Chest X-ray Abnormalities Detection

ECE-GY 9123 | Deep Learning
Spring 2021

Team Members

Kunwar Shivam Srivastav
kss519@nyu.edu
M.S. Computer Engineering

Sagar Patel
sp5894@nyu.edu
M.S. Computer Engineering

Repository Link

Kindly access the Project Repository from the link mentioned below:
https://github.com/Sagar-py/chestXRay-Detection

Table of Contents

Objective

The goal is to automatically localize and classify 14 types of thoracic abnormalities from chest radiographs while working with a dataset consisting of 18,000 scans that have been annotated by experienced radiologists. The model can be trained with 15,000 independently labeled images and will be evaluated on a test set of 3,000 images.
The annotations were to be collected via VinBigData's web-based platform, VinLab. Details on building the dataset were found in the paper “VinDr-CXR: An open dataset of chest X-rays with radiologist's annotations”.

Introduction

An X-ray is an imaging test that uses small amounts of radiation to produce pictures of the organs, tissues, and bones of the body. When focused on the chest, it can help spot abnormalities or diseases of the airways, blood vessels, bones, heart, and lungs. Chest X-rays can also determine if you have fluid in your lungs, or fluid or air surrounding your lungs. The doctor could order a chest X-ray for a variety of reasons, including to assess injuries resulting from an accident or to monitor the progression of a disease, such as cystic fibrosis. A chest X-ray is an easy, quick, and effective test that has been useful for decades to help doctors view some of the most vital organs.
When we have a broken arm, radiologists help save the day - and the bone. These doctors diagnose and treat medical conditions using imaging techniques like CT and PET scans, MRIs, and X-rays. Yet, as it happens when working with such a wide variety of medical tools, radiologists face many daily challenges, perhaps the most difficult being the chest radiograph. The interpretation of chest X-rays can lead to a medical misdiagnosis, even for the best practicing doctor. Computer-aided detection and diagnosis systems would help reduce the pressure on doctors at metropolitan hospitals and improve diagnostic quality in rural areas.
Existing methods of interpreting chest X-ray images classify them into a list of findings. There is currently no specification of their locations on the image which sometimes leads to inexplicable results. A solution for localizing findings on chest X-ray images is needed for providing doctors with more meaningful diagnostic assistance.

Dataset

We intend to classify common thoracic lung diseases and localize them. To achieve this feat, we will work with a dataset consisting of *18,000* scans that have been annotated by radiologists. *15,000* of which will be *independently-labeled training images* while the remaining *3,000* images will be used for *validation*. The dataset comprises *18,000 posteroanterior (PA) CXR* scans in *DICOM* format, which were de-identified to protect patient privacy. All the images were labeled by a panel of experienced radiologists for the presence of 14 critical radiographic findings as listed —

0 - Aortic enlargement
1 - Atelectasis
2 - Calcification
3 - Cardiomegaly
4 - Consolidation
5 - ILD
6 - Infiltration
7 - Lung Opacity
8 - Nodule/Mass

9 - Other lesions **11 - Pleural thickening** **13 - Pulmonary fibrosis**
10 - Pleural effusion **12 - Pneumothorax** 14 - No finding

The *No finding* observation (14) was intended to capture the absence of all findings above. The columns in metadata files in the dataset are given below

- **image_id** - unique image identifier
- **class_name** - the name of the class of detected object (or "No finding")
- **class_id** - the ID of the class of detected object
- **rad_id** - the ID of the radiologist that made the observation
- **x_min** - minimum X coordinate of the object's bounding box
- **y_min** - minimum Y coordinate of the object's bounding box
- **x_max** - maximum X coordinate of the object's bounding box
- **y_max** - maximum Y coordinate of the object's bounding box

Rather than obtaining the dataset, pre-processing, segmenting, and focusing on converting the image from X-ray image format (DICOM) to a normal PNG format which is clearly a superior choice for visualization, we decided to prioritize the modeling aspects of the problem.
We obtained the pre-processed image dataset that required metadata from the popular Kaggle user xhulu. Keeping our computation power in mind, we decided to work with the resized dataset of 256 x 256 images.

