



**RSNA**  
Radiological Society  
of North America



Peter and 3 collaborators


**Exploratory Data Analysis**

last run 2 months ago · IPython Notebook HTML · 14,647 views  
using data from [RSNA Pneumonia Detection Challenge](#) ·  Public

▲

232

voters



Notebook

Code

Data (1)

Comments (36)

Log

Versions (6)

Forks (556)

Fork Notebook

Notebook

## Overview

Welcome to the 2018 RSNA Challenge co-hosted by Kaggle. In this competition, the primary endpoint will be the detection of bounding boxes corresponding to the diagnosis of pneumonia (e.g. lung infection) on chest radiographs, a special 2D high resolution grayscale medical image. Note that pneumonia is just one of many possible disease processes that can occur on a chest radiograph, and that any given single image may contain 0, 1 or many boxes corresponding to possible pneumonia locations.

My name is Peter Chang, MD. I am both a radiologist physician and a data scientist / software engineer with machine learning experience. Today, in this Jupyter notebook, we will explore the 2018 RSNA Challenge dataset including underlying data structures, imaging file formats and label types.

```
In [1]: import glob, pylab, pandas as pd
import pydicom, numpy as np
```

## Challenge Data

The challenge data is organized in several files and folders. If you are following along in the Kaggle kernel, this data will be preloaded in the `../input` directory:

```
In [2]: !ls ../input
```

```
GCP Credits Request Link - RSNA.txt  stage_1_test_images
stage_1_detailed_class_info.csv       stage_1_train_images
stage_1_sample_submission.csv         stage_1_train_labels.csv
```

The several key items in this folder:

- `stage_1_train_labels.csv`: CSV file containing training set patientIds and labels (including bounding boxes)
- `stage_1_detailed_class_info.csv`: CSV file containing detailed labels (explored further below)
- `stage_1_train_images/`: directory containing training set raw image (DICOM) files

Let's go ahead and take a look at the first labels CSV file first:

```
In [3]: df = pd.read_csv('../input/stage_1_train_labels.csv')
print(df.iloc[0])
```

```
patientId    0004cfab-14fd-4e49-80ba-63a80b6bddd6
x            NaN
y            NaN
width        NaN
height       NaN
Target       0
Name: 0, dtype: object
```

As you can see, each row in the CSV file contains a `patientId` (one unique value per patient), a `target` (either 0 or 1 for absence or presence of pneumonia, respectively) and the corresponding abnormality bounding box defined by the upper-left hand corner (`x`, `y`) coordinate and its corresponding width and height. In this particular case, the patient does *not* have pneumonia and so the corresponding bounding box information is set to `NaN`. See an example case with pneumonia here:

In [4]:

```
print(df.iloc[4])
```

```
patientId    00436515-870c-4b36-a041-de91049b9ab4
x            264
y            152
width        213
height       379
Target       1
Name: 4, dtype: object
```

One important thing to keep in mind is that a given `patientId` may have **multiple** boxes if more than one area of pneumonia is detected (see below for example images).

## Overview of DICOM files and medical images

Medical images are stored in a special format known as DICOM files ( `*.dcm` ). They contain a combination of header metadata as well as underlying raw image arrays for pixel data. In Python, one popular library to access and manipulate DICOM files is the `pydicom` module. To use the `pydicom` library, first find the DICOM file for a given `patientId` by simply looking for the matching file in the `stage_1_train_images/` folder, and then use the `pydicom.read_file()` method to load the data:

In [5]:

```
patientId = df['patientId'][0]
dcm_file = '../input/stage_1_train_images/%s.dcm' % patientId
dcm_data = pydicom.read_file(dcm_file)

print(dcm_data)
```

```

(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: 1.2.276.0.723001
0.3.1.4.8323329.28530.1517874485.775526
(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: '0004cfab-14fd-4e
49-80ba-63a80b6bddd6'
(0010, 0020) Patient ID                     LO: '0004cfab-14fd-4e
49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date           DA: ''
(0010, 0040) Patient's Sex                  CS: 'F'
(0010, 1010) Patient's Age                  AS: '51'
(0018, 0015) Body Part Examined             CS: 'CHEST'
(0018, 5101) View Position                  CS: 'PA'
(0020, 000d) Study Instance UID             UI: 1.2.276.0.723001
0.3.1.2.8323329.28530.1517874485.775525
(0020, 000e) Series Instance UID            UI: 1.2.276.0.723001
0.3.1.3.8323329.28530.1517874485.775524
(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: ['0.143000000000000
0002', '0.143000000000000002']
(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 142006 b
ytes

```

Most of the standard headers containing patient identifiable information have been anonymized (removed) so we are left with a relatively sparse set of metadata. The primary field we will be accessing is the underlying pixel data as follows:

