Pneumonia Detection

UPC Artificial intelligence with Deep Learning Postgraduate Course

Students: Inés de Val, Javier Moreno, Pau Dura, Alfred Subietas

Advisor: Kevin McGuinness

Repository structure

Here is a brief description on what can be found in the different files of this repository:

The main repository for this project is called “aidl-medicine” and is based on Git (https://github.com/paudura/aidl-medicine) and the entire solution is built within the Google Cloud Platform (GCP) using 2 services:

1. Google Cloud Storage: Bucket where the user can interact with the solution (input/output).
2. Google VM instance: Where the process runs and where the images are evaluated

The scripts that contain the solution are:

data_input.py: exploratory data analysis and dataset class required to feed the algorithm

FasterRCNNModel.py: implementation of the FasterRCNN ResNet 50 model, training and testing loop

MAIN_PRODUCTION.py: MAIN script for a fake production environment**:** checks if there are new images in the Google Cloud Storage folder and in case there are, it evaluates them and stores the results in the output folder. This is the script that is linked with the crontab to automatically run the process every 2 minutes. Is the one to go script once the model is trained.

auxiliar.py: Auxiliar functions: metric used for evaluation and other complementary code.

config_file.py: File to set up auxiliar parameters like paths and thresholds. The idea of centralizing in a single file all the hardcoded parameters tries to build a cleaner solution.

install_packages.py: Procedure to install the different packages needed to execute the other codes.

classification_results.py: Procedure to run the classification model and extract the classification metrics (precision, recall, accuracy).

General configuration

If the user wants to run the code, there are several steps that has to be followed:

1. Download the required dataset with the Pneumonia images and the training labels:
    a. Go to the link: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data
    b. Download the files:
        i. stage_2_train_images (all the images that are in the folder)
        ii. stage_2_test_images (all the images that are in the folder)
        iii. stage_2_train_labels.csv
    c. Upload them in a Google Storage folder or in a local folder
2. Clone the repository
3. Change the paths in the code in order to link them with your folder architecture
4. Run the FasterRCNNModel.py in order to train the model
5. Run MAIN_PRODUCTION.py in order to evaluate new images

Motivation

All of us are interested in biomedical applications and we wanted to contribute to the detection of one of the most important global health problems that is Pneumonia. Pneumonia accounts for over 15% of all deaths of children under 5 years old internationally and an accurate diagnosis to identify this pathology is one of the key issues for correctly treating it.

Pneumonia is an infectious disease which causes the lungs to fill with pus or other liquids reducing the ability to hold air. In the event of a possible case of pneumonia, the physician performs a chest X-ray in order to establish a more accurate diagnosis. When interpreting an X-ray by a professional, factors such fatigue, subjectivity can have an impact in detecting pneumonia. Thus, there is a need to support these professionals with CAD systems (Computer-Aided Diagnosis) in order to minimize human errors.

Purpose of our project

The main objective of this work is to build an implementation based on Deep Learning that allows to determine whether pneumonia exists in a given chest radiograph and to detect the location of this anomaly. As locations, our implementation will predict bounding boxes around the areas that indicate presence of pneumonia. We intend to create an infrastructure that will allow health professionals to upload chest radiographs in the model and they will receive the diagnostic results.

