

RSNA-Intracranial-Hemorrhage-Detection

Repo for RSNA Intracranial Hemorrhage Detection

Intracranial Hemorrhage Detection

This blog post is about the challenge that is hosted on kaggle on [RSNA Intracranial Hemorrhage Detection](#).

This post is divided into following parts

1. Overview
2. Basic EDA [Ipython Notebook](#)
3. Data Visualization & Preprocessing
4. Deep Learning Model

1. Overview

What is Intracranial Hemorrhage?

An intracranial hemorrhage is a type of bleeding that occurs inside the skull. Symptoms include sudden tingling, weakness, numbness, paralysis, severe headache, difficulty with swallowing or vision, loss of balance or coordination, difficulty understanding, speaking, reading, or writing, and a change in level of consciousness or alertness, marked by stupor, lethargy, sleepiness, or coma. Any type of bleeding inside the skull or brain is a medical emergency. It is important to get the person to a hospital emergency room immediately to determine the cause of the bleeding and begin medical treatment. It requires highly trained specialists review medical images of the patient's cranium to look for the presence, location and type of hemorrhage. The process is complicated and often time consuming. So as part of this we will be deep learning techniques to detect acute intracranial hemorrhage and its subtypes.

Hemorrhage Types

1. Epidural
2. Intraparenchymal
3. Intraventricular
4. Subarachnoid
5. Subdural
6. Any

What am i predicting?

In this competition our goal is to predict intracranial hemorrhage and its subtypes. Given an image the we need to predict probability of each subtype. This indicates its a multilabel classification problem.

Evaluation Metric

Competition evaluation metric is **weighted log loss** but weights for each subtype is not disclosed as part of the competition but in the discussion forms some of the teams found it out that the any label has a weight of 2 compared to other subtypes, you can check more details [here](#). But as part of this tutorial i'm going to use normal accuracy as evaluation metric and loss as **binary cross entropy loss** and checkpointing the models based on the loss.

2. Basic EDA

Lets look at the [data](#) that is provided.

We have a train.csv containing file names and label indicating whether hemorrhage is present or not and train images folder which is set of [Dicom](#) files (Medical images are stored in dicom formats) and test images folder containing test dicom files.

```
# load the csv file
train_df = pd.read_csv(input_folder + 'stage_1_train.csv')
train_df.head()
```

	ID	Label
0	ID_63eb1e259_epidural	0
1	ID_63eb1e259_intraparenchymal	0
2	ID_63eb1e259_intraventricular	0
3	ID_63eb1e259_subarachnoid	0
4	ID_63eb1e259_subdural	0

It consists of two columns ID and Label. ID has a format FILE_ID_SUB_TYPE for example ID_63eb1e259_epidural so ID_63eb1e259 is file id and epidural is subtype and Label indicating whether subtype hemorrhage is present or not.

Lets separate file names and subtypes

```
# extract subtype
train_df['sub_type'] = train_df['ID'].apply(lambda x: x.split('_')[-1])
# extract filename
train_df['file_name'] = train_df['ID'].apply(lambda x: '_'.join(x.split('_')[:2]) + '.dcm')
train_df.head()
```

	ID	Label	sub_type	file_name
0	ID_63eb1e259_epidural	0	epidural	ID_63eb1e259.dcm
1	ID_63eb1e259_intraparenchymal	0	intraparenchymal	ID_63eb1e259.dcm
2	ID_63eb1e259_intraventricular	0	intraventricular	ID_63eb1e259.dcm
3	ID_63eb1e259_subarachnoid	0	subarachnoid	ID_63eb1e259.dcm
4	ID_63eb1e259_subdural	0	subdural	ID_63eb1e259.dcm

```
train_df.shape
```

Output : (4045572, 4)