Working with detectron2

About detectron2

Detectron2 is Facebook AI Research's next-generation software system that implements state-of-the-art object detection algorithms. It is a ground-up rewrite of the previous version, Detectron, and it originates from mask-rcnn-benchmark. Detectron2 is one of the most famous PyTorch object detection libraries available.
Initially, we considered working with the trained models of YOLOv4 which is a family of compound-scaled object detection models trained on the COCO dataset and includes simple functionality for Test Time Augmentation, model ensembling, and hyperparameter evaluation. However, the lack of resources available on YOLOv4 and the functionality of detectron2's faster R-CNN (with the availability of resources) helped us choose detectron2. Other than that, customizing a trainer using detectron2 is far more convenient and easier than that of YOLOv4. However, there are a lot of similarities between the two architectures. For Example - they both use an anchor box-based network structure and both the architectures use bounding booth regression. The major difference between the two is that YOLO makes classification and bounding box regression at the same time.

Installation

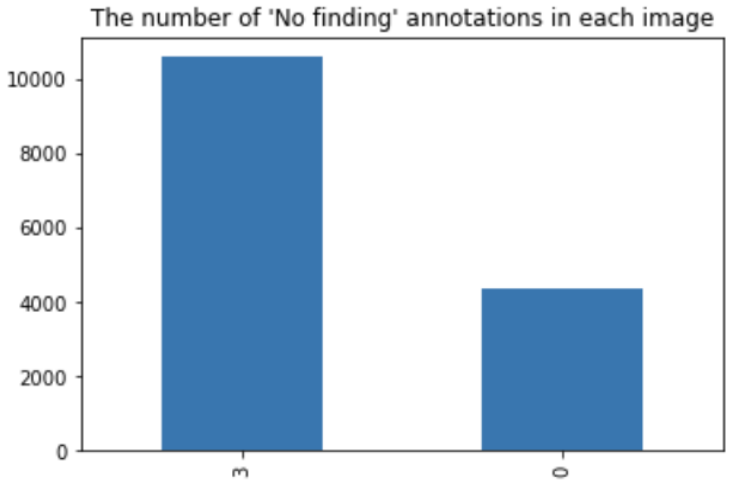
Installing detectron2 could get a bit tricky as we need to know the right version of CUDA and PyTorch to install detectron2. We decided to install CUDA 10.2 as our PyTorch version was 1.7.0. The detectron2 with the right version of CUDA and PyTorch can be installed using the command

```
!pip install detectron2 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu102/torch1.7/index.html
```

The distribution between Normal and Abnormal Classes

While reviewing the metadata file, we noticed that the same `image_id` has been annotated by multiple radiologists and they do not seem to match. An annotation is considered “normal” when all the radiologists unanimously agree to think that the given image is normal or accurate. So we decided to check the number of “No finding” (Class - 14) annotations for each image and if the opinions are in complete agreement, the number of “No finding” annotations should be 0 (Abnormal) or 1 (Normal). Upon further analysis, we found that nearly 70% of the data is actually “Normal” and only 30% of the images needed thoracic abnormality location detection.

	image_id	num_normal_annotations
0	000434271f63a053c4128a0ba6352c7f	3
1	00053190460d56c53cc3e57321387478	3
2	0005e8e3701dfb1dd93d53e2ff537b6e	0
3	0006e0a85696f6bb578e84fafa9a5607	3
4	0007d316f756b3fa0baea2ff514ce945	0



Model

For the model, we implemented *Mask R-CNN* which is a combination of *Faster R-CNN* and *Fully Connected Network*. Faster R-CNN consists of two stages — The first stage (Region Proposal Network), proposes candidate object bounding boxes. The second stage extracts feature using RoIPool from each candidate box and perform classification and bounding-box regression. Faster R-CNN has two outputs for each candidate object, a class label, and a bounding-box offset; to this, we add a third branch that outputs the object mask — which is a binary mask that indicates the pixels where the object is in the bounding box. But the additional mask output is distinct from the class and box outputs, requiring extraction of a much finer layout of an object. To do this, Mask R-CNN uses the Fully Convolutional Network (FCN).