In [6]:

```
im = dcm_data.pixel_array
print(type(im))
print(im.dtype)
print(im.shape)
```

```
<class 'numpy.ndarray'>
uint8
(1024, 1024)
```

## Considerations

As we can see here, the pixel array data is stored as a Numpy array, a powerful numeric Python library for handling and manipulating matrix data (among other things). In addition, it is apparent here that the original radiographs have been preprocessed for us as follows:

- The relatively high dynamic range, high bit-depth original images have been rescaled to 8-bit encoding (256 grayscales). For the radiologists out there, this means that the images have been windowed and leveled already. In clinical practice, manipulating the image bit-depth is typically done manually by a radiologist to highlight certain disease processes. To visually assess the quality of the automated bit-depth downscaling and for considerations on potentially improving this baseline, consider consultation with a radiologist physician.
- The relatively large original image matrices (typically acquired at >2000 x 2000) have been resized to the data-science friendly shape of 1024 x 1024. For the purposes of this challenge, the diagnosis of most pneumonia cases can typically be made at this resolution. To visually assess the feasibility of diagnosis at this resolution, and to determine the optimal resolution for pneumonia detection (oftentimes can be done at a resolution *even smaller* than 1024 x 1024), consider consultation with a radiologist physician.

## Visualizing An Example

To take a look at this first DICOM image, let's use the `pylab.imshow()` method:

In [7]:

```
pylab.imshow(im, cmap=pylab.cm.gist_gray)
pylab.axis('off')
```

Out[7]:

```
(-0.5, 1023.5, 1023.5, -0.5)
```





## Exploring the Data and Labels

As alluded to above, any given patient may potentially have many boxes if there are several different suspicious areas of pneumonia. To collapse the current CSV file dataframe into a dictionary with unique entries, consider the following method:

```
In [8]: def parse_data(df):
        """
        Method to read a CSV file (Pandas dataframe) and parse the
        data into the following nested dictionary:

        parsed = {

            'patientId-00': {
                'dicom': path/to/dicom/file,
                'label': either 0 or 1 for normal or pneumonia,
                'boxes': list of box(es)
            },
            'patientId-01': {
                'dicom': path/to/dicom/file,
                'label': either 0 or 1 for normal or pneumonia,
                'boxes': list of box(es)
            }, ...
        }

        """
        # --- Define lambda to extract coords in list [y, x, height, width]
        extract_box = lambda row: [row['y'], row['x'], row['height'], row[
'width']]

        parsed = {}
        for n, row in df.iterrows():
            # --- Initialize patient entry into parsed
            pid = row['patientId']
            if pid not in parsed:
                parsed[pid] = {
                    'dicom': '../input/stage_1_train_images/%s.dcm' % pid,
                    'label': row['Target'],
                    'boxes': []}

            # --- Add box if opacity is present
            if parsed[pid]['label'] == 1:
```

```

        parsed[pid]['boxes'].append(extract_box(row))

    return parsed

```

Let's use the method here:

```

In [9]:
    parsed = parse_data(df)

```

As we saw above, patient 00436515-870c-4b36-a041-de91049b9ab4 has pneumonia so let's check our new `parsed` dict here to see the patient's corresponding bounding boxes:

```

In [10]:
    print(parsed['00436515-870c-4b36-a041-de91049b9ab4'])

{'dicom': '../input/stage_1_train_images/00436515-870c-4b36-a041-de91049b9ab4.dcm', 'label': 1, 'boxes': [[152.0, 264.0, 379.0, 213.0], [152.0, 562.0, 453.0, 256.0]]}

```

## Visualizing Boxes

In order to overlay color boxes on the original grayscale DICOM files, consider using the following methods (below, the main method `draw()` requires the method `overlay_box()`):

```

In [11]:
    def draw(data):
        """
        Method to draw single patient with bounding box(es) if present

        """
        # --- Open DICOM file
        d = pydicom.read_file(data['dicom'])
        im = d.pixel_array

        # --- Convert from single-channel grayscale to 3-channel RGB
        im = np.stack([im] * 3, axis=2)

        # --- Add boxes with random color if present
        for box in data['boxes']:
            rgb = np.floor(np.random.rand(3) * 256).astype('int')
            im = overlay_box(im=im, box=box, rgb=rgb, stroke=6)

        pylab.imshow(im, cmap=pylab.cm.gist_gray)
        pylab.axis('off')