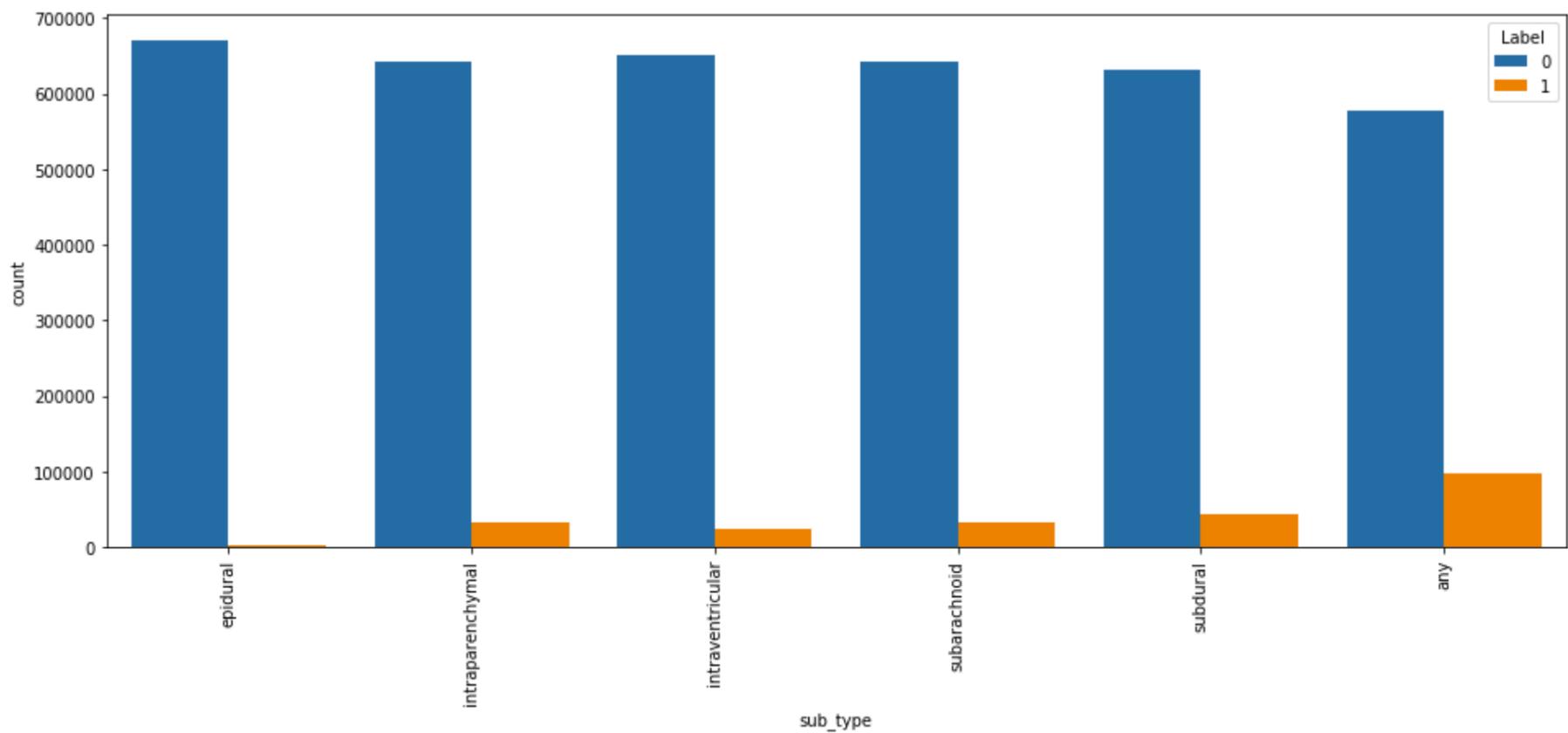
```
print("Number of train images available:", len(os.listdir(path_train_img)))
```

Output : Number of train images available: 674258

The csv file has a shape of (4045572, 4). For every file(dicom file) present in the train folder has 6 entries in csv indicating possible 6 subtype hemorrhages.

Lets check the files available for each subtype

```
plt.figure(figsize=(16, 6))
graph = sns.countplot(x="sub_type", hue="Label", data=train_df)
graph.set_xticklabels(graph.get_xticklabels(), rotation=90)
plt.show()
```



Lets check the counts for each subtype

Epidural

```
train_df[train_df['sub_type'] == 'epidural']['Label'].value_counts()
```

Output:

0 671501

1 2761

Name: Label, dtype: int64

For epidural sub type we have 6,71,501 images labeled as 0 and 2,761 labelled as 1.

