REPORT

AMIA Kaggle Challenge 2024 Project Report by Maximilian Otto

Maximilian Otto[*†], Florian Herzler[†] and Maike Herkenrath[†]

[*]Correspondence:
max.otto@fu-berlin.de
Department of Mathematics and Computer Science, Free University of Berlin, Takustraße 9, 14195 Berlin, Germany
Full list of author information is available at the end of the article
[†]Equal contributor

Abstract

Goal of the project: Disease prediction of up to 14 classes per image using bounding box regression on chest x-ray images from the Kaggle challenge "AMIA".

Methods used in the project: Explorative data analysis, K-Means, Faster-R-CNN, RetinaNet.

Main results of the project: We achieved a mAP of 0.24 within relatively short training time (2.5h) using Faster-R-CNN.

Possible improvements: Fine tuning of hyperparameters, grid-search, warm-up epochs, class-balancing dataloader or loss-adjustment by class-weight or using focal loss in Faster-R-CNN, further image curation

Keywords: Classifiers; Deep Learning; Object Detection; RetinaNet; Chest; X-Ray; Semester Project; Faster R-CNN; ResNet50

Introduction

In this semester project, we tried to solve the AMIA Kaggle challenge [1]. The challenge is set up to detect thoracic abnormalities from chest radiographs. Since these abnormalities lie within subtle relations of pixels of each image, they can be considered as unstructured data, that need to be extracted. For this task, machine learning and deep learning methods are commonly used to train a model on the images and their ground truth annotations in a supervised fashion to then automatically detect abnormalities on new, unseen images.
Our approach began with an exploratory data analysis to identify key features and gain an initial understanding of the dataset. This step was crucial for assessing the data before proceeding with the preprocessing of images and bounding boxes. The task required a specific result format, including the bounding box coordinates of each detected object and the corresponding confidence score. Consequently, the challenge encompassed two primary aspects of contemporary machine learning:

- Multi-class prediction of images
- Object detection

To address both aspects, various approaches can be considered depending on the dataset and available hardware. These might include locating abnormalities first and then classifying them, or predicting all classes present in an image followed by regressing their locations using methods such as sliding windows, masks, or even explainability techniques.

We focused on solving the challenge by employing a single model, Faster R-CNN [2], which encodes the image, passes it through a region proposal network, and then classifies each detected bounding box.
We aimed at solving the challenge by using one model, Faster R-CNN [2], which encodes the image, passes it to a region proposal network and then classifies each found bounding box.

Data

Data sets used

The AMIA Kaggle challenge uses a subset of the original VinDr-CXR data set [3], which released 18.000 chest X-ray images with annotations of radiologists. The images were provided with a resolution of 1024x1024 in greyscale PNG format, already down scaled from their original resolution. The subset is based on a dataset consisting of a total of 15,000 scans that have been annotated and checked by up to 17 radiologists. From those, 8,573 images can be used for training and 6,427 images for testing. The train CSV contains 45,925 rows and the test CSV 21,989, where each bounding box is stored in a separate row.
Each bounding box contains one of the fourteen abnormalities listed below, which range from larger findings as aortic enlargements and cardiomegaly to very small findings like nodules and lesions:

0	Aortic enlargement	5	ILD	10	Pleural effusion
1	Atelectasis	6	Infiltration	11	Pleural thickening
2	Calcification	7	Lung Opacity	12	Pneumothorax
3	Cardiomegaly	8	Nodule/Mass	13	Pulmonary fibrosis
4	Consolidation	9	Other lesion	14	No finding

Explorative data analysis

We then checked the distribution of the findings among all images of the train set and the frequency of annotations performed per radiologist-ID. The data classes are imbalanced, as depicted in figure 1, which can lead to major problems when it comes to object detection.
Each image may contain multiple boxes of any class label, assigned by any radiologist independently and therefore overlap, or has no findings at all and is therefore assigned as class 14. IN the latter case, it seems like multiple radiologists confirmed that there is no finding in an image at all. Our Jupyter Notebook *exploration.ipynb* provides interactive histograms for further inspection.
To get an overview of the expected bounding boxes before setting up an abject detection model, we calculated the bounding box dimensions. These can be seen in figure 2. The wider horizontal boxes are mostly coming from enlarged hearts and cardiomegaly, as depicted in figure 4.

