label_paths(train_data)

valid_image_paths = get_images_paths(valid_data)
valid_label_paths = get_label_paths(valid_data)
```

Then we save these to text files.

In [38]:
```python
def save_path(path, file):
    with open(file, 'w') as f:
        for p in path:
            f.write(p+'\n')
```

In [39]:
```python
metadata = os.path.join(os.getcwd(), 'metadata')
save_path(train_image_paths, os.path.join(metadata, 'train.txt'))
save_path(valid_image_paths, os.path.join(metadata,'valid.txt'))
```

0.3 3. Train the Model

I split the images to train set and validation set with the ratio 9:1. For better using yolov3, I add the same number of negative images(non-pneumonia) with positive images(pneumonia).

I use the initial weight `darknet53.conv.74`. The training process is long and the log is very large. It takes about 1000 iterations an hour in my Nvidia 1080Ti GPU. So I show the results directly. The loss is stable after 20000 iterations, so I haven't plotted here.

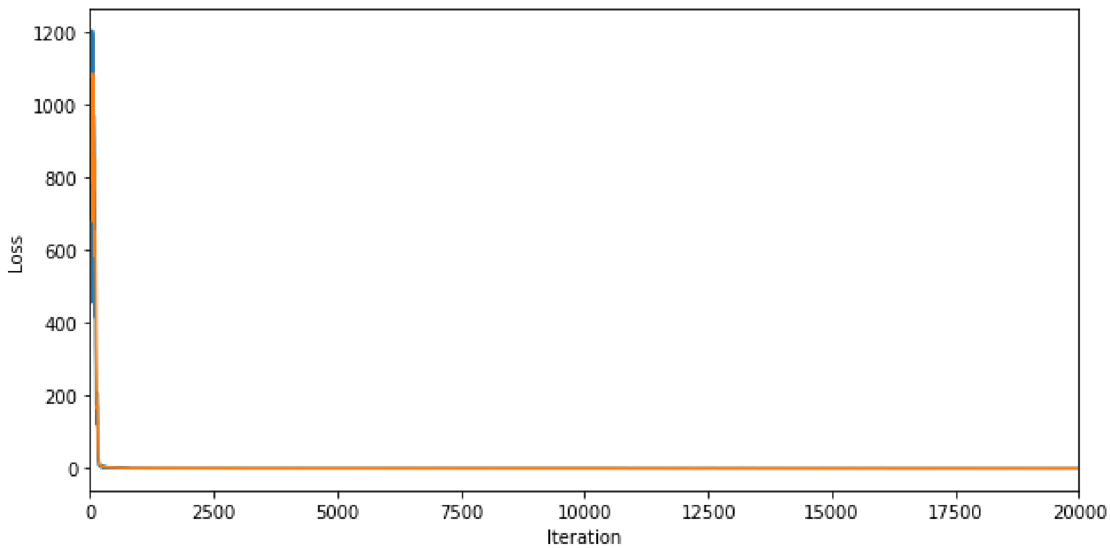
In [ ]:
```
# train command
./darknet_gpu detector train cfg/rsna.data cfg/rsna_yolov3.cfg_train darknet53.conv.74 -
```

In [40]:
```python
# train history log
import matplotlib.pyplot as plt
import seaborn as sns

def get_loss_from_log(log_file, loss_file, num):
    with open(log_file, 'r') as log:
        with open(loss_file, 'a+') as loss:
```

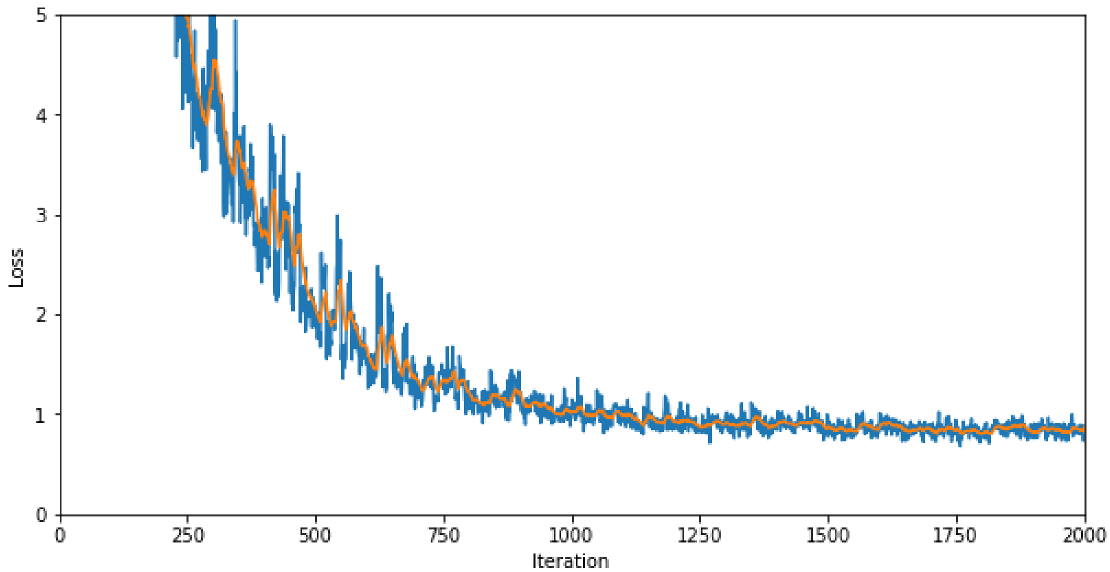
```
                lines = log.readlines()
                i = 0
                for line in lines:
                    if 'avg' in line:
                        loss.write(line)
                        i += 1
                    if i >= num:
                        break
                        loss.close()
                        log.close()
```