```

```

pylab.axis( off )

def overlay_box(im, box, rgb, stroke=1):
    """
    Method to overlay single box on image

    """
    # --- Convert coordinates to integers
    box = [int(b) for b in box]

    # --- Extract coordinates
    y1, x1, height, width = box
    y2 = y1 + height
    x2 = x1 + width

    im[y1:y1 + stroke, x1:x2] = rgb
    im[y2:y2 + stroke, x1:x2] = rgb
    im[y1:y2, x1:x1 + stroke] = rgb
    im[y1:y2, x2:x2 + stroke] = rgb

    return im

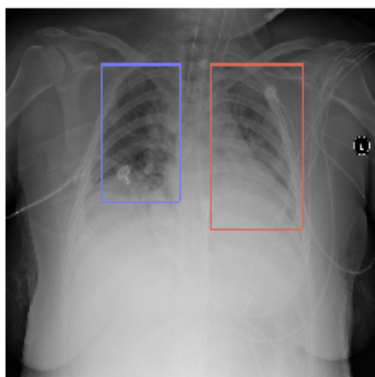
```

As we saw above, patient 00436515-870c-4b36-a041-de91049b9ab4 has pneumonia so let's take a look at the overlaid bounding boxes:

```

In [12]: draw(parsed[ '00436515-870c-4b36-a041-de91049b9ab4' ])

```



## Exploring Detailed Labels

In this challenge, the primary endpoint will be the detection of bounding boxes consisting of a binary classification---e.g. the presence or absence of pneumonia. However, in addition to the binary classification, each bounding box *without* pneumonia is further categorized into *normal* or *no lung opacity / not normal*. This extra third class indicates that while pneumonia was determined not to be present, there was nonetheless some type of abnormality on the image---and oftentimes this finding



may mimic the appearance of true pneumonia. Keep in mind that this extra class is provided as supplemental information to help improve algorithm accuracy if needed; generation of this separate class **will not** be a formal metric used to evaluate performance in this competition.

As above, we saw that the first patient in the CSV file did not have pneumonia. Let's look at the detailed label information for this patient:

```
In [13]: df_detailed = pd.read_csv('../input/stage_1_detailed_class_info.csv')
print(df_detailed.iloc[0])
```

```
patientId    0004cfab-14fd-4e49-80ba-63a80b6bddd6
class                No Lung Opacity / Not Normal
Name: 0, dtype: object
```

As we see here, the patient does not have pneumonia however *does* have another imaging abnormality present. Let's take a closer look:

```
In [14]: patientId = df_detailed['patientId'][0]
draw(parsed[patientId])
```



While the image displayed inline within the notebook is small, as a radiologist it is evident that the patient has several well circumscribed nodular densities in the left lung (right side of image). In addition there is a large chest tube in the right lung (left side of the image) which has been placed to drain fluid accumulation (e.g. pleural effusion) at the right lung base that also demonstrates overlying patchy densities (e.g. possibly atelectasis or partial lung collapse).

As you can see, there are a number of abnormalities on the image, and the determination that none of these findings correlate to pneumonia is somewhat subjective even among expert physicians. Therefore, as is almost always the case in medical imaging datasets, the provided ground-truth labels are far from 100% objective. Keep this in mind as you develop your algorithm, and consider consultation with a radiologist physician to help determine an optimal strategy for mitigating these

discrepancies.

## Label Summary

Finally, let us take a closer look at the distribution of labels in the dataset. To do so we will first parse the detailed label information:

```
In [15]: summary = {}
for n, row in df_detailed.iterrows():
    if row['class'] not in summary:
        summary[row['class']] = 0
    summary[row['class']] += 1

print(summary)

{'No Lung Opacity / Not Normal': 11500, 'Normal': 8525, 'Lung Opacity': 8964}
```

As we can see, there is a relatively even split between the three classes, with nearly 2/3rd of the data comprising of no pneumonia (either completely *normal* or *no lung opacity / not normal*). Compared to most medical imaging datasets, where the prevalence of disease is quite low, this dataset has been significantly enriched with pathology.

Did you find this Kernel useful?  
Show your appreciation with an upvote

232



### Comments (36)

All Comments

Sort by

Hotness



Click here to enter a comment...



**Gabriel** • Posted on Latest Version • 2 months ago • Options • Reply

11

"this dataset has been significantly enriched with pathology" - can we reasonably assume a similar distribution for the test set? It seems like there's risk of overfitting to the distribution of the training set vs. real-world data. I.e. if the test & train data skews towards pneumonia, our trained models generally should as well which may not be as useful for the real-world case.

**Matt** • Posted on Latest Version • 2 months ago • Options • Reply

5



As you seem to note, the bounding box is fairly coarse. Will the accuracy of the bounding box be a consideration? How will you judge accuracy? It seems like there may be a benefit to being overly broad with the boxes.