Preprocessing

Before we can use the raw images and the CSV files containing the box coordinates as inputs for neural networks, some preprocessing steps are necessary. First, we

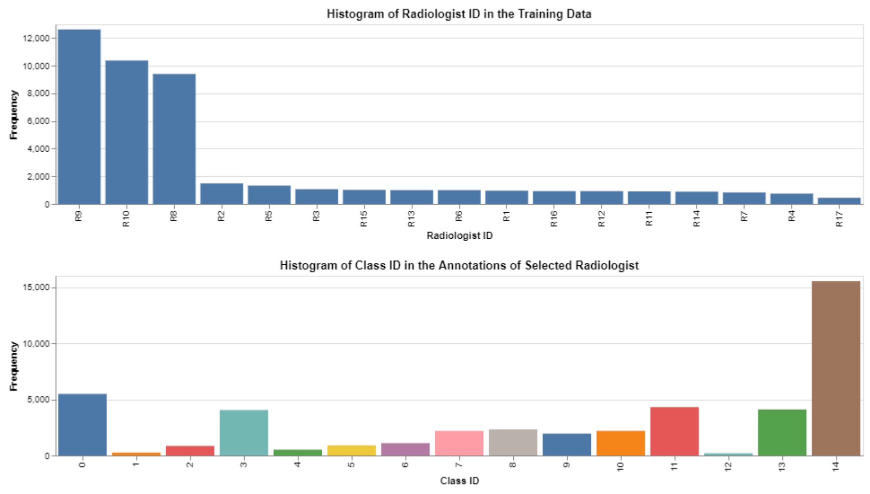

Figure 1 Top: Annotation frequency per radiologist-ID. Bottom: Frequency of class labels assigned.

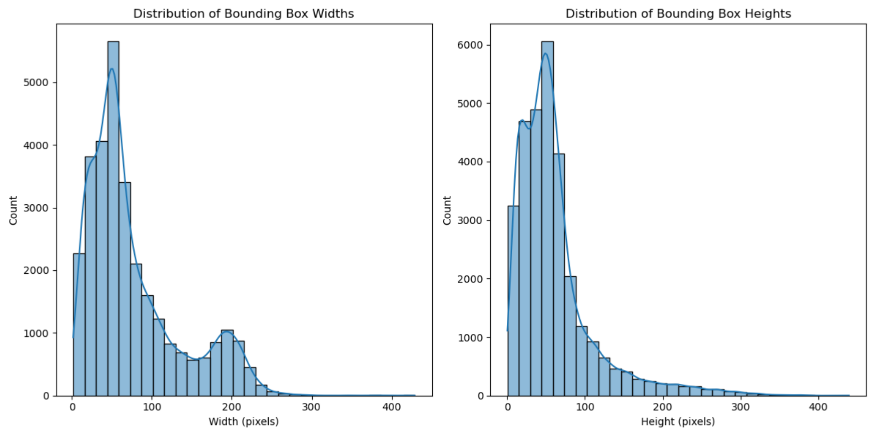

Figure 2 Histograms of the bounding box heights and widths.

reformatted the CSV file containing the annotations into a dictionary, providing a structured and compressed data format, which can be stored directly as JSON file. This step included scaling the box coordinates to $[0, 1]$ by dividing the coordinates by the respective original image resolution (resolutions are publicly available in *img_size.cvs*), because the box coordinates were not already down scaled as the images.

Several data augmentations were added for training for segmentation and classification to further increase the virtual sample size. These augmentations are executed during runtime and are controlled by the set random seed. We associate data augmentations as a part of preprocessing, since it is possible to compute the augmentations once and store them before training [4].

The data augmentations are used to simulate more data to tackle an immediate overtraining and give the model a chance to learn more generalized features. When training the models, our fixed set of augmentations gets applied to each image in-

dividually:
After downsizing the image to $448x448$ pixels, a random rotation of each image within $(-6, 6)$ degrees, a random color jitter (settings: `brightness=0.1, contrast=0.1, saturation=0.0, hue=0.0`), a random view perspective (settings: `distortion_scale=0.1, probability=0.1`) gets applied. We then convert the image to float32 tensors while scaling the pixel values to $[0, 1]$ to match the expected input range of pre-trained networks.
We then normalize each pixel value by the mean and standard deviation of all pixels within the train set. By doing such standardization, the overall values of the whole data set are expected $\mu = 0, \sigma = 1$. It is important to note that this expectation is not matched, since our calculation of μ and σ excludes pixels with a value of zero. This is done because we are not interested in the pitch black background altering the mean of an image and wanted to focus on the X-ray signal only, e.g. a cropped images or smaller person would have a different distribution when including zero pixels. Thanks to *torchvision v2*, the bounding boxes get transformed accordingly.
Before loading each image and its boxes into the augmentations, we automatically perform non-maximum suppression (nms) on boxes of the same class with an intersection over union (IoU) threshold of 0.5. This reduces the amount of overlapping boxes that were annotated by different radiologists. The effect of nms on the boxes can be seen in figure 3. The incent is to simplify training and ensuring the model does not have to predict strongly overlapping boxes of the same class.

