Pneunomia Detection with RetinaNet

A deep neural network project

Manuel Ivagnes
Riccardo Bianchini
Valerio Coretti

La Sapienza University of Rome — April 17, 2021

Contents

1 Background

Pneumonia is an inflammatory condition of the lung primarily affecting the small air sacs known as alveoli. Symptoms typically include some combination of productive or dry cough, chest pain, fever and difficulty breathing. The severity of the condition is variable. Pneumonia is usually caused by infection with viruses or bacteria, and less commonly by other microorganisms. Identifying the responsible pathogen can be difficult. Diagnosis is often based on symptoms and physical examination. Chest X-rays, blood tests, and culture of the sputum may help confirm the diagnosis. - wikipedia

Therefore, identifying cases of Pneumonia is tedious and often leads to a disagreement between radiologists. However, computer-aided diagnosis systems showed the potential for improving diagnostic accuracy. In this work, taking inspiration from the reference paper [1], we replicate and build some computational approaches for pneumonia regions detection.

2 Input data

The labelled dataset of the chest X-ray images and patients metadata was publicly provideds by the *US National Institutes of Health Clinical Center* [2]. This database comprises frontal-view X-ray images from 26684 unique patients. Each image was labelled with one of three different classes from the associated radiological reports:

- The **"Normal"** class contained data of healthy patients without any pathologies found (including, but not limited to pneumonia, pneumothorax, atelectasis, etc.).
- The **"Lung Opacity"** class had images with the presence of fuzzy clouds of white in the lungs, associated with pneumonia. The regions of lung opacities were labelled with bounding boxes. Any given patient could have multiple boxes if more than one area with pneumonia was detected. There are different kinds of lung opacities, some are related to pneumonia and some are not.
- The class **"No Lung Opacity / Not Normal"** illustrated data for patients with visible clouds on CXR lung opacity regions, but without diagnosed pneumonia.

2.1 Dataset overview

In practice, our dataset is composed by:

- The 'stage_2_train_images' folder, containing all the images in *dicom* format;
- The 'stage_2_train_labels.csv' file, containing the bounding boxes coordinates and the target for each image[1];
- The 'stage_2_detailed_class_info.csv' file, containing the class for each image;

Both CSV files have 30227 rows, while the number of unique patients is 26684. This results in 3543 duplicates for the 'patientId' attribute, that actually corresponds to the presence of more bounding boxes in the same image.

	patientId	x	y	width	height	Target
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1
5	00436515-870c-4b36-a041-de91049b9ab4	562.0	152.0	256.0	453.0	1
8	00704310-78a8-4b38-8475-49f4573b2dbb	323.0	577.0	160.0	104.0	1
9	00704310-78a8-4b38-8475-49f4573b2dbb	695.0	575.0	162.0	137.0	1

Carrying on the analysis, we can find the distribution of duplicates for each patient. Indeed, in the table below we have that each exam corresponds to a bounding box inside the image.

[1] Each image is identified by a unique patientId

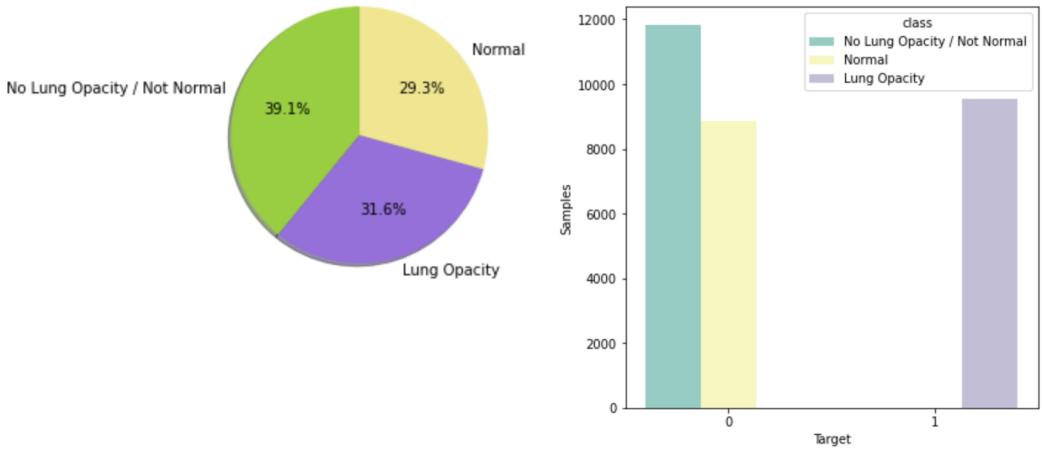
	# Exams	Target	Class	# Entries
0	1	0	No Lung Opacity / Not Normal	11821
1	1	0	Normal	8851
2	1	1	Lung Opacity	2614
3	2	1	Lung Opacity	3266
4	3	1	Lung Opacity	119
5	4	1	Lung Opacity	13

Now, we can analyse the classes distribution and corresponding targets in the dataset. Specifically, target 1 or 0 indicates whether pneumonia is diagnosed or not.

Class	Target	Patients
Lung Opacity	1	9555
No Lung Opacity / Not Normal	0	11821
Normal	0	8851

The graphs below shows the following:

- The graph on the left shows the percentage of samples in each class;
- The graph on the right shows how many samples we have for each target, in the corresponding class;



