BENCHMARKING SOTA VISUAL OBJECT DETECTION TECHNIQUES FOR MEDICAL APPLICATIONS

Alessandro Pianese
S4106431

a.pianese@student.rug.nl

1. INTRODUCTION

Object detection and classification in computer vision is an active field of research and has been for quite some time[1]. The aim of object detection is to spatially locate and identify a specific semantic object in images or videos[2].

Object detection can be performed with two approaches: by instance segmentation or by object localization. In instance segmentation, each pixel is assigned one or more labels depending to the class they belong to[3]. Usually, a model outputs a three dimensional object having the same spatial dimension of the input image but with a third dimension as deep as the number of classes where each dimension highlights the pixel belonging to that class. In object localization, for each possible object, the model returns a set of coordinates that specify an enclosing box around the said object and a predicted class for the proposed bounding box. Different networks may use different classification algorithms and assign more than a single class to the same object, such as the labels *person* and *woman*.

These techniques are usually applied to natural images, such as RGB pictures taken with standard cameras, but they are also applied to medical images due to the ability of deep networks to learn high-level features[4]. Although difficult to come by due to their scarcity or privacy issues, medical data sets need different approaches compared to large scale data sets[5]. Detection targets in this kind of data sets [6, 7] are similar in size and shape and have ill defined edges which makes targets detection harder.

However, deep convolutional neural networks are not the only means to perform object detection and classification. Before and also afterwards the release of the first deep CNN, hand-crafted methods targeted to a specific class of problems were employed. These methods required the algorithm to manually extract and manipulate the handcrafted features before using a learning model for prediction.

1.1. Previous work

Handcrafted models are usually built on top of existing computer vision methods that performs feature extraction. One of the most famous example is AdaBoost[8]. This method is based on boosting which consists of producing a highly accurate prediction by combining and weighting different slightly inaccurate rules. Boosting proceeds as following: first, the booster receive a collection of labelled examples $\{(x_i, y_i)\}_{i=1..N}$ where x_i is the actual data and y_i is the corresponding label; secondly, the booster creates a distribution D_t over the set of examples for each epoch $t = 1..T$ and obtains *weak hypothesis*[8] with their corresponding errors. By combining this information, the booster can, after T epochs, adapt the weights of each rule based on their accuracy throughout the examples. Differently by AdaBoost authors previously published methods[9, 10], AdaBoost does not need previous knowledge on the accuracy of the weak hypothesis.

A first famous example of handcrafted method for object detection is the Viola-Jones detector[11]. Three key points made this detector perform faster than its predecessor: a new image representation called *Integral image* that allow the authors to calculate a set of features close to the Haar-like features in constant time; using a modified version of AdaBoost where each weak classifier depends only on one feature so that the algorithm is actively selecting a small subset of features from the one extracted through the previous step; a degenerate decision tree that initially employs a classifier whose task is to quickly discard negative results and send positive results to a second classifier which has been adjusted to have higher detection rates w.r.t the predecessor and that can also trigger a third classifier and so on. This method allows for a computationally expensive classifier to focus on only a subset of features that are deemed relevant.

In computer vision, another famous tool employed in object detection is SIFT[12], which stands for scale-invariant features. This method employs keypoints, a series of salient features extracted from images: first, a set of images is processed for keypoints to be extracted and then stored in a database; secondly, a new image goes through the processing stage to extract another set of keypoints. These are compared with the already stored keys to see if then new image matches any of the stored ones or not. Such keypoints are obtained by applying a difference of Gaussian functions in the scale space to smoothed and re-sampled images. Obtaining these point in such a way allows them to be strongly resistant to rotation

and scale operations while also being slightly resistant to shear operations and change in illumination of the image. To reduce the number of false matches, the authors also rejected the matches where the ratio of the nearest neighbour distance to the second neighbour distance was over a certain threshold.

In [13], the authors combined the SIFT keypoints extraction with graph theory to perform object detection in satellite images. Their objective was to successfully detect manmade structure in such images which is a tedious task if done manually. Previous attempts in the literature considered elements such as the buildings shape, corners or even the texture of the satellite image itself. Detecting building, however, resulted to be difficult due to the changes observed by the illumination setting, differences in the contrast, occlusions, such from trees, and the fact that building does not have a standard shape. To overcome these obstacles, the authors applied the SIFT keypoints extraction process to the satellite image, after it was upsampled and smoothed to reduce the noise, and then used the keypoints to build a graph. In this graph, the keypoints would act as vertices and the neighbourhood of each keypoint is used to build the edges. Afterwards, the task of building detection is formulated as a graph cut problem to match the obtained subsets.