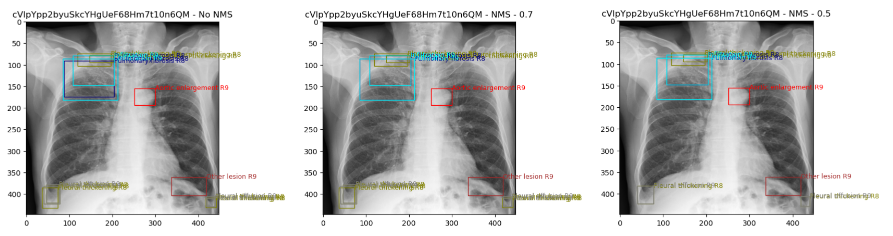
cVlpYpp2byuSkcYHgUeF68Hm7t10n6QM - No NMS
cVlpYpp2byuSkcYHgUeF68Hm7t10n6QM - NMS - 0.7
cVlpYpp2byuSkcYHgUeF68Hm7t10n6QM - NMS - 0.5

Figure 3 Left: Raw boxes with annotations. Middle: Boxes after applying nms with IoU-threshold 0.7, Right: Boxes after applying nms with IoU-threshold 0.5

Methods

In regards to the overall project goal and the given data, we found supervised learning methods to be suited for this task. They require a ground truth data set (annotations) to learn features from, which got preprocessed as described above. Using PyTorch, we were able to modify a Faster R-CNN network [2] provided from PyTorch [5].
We tested their pre-trained model, which was trained on ImageNet [6], but quickly switched to using a pre-trained ResNet-architecture specifically for chest X-ray images from torchxrayvision [7]. To adapt the network and incorporate it into a Faster R-CNN architecture, building a feature pyramid network (fpn) was the first step. We removed its last two layers and passed the following configuration to `BackboneWithFPN`:

```
# use pre-trained on chest x-rays
backbone = xrv.models.ResNet(weights="resnet50-res512-all")
backbone = torch.nn.Sequential(*list(backbone.model.children())[:-2])

# Define the layers to return feature maps from
return_layers = {
    '4': '0',  # Corresponds to layer1
    '5': '1',  # Corresponds to layer2
    '6': '2',  # Corresponds to layer3
    '7': '3',  # Corresponds to layer4
}

# Construct the BackboneWithFPN
backbone_with_fpn = BackboneWithFPN(
    backbone,
    return_layers=return_layers,  # The layers we want to use
    in_channels_list=[256, 512, 1024, 2048],  # Corresponding in_channels for these layers
    out_channels=256  # Out channels for FPN layers
)
roi_align = torchvision.ops.MultiScaleRoIAlign(featmap_names=['0', '1', '2', '3', '4'], output_size=7, sampling_ratio=2)
```

We set the `num_classes` to 15, where the background is 0 and the original class-IDs got increased by one. The model is capable of nms before outputting boxes, so we set that to 0.5 according to our preprocessing. We limited box detections per image to 50 to increase training speed and only allowed boxes with a score of 0.1. By setting $\mu = 0$ and $\sigma = 1$, we ensured the network does not alter the images with its own normalization.
To further improve speed, we set `fixed_size=(448x448)`. Otherwise, the network would set images to $800x1300$ pixels, hence interpolating our small images. Faster R-CNN uses a region proposal network, which tries to find boxes in which an object of interest could be prominent, before classifying the box. To set up initial boxes that fit our data classes and reject larger default boxes, we ran a k-Mean analysis on the bounding box dimension with $k = 6$. This gave us the means of each cluster (s. figure 4) and therefore the height and widths of each cluster. From this we calculated the box ratios by $ratio = height/width$, as PyTorch expects it. By analyzing the `anchor_generator`, we were able to calculate the correct input sizes for these cluster. We increased them a little bit to ensure our initial boxes are a bit larger than the means of the cluster, to include most objects before regressing. Every ratio and base size for the boxes got applied to each of the five fpn layers.

```
anchor_generator = rpn.AnchorGenerator(
sizes=((30,), (60,), (130,), (200,))*5, # times 5 'cause we want all sizes on all layers
aspect_ratios=((0.32, 1., 1.8,),)*5, # times 5 'cause 5 feature maps'
)
```