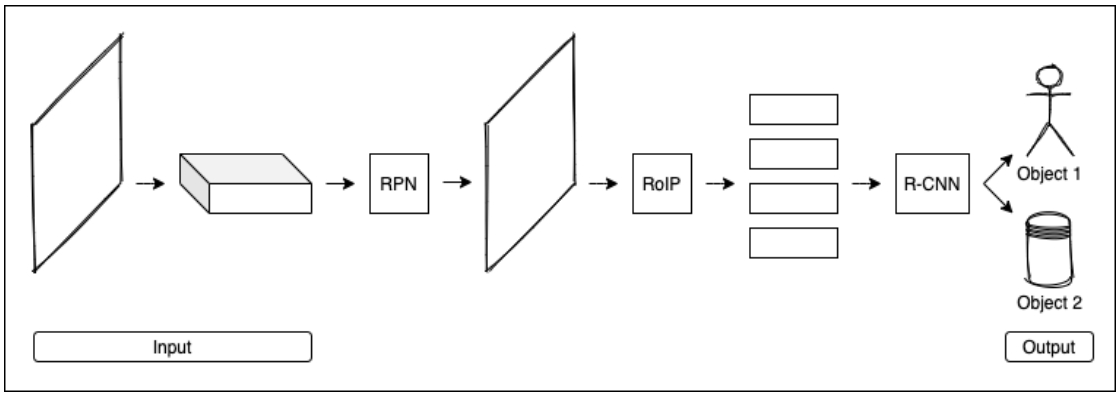
Implementation

The *input images* are represented as `(Height*Width*Depth Height)×Width×(Depth Height*Width*Depth)` tensors (multidimensional arrays), which are then passed through a pre-trained CNN up until an intermediate layer, ending up with a convolution feature map. This is then used as a feature extractor for the next stage. The next stage is RPN. Using the CNN computed features, it

is then used to create a predefined number of regions (bounding boxes), which may contain objects. After having a list of possible relevant objects and their locations in the original image, the solution becomes quite direct. Using the features extracted by the CNN and the bounding boxes with relevant objects, we apply Region of Interest Pooling (RoIPool) and extract those features which would correspond to the relevant objects into a new tensor. Finally, the R-CNN module uses that information to classify the content in the bounding box (or discard it using "background" as a label) and adjust the bounding box coordinates. The *output returns a tensor* in the format —

```
[classID y_max x_min y_min x_max confidenceScore]
```

Example: [14 1 0 0 1 1] would indicate no finding with a 1-pixel bounding box and a confidence score of 0.



Training Method

For training, we decided to experiment with Facebook Research's famous `detectron2` PyTorch library. The reason for doing so is that detectron2 provides a high-level API for training customized datasets. To define a custom dataset, we created a list of dictionaries where each dictionary contains the following:

- **`file_name`**: The filename of the image.
- **`image_id`**: id of the image (Can be done by just normal indexing).
- **`height`**: Height of the image.
- **`width`**: Width of the image.
- **`annotation`**: An annotation class which is the ground truth annotation data for object detection contains
  - Abounding box pixel location with shape (`n_boxes, 4`)
  - A box mode to store the absolute value of `y_max, x_min, y_min,` and `x_max`.
  - Class label id for each bounding box, with shape (`n_boxes, `)

We then decided to create two functions `get_vinbigdata_dicts` for preparing the train dataset and `get_vinbigdata_dicts_test` for the test dataset preparation. Apart from that, we also created a metadata dictionary called `dataset_dicts` for actual data that is fed into the neural network. It is loaded beforehand of the training *on memory*, so it should contain all the metadata (image file path) to construct the training dataset, but *should not contain any heavy data*.
In practice, loading all the training images image arrays are too heavy to be loaded on memory, so these are loaded inside the DataLoader.

Customizing the trainer in detectron2

We wanted to customize the training behavior more to improve the model's performance. Therefore, we decided to make our `Trainer` class, and the `Trainer` class (labeled as `MyTrainer`) is also used to override methods to provide customized behavior.