Another famous method to extract feature for object detection is histogram of oriented gradients or HOG. The idea behind this method was to divide an image into connected blocks and then obtaining a descriptor by computing and, for each of these blocks, and concatenating histograms of gradient directions. It was first introduced in a patent[14] but it became more widely used in 2005 thanks to [15]. In this paper, the authors used the HOG descriptors to detect humans. The pipeline they set up is rather simple: first, the image is gamma and colour normalized to compensate for illumination variations; secondly, the gradients are computed and, for each pixel, a weighted vote, collected in orientation bins for each spatial region, based on the orientation of an edge centred on that pixel; lastly, gradients are normalized based on the sliding window in question before being concatenated and sent to a linear support vector machine[16] for classification. This method turned out to perform better than previous methods for this task.

A more recent handcrafted method that employs keypoints is the trainable COSFIRE filters[17] one. The name stands for Combination Of Shifted FIlter REsponses since the response of a COSFIRE filter in a specific point is computed as a function of the shifted responses of a simpler filter[17]. The concept of this filter is very simple: a set of Gabor filters[18] is applied to an image, the filter response is Gaussian blurred, the outcome is then shifted using specific, different vectors and finally, all the results are combined by applying a Geometric Mean. Multiplication was chosen over addition due to the intrinsic benefits over shape recognition. Furthermore, the COSFIRE filters can provide strong invariance to rotation, scale, mirroring and contrast inversion. An advantage of this method is its versatility: in fact, COSFIRE filters can be configured on a sample pattern image and then produce excellent results in recognizing such patterns in new images.

The 2015 ImageNet Large Scale Visual Recognition Challenge[19], or ILSVRC for short, marks the transition to deep learning based object detection. The challenge contains two main competition: the first one is the object detection challenge, where the algorithm is expected to provide for each image a class label, confidence score and a bounding box proposal; the second challenge is the object localization one where an algorithm is expected to produce five class prediction, each with its bounding box, in decreasing order of confidence; the main idea behind this challenge was to evaluate the localization procedure by considering the label and bounding boxes that best match the ground truth and, also, to not penalize the prediction that is, in fact, present in the image but not in the ground truth.

Looking at the results of the previous years, we can see how handcrafted methods started getting less frequent than deep learned based methods. One of the last and most successful handcrafted approach was a method proposed during the 2013 challenge by the UvA-Euvision team. Their approach used a selective search method proposed in [20] to create candidate bounding boxes. The focus of this selective search is to generate between a 1́00 an 10́00 locations by looking at object contours. The sheer number of locations allows to virtually include every possible object in the image as stated by the authors. These boxes are then represented by extracted SIFT color descriptor at different scales. Lastly, the proposed boxes are also encoded using a multi level spatial pyramid which allows this approach to run faster than the previous bag of words method.

Looking at the results of 2015 challenge, howver, we can see that the most successful team propositions mostly rely on the Faster R-CNN[21] model.

Faster R-CNN[21] presents itself as an improvement to the 2015 Fast R-CNN showed in [22]. The Faster R-CNN network builds, on top of a VGG16[23] backbone, two modules: A **region proposal network** and a **region of interest pooling**. The former is used to extract a set of rectangular object proposal from the feature map and feed these with the feature maps to the latter, which then classifies each proposal. The novelty in this approach was to allow different modules to share convolutional layers and, therefore, reduce the number of operations needed. Although this model was able to establish itself as state of the art, the efficiency of Fast R-CNN, by today standards, has been long surpassed by other models.

Researchers in deep neural networks are constantly looking to develop breakthroughs in object detection that allow the models to have better efficiency as is the case with R-CNN[24], which introduced a new selective search method that was able to generate and evaluate 2 000 predictions per image. These proposals are then fed to a CNN that extracts

features which are classified via SVM. This improvement was obtained by fixing the number of regions that need to be investigated. Although this method was faster than previously available region proposers, the model still needed around 47 seconds for inference on a single image. Moreover, the selective search algorithm does not encompass automatic parameter tweaking to allow *learning* from examples.

Afterwards, the U-Net model aimed to improve the square window pixel-wise classification method reported in [25] by Ciresan et al in 2012. In fact, the authors of [26] wanted to improve on the two major drawbacks of the square window pixel-wise classification strategy: their sliding windows approach could be accelerated by not looking at individual patches one at a time and the trade-off between accuracy and patch dimensions is quite inconvenient. The idea behind U-Net is to pair an usual contracting network, a network where the output of the convolutional layers gets smaller as we progress through the network, to another network whose pooling layers are replaced by symmetric up-sampling layers to increase the resolution of the output. This, combined with the high number of feature channels, allows the network to propagate high resolution features[26] to the later stages of the network. Hence, the network is able to look for features in lower levels, avoiding the creation of patches and their redundancy, and looks for them just once.