Ultimately, we chose our hyperparamters (table 1) of smaller batch sizes due to the GPU memory limit of 16GB that are available on Kaggle. We trained our models on 80% of the training set and used 20% as validation set, before we ran the trained models on the test set for the Kaggle-submission. We evaluated the mean average precision according to the challenge evaluation, using PASCAL VOC 2010 mean Average Precision (mAP) at $IoU > 0.4$ on our 20% validation split [8].

As a second network to compare our results to, we chose RetinaNet [9]. This model can produce bounding boxes in just one forward pass and does not need an additional region proposal network, while still relying on a fpn. It implements a focal loss, which could help resolve our class imbalance problem. We set it up using mostly the same hyperparameters and configuration, but modified the anchor generator to include the initial bounding box size of 90 and the standard scaling per feature map.

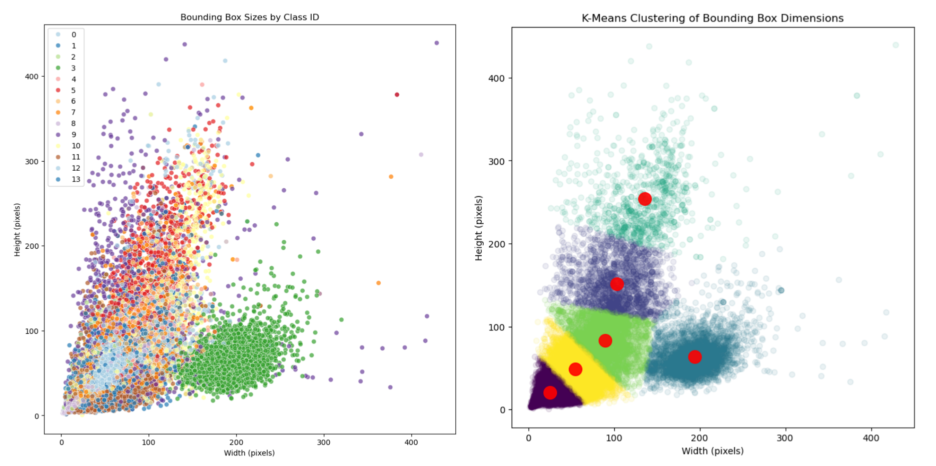

Figure 4 Left: Bounding box dimensions colored by class. Right: Cluster after K-Means with $k = 6$ got applied. The mean of each cluster is marked in red.

Parameter	f*Value*	*Short description*
Number of epoch	30	How many epochs the network runs through
Number of classes	15	Output classes to predict, incl. background
Learning rate	0.0005	Rate with which the weights are modulated
Batch size	10	Images per iteration
Optimizer	ADAMAX with weight decay of 0.00005	Gradient descent method
Learning rate scheduler	Exponential, $\gamma = 0.95$	Adjustment of learning rate after each epoch

Table 1 Main hyperparameters for training our neural networks

To check whether the network learnt features we can understand, we used Eigen-CAM [10], which highlights the important areas of an image based on class activation maps using principle components instead gradients.

Results

After finding a suitable configuration, which we tested for by training for one or two epochs and comparing the mean average precisions and losses, we achieved rank 4 compared to the leader board of this challenge on Kaggle with an mAP of 0.243. This was achieved after 30 epochs of training, as it can be seen in figure 5 with the previously defined parameters.
As expected, the class imbalance problem leads to strongly discrepant accuracies when training a classifier with cross entropy loss (s. figure 6), with cardiomegaly achieving a relatively good score of 0.84.
After training RetinaNet, we observed a more linear rise of mAP without immediate stagnation, indicating that the model could be trained far longer (s. figure 7). The mAP had its peak at 0.168.
The class imbalance problem is still persistent in this scenario, as the resulting discrepancies in the test curves look similar to Faster R-CNN (s. figure 8).