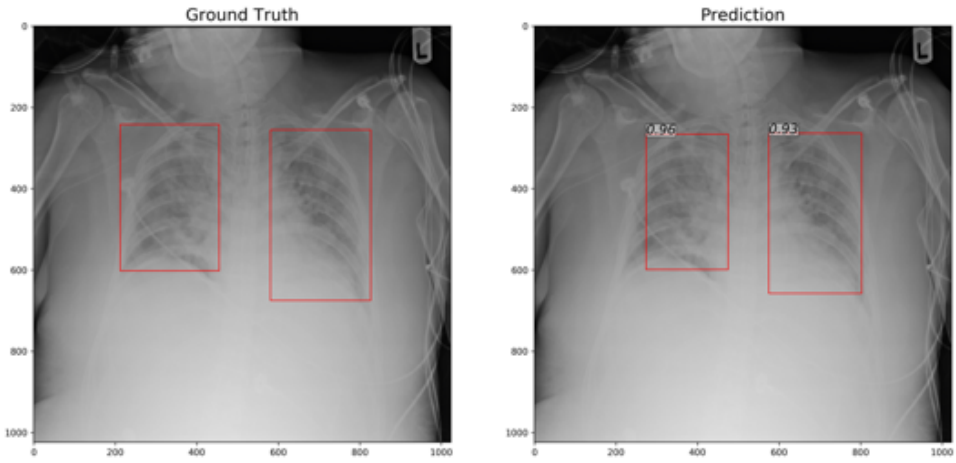

Figure 1. Example of what we want to achieve

About the Radiological Society of North America Data Challenge

The Radiological Society of North America (RSNA®) has posted with the help of Kaggle, the US National Institute of Health, The Society of Thoracic Radiology and MD.ai a dataset for this Pneumonia Detection challenge. This dataset can be accessed on this link. This is the largest publicly available dataset and is the results of work during 6 months of 18 radiologists with an average experience of 10.6 years.

Input dataset

The input dataset consists of two excel files "stage_2_train_labels.csv", "stage_2_detailed_class_info.csv" and two folders "stage_2_train_images" and "stage_2_test_images". The train and test sets consist of 26.684 and 3.000 images respectively, all of them representing chest X-rays in DICOM format. All the images are 8-bit (256 grayscales) with dimensions 1024 x 1024.

Normally in datasets like ImageNet, images have the jpg format or similar. However in this medical task, the ***dcm*** format is used. A DCM file is an image file saved in the Digital Imaging and Communications in Medicine (DICOM) image format. It stores a medical image, such as a CT scan or ultrasound, and may also include patient information to pair the image with the patient; it is commonly used in medical imaging. The format has become a standard for storing, transmitting, viewing, processing, and printing medical images, such as MRIs, CT scans, and ultrasound images.

As mentioned above, "stage_2_train_labels.csv" contains the training labels. The file is a table with the following fields:

- PatientId: each patientId corresponds to a unique image. It links the X-ray images and the labels for each of the samples.
- x: the upper-left x coordinate of the bounding box
- y: the upper-left y coordinate of the bounding box
- width: the width of the bounding box
- height: the height of the bounding box
- Target: the binary Target indicating whether this sample has evidence of pneumonia.

The other file “stage_2_detailed_class_info.csv” provides information about the type of positive or negative class for each image. The data field “Class” categorizes the image in:

- Normal: sample without any trait of pneumonia or any other pulmonary pathology.
- Lung opacity: case of pneumonia.
- Not-normal: sample that does not constitute a case of pneumonia, but which reflects any other pathology.

Given the circumstances we will categorize “Lung opacity” as the positive case for the classification algorithm and the other two as the negative case.

Architecture

In this section we will describe the proposed architecture implemented. This architecture has two main blocks:

1. Data preparation: includes the processing of the images and labels with which we feed the model.

data_input.py

The first step before starting to build the model is to analyze and explore the data. In order to link the image with its annotations we have created a function called train_dataframe() that returns a matrix of shape (30227, 8). The columns are ‘patientId’, ‘x’, ‘y’, ‘width’, ‘height’, ‘target’, ‘class’ and ‘path’. Analysing the data frame, we have seen that some patientId were repeated, that means that this patient is diagnosed with pneumonia and has more than one bounding box where the pneumonia is detected.

In order to feed the model with one image for each patientId with all the bounding boxes detected, we have created another function which creates a dictionary for each variable x, y, width, height with a list of the bounding boxes coordinates.

patientId	x	y	width	height	Target	class	path
0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	/home/medicine_project/input_data/stage_2_train_images/0004cfab-14fd-4e49-80ba-63a80b6bddd6
000924cf-0f8d-42bd-9158-1af53881a557	NaN	NaN	NaN	NaN	0	Normal	/home/medicine_project/input_data/stage_2_train_images/000924cf-0f8d-42bd-9158-1af53881a557
000db696-cf54-4385-b10b-6b16fbb3f985	316.0	318.0	170.0	478.0	1	Lung Opacity	/home/medicine_project/input_data/stage_2_train_images/000db696-cf54-4385-b10b-6b16fbb3f985
000fe35a-2649-43d4-b027-e67796d412e0	570.0	282.0	269.0	409.0	1	Lung Opacity	/home/medicine_project/input_data/stage_2_train_images/000fe35a-2649-43d4-b027-e67796d412e0