SSD[27], which stands for Single Shot multi-box Detector, aimed to achieve real-time performance while preserving a high mean average precision. Starting from the premise that Faster R-CNN was not suited for real-time applications, this model removed the bounding box proposals and therefore all the re-sampling operations usually performed on these ones. Furthermore, to compensate for the loss of precision due to the last step, the authors added convolutional filters to obtain a 74.1% mAp and a 59fps detection speed outperforming the 7fps of Faster R-CNN.

One of the most recent addition is YOLOv4, presented in [28]. This network is intended to be an improvement of the previous release of the YOLO network proposed in 2018 in [29].

The first version of YOLO[30] provided a simple yet efficient network able to process images at 45fps that can be brought up to 155fps if using the Fast YOLO variant, while still keeping a high mean average value if compared to contemporary models. The authors approach was to handle bounding box prediction and classification as a regression problem[30] allowing the network to perform such prediction in one pass. In the second release of YOLO, called YOLO9000[31], the authors aimed to improve the performances by replacing the original bounding box proposer with a anchor based one. The speed of the model was not affected by this change due to the shared convolutional layers between the CNN and the proposal generator while it helped increasing the accuracy. In the third installement of the model, YOLOv3[29], two issues were adressed. First, they introduced a independent logistic classifier for each class allowing the model to predict multiplle classes for one object. Secondly, the authors added shortcuts connection to allow some information from the first layers in the CNN to *survive* the downsampling until the last layers, increasing the detection rate of small objects. In the last release, YOLOv4[28], several changes were introduced. On top of a new backbone, the authors applied the CSPNet[32] architecture, added a spatial pyramid pooling block[33] to accept input of arbitrary size and the PANet[34] neck to boost the speed of the network by adding more shortcuts in the CNN.

1.2. Medical data sets for computer vision

Most medical data sets for machine learning share the same issues: the amount of data is scarce, images can be better collected, annotated and shared[35]. Hospitals tend to collect and store images and data collected from patients during regular exams but sharing said information may incur in privacy issues[6]. Furthermore, images need to be hand annotated from medical experts to be used for research. Building a medical data set is more difficult than to build a database of natural images. If we consider, for example, a data set of different types of tumors as seen by CT scans, we have to keep in mind that some are rarer than others resulting in a heavily imbalanced data set. Nonetheless, efforts to build and share these data sets are constantly being made.

As a first example, we can look at the DeepLesion[6] data set, the one employed in this report. This data set, released in 2017, contains 32 735 slices of CT scans annotated with different kinds of internal lesions. This data set was built with decades of images stored into the internal picture archive system.

In the same year, the chestX-ray8[7] data set was released. It contains 108 948 frontal view chest x-ray images of 32 717 unique patients collected from 1992 to 2015. To obtain the annotations, the authors used natural language processing to extract labels from the associated description. This step was deemed necessary since radiologists often use complex, abstract sentences while describing their findings. This data set was built with the idea of tackling three issues:

1. Crowd-sourcing annotation is not possible when considering medical data sets, since experts in a specific filed are always required. Automatic labels harvesting, as discussed above, is required to obtain efficient results.

2. Such X-rays images are as big as 3000x2000 pixel and the corresponding region of interest, while varying in sizes, are often very small compared to the full image size. An automatic framework for multi label image classification and disease localization is developed in order to face this issue.

3. Image classification models are usually strongly dependent on the ImageNet[36] data set for initialization. While this data set is able to teach a network to recognize a different number of classes due to its huge amount of instances, it may not be suited for a medical image classification model.

The Digital Retinal Images for Vessel Extraction[37], or DRIVE, data set has been created to allow studies on automatic vessel segmentation. Eye vessels characteristics, such as width, length or branching patterns, are used to diagnose and examine a wide range of diseases. Automatic detection and analysis of such vessels' characteristics can aid medical experts to provide more efficiently treatment for such conditions. The data set was built from a diabetic retinopathy screening program in the Netherlands. The study comprehended 400 subjects within the 25-90 years old range. The final data set consists of forty photographs, equally divided in training and validation, in which thirty-three images do not report any sign of diabetic retinopathy while the remaining seven do. Furthermore the training set contains a single manually built segmentation mask highlighting the blood vessels while the test set contains two: one *gold* standard mask and another one to confront the computer generated output with the one of a manual annotator. It is obvious that this data set is very small and cannot be used to train deep models, since we would only have seven images we can learn from.