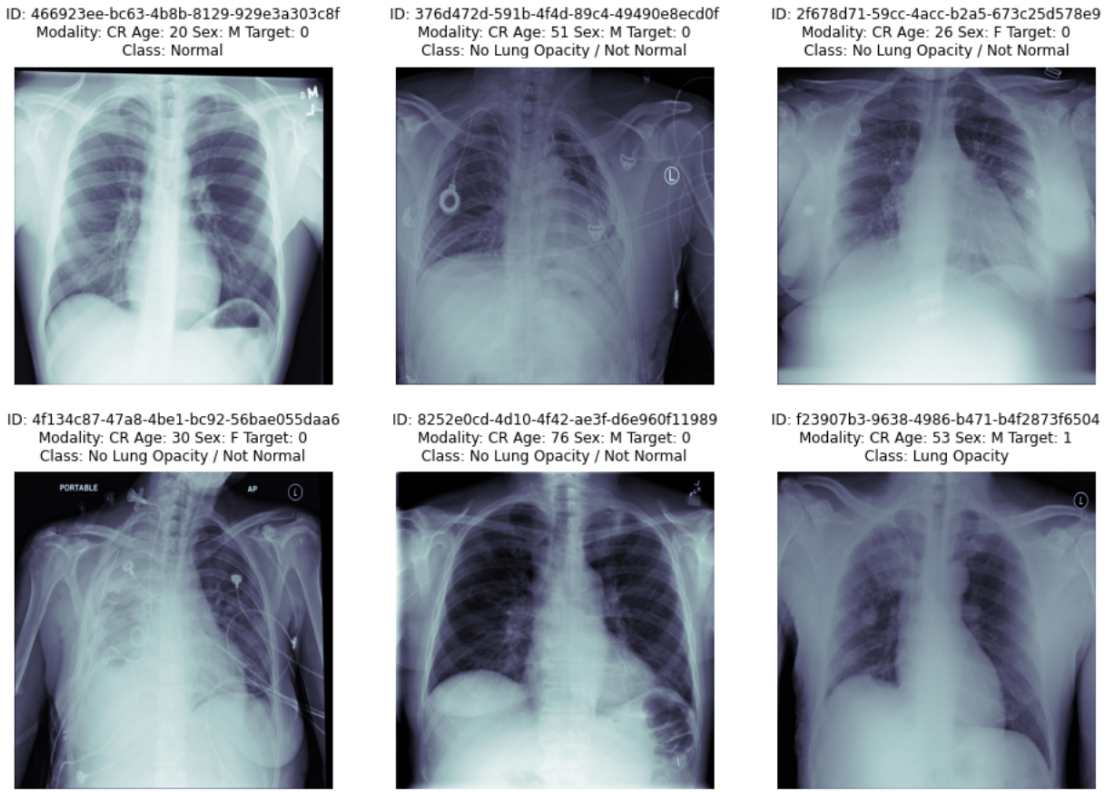
2.2 Sample images

The *Digital Imaging and Communications in Medicine* (DICOM) format is the standard for the communication and management of medical imaging information and related data. Here we can see an example of metadata included with a sample image.

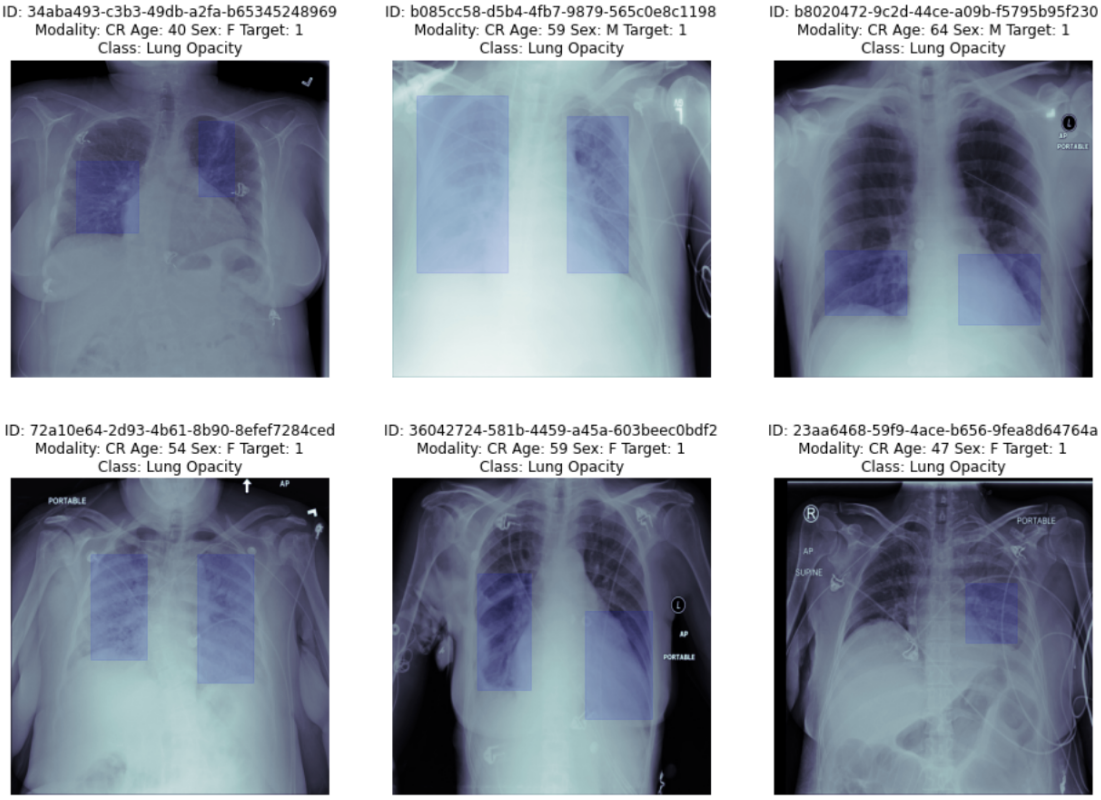
```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 202
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
(0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
(0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
```

```
-------------------------------------------------
(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: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(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-4e49-80ba-63a80b6bddd6'
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-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: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.1517874485.775524
(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: [0.14300000000000002, 0.14300000000000002]
(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 elements
```

At this point, we can visualise some sample images, taken at random from all the different classes, with all the corresponding information.

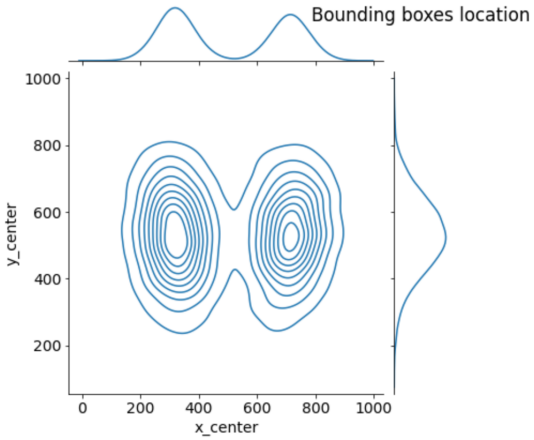


For completeness, we visualise also some samples taken from the 'Lung opacity' class only, with all the corresponding information and the bounding boxes.