Table1. Train dataframe

Function test_dataframe() outputs a matrix of shape(30227,2) which links the file path of the image and patientId.

path	patientId
/home/medicine_project/input_data/stage_2_test_images/2317d609-bef9-4488-9b56-591954dc6742	2317d609-bef9-4488-9b56-591954dc6742
/home/medicine_project/input_data/stage_2_test_images/215d4b0a-b546-4eae-ab7a-39b303f7248b	215d4b0a-b546-4eae-ab7a-39b303f7248b
/home/medicine_project/input_data/stage_2_test_images/ 003ec9e3-512e-4f6e-923d-daa9f9f3db9a	003ec9e3-512e-4f6e-923d-daa9f9f3db9a
/home/medicine_project/input_data/stage_2_test_images/279433ff-5347-417b-ab70-51f06e227130	279433ff-5347-417b-ab70-51f06e227130

Table2. Test dataframe

Pneumonia Dataset:

Once we have defined the data frames, we have generated a PneumoniaDataset class which loads the images from the image folder path depending if we are using the dataset for training or testing. In the __getitem__function, we read the image with pydicom library and convert

it to RGB for consistency. Then, we update the list of boxes variable by adding the boxes coordinates (it can have more than one bounding box) and set the variable labels. Finally, we convert the image, boxes and labels to tensors to feed our model.

```
>>> pneumonia_train_dataset[0]

(tensor([[[0.0000, 0.0039, 0.0118,  ..., 0.0000, 0.0000, 0.0000],

        [0.0000, 0.0078, 0.0157,  ..., 0.0000, 0.0000, 0.0000],

        [0.0000, 0.0078, 0.0157,  ..., 0.0000, 0.0000, 0.0000],

       …,

        [0.0000, 0.0745, 0.2549,  ..., 0.0000, 0.0000, 0.0000],

        [0.0078, 0.0431, 0.1922,  ..., 0.0000, 0.0000, 0.0000],

        [0.0157, 0.0000, 0.0275,  ..., 0.0000, 0.0000, 0.0000]]]), {'boxes': tensor([[160., 463., 339., 725.],

       [538., 387., 705., 800.]]), 'labels': tensor([1, 1])})
```

Figure 2. Pneumonia Dataset returns the image, boxes and labels as tensors

Dataset splitting:

This dataset had already defined the splitting of the dataset in training and test sets of images. The training and validation splits are generated randomly. To train the model we have used a set of 6.500 images with Pneumonia. As for validation, we have used 1.000 images from the training set to validate how the model was doing and as for testing, we have used 1.000 images.

Model

As the structure of the network we have used a two-stage detector Faster R-CNN, a Deep Convolutional Neural Network used for object detection. We have selected this architecture because region-based CNN has shown great success in object detection tasks and because it improves on Fast R-CNN by using a Region Proposal Network (RPN).

This network is composed of the following modules:

1. **Feature Extraction Network (FPN):** the first part of the network is a backbone for feature extraction. To do it, we have used a Feature Pyramidal Network with a ResNet50 which generates multiple feature maps layers. FPN is composed of a bottom-up and a top-down pathway:

Bottom-up: is the feed-forward computation of the ResNet, which computes feature maps at several scales. As we go up, the spatial resolution decreases and the semantic value for each layer increases. The output of each convolution module is used in the top-down pathway.

Top-down: enhances the feature maps with lateral connections from the bottom-up pathway. This process is iterated until the finest resolution map is generated.