The most recent examples of medical data set for computer vision are the numerous data set containing CoVid-19 related lesions and symptoms. For instance, the COVID-CT[38] contains 349 images positive with CoVid-19, extracted from several paper pre-prints, and 397 images that do not present the disease. Unfortunately, this data set is very useful only during an outbreak. In fact, when there are more suspected cases of CoVid-19 pneumonia, it is hard to gather enough kit to test all the patients. With this data set, one can detect if a patient has pneumonia, that may be CoVid-19 related or not, just by looking at a CT scan and avoiding the use of a testing kit. During such outbreak, a patient with pneumonia is very likely to also have the CoVid-19 virus and, therefore, this allows the use of artificial intelligence to aid medics in handling a large number of patients. One of the concerns of this data set is that, due to the source of the CT scans, their quality was not high enough to be employed for this task but, as said in the data set paper, radiologists have confirmed that the quality of a CT scan can be decreased without incurring in loss of information.

1.3. Motivation

The latest approach for object detection are based in deep convolutional neural networks and trained on huge data sets that are built with ten of thousands different annotated images an impressive number of classes. If we take a look at the Faster R-CNN paper, it is reported that they used two different data set for training and evaluation: both Pascal VOC 2007[39] and 2012 data sets and the Microsoft Common Object in Context[40], or COCO : the 2012 release of the Pascal VOC provided 11́530 images with 27́450 different annotations and 6́929 segmentations all distributed within twenty classes; the COCO data set, insead, presents more than 250́000 annotated images with 1.5 million object instances and 80 different classes.

Comparing this data set with the medical ones previously introduced, it is easy to observe a disparity. It is unknown if these deep learning based models can perform well for applications for which very limited amount of data is available

With this report, we wanted to investigate this matter. We selected Faster R-CNN and YOLOv4 and trained them on the DeepLesion data set to observe if the chosen model returned comparable performances to the one reported in their respective papers.

In section 2 we further explain all the details about the DeepLesion data set and the Faster R-CNN and YOLO methods. In section 3 we explain the different settings we used to perform the experiments and the evaluation procedure. In section 4 we presents our results and in section 5 we discuss them. In section 6 we report our conclusions and some additional future developments.

2. METHODOLOGY

2.1. Data set

As a data set, we are going to use the Deep Lesion data set presented in [6]. It was built by using the picture archiving and communication system (PACS) of the national health institute situated in Bethesda, Maryland, United States of America. Radiologists in the institute used bookmarks, as showed in figure 1, to annotate internal lesions in images from CT scans. The goal of [6] authors was to be able to mine hospitals' PACS, almost effort-free, to be able to create a large scale medical data set.

The data set provides 32 735 slices obtained only from CT scans since this source had the largest collection of images. The CSV file contained in the data set provided a different number of information on each lesion which are reported in table 1.

For the experiment we executed we only used the image path, the bounding box corners and the lesion type.

As for the labels, stored as integers, we can encounter eight different kind of lesions, each with their corresponding value:

1. Bone lesions
2. Abdomen lesions
3. Mediastinum lesions
4. Liver lesions

1	File name
2	Patient index starting from 1
3	Study index for each patient starting from 1. There are up to 26 studies for each patient
4	Series ID
5	Slice index of the key slice containing the lesion annotation, starting from 1
6	8D vector containing the coordinates of the diameter of the lesion and of its normal axis
7	4D vector containing the estimated bounding box for the lesion
8	2D vector containing the length of the long and short axis
9	The relative body position of the center of the lesion
10	The type of the lesion
11	Field to note if the slice is possibly noisy or not
12	The slice range: in fact for each slice the authors also included additional contiguous slices
13	Spacing (mm per pixel) of the x, y, and zaxes. The 3rd value is the slice interval, or the physical distance between two slices
14	Image size
15	The windowing (min max) in Hounsfield unit extracted from the original tridimensional file
16	Patient gender. F for female and M for male
17	Patient age
18	Official randomly generated patient-level data split

Table 1: Table describing the data set's CSV file.