When inspecting the Eigen-CAM of a network (s. figure 9), we can see, that the Faster R-CNN has a focus on certain regions when predicting bounding boxes

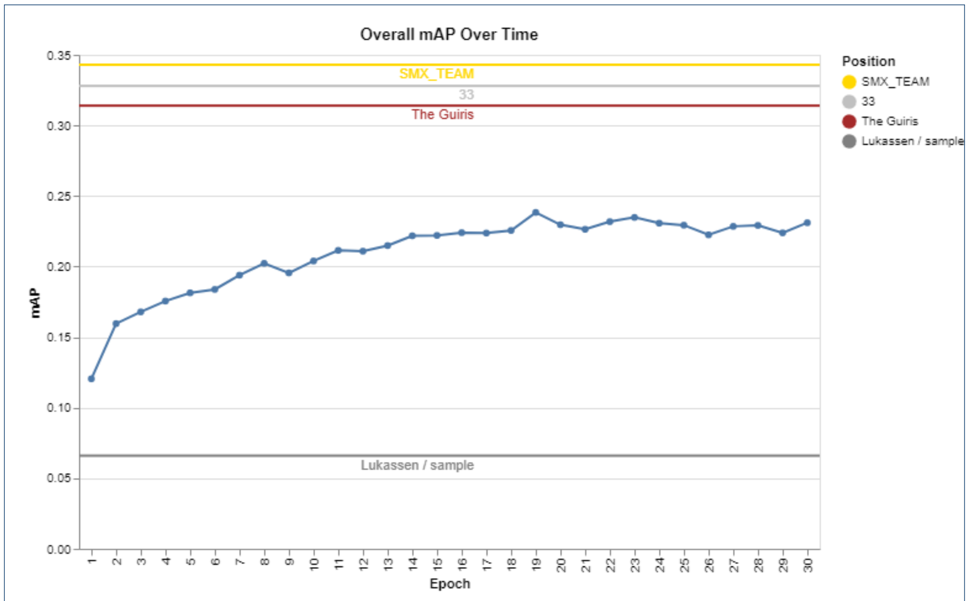

Figure 5 Faster R-CNN results: Blue: Mean Average Precision of our Faster R-CNN, according to PASCAL VOC 2010 at $IoU > 0.4$. Horizontal lines: Kaggle leader board scores with team names

and finds objects of interest in those. It seems like the network started to learn relevant features for this task. Due to insufficient RAM we were unable to produce Eigen-CAM visualizations for the RetinaNet.

Conclusion

Finding a good configuration for a specific task is a challenge in itself, but can be rewarding. With a good anchor box generator, the Faster R-CNN setup is straightforward. The huge class-imbalance problem could be tackled by adjusting the class-loss based on the class-distribution of the bounding boxes, and therefore re-writing the `compute()` function of `FasterRCNN`. We did not really did a hyperparameter search and believe it could be beneficial to find a good configuration by testing for the first two epochs in a grid search of many different hyperparameters.
The RetinaNet is a promising approach with a slightly faster runtime and an in-built loss to counter the class-imbalance, but needs more tinkering and training time from our side to achieve comparable results.
Training our own backbone, a multi-class ResNeXt for example, could lead to a better encoding and would allow us to freeze most of the backbone layers of the two object detection models. Training a pre-trained standard classifier with a good hyperparameter-search could elevate the results.

Note: We chose short training times to be able to test different configurations and would like to continue beating the challenge. To us, Faster R-CNN seems to be a valid choice for this task. Moving to an NVIDIA RTX3090 in the middle of the project and setting the `fixed_size` parameter decreased the runtime by a factor of 4, effectively training and testing each epoch within under six minutes instead 20m