```
In [42]: def plot_loss(loss_file, xmin=None, xmax=None, ymin=None, ymax=None):
             iteration = []
             loss = []
             avg_loss = []
             with open(loss_file, 'r') as f:
                 lines = f.readlines()
                 for line in lines:
                     # pattern  1: 1384.613525, 1384.613525 avg, 0.000000 rate, 4.780301 seconds
                     data = line.split(',')
                     iteration.append(int(data[0].split(':')[0]))
                     loss.append(float(data[0].split(':')[1].split()[0]))   # the first one afte
                     avg_loss.append(float(data[1].split()[0]))
             plt.figure(figsize=(10, 5))
             sns.lineplot(iteration, loss)
             sns.lineplot(iteration, avg_loss)
             plt.xlabel('Iteration')
             plt.ylabel('Loss')
             if xmin is not None and xmax is not None:
                 plt.xlim(xmin, xmax)
             if ymin is not None and ymax is not None:
                 plt.ylim(ymin, ymax)
```

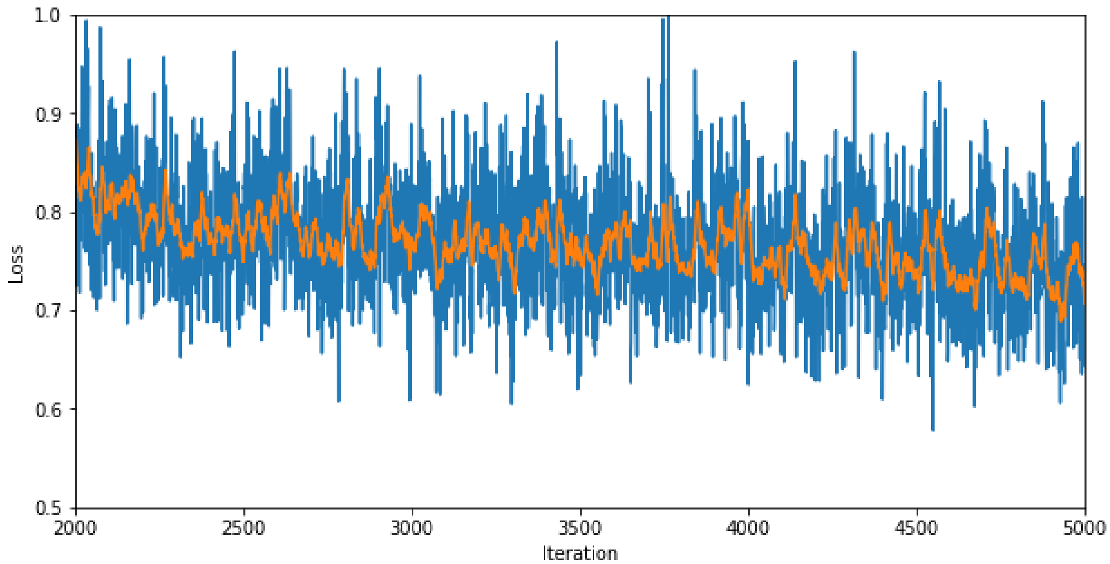
```
In [43]: plot_loss('logs/loss.txt', xmin=0, xmax=20000)
```



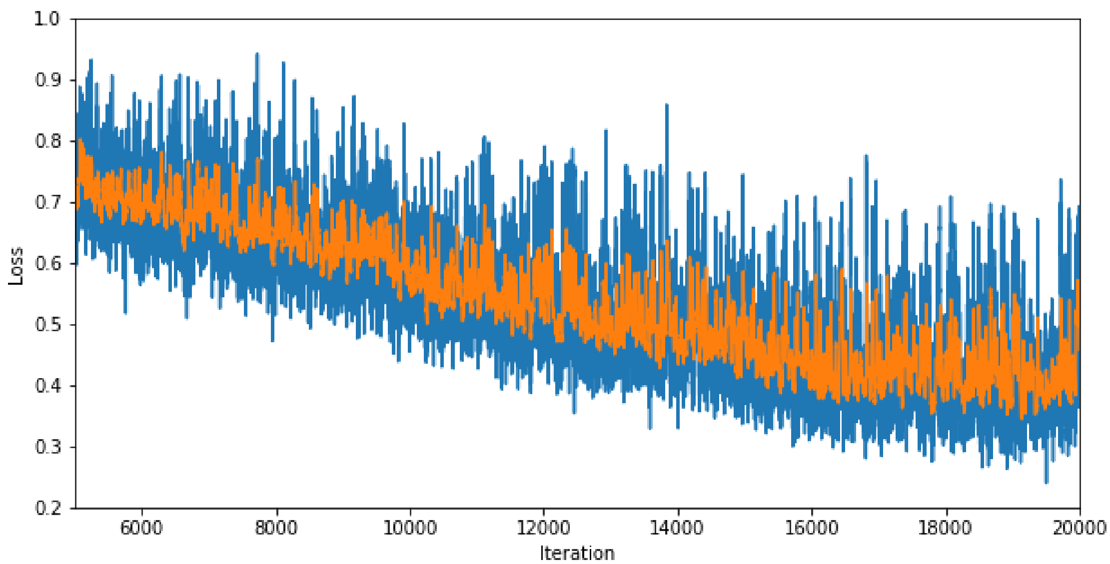