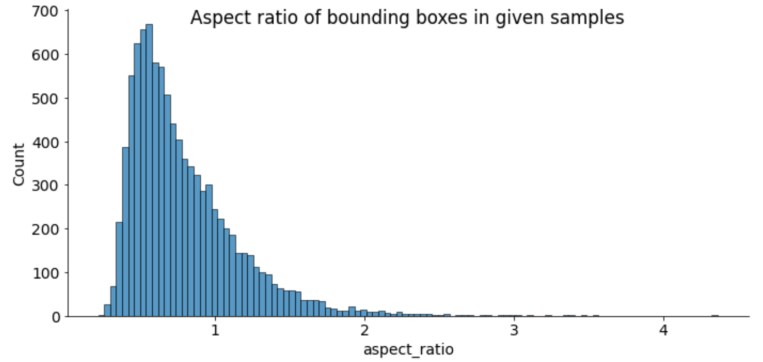
2.3 Bounding boxes analysis

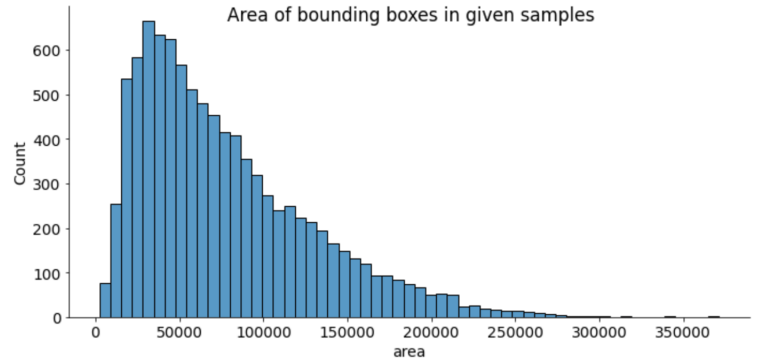
In this project, we are talking about *Object detection*, a computer vision and image processing task that deals with detecting instances of semantic objects of a certain class in digital images and videos. Therefore, it is important to understand the nature of the bounding boxes that defines the position of these semantic objects.

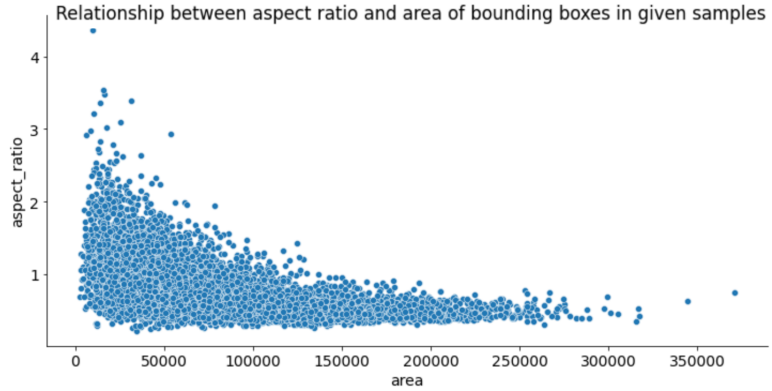


In this first graph, we can visualise a joint plot for the x and y locations in each image. As expected, they are all over the lungs region. However, there is a slightly greater probability to find infected areas on the left lung (note: we are talking about our dataset only).

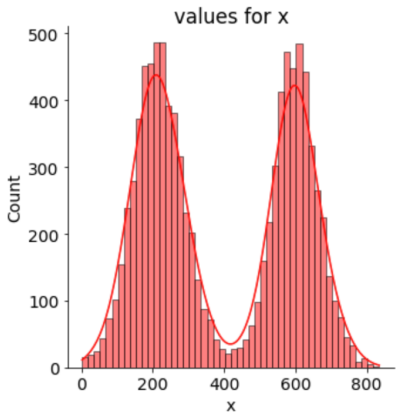
Now, let's plot the aspect ratio (width/height), area and the relationship between them.

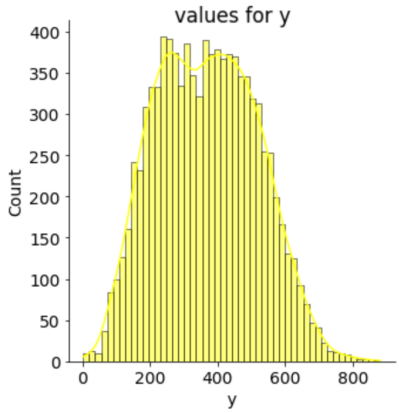


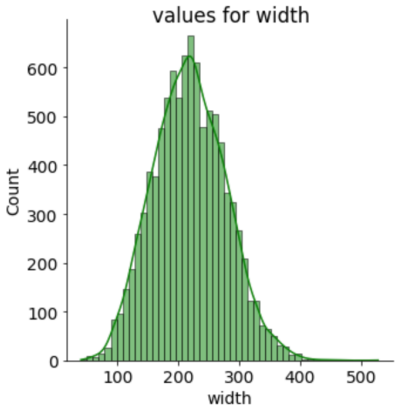


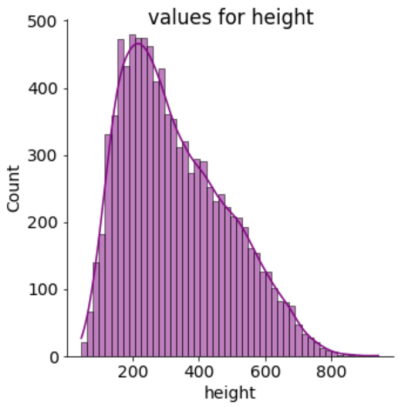


Finally, the density for the size of x, y, width and height.