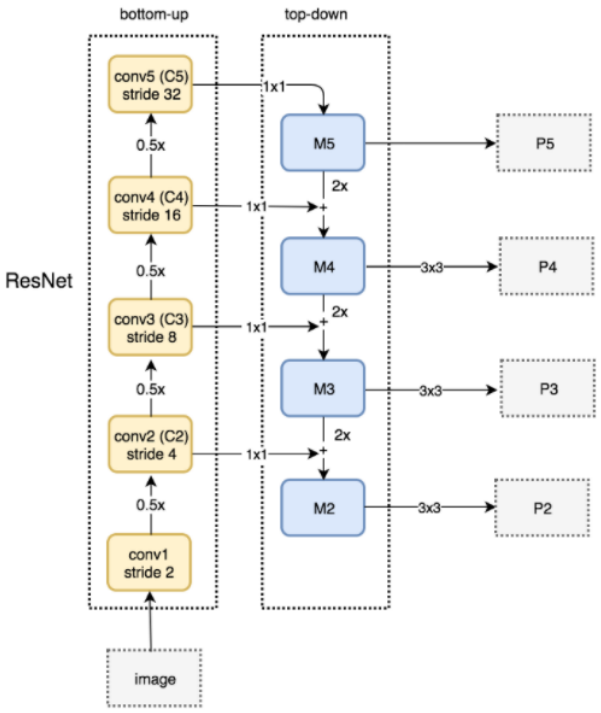

Figure 3. Bottom-up and top-down pathways in ResNet50

2. **Region Proposal Network (RPN):** once the feature maps are extracted, they are fed into the RPN. The purpose of this network is to generate a number of bounding boxes called Regions of Interests (ROIs) that have a high probability of containing an object. The RPN slides a small network over each pixel in the output feature map and checks whether the anchors contain objects.

   Anchors: at each sliding -window location, multiple region proposals are predicted simultaneously, where the number of maximum possible proposals for each location is denoted as *k.* So the regression layer has *4k* outputs encoding the coordinates of *k* boxes and the classification layer has *2k* scores that estimate the probability of an object or not for each proposal.

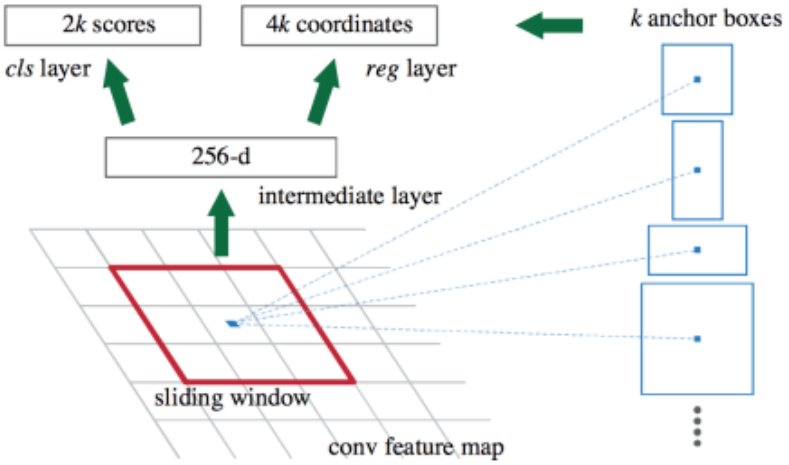

Figure 4. Anchor generator in the Region Proposal Network

3. **Fast R-CNN Detector:** the detector network takes input from the Feature Extraction Network and the Region Proposal Network. It is composed normally of 4 Fully Connected Layers. The last layers are 2 sibling FC layers one for bounding box regression and the other for classification.

   ROI Pooling: given the feature map and the proposals from the RPN, we have to rescale the proposals to the feature map level. We can do this using ROIAlign and then ROI Pooling so that the ROIs have the same size as the FC Layers.