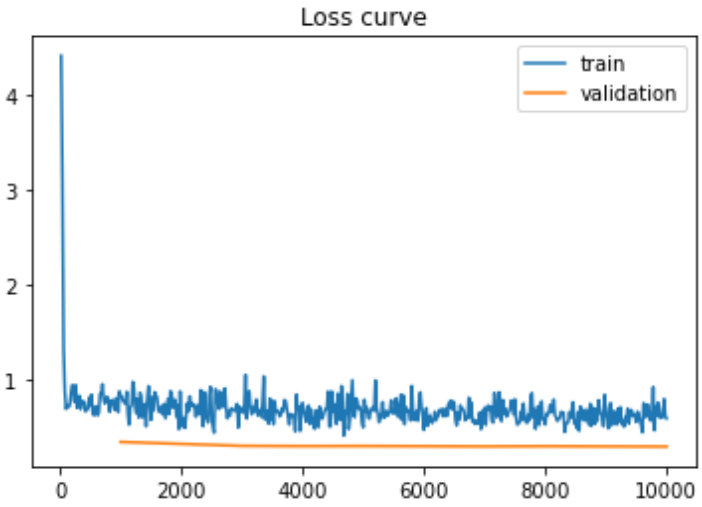
The `Mapper` class is used inside PyTorch's `DataLoader`. The main responsibility of the mapper class is to convert `dataset_dicts` into actual data which is being fed into the neural network so we can insert the augmentation process in the `Mapper` class [8].

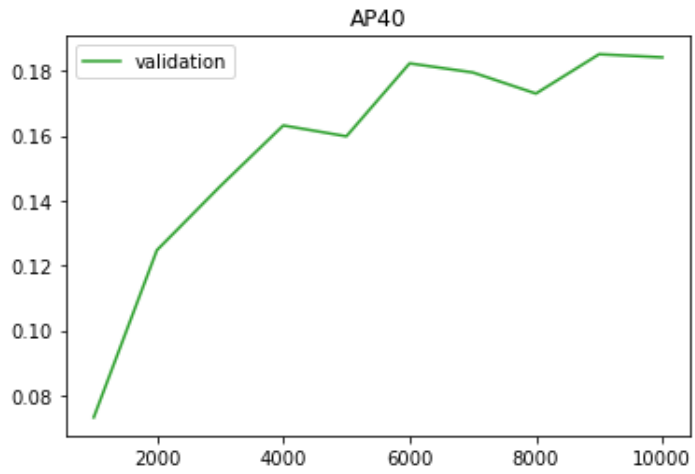
Evaluator and Loss Function

To evaluate the validation dataset to calculate the competition metric, we need an Evaluator. Famouns dataset's evaluator is already implemented in detectron2. For example, many kinds of AP (Average Precision) are calculated in `COCOEvaluator`. `COCOEvaluator` only calculates AP with IoU from 0.50 to 0.95, but we need AP with IoU 0.40. Here, we modified COCOEvaluator implementation to calculate *AP with IoU 0.40* and replaced it to show this value instead of *AP with IoU 0.70.*

Later, we implemented Evaluator and now we can calculate competition metric, however, validation loss is not calculated inside Evaluator. This is because the model's evaluation is done in `model.eval()` mode and it outputs bounding box prediction but does not output loss. To calculate validation loss, we can add *Hook* which calculates the loss to the trainer. Trainer has attribute *storage* and calculated metrics are summarized. Its content is saved to metric.json (JSON format) during training.

Note that the current implementation is inefficient because the Evaluator's evaluation and LossEvalHook's loss calculation run separately, even if both need a model forward calculation for the same validation data. The calculated metrics are saved in metrics.json. We can plot them to check how the training proceeded. AP40 stands for AP with IoU 0.40





Hyperparameters for Loading and Tuning the Data

The `Flags` class is used to manage the tuning. We referred to a couple of notebooks to get an approximate idea of the parameter values that need to be tuned to and those parameters with values were then stored in a dictionary called `flags_dict` and the parameters are given by