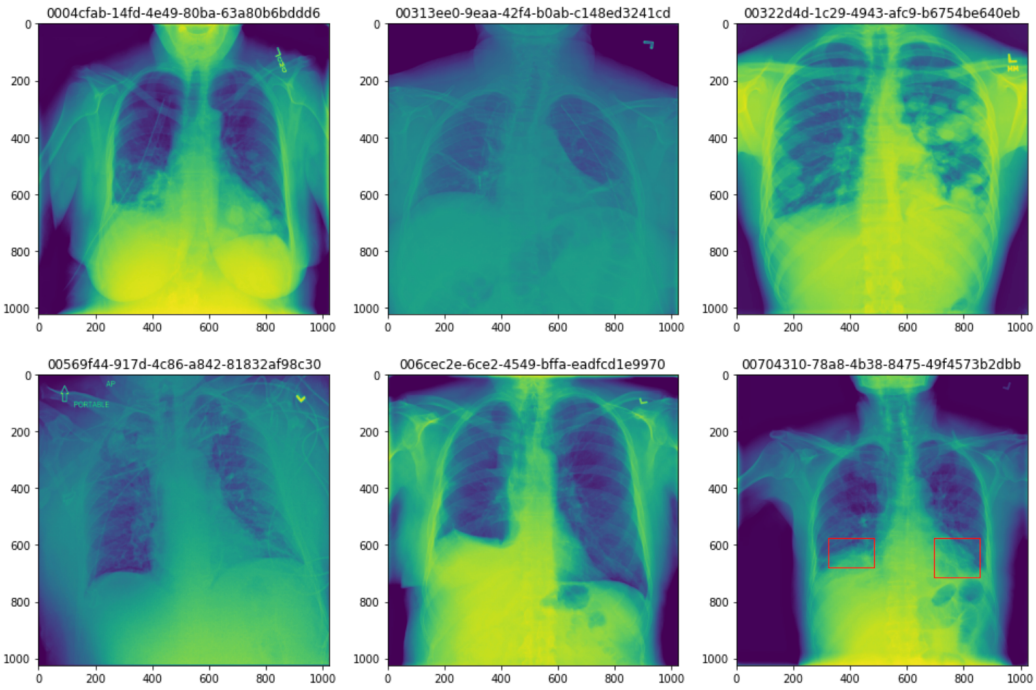




3 Preprocessing & Augmentation

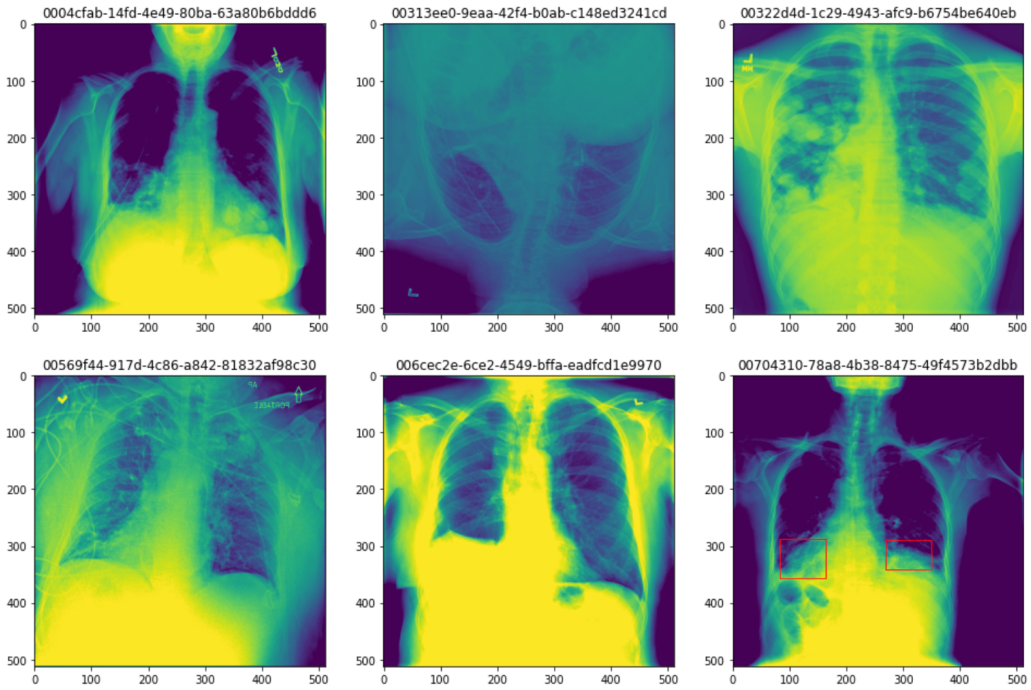
As first step, we perform the *Train / Validation / Test* dataset split, taking around 80% of the samples for the training set, 10% for the validation set, and 10% for the test set[2]. This process is needed to obtain an *unbiased evaluation* for the predictive performance of the model. Indeed, the training set is applied to train, or fit, the model. While the validation set is used for unbiased model evaluation during hyperparameter tuning.

Following the "Torchvision object detection tutorial" [3], we first update the *annotations* data-sheet, in order to obtain the *Pascal VoC* encoding for the bounding boxes, i.e. each bounding box is identified by the top left-hand corner and bottom right-hand corner (xmin-top left, ymin-top left, xmax-bottom right, ymax-bottom right). Accordingly, the custom DataLoader does also provide all the required metadata for each image, that is converted into a pixel array before the augmentations.



[2]Different studies show that deep learning requires a huge amount of data for training, and therefore 80-20 are common values for this operation

In practice, for the augmentations, we use the **imgaug** library [4], that provides an easy implementation for the automatic bounding boxes resizing. From above, we can see sample images before the augmentation. While in the picture below, we can observe the same images after the augmentation.



Specifically, in this example (not used for the training) we applied:

- iaa.Resize(512), → resize to 512x512
- iaa.Fliplr(0.5), → horizontally flip 50% of the images
- iaa.Flipud(0.2), → vertically flip 20% of all images
- iaa.Multiply((0.5, 1.5), per_channel=0.5), → Brightness (50-150% of original)
- iaa.LinearContrast((0.5, 2.0), per_channel=0.5), → Improve or worsen the contrast

Note: From the last picture we can see how the library correctly finds the new position for the bounding boxes on flipped and resized images.