```
In [44]: plot_loss('logs/loss.txt', xmin=0, xmax=2000, ymin=0, ymax=5)
```




```
In [45]: plot_loss('logs/loss.txt', xmin=2000, xmax=5000, ymin=0.5, ymax=1)
```



In [46]: plot_loss('logs/loss.txt', xmin=5000, xmax=20000, ymin=0.2, ymax=1)



0.4 4. Validate the Model

In validation we use the command map and Yolo-AlexeyAB branch to get the files for weight iteration from 2000 to 27000.

In [51]: !ls backup/

```
rsna_yolov3_10000.weights  rsna_yolov3_18000.weights  rsna_yolov3_26000.weights
rsna_yolov3_11000.weights  rsna_yolov3_19000.weights  rsna_yolov3_27000.weights
rsna_yolov3_12000.weights  rsna_yolov3_20000.weights  rsna_yolov3_3000.weights
rsna_yolov3_13000.weights  rsna_yolov3_2000.weights   rsna_yolov3_4000.weights
rsna_yolov3_14000.weights  rsna_yolov3_21000.weights  rsna_yolov3_5000.weights
rsna_yolov3_15000.weights  rsna_yolov3_22000.weights  rsna_yolov3_6000.weights
rsna_yolov3_15300.weights  rsna_yolov3_23000.weights  rsna_yolov3_7000.weights
rsna_yolov3_16000.weights  rsna_yolov3_24000.weights  rsna_yolov3_8000.weights
rsna_yolov3_17000.weights  rsna_yolov3_25000.weights  rsna_yolov3_9000.weights
```

In [ ]:
```
## Because this output is too long, so I just show this code for convenience