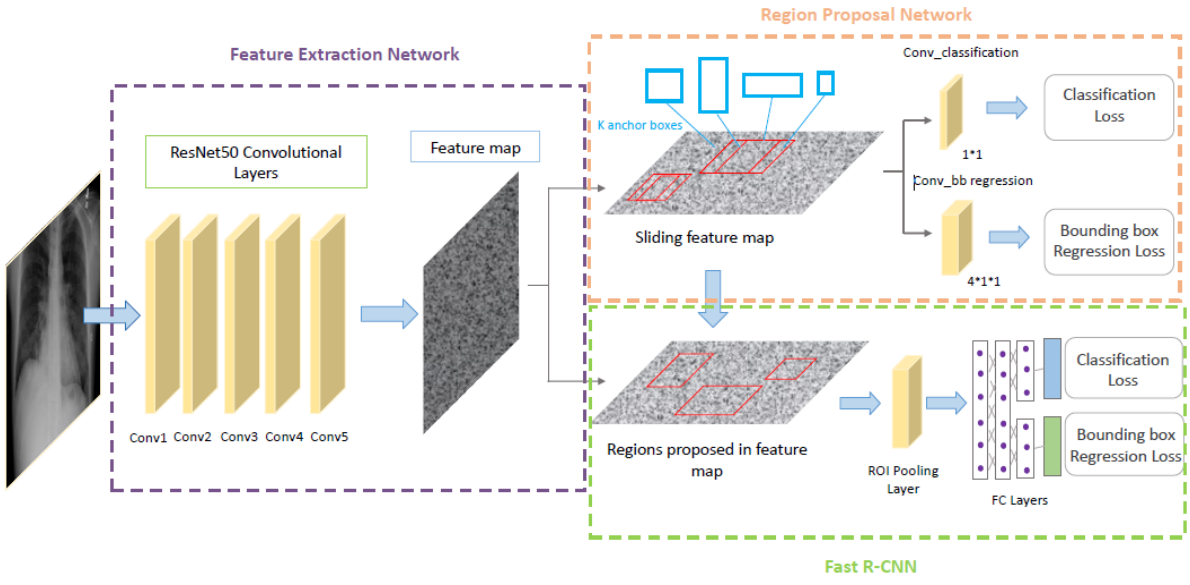

Figure 5. Structure of how the model works

To do transfer learning of this pre-trained model in COCO dataset, first we have loaded it and we have kept the min_size of 1024 pixels to ensure better results during training and testing. We have changed the number of classes to 2 as we have two classes, pneumonia or

background and we have replaced the pre-trained head so that it classifies the images based on our data input features. We have also used Adam Optimizer with a weight decay of 0.0001 to train the new FastRCNNPredictor.

Training and testing

During training, the RPN and Fast R-CNN are trained. The FPN is already pre-trained in ImageNet.

Training RPN:

As mentioned above, the RPN generates a number of bounding boxes called ROIs that have a high probability of containing an object. Some of these proposals highly overlap with each other, so to reduce redundancy non-maximum suppression (NMS) is adopted and applied to the proposal regions based on their scores. If a box overlaps with another box that has a higher score, the box will be removed. To generate the labels for RPN classification, the Intersection-over-Union of all the bounding boxes against all the ground truth boxes are calculated. Then, the IoUs are used to label the ROIs. In addition to classification, the RPN tries to tighten the center and the size of the anchor boxes around the target.

The RPN loss for an image is defined as:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*).$$

Figure 6. Equation of RPN loss for images

i = index of an anchor

pi = predicted probability of anchor i being an object

pi* = ground-truth label

ti = vector that represents the 4 coordinates of the predicted bounding box

ti* = vector that represents the 4 coordinates of the ground-truth box associated with a positive anchor

Lcls = cross-entropy loss

Lreg = smooth L1 loss

Training Fast R-CNN:

Training the Detection Network is similar to that of RPN. Fast R-CNN receives as input the proposals and the feature maps. ROI Pooling is used to crop and resize the feature maps in the locations indicated by the bounding boxes. Then the TwoMLPHead takes the cropped feature maps as input and box_predictor takes the output of box_head and returns the classification logits and box regression deltas. The classification loss in the detection network is also cross-entropy loss and the regression loss Smooth-L1 loss.

During testing, the model only requires the input tensors and returns the post-processed predictions as a List[Dict[Tensor]] one for each input image. The fields of Dict are:

- Boxes: the predicted boxes in [x1, y1, x2, y2] format
- Labels: the predicted labels for each detection
- Scores: the scores of each detection

Experiment

To train our model we have fine-tuned the final layers of the detection network (Fast R-CNN) so that it predicts two classes. We have used Adam Optimizer with a learning rate of 0.001 and a weight decay of 0.0001. The batch size for training is 10 and for validation is 5. As we haven't been able to train the model with GPU, we have only trained it for 1 epoch using CPU. The training loop returns the regression and classification losses of the Region Proposal Network and the regression and classification losses of the Fast R-CNN.

Final results

The results shown in this section are by no means finished since we haven't been able to train the model long enough to either get meaningful loss values or tune hyper-parameters. This is due to the lack of time and computational resources we had available for this project. Also, because of lack of time we could only train one epoch of the transfer learning process (from the ResNet50 to our 2 classes).