3.1 Training set augmentations

Inspired by the paper, we defined four different types of augmentations, by which the user can decide before running the code. These are defined in the following function:

```
def augmentation_level(level):
  if level == 'resize_only':
    list_augmentations = [
      iaa.Resize(512)
    ]

  elif level == 'light':
    list_augmentations = [
      iaa.Resize(512),
      iaa.Affine(
        scale=1.1,
```

```
        shear=(2.5,2.5),
        rotate=(-5, 5),
      ),
    ]

  elif level == 'heavy': #no rotation included
    list_augmentations = [
      iaa.Resize(512),
      iaa.Affine(
        scale=1.15,
        shear=(4.0,4.0),
      ),
      iaa.Fliplr(0.2), # horizontally flip 20% of the images
      iaa.Sometimes(0.1, iaa.CoarseSaltAndPepper(p=(0.01, 0.01), size_percent
    =(0.1, 0.2))),
      iaa.Sometimes(0.5, iaa.GaussianBlur(sigma=(0.0, 2.0))),
      iaa.Sometimes(0.5, iaa.AdditiveGaussianNoise(scale=(0, 0.04 * 255))),

    ]

  elif level == 'heavy_with_rotations':
    list_augmentations = [
    iaa.Resize(512),
      iaa.Affine(
        scale=1.15,
        shear=(4.0,4.0),
        rotate=(-6, 6),
      ),
      iaa.Fliplr(0.2), # horizontally flip 20% of the images
      iaa.Sometimes(0.1, iaa.CoarseSaltAndPepper(p=(0.01, 0.01), size_percent
    =(0.1, 0.2))),
      iaa.Sometimes(0.5, iaa.GaussianBlur(sigma=(0.0, 2.0))),
      iaa.Sometimes(0.5, iaa.AdditiveGaussianNoise(scale=(0, 0.04 * 255))),
    ]

  return list_augmentations
```

The corresponding descriptions on the paper are:

1. No augmentations: after resizing and normalisation, no changes were applied to the images
2. Light augmentations: affine and perspective changes (scale=0.1, shear=2.5), and rotations (angle=5.0)
3. Heavy augmentations: random horizontal flips, affine and perspective changes (scale=0.15, shear=4.0), rotations (angle=6.0), occasional Gaussian noise, Gaussian blur, and additive noise
4. Heavy augmentations without rotation: heavy augmentations described above without rotations

Note: we exchanged the name for the versions with rotations.

3.2 Validation/test sets and some extra info

On the validation and test sets, we applied the resize process only, as standard procedure.

All the operations referring to the split phase can be found inside the **merge_and_split_dataset.py** script, that saves the new labels inside a 'tmp' folder. The example augmentations, with the corresponding pictures, can be found inside the **Augmentation.ipynb** notebook.

4 The model

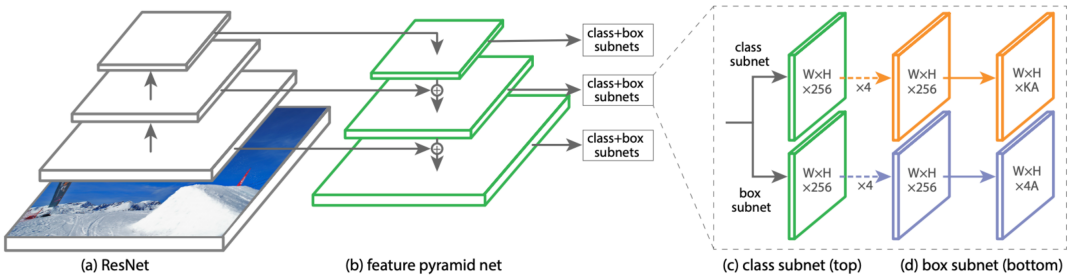
The architectures for networks tasked with object detection is usually split in two categories, namely single-stage (or one-stage) and two-stage object detectors.

In two-stage detectors such as R-CNN, a region proposal network (RPN) is used to generate ROIs in the first stage. Subsequently, these ROI proposals are transferred down the pipeline for object classification and bounding-box regression in the second stage. These two-stage models are very slow; however, they yield a high accuracy because they maintain a manageable balance between the foreground and the background.

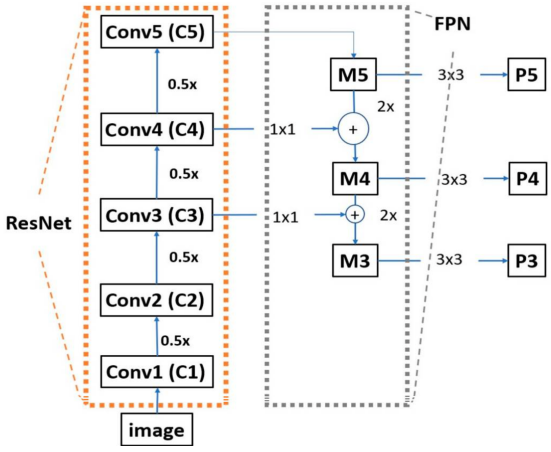
On the other hand, one-stage detectors such as You Only Look Once (YOLO) and single shot multibox detector (SSD) do not have a pre-selection step for detection of foreground candidates and they treat object detection as a simple regression problem. These one-stage methods normally use 10,000 100,000 box proposals per image, compared to only 2000 proposals generated by two-stage methods like Faster R-CNN. Therefore, they often yield a lower detection accuracy; however, they are faster than two-stage object detectors.

4.1 RetinaNet Overview

In our case, the model is based on the well known **RetinaNet** architecture [5], that is a one-stage object detector, composed of a backbone network and two task-specific subnetworks. The backbone is responsible for computing a convolutional feature map over an entire input image and is an off-the-self convolutional network (a-b). The first subnet performs convolutional object classification on the backbone's output (c); the second subnet performs convolutional bounding box regression (d). The overall architecture is shown in the picture.