Intraparenchymal

```
train_df[train_df['sub_type'] == 'intraparenchymal']['Label'].value_counts()
```

Output:

0 641698

1 32564

Name: Label, dtype: int64

For intraparenchymal sub type we have 6,41,698 images labeled as 0 and 32,564 labelled as 1.

Intraparenchymal

```
train_df[train_df['sub_type'] == 'intraparenchymal']['Label'].value_counts()
```

Output:

0 650496

1 23766

Name: Label, dtype: int64

For intraparenchymal sub type we have 6,50,496 images labeled as 0 and 23,766 labelled as 1.

Subarachnoid

```
train_df[train_df['sub_type'] == 'subarachnoid']['Label'].value_counts()
```

Output:
0 642140
1 32122
Name: Label, dtype: int64

For subarachnoid sub type we have 6,42,140 images labeled as 0 and 32,122 labelled as 1.

Subdural

```
train_df[train_df['sub_type'] == 'subdural']['Label'].value_counts()
```

Output:
0 631766
1 42496
Name: Label, dtype: int64

For Subdural sub type we have 6,31,766 images labeled as 0 and 42,496 labelled as 1.

Any

```
train_df[train_df['sub_type'] == 'any']['Label'].value_counts()
```

Output:
0 577159
1 97103
Name: Label, dtype: int64

For any sub type we have 5,77,159 images labeled as 0 and 97,103 labelled as 1.

3. Data Visualization & Preprocessing

Lets look at the dicom files in the dataset

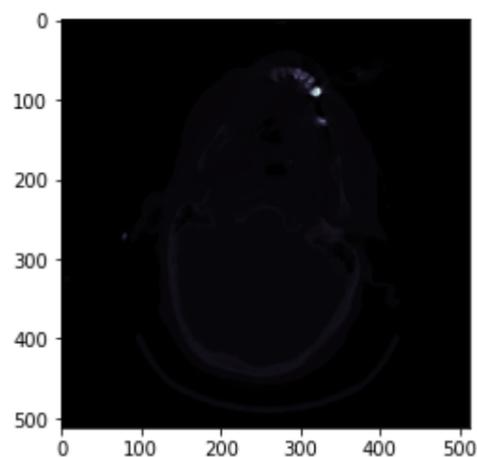
```
dicom = pydicom.read_file(path_train_img + 'ID_ffff922b9.dcm')  
print(dicom)
```

```
(0008, 0018) SOP Instance UID          UI: ID_ffff922b9  
(0008, 0060) Modality                  CS: 'CT'  
(0010, 0020) Patient ID                LO: 'ID_5964c5e5'  
(0020, 000d) Study Instance UID        UI: ID_b47ca0ad05  
(0020, 000e) Series Instance UID       UI: ID_6d2a9b2810  
(0020, 0010) Study ID                  SH: ''  
(0020, 0032) Image Position (Patient)  DS: ['-126.408875', '-126.408875', '-235.611511']  
(0020, 0037) Image Orientation (Patient) DS: ['1.000000', '0.000000', '0.000000', '0.000000', '1.000000', '0.000000']  
(0028, 0002) Samples per Pixel         US: 1  
(0028, 0004) Photometric Interpretation CS: 'MONOCHROME2'  
(0028, 0010) Rows                      US: 512  
(0028, 0011) Columns                   US: 512  
(0028, 0030) Pixel Spacing             DS: ['0.494750976563', '0.494750976563']  
(0028, 0100) Bits Allocated            US: 16  
(0028, 0101) Bits Stored                US: 16  
(0028, 0102) High Bit                  US: 15  
(0028, 0103) Pixel Representation       US: 1  
(0028, 1050) Window Center              DS: "35.000000"  
(0028, 1051) Window Width              DS: "135.000000"  
(0028, 1052) Rescale Intercept          DS: "-1024.000000"  
(0028, 1053) Rescale Slope             DS: "1.000000"  
(7fe0, 0010) Pixel Data                 OW: Array of 524288 elements
```

Dicom data format files contain pixel data of image and other meta data like patient name, instance id, window width etc...