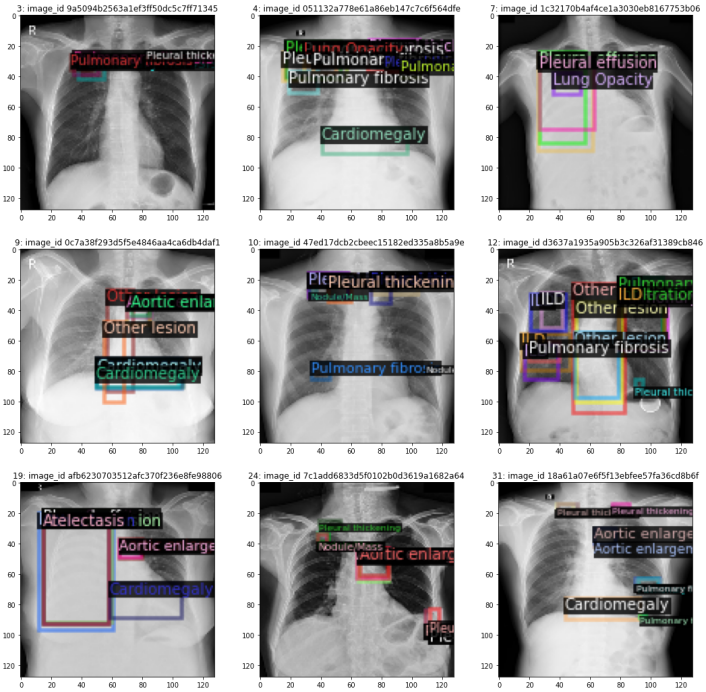
```
flags_dict = {
    "debug": False,
    "split_mode": "valid20",
    "iter": 10000,
    "roi_batch_size_per_image": 512,
    "eval_period": 1000,
    "lr_scheduler_name": "WarmupCosineLR",
    "base_lr": 0.001,
    "num_workers": 4,
    "outdir": "results",
    "imgdir_name": "vinbigdata-chest-xray-resized-png-256x256",
    "aug_kwargs": {
        "HorizontalFlip": {"p": 0.5},
        "ShiftScaleRotate": {"scale_limit": 0.15, "rotate_limit":
10, "p": 0.5},
```

```
        "RandomBrightnessContrast": {"p": 0.5}
    }
}
```

Visualizing the Augmented Images by Mapper

Since mapper is used inside the DataLoader, we can check the behavior by constructing DataLoader and visualizing the data processed by the DataLoader. The defined Trainer class has a method build_train_loader. We can construct train_loader purely from cfg (configs), without representing the trainer since it's a class method.

At first, we were using `detectron2.data.transforms` with `MyMapper` class as it provides basic augmentations. We then observed that many augmentations in albumentations can be used, so we implemented `AlbumentationsMapper` to support it.



Prediction and Results

Everything that has been done so far was done in the notebook-train.ipynb file. For prediction, we move on to the notebook-prediction.ipynb file. The setup is pretty much the same as that of the training file. The object detection model that we chose to implement was the R50-FPN model.

The trainer is then customized and the functions `get_vinbigdata_dicts` and `get_vinbigdata_dicts_test` for preparing the training and test datasets respectively.

Output File

As per the requirements, we were asked to have our output returned in the tensor of the format

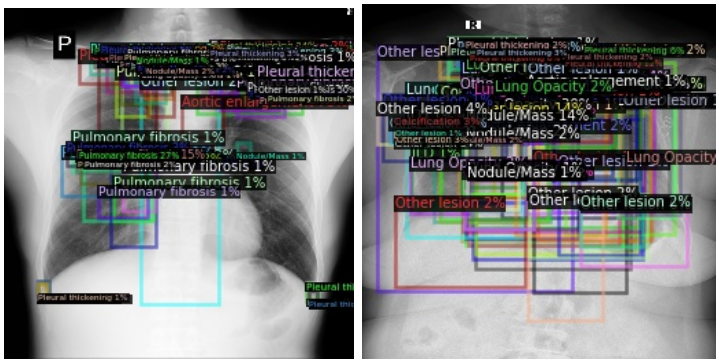
```
[classID y_max x_min y_min x_max confidenceScore]
```