The **Feature Pyramid Network** (FPN) in the backbone augments a standard convolutional network with a top-down pathway and lateral connections so the network efficiently constructs a rich, multi-scale feature pyramid from a single resolution input image. Indeed, each level of the pyramid can be used for detecting objects at a different scale.



The picture shows an example architecture of RetinaNet with a revised ResNet encoder (block in orange box) and FPN (block in gray box). M3-5 means the feature maps obtained from Conv3-5, respectively, whereas P3-5 shows the feature maps for prediction.

At this point, the P3-5 outputs are passed to the classification and regression subnets, that accomplish the following tasks:

- **Classification subnet:** It predicts the probability of object presence at each spatial position for each of the A anchors and K object classes.
- **Box Regression Subnet:** Outputs the object location with respect to anchor box if an object exists.

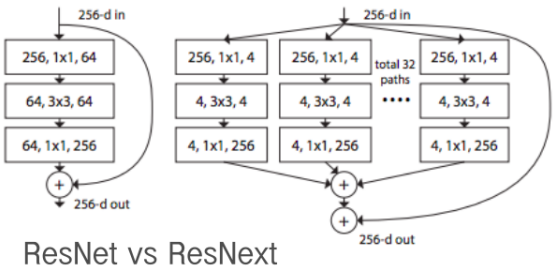
Moreover, our model adds to the above mentioned subnets a third **'global classification subnet'**, that is used in order to predict the class for the overall image, i.e. it predicts wether the image is 'Normal', 'Lung Opacity' or 'No Lung Opacity / Not Normal'.

Finally, the most important feature introduced by RetinaNet is **Focal Loss**, that is an enhancement over Cross-Entropy Loss. It is introduced to handle the **class imbalance problem** with single-stage object detection models. Indeed, single-stage models suffer from an extreme foreground-background class imbalance problem due to dense sampling of anchor boxes. In RetinaNet, at each pyramid layer there can be thousands of anchor boxes. However, only a few will be assigned to a ground-truth object while the vast majority will be background class. These easy examples (detections with high probabilities) although resulting in small loss values can collectively overwhelm the model. Therefore, Focal Loss reduces the loss contribution from easy examples and increases the importance of correcting misclassified examples. This allows RetinaNet to outperform even complex two-stage detectors.

4.2 Our encoders

The standard RetinaNet architecture uses a ResNet encoder, in combination with the FPN, for the backbone. However, we can apply other networks in the backbone to obtain different results. Therefore, we tried three different encoders, that are:

- **ResNet50:** short for *Residual Networks*, it is a classic neural network used as a backbone for many computer vision tasks. It is based on the idea of skip connections, or shortcuts to jump over some layers. In this case, we are using the 50 layers version. [6]
- **Pnasnet5:** short for *Progressive Neural Architecture Search*, that is based on the idea of Neural architecture search, which is a technique for automating the design of artificial neural networks. It uses a sequential model-based optimization (SMBO) strategy, that searches for structures in order of increasing complexity, while simultaneously learning a surrogate model to guide the search through structure space. [7]
- **Se_resnext50:** inherited from ResNet, VGG, and Inception, the basic ResNeXt includes shortcuts from the previous block to next block, stacking layers and adapting split-transform-merge strategy. Moreover, in this version we have the *Squeeze-and-Excitation* blocks, that adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels. [8]



ResNet vs ResNext

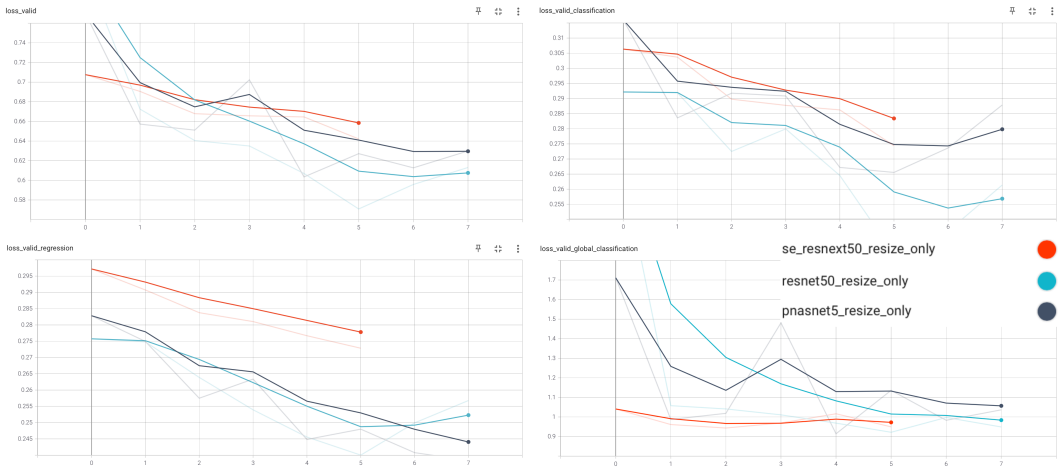
5 Experiments & Results

Given that we have a very limited computing power, we trained our models on *Paperspace gradient* [9], using the basic developer account with the P5000 GPUs. The service's cost may increase a lot, and therefore, we limited our tests to a reduced amount of samples and maximum eight epochs for each test.