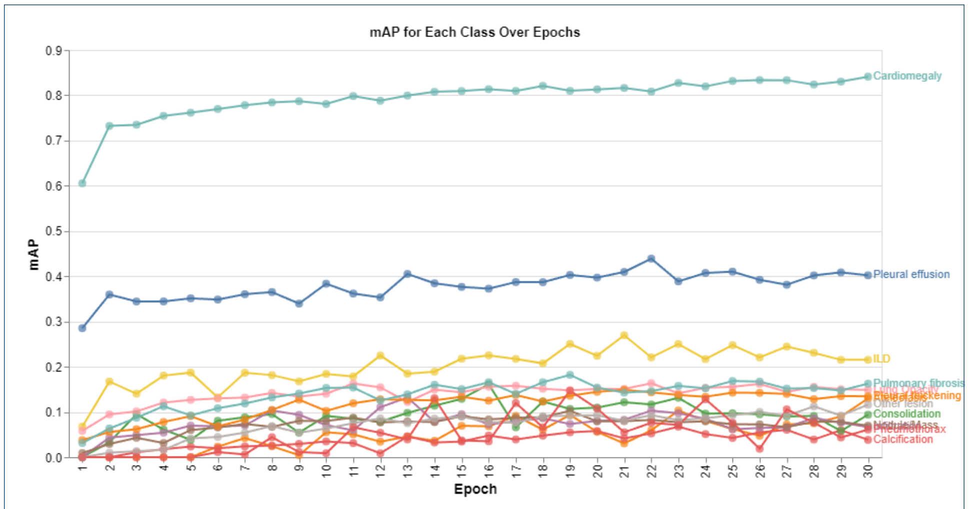

Figure 6 Faster R-CNN results: mAP per class over time.

on Kaggle. It was a very interesting project and I believe we learned a lot about object detection architectures, PyTorch, and the impact of different parameters.

Code availability

The codes are available under https://github.com/scaramir/cv upon request, becasue the repository is currently set to "private".

References

1. Marti, R.: AMIA Public Challenge 2024 AMAI Public Challenge on X-ray Anomaly Detection. https://www.kaggle.com/competitions/amia-public-challenge-2024
2. Gavrilescu, R., Zet, C., Foșalău, C., Skoczylas, M., Cotovanu, D.: Faster r-cnn:an approach to real-time object detection. In: 2018 International Conference and Exposition on Electrical And Power Engineering (EPE), pp. 0165–0168 (2018). doi:10.1109/ICEPE.2018.8559776
3. Nguyen, H.Q., Lam, K., Le, L.T., Pham, H.H., Tran, D.Q., Nguyen, D.B., Le, D.D., Pham, C.M., Tong, H.T.T., Dinh, D.H., Do, C.D., Doan, L.T., Nguyen, C.N., Nguyen, B.T., Nguyen, Q.V., Hoang, A.D., Phan, H.N., Nguyen, A.T., Ho, P.H., Ngo, D.T., Nguyen, N.T., Nguyen, N.T., Dao, M., Vu, V.: Vindr-cxr: An open dataset of chest x-rays with radiologist's annotations. Scientific Data **9**(1) (2022). doi:10.1038/s41597-022-01498-w
4. Spanhol, F.A., Oliveira, L.S., Petitjean, C., Heutte, L.: A Dataset for Breast Cancer Histopathological Image Classification. IEEE Transactions on Biomedical Engineering **63**(7), 1455–1462 (2016). doi:10.1109/TBME.2015.2496264. Accessed 2022-11-18
5. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. CoRR **abs/1912.01703** (2019). 1912.01703
6. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009). doi:10.1109/CVPR.2009.5206848
7. Cohen, J.P., Viviano, J.D., Bertin, P., Morrison, P., Torabian, P., Guarrera, M., Lungren, M.P., Chaudhari, A., Brooks, R., Hashir, M., Bertrand, H.: TorchXRayVision: A library of chest X-ray datasets and models. In: Medical Imaging with Deep Learning (2022). https://github.com/mlmed/torchxrayvision
8. The PASCAL Visual Object Classes Challenge (VOC2010). http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2010/index.html
9. Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **42**(2), 318–327 (2020). doi:10.1109/TPAMI.2018.2858826
10. Bany Muhammad, M., Yeasin, M.: Eigen-cam: Visual explanations for deep convolutional neural networks. SN Computer Science **2**(1) (2021). doi:10.1007/s42979-021-00449-3