Original image

```
plt.imshow(dicom.pixel_array, cmap=plt.cm.bone)
plt.show()
```



The original image seems to have difficult to understand, lets check meta data features like Window Center, Window Width, Rescale Intercept, Rescale Slope

```
(0028, 1050) Window Center      DS: "35.000000"
(0028, 1051) Window Width      DS: "135.000000"
(0028, 1052) Rescale Intercept  DS: "-1024.000000"
(0028, 1053) Rescale Slope     DS: "1.000000"
```

We can use these features to construct the new image.

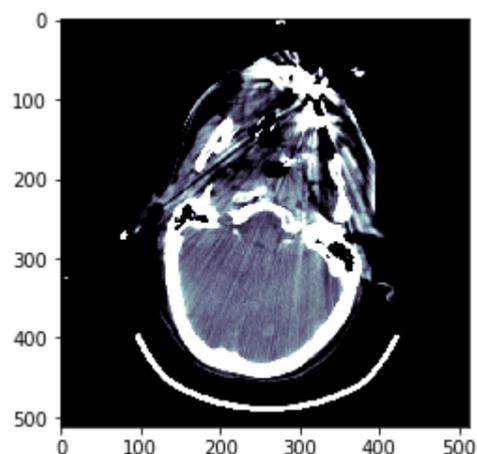
```
def get_dicom_field_value(key, dicom):
    """
    @param key: key is tuple
    @param dicom: dicom file
    """
    return dicom[key].value

window_center = int(get_dicom_field_value(('0028', '1050'), dicom))
window_width = int(get_dicom_field_value(('0028', '1051'), dicom))
window_intercept = int(get_dicom_field_value(('0028', '1052'), dicom))
window_slope = int(get_dicom_field_value(('0028', '1053'), dicom))
window_center, window_width, window_intercept, window_slope

def get_windowed_image(image, wc, ww, intercept, slope):
    img = (image*slope +intercept)
    img_min = wc - ww//2
    img_max = wc + ww//2
    img[img<img_min] = img_min
    img[img>img_max] = img_max
    return img

windowed_image = get_windowed_image(dicom.pixel_array, window_center, window_width, \
                                   window_intercept, window_slope)

plt.imshow(windowed_image, cmap=plt.cm.bone)
plt.show()
```

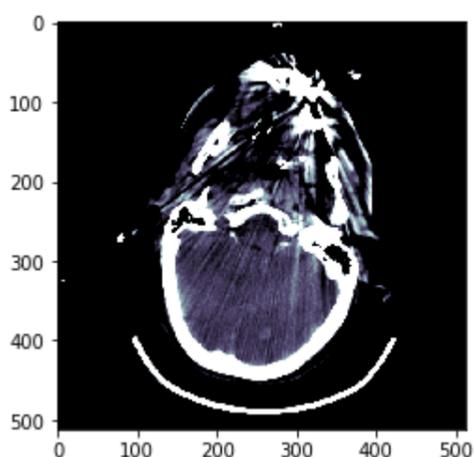


The windowed image using meta data is much better than the original image this is because the dicom pixel array which contain pixel data contain raw data in Hounsfield units (HU).

Scaling the image:

Rescale the image to range 0-255.

```
def get_scaled_windowed_image(img):  
    """  
    Get scaled image  
    1. Convert to float  
    2. Rescale to 0-255  
    3. Convert to unit8  
    """  
    img_2d = img.astype(float)  
    img_2d_scaled = (np.maximum(img_2d, 0) / img_2d.max()) * 255.0  
    img_2d_scaled = np.uint8(img_2d_scaled)  
    return img_2d_scaled  
  
scaled_image = get_scaled_windowed_image(windowed_image)  
plt.imshow(scaled_image, cmap=plt.cm.bone, vmin=0, vmax=255)  
plt.show()
```