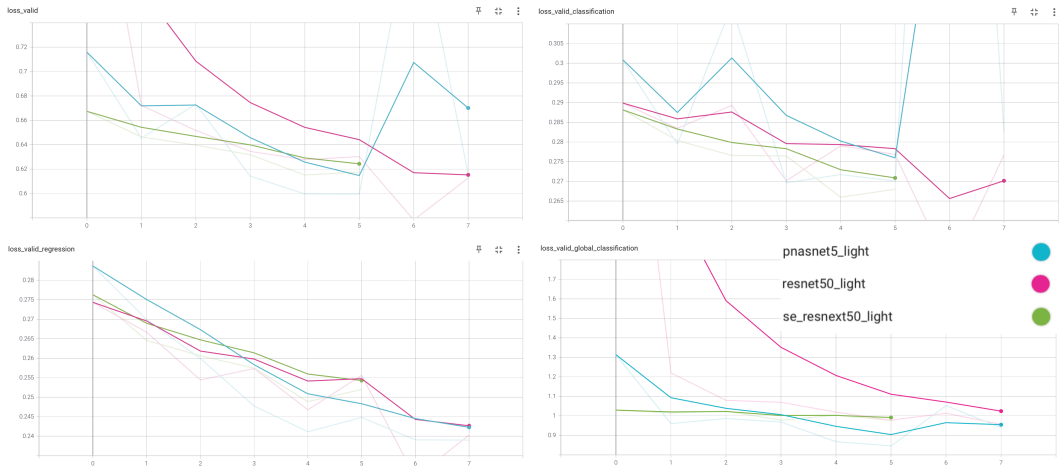
We first tried any combination between the available encoders and augmentations, using a sample of about 3000 images[3], leading us with $3 * 4 = 12$ tests. Then, to prove that Deep Learning models require a massive amount of data for training, we trained our best model (SE_resnext50 encoder), doubling the number of samples to about 6000 images.

First, we show the graphs comparing the performances of each encoder with a given augmentation, regarding the case of 3000 samples. The graphs refers to the validation and they are in the following order: *running_loss*, *classification_loss*, *regression_loss*, *global_loss*.

resize_only

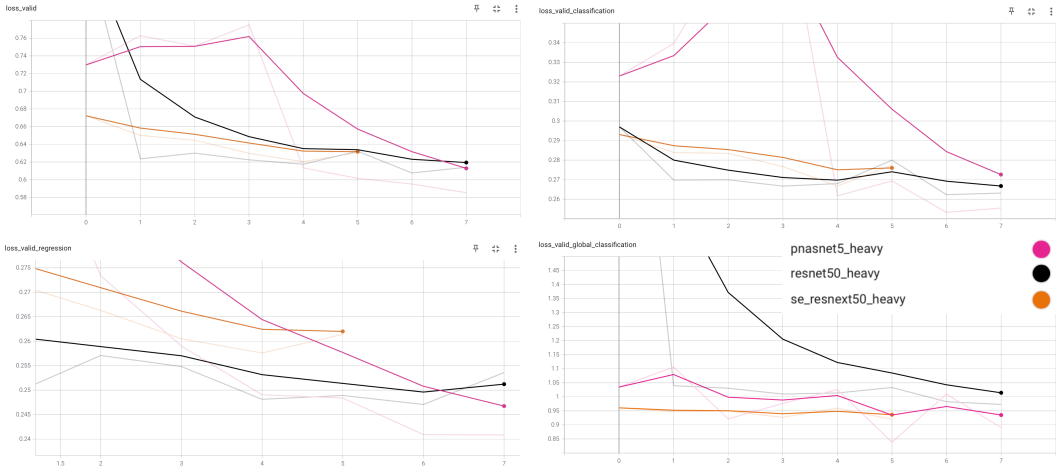


light

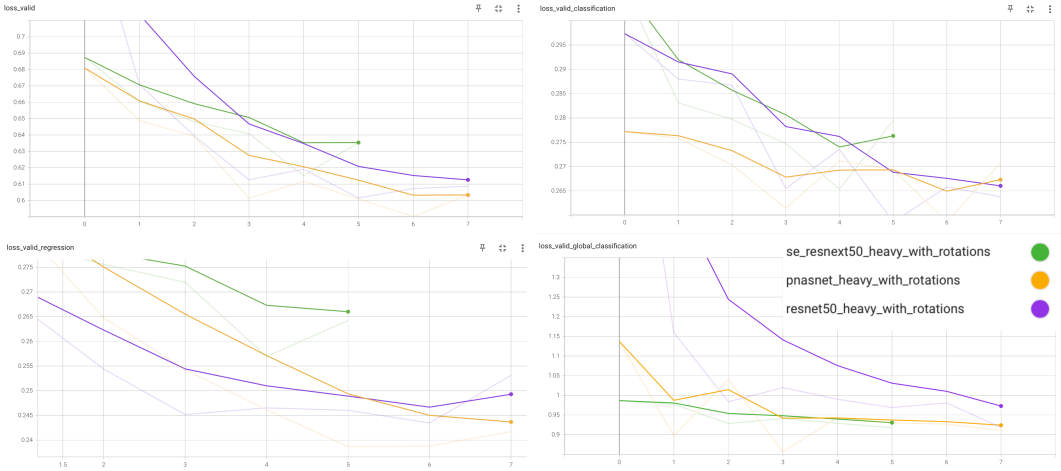


[3]We are sampling the CSV without considering that an image might have more bounding boxes. This procedure introduces an error, but it is not relevant for the purpose of this project.