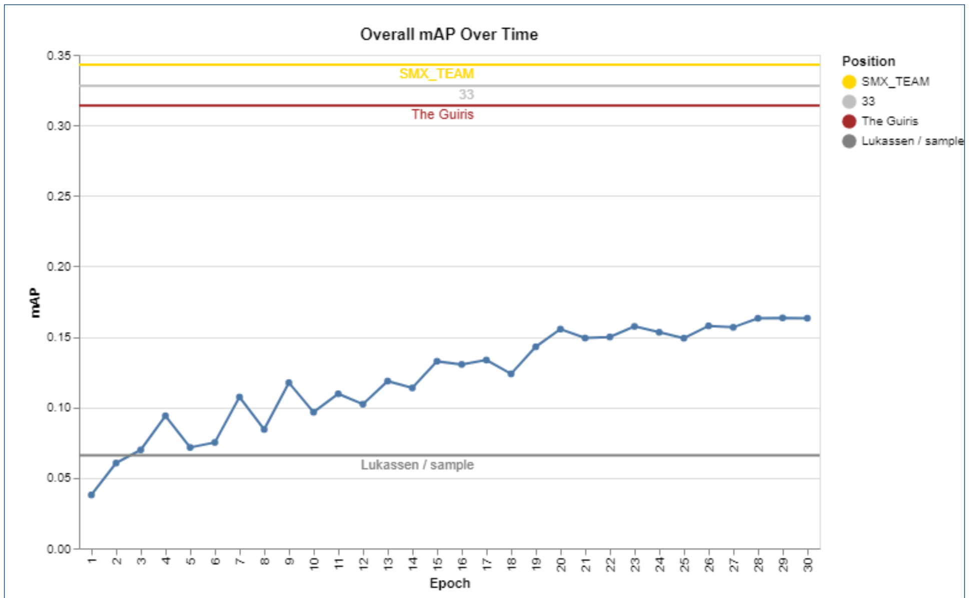

Figure 7 RetinaNet results: Blue: Mean Average Precision of our Faster R-CNN, according to PASCAL VOC 2010 at $IoU > 0.4$. Horizontal lines: Kaggle leader board scores with team names.

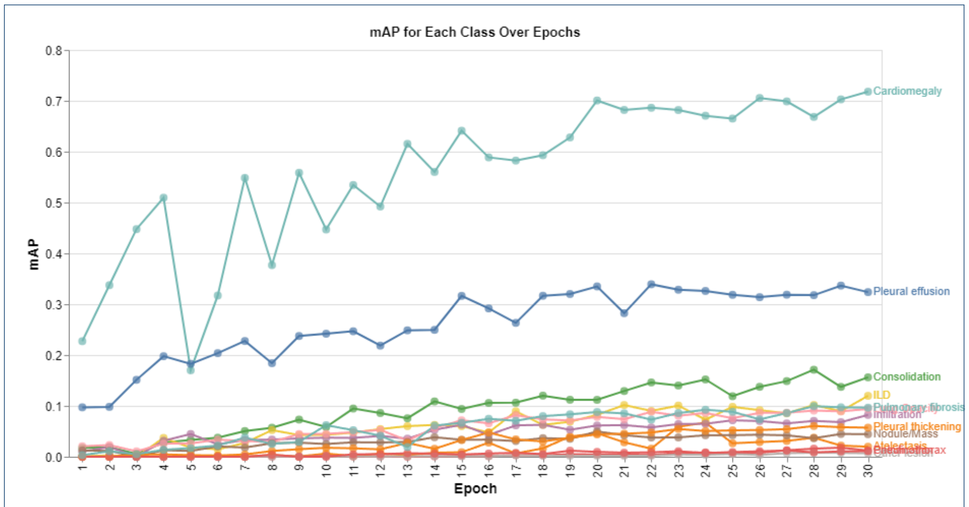

Figure 8 RetinaNet results: mAP per class over time.

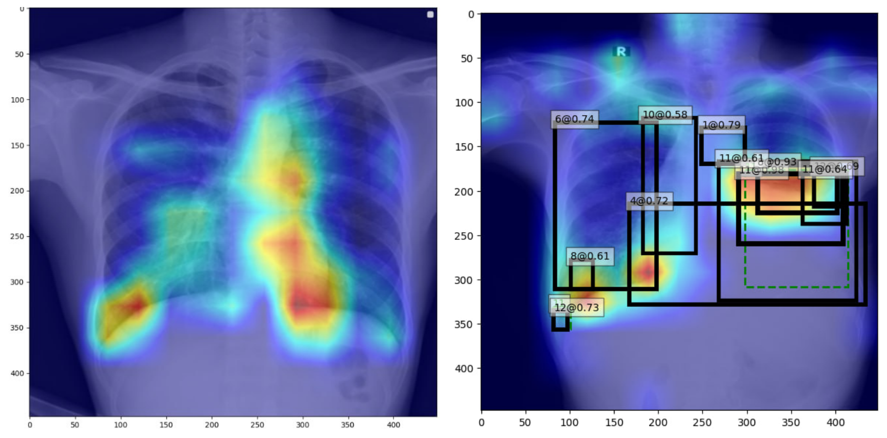

Figure 9 Eigen-CAM for two random images of the validation set. The target box (green) and the predictions (black) are added on the right.