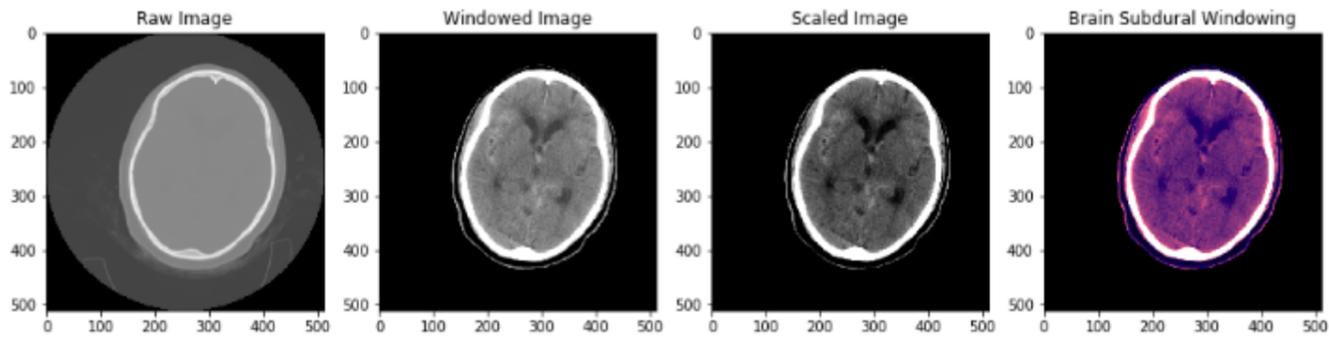


Hounsfield Units (HU) are the best source for constructing CT images. [Here](#) is detailed table showing the substance and HU range.

A detailed explanation of all the possible windowing techniques can be found in this great kernel ([Gradient Sigmoid Windowing](#))

```
def correct_dcm(dcm):  
    # Refer Jeremy Howard's Kernel https://www.kaggle.com/jhoward/from-prototyping-to-submission-fastai  
    x = dcm.pixel_array + 1000  
    px_mode = 4096  
    x[x >= px_mode] = x[x >= px_mode] - px_mode  
    dcm.PixelData = x.tobytes()  
    dcm.RescaleIntercept = -1000  
  
def window_image(dcm, window_center, window_width):  
  
    if (dcm.BitsStored == 12) and (dcm.PixelRepresentation == 0) and (int(dcm.RescaleIntercept) > -100):  
        correct_dcm(dcm)  
  
    img = dcm.pixel_array * dcm.RescaleSlope + dcm.RescaleIntercept  
    img_min = window_center - window_width // 2  
    img_max = window_center + window_width // 2  
    img = np.clip(img, img_min, img_max)  
  
    return img  
  
def bsb_window(dcm):  
    brain_img = window_image(dcm, 40, 80)  
    subdural_img = window_image(dcm, 80, 200)  
    soft_img = window_image(dcm, 40, 380)  
  
    brain_img = (brain_img - 0) / 80  
    subdural_img = (subdural_img - (-20)) / 200  
    soft_img = (soft_img - (-150)) / 380  
    bsb_img = np.array([brain_img, subdural_img, soft_img]).transpose(1, 2, 0)  
  
    return bsb_img
```

```
display_dicom_image('ID_0005d340e.dcm')
```



It looks like Brain + Subdural is a good start for our models it has three channels and can be easily fed to any pretrained models.

4. Deep Learning Model

The whole code for the training of the model can be found [here](#)

We will use normal windowed images for training the model with augmentations like flip left right and random cropping.

Here are steps for training the model

1. Prepare train and validation data generators we will be splitting the data by stratifying the labels here is the link to [multilabel stratification](#). We will make two splits and only work on the first split and check the results.
2. Load pretrained EfficientNet B0 model.
3. For the first epoch use all the train images for training the model with the first head layers using as is by setting trainable as False but train all the later images and save the model.
4. Load the saved model and for the further epochs we train whole model except the last layer thus our model will learn most complicated features.
5. Make predictions.