In order to store that, we created a file submission.csv which stored the details of the training dataset. The output was in the tabular format with 3000 rows and 2 columns as shown

	image_id	PredictionString
0	8dec5497ecc246766acfba5a4be4e619	0 0.775484561920166 1010 603 1248 893 13 0.501...
1	287422bed1d9d153387361889619abed	3 0.9862767457962036 666 1289 1865 1820 0 0.78...
2	1d12b94b7acbeadef7d7700b50aa90d4	0 0.8117097616195679 1173 896 1433 1138 3 0.78...
3	6b872791e23742f6c33a08fc24f77365	11 0.3285936415195465 1799 2196 1900 2336 10 0...
4	d0d2addff91ad7beb1d92126ff74d621	0 0.850891649723053 1422 828 1707 1140 3 0.741...
...	...	...
2995	78b44b96b121d6075d7ae27135278e03	0 0.5035402178764343 1036 771 1214 949 11 0.10...
2996	afee8ff90f29b8827d0eb78774d25324	0 0.27024954557418823 1028 714 1243 947 11 0.0...
2997	6e07fab2014be723250f7897ab6e3df2	0 0.990529477596283 1667 801 1972 1131 3 0.977...
2998	690bb572300ef08bbbb7ebf4196099cf	0 0.5590820908546448 1085 689 1337 956 8 0.464...
2999	0a08191a658edb1327e7282045ec71cf	11 0.45100387930870056 565 379 797 479 11 0.16...

3000 rows × 2 columns

Data Visualization



Conclusion

This model has helped build a valuable second opinion for radiologists. An automated system which could accurately identify and localize findings on chest radiographs will relieve the stress the of busy doctors while also providing patients with a more accurate diagnosis.

Possible Extensions

- Selecting the optimal threshold value.
  Because while the NMS value is low, the confidence score would drop so picking the right value of the threshold and the right ensemble weights are necessary. Also, it is (sometimes) difficult to observe stable incremental gains when we switch from a single fold to a multi-fold ensemble.
- It is better to include normal images for training to learn where there is no abnormality. Maybe a model which is trained without any abnormalities.

- Maybe re-try the whole model including "No finding" class during detection training by adding virtual "No finding" boxes, or by adding global classifier together with the detection.
- A real-time application which works on unknown data and is able to predict the output would also be a great additional feature.

References

[1] VinLab: Data Platform for Medical AI (https://vindr.ai/vinlab)

[2] VinDr-CXR: An open dataset of chest X-rays with radiologist's annotations (https://arxiv.org/pdf/2012.15029.pdf)

[3] VinBigData Chest X-ray Resized PNG 256 x 256 dataset by xhulu (https://www.kaggle.com/xhlulu/vinbigdata-chest-xray-resized-png-256x256)

[4] Detectron2 installation instructions (https://github.com/facebookresearch/detectron2/blob/master/INSTALL.md)

[5] Mask R-CNN (https://arxiv.org/abs/1703.06870)

[6] Detectron2 Beginner Tutorial Collab (https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5#scrollTo=QHnVupBBn9eR)

[7] Detectron2 DataLoader and comparing Models (https://detectron2.readthedocs.io/en/latest/tutorials/data_loading.html)

[8] Detectron2 Compare Models Augmentation (https://www.kaggle.com/dhiiyaur/detectron-2-compare-models-augmentation/#data)

[9] COCOdataset detection-eval and COCOdataset keypoints-eval (http://cocodataset.org/#detection-eval) (http://cocodataset.org/#keypoints-eval)

[10] Detectron2 JSON Evaluator (https://github.com/facebookresearch/Detectron/blob/a6a835f5b8208c45d0dce217ce9bbda915f44df7/detectron/datasets/json_dataset_evaluator.py)

[11] Understanding Loss Evaluation Hook and training on Detectron2 with a Validation set (https://ortegatron.medium.com/training-on-detectron2-with-a-validation-set-and-plot-loss-on-it-to-avoid-overfitting-6449418fbf4)

[12] Understanding Albumentations (https://github.com/albumentations-team/albumentations)

[13] Visualizing the data on detectron2 and spatial level transformations (https://github.com/facebookresearch/detectron2/blob/22b70a8078eb09da38d0fefa130d0f537562bebc/tools/visualize_data.py#L79-L88)

[14] Prediction Batches (https://github.com/sphinx-doc/sphinx/issues/4258)