heavy

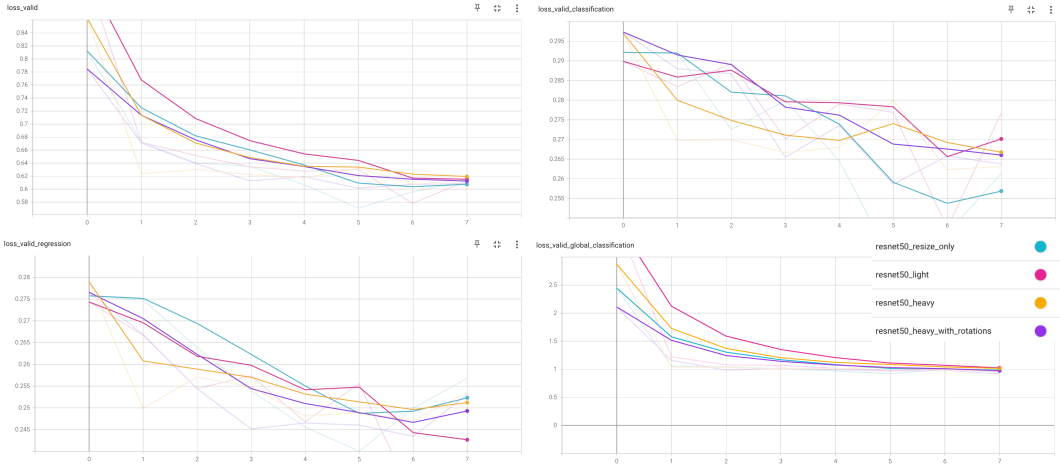


heavy_with_rotations

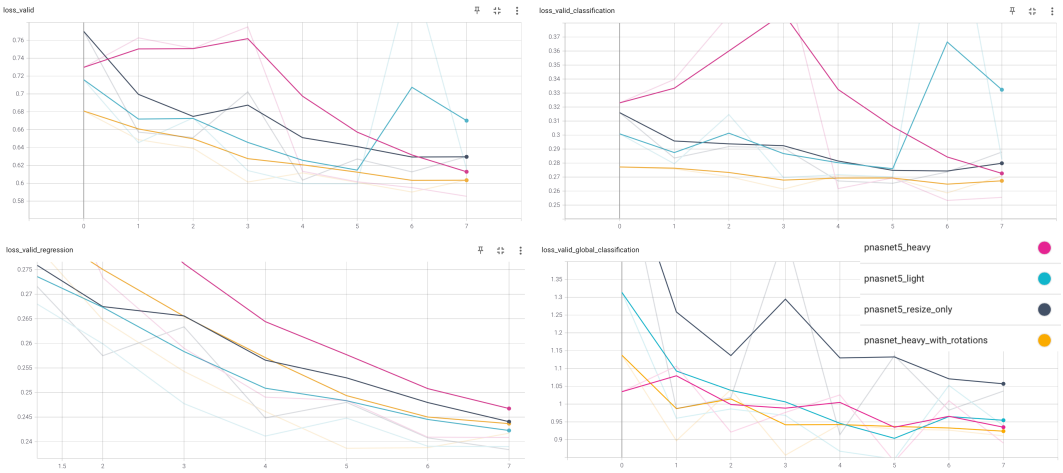


Now, we show the graphs comparing the performances of each augmentation with a given encoder, regarding the case of 3000 samples.

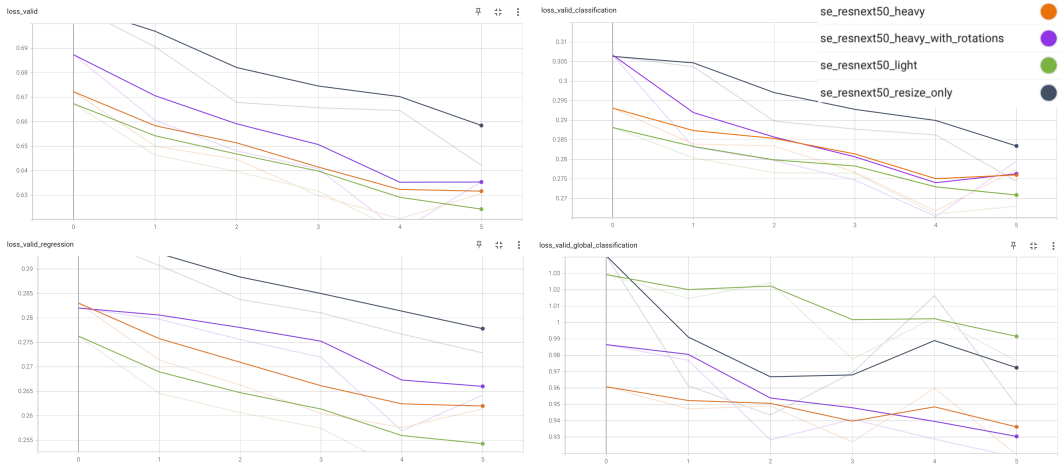
Resnet



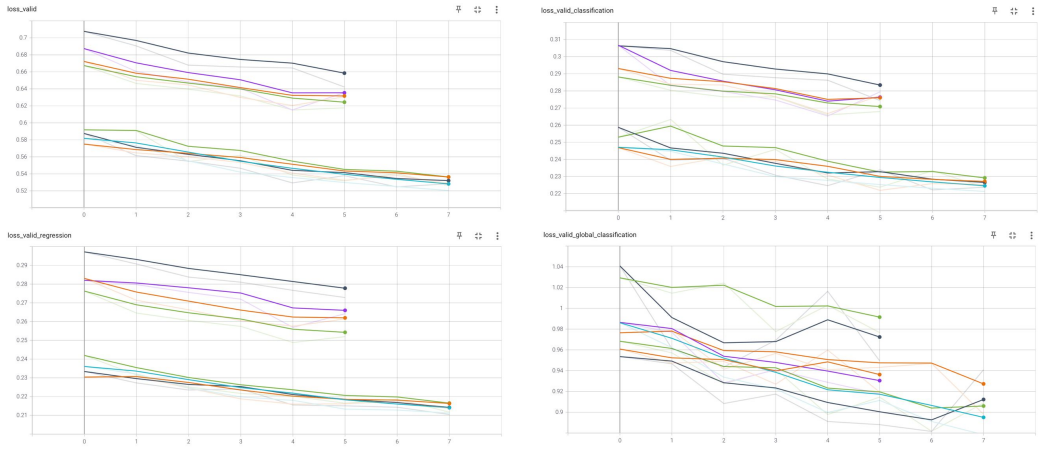
PnasNet



Se_resnext



Finally, we show the graphs comparing the performances of the various SE_resnext with 3000 samples and with 6000 samples. The five epochs lines corresponds to the 3000 samples, while the eight epochs lines to the 6000 samples.



In conclusion, with small samples, the augmentations do not really provide relevant differences between the tests. A different encoder may provide improvements, but the real difference is in the number of samples we provide to the network.

References

[1] Gabruseva, Tatiana and Poplavskiy, Dmytro and Kalinin, Alexandr A.. Deep Learning for Automatic Pneumonia Detection, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. June, 2020

[2] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R.M. Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In IEEE CVPR, 2017.

[3] https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html

[4] https://imgaug.readthedocs.io/en/latest/

[5] Lin T.Y., Goyal P., Girshick R., He K., and Dollr P. Focal loss for dense object detection. IEEE International Conference on Computer Vision, page 29993007, 2017.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[7] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search, 2017.

[8] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[9] https://www.paperspace.com/