Nevertheless, we understand that they reach what was intended in the scope of the course and this final project. The results that will be shown in the following chapters are always based on the test dataset:

Classification Algorithm

In order to find which threshold suits best for our project, we took a look at the different metrics:

1. Precision: out of all the cases that the model is detecting as pneumonia, how many really were.
2. Recall: out of all the cases that were pneumonia, how many our model is able to detect.
3. Accuracy: How many cases is the model labeling correctly (Pneumonia or not Pneumonia)

For the classification algorithm we tested several thresholds which would indicate the algorithm how sure it needs to be to trigger the recognition or not. The thresholds used were 0.1, 0.2 ,0.3, 0.4 and 0.5. Each threshold gave us the following results:

THRESHOLD	PRECISION	RECALL	ACCURACY
0.1	44%	100%	45%
0.2	47%	90%	44%
0.3	49%	80%	40%
0.4	55%	73%	51%
0.5	59%	64%	54%

Based on those metrics, the doctors should take the decision of whether they want to take the risk or not. If they want to play conservative, we would advise a threshold of 0.1: With this threshold we would not lose any case with Pneumonia (False Negatives) but we would be saying in many cases that the patients have Pneumonia when they actually don't.

We will fix this threshold on the hyperparameters and use it for the real-life future predictions.

Detection Algorithm

The detection algorithm draws a square on the radiography pointing the location of the vacants on the lungs. These vacants can indicate the presence of pneumonia. On the training set, each image contains the ground truth box (represented in green) and our prediction box (represented in red).

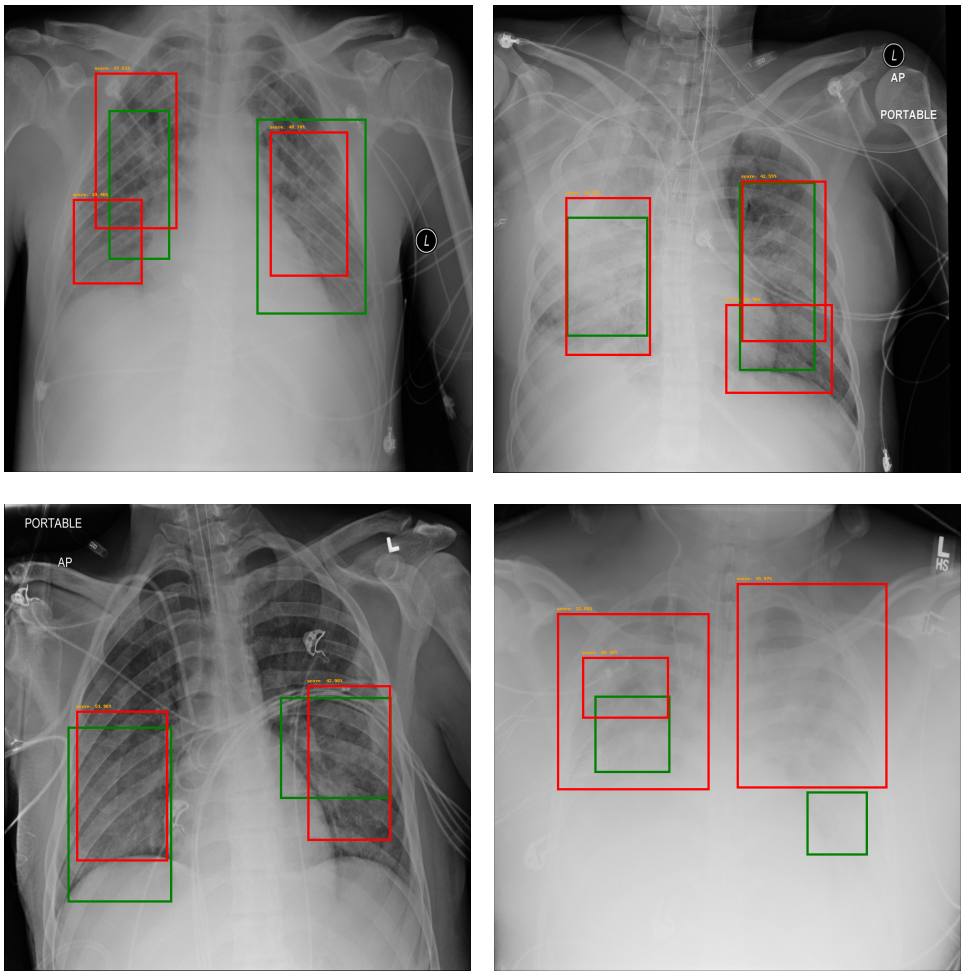

Figure 7. Radiographies with ground-truth boxes (in green) and prediction boxes (in red)