weights = np.linspace(2000, 27000, 26)
for weight in weights:
    weight_path = os.path.join(os.getcwd(), 'backup/rsna_yolov3_' + str(int(weight))+ ".
    file_path = os.path.join(os.getcwd(), 'darknet/results/' + str(int(weight))+ "_0.01_
    !cd ../Kaggle/Yolo-AlexeyAB/darknet/ && ./darknet detector map ../cfg/rsna.data ../c
```

In [58]:
```
!ls darknet/results/
```

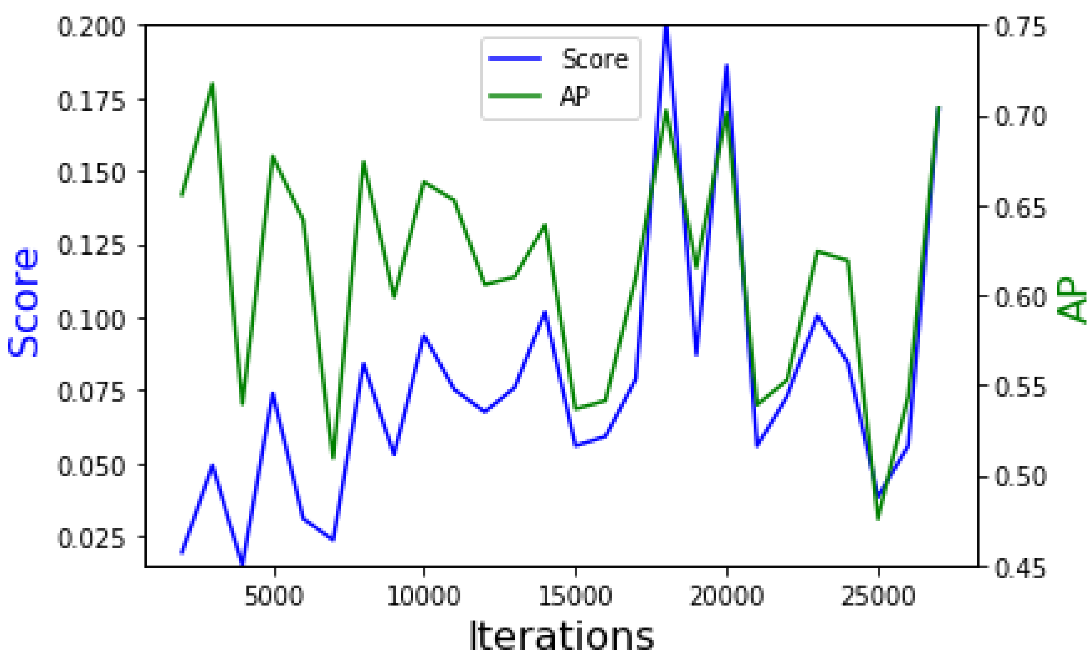
```
10000_0.01_iou_thresh.txt  19000_0.01_iou_thresh.txt  27000_0.01_iou_thresh.txt
11000_0.01_iou_thresh.txt  20000_0.01_iou_thresh.txt  3000_0.01_iou_thresh.txt
12000_0.01_iou_thresh.txt  2000_0.01_iou_thresh.txt   4000_0.01_iou_thresh.txt
13000_0.01_iou_thresh.txt  21000_0.01_iou_thresh.txt  5000_0.01_iou_thresh.txt
14000_0.01_iou_thresh.txt  22000_0.01_iou_thresh.txt  6000_0.01_iou_thresh.txt
15000_0.01_iou_thresh.txt  23000_0.01_iou_thresh.txt  7000_0.01_iou_thresh.txt
16000_0.01_iou_thresh.txt  24000_0.01_iou_thresh.txt  8000_0.01_iou_thresh.txt
17000_0.01_iou_thresh.txt  25000_0.01_iou_thresh.txt  9000_0.01_iou_thresh.txt
18000_0.01_iou_thresh.txt  26000_0.01_iou_thresh.txt
```

In [54]:
```
def calculate_score_and_AP(file_path):
    with open(file_path, 'r') as f:
        lines = list(f.readlines())
        line = lines[-3]
        data = line.split(',')
        TP = float(data[1].split('=')[1])
        FP = float(data[2].split('=')[1])
        FN = float(data[3].split('=')[1])
        score = TP / (TP + FP + FN)

        last_line = lines[-1]
        AP = float(last_line.split(',')[0].split("=")[1])
    return score, AP
```

In [55]:
```
weights = np.linspace(2000, 27000, 26)

scores = []
```

```
APs = []
for weight in weights:
    file_path = os.path.join(os.getcwd(), 'darknet/results/' + str(int(weight))+ "_0.01
    score, AP = calculate_score_and_AP(file_path)
    scores.append(score)
    APs.append(AP)
```

In [56]:
```
fig = plt.figure()
score = fig.add_subplot(111)
precision = score.twinx()

score.set_ylim(0.015, 0.2)
precision.set_ylim(0.45, 0.75)
score.set_xlabel('Iterations', fontsize=16)
score.set_ylabel('Score', fontsize=16)
precision.set_ylabel('AP', fontsize=16)

p1 = score.plot(weights, scores, color='b')
p2 = precision.plot(weights, APs, color='g')

plt.legend((p1[0], p2[0]), ('Score', 'AP'), prop={'size':10}, loc=9)
score.yaxis.label.set_color('b')
precision.yaxis.label.set_color('g')
plt.savefig("valid")
plt.show()
```



0.5 5. Test the Model

I write a script for testing in `darknet` directory. You can use it in command by inputting `bash test.sh num` and then the prediction popups. The `num` is from 1 to 4 where 1 and 2 are right predictions, 3 and 4 are wrong predictions. We can see the original images here.

In [77]:
```
patientId1 = '401b69a0-3d07-47f8-bfea-4bc30aa51403'
patientId2 = 'f5926c42-623f-4a7d-9214-bf9f0964e9fd'
patientId3 = '401b69a0-3d07-47f8-bfea-4bc30aa51403'
patientId4 = '1794b7b4-de8c-479a-bfa7-2ab868c3061b'

draw_image(patientId2)
```