Sample code:

```
# 1. -----prepare data generators-----#
# https://github.com/trent-b/iterative-stratification
# Multilabel stratification
splits = MultilabelStratifiedShuffleSplit(n_splits = 2, test_size = TEST_SIZE, random_state = SEED)
file_names = train_final_df.index
labels = train_final_df.values
# Lets take only the first split
split = next(splits.split(file_names, labels))
train_idx = split[0]
valid_idx = split[1]
submission_predictions = []
len(train_idx), len(valid_idx)
# train data generator
data_generator_train = TrainDataGenerator(train_final_df.iloc[train_idx],
                                          train_final_df.iloc[train_idx],
                                          TRAIN_BATCH_SIZE,
                                          (WIDTH, HEIGHT),
                                          augment = True)

# validation data generator
data_generator_val = TrainDataGenerator(train_final_df.iloc[valid_idx],
                                       train_final_df.iloc[valid_idx],
                                       VALID_BATCH_SIZE,
                                       (WIDTH, HEIGHT),
                                       augment = False)

# 2. -----load efficient net B0 model-----#
base_model = efn.EfficientNetB0(weights = 'imagenet', include_top = False, \
                                pooling = 'avg', input_shape = (HEIGHT, WIDTH, 3))

x = base_model.output
x = Dropout(0.125)(x)
output_layer = Dense(6, activation = 'sigmoid')(x)
model = Model(inputs=base_model.input, outputs=output_layer)
model.compile(optimizer = Adam(learning_rate = 0.0001),
              loss = 'binary_crossentropy',
              metrics = ['acc', tf.keras.metrics.AUC()])
```


Output: (78545, 6)

As per sample submission given by kaggle it is in a different format, the submission should be made with ID and Label column where ID is in the form of **dicomId_subType**(Ex:ID_0fbf6a978_subarachnoid) so we need format this to convert each prediction to 6 rows each indicating the id with sub type and its probability. The following code generates the required format for submission.

```
def create_download_link(title = "Download CSV file", filename = "data.csv"):
    """
    Helper function to generate download link to files in kaggle kernel
    """
    html = '<a href={filename}>{title}</a>'
    html = html.format(title=title,filename=filename)
    return HTML(html)

def generate_submission_file(preds):
    from tqdm import tqdm

    cols = list(train_final_df.columns)

    # We have predictions for each of the image
    # We need to make 6 rows for each of file according to the subtype
    ids = []
    values = []
    for i, j in tqdm(zip(preds, test_df.index.to_list()), total=preds.shape[0]):
        # print(i, j)
        # i=[any_prob, epidural_prob, intraparenchymal_prob, intraventricular_prob, subarachnoid_prob, subdural_prob]
        # j = filename ==> ID_xyz.dcm
        for k in range(i.shape[0]):
            ids.append([j.replace('.dcm', '_' + cols[k])])
            values.append(i[k])

    df = pd.DataFrame(data=ids)
    df.head()

    sample_df = pd.read_csv(input_folder + 'stage_1_sample_submission.csv')
    sample_df.head()

    df['Label'] = values
    df.columns = sample_df.columns
    df.head()

    df.to_csv('submission.csv', index=False)

    return create_download_link(filename='submission.csv')
```

```
df = pd.read_csv('submission.csv')
df.head()
```

	ID	Label
0	ID_000012eaf_any	0.024590
1	ID_000012eaf_epidural	0.000314
2	ID_000012eaf_intraparenchymal	0.005449
3	ID_000012eaf_intraventricular	0.000154
4	ID_000012eaf_subarachnoid	0.002607

All notebooks can be found [here](#)

References

<https://my.clevelandclinic.org/health/diseases/14480-intracranial-hemorrhage-cerebral-hemorrhage-and-hemorrhagic-stroke>

https://github.com/MGH-LMIC/windows_optimization

<https://arxiv.org/abs/1812.00572>(Must read) <https://www.kaggle.com/c/rsna-intracranial-hemorrhage-detection/discussion/111325#latest-650043> <https://www.kaggle.com/c/rsna-intracranial-hemorrhage-detection/discussion/109261#latest-651855>

Kaggle Kernels

<https://www.kaggle.com/jhoward/some-dicom-gotchas-to-be-aware-of-fastai> <https://www.kaggle.com/reppic/gradient-sigmoid-windowing>
<https://www.kaggle.com/jhoward/from-prototyping-to-submission-fastai> <https://www.kaggle.com/suryaparsa/rsna-basic-eda-part-1>
<https://www.kaggle.com/suryaparsa/rsna-basic-eda-part-2>

This page was generated by [GitHub Pages](#).