To evaluate the accuracy of our prediction we use the Intersection over Union (IoU) metric for object detection.

$$IoU = \frac{Area\ overlap}{Area\ of\ Union} = \frac{Ground\ truth\ area \cap Prediction\ area}{Ground\ truth\ area \cup Prediction\ area}$$

Figure 8. IoU Equation

The IoU creates a ratio metric that compares the overlap of the ground truth area and our prediction with the union of both areas. This prevents bad predictions in the form of placing a higher score on those predictions that adjust the best to the ground truth without being too big of an area or too small. The IoU metric will be the one to be used to measure and refine the algorithm.

For the detection algorithm we also tested several thresholds and calculated the IoU. The results were as follows:

Threshold	IoU
0.1	37%
0.15	43%
0.2	42%
0.25	39%
0.30	36%
0.35	32%
0.40	29%
0.45	20%
0.50	13%

As shown on the previous table, the threshold that showed the best results was the 0.2 which will be stored on the hyperparameters and used by the users when uploading their own images for recognition.

Challenges we faced

We can split the challenges we faced in 3 main blocks:

1. Images format:

   In the input data there is one Pneumonia zone for each image. If a patient has 4 pneumonia zones, there will be 4 different images. Our process had to merge all the different zones in a single image for each patient in order to train our model properly.

2. Computational resources:

   As we had many images and we wanted to use a complex deep learning architecture, the computational resources were a very important point. We have struggled with GCP and its resources (quotas and GPUs basically). Even though we had requested many quota increases, we were not able to get the required GPUs to run our model fastly. Lately, once we got the GPUs, we struggled with the CUDA drivers version and we could not link the GPU with the Virtual Machine Python3 interpreter.

3. IoU for multiple boxes:

   The lungs radiographies can have more than one pneumonia, so our model had to be able to predict more than one pneumonia. In order to know if the model is fitting well the target with the predicted bounding boxes we used the IoU metric. This metric only take into account one bounding box, so to calculate the total IoU for more than one bounding box, we did it by the following way:

   3.1. We calculate the IoU of every predicted bounding box with every target bounding box, and we get the maximum one (ej. if we had three predicted bounding boxes and two target bounding boxes, we calculate the IoU of the predicted bounding box with the two target bounding boxes, and we get the maximum).
   3.2. Finally we calculate the mean of all bounding boxes. If there are more targets than predicted bounding boxes, we add all IoU of the bounding boxes and we divide by the number of targets bounding boxes. In order to penalize the model if it predicts less bounding boxes. Otherwise we did the mean of the IoUs.

Conclusions

Our aim for this project has been to develop an implementation that allows diagnosing cases of pneumonia within a Chest X-ray, providing a binary classification of presence / absence or pneumonia and identifying it with bounding boxes. Our solution could be an efficient tool that can help doctors improve their diagnosis and pay close attention to those areas that can be indicators of pneumonia.

To achieve this goal a cloud solution was implemented that would automatically load the pre-trained model and would require minimal tweaking on the doctor's side, creating a product that would be easy for doctors and other health professionals not familiar with artificial intelligence to use.

Our implementation is based on the Faster R-CNN model used for object detection and we have fine-tuned the last layers to predict 2 classes. As evaluation metrics we have used IoU for the detection task and accuracy for the classification task, obtaining good results, given that we had trained the model for 1 epoch. This can also be seen in the evaluation images where most of the predicted boxes are quite similar to the ground-truth boxes.

To conclude, despite the difficulties encountered we have been able to build a Deep Learning architecture and execute a model that performs well, but that can be considerably improved with the solutions that we propose in the next section.

Future work

As future work for this project, we plan on working in two different tasks:

1. Keep training the actual model and do hyper-parameter fine tuning to be able to find the neural network architecture that gives the best results.
2. Optimize the data input pipeline and training schedule by using more torch native functions.
3. Increase the number of epochs and check loss functions