MichelleS Lee • Posted on Latest Version • 2 months ago • Options • Reply

2



As the evaluation criteria states the consideration of precision of each image (using intersection over union of the prediction vs ground truth) I think the accuracy of the bounding box will affect the evaluation.



Tim Hoolihan • Posted on Latest Version • 2 months ago • Options • Reply

0

Per the Intersection over Union section of <https://www.kaggle.com/c/rsna-pneumonia-detection-challenge#evaluation>, the evaluation using IoU will discount oversized or off-center boxes using that formula.



Kanwalinder Singh... • Posted on Latest Version • 2 months ago • Options • Reply

2

Hi Peter,

I got the following results on unique examples:

Normal examples (Normal): 8525

Pneumonia examples (Lung Opacity): 5659

Other abnormal examples (No Lung Opacity / Not Normal): 11500

Total examples:: 25684

My results differ from your **Lung Opacity** number of **8964**. Did I make a mistake or your Lung Opacity number is total examples, not unique ones?

Please see my [First Pass Analysis](#). I computed the numbers in shell, so this might provide a sanity check.



Larry Tenny • Posted on Latest Version • 2 months ago • Options • Reply

1



Patients can be represented more than once...it looks like you filtering thru uniq and removing duplicated, that would explain the lower numbers.



Phillip Cheng • Posted on Latest Version • 2 months ago • Options • Reply

1

8964 is the total number of ground truth bounding boxes. 5659 is the total number of patients with pneumonia. Each of these patients has 1-4 ground truth bounding boxes.



**taindow** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Really useful kernel, thank you very much for the introduction.



**Brian** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Thanks



**Shreyas Prabhud...** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Thank you



**Rascale** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Nice exploratory kernel



**Deva GN** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Nice one



**Rafael** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Thanks a lot! Really nice overview of the data



**Achal Agarwal** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Nice introduction. Thank you!



**tonami** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

nice one



**JuanOI** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Very nice explanation. Thanks!



**Larry Tenny** • Posted on Latest Version • 2 months ago • Options • Reply

0

Thanks..great starting point. I'm looking forward to working on this.



**TSHLNIHAO** • Posted on Latest Version • 2 months ago • Options • Reply

0

good eda,it is very useful to understang the picture and the label by show them



**Carlos Ferrin** • Posted on Latest Version • 2 months ago • Options • Reply

0

Thanks!



**anad** • Posted on Latest Version • 2 months ago • Options • Reply

0

Thank you. Very well explained your findings.



**Makundi.Jr** • Posted on Latest Version • 2 months ago • Options • Reply

0

Nice one.



**Yuho** • Posted on Latest Version • 2 months ago • Options • Reply

0

Thank you! Your kernel is so helpful as I didn't know anything about DICOM format.



**Jack** • Posted on Latest Version • 2 months ago • Options • Reply

0

Thank you. Very helpful kernel.



**Sven** • Posted on Latest Version • 2 months ago • Options • Reply

0

Very helpful this explore data analysis notebook, shows the data files, structures, and basic analysis. Very useful to do next step analysis, thanks.



**Nataly Lundqvist** • Posted on Latest Version • 2 months ago • Options • Reply

0

What tools one suppose to use?? May I use TensorFlow and Google DataLab with suitable instance?



**Chirag** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Great intro to the challenge! Thanks very much for the writeup



**Kelvin** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Is anyone getting the following error when trying to obtain DICOM file's pixel array?

```
sample_dcm_data = pydicom.read_file('stage_1_train_images/0004cfab-14fd-4e49-80ba-63a80b6bddd6.dcm')
sample_image_array = sample_dcm_data.pixel_array
```

NotImplementedError: No available image handler could decode this transfer syntax JPEG Baseline (Process 1)



**Kelvin** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Nevermind, just needed to install Pillow



**Alpamys Tolegen** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

Thanks a lot. I hope it will help me



**Ejskwk** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

While visualizing some images through 1- 30, the dataset seems to contain duplicate images. Is this ok?



**Ömer Şayli** • Posted on Latest Version • 2 months ago • Options • Reply

^ 0 v

They are not duplicate, one image can contain more than one "bounding boxes". Each row is for another image or box of the same image.



**Ejskwk** • Posted on Latest Version • a month ago • Options • Reply

^ 0 v

Thanks



**Shanshan\_Wang** • Posted on Latest Version • a month ago • Options • Reply

^ 0 v

Very nice one! It helps me a lot in understanding the data. Thanks!



**runbotic** • Posted on Latest Version • a month ago • Options • Reply

^ 0 v

nice kernel, thx



**DineshCTech** • Posted on Latest Version • a month ago • Options • Reply

^ 0 v

An initial insight from a radiologist helped me to get started on this.



2 months ago

This Comment was deleted.