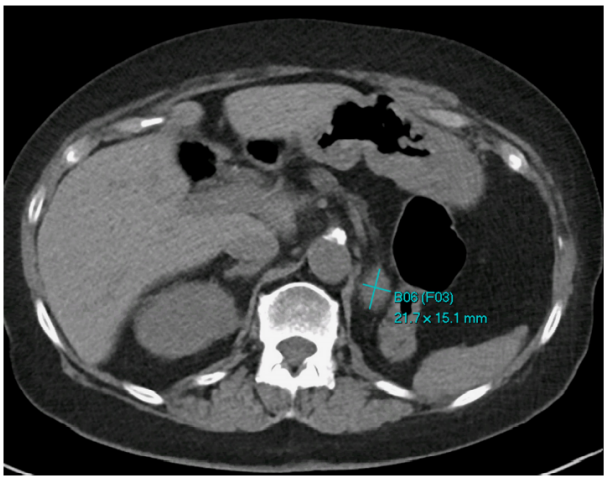


Fig. 1: An example of bookmarked CT scan image from the DeepLesion paper.

5. Lung lesions
6. Kidney lesions
7. Soft tissue lesions, such as fat, muscle or skin lesions
8. Pelvis lesions

Moreover, there was a ninth label used to identify other type of lesions with value -1. This class, despite being present in the data set, is not employed while training the universal lesion detector in [6], therefore we will also ignore this class and only focus on the remaining 9 816 slices. In figure 2 we can observe how many samples are available0 for each class and we can appreciate that this data set is heavily imbalanced. In fact, the *lung lesion* label is the most predominant, with 2 394 occurrences out of the total 9 816. On the other hand, the least encountered are bone lesions with just 247 labels.

For each annotated slice, the data set also provides additional unmarked sections up to 30mm above and under the annotated slice. In the paper it is unclear if all these extra slices contain the lesion or not and it is not reported if the lesion increases in volume, decreases or does not undergo any change. Due to these unknown factors, it has been decided to ignore these additional slices to avoid the risk of introducing false positives into the data set.

2.2. Convolutional neural networks

Convolutional neural networks, CNNs or ConvNets, are a specific type of feed-forward neural networks that take im-

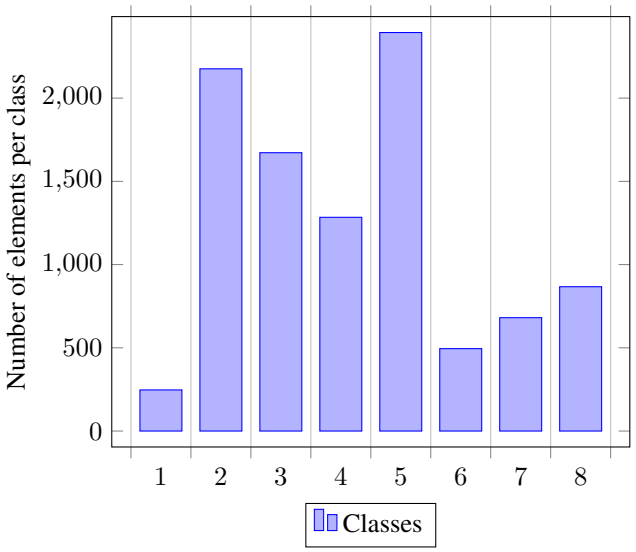


Fig. 2: Bar chart displaying the imbalance of the data set

ages as input. An example of a CNN is reported in figure 3. Inside the network, images are processed with the use of different kinds of layers. The ones that are ubiquitous are convolutional layers, pooling layers and fully-connected layers.

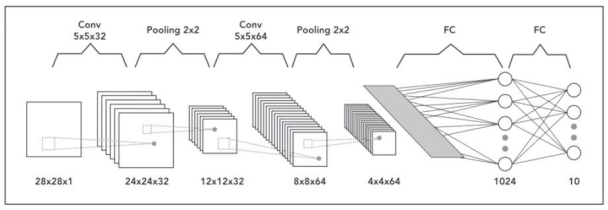


Fig. 3: Example of convolutional neural network. Images from Muhammad Rizwan's *Convolutional Neural Networks - In a Nutshell*

A convolutional layer simply performs a filtering operation on the image with a number of kernels. Each kernel slides over each pixel of the image spatial dimension and performs a dot product between the filter and the corresponding highlighted image subsection. Here, among the values of the kernel itself, the filter size can be set and the most common dimensions are 3x3xC, 5x5xC, 7x7xC and 11x11xC, where C is the number of channel of the input image. Small dimensions allow to highlight finer details more easily while bigger filter sizes tend to ignore these smaller feature. The stride of a kernel can also be changed: a kernel with stride one will move one pixel to the right after each dot product; if the stride were two, the process would skip a pixel while performing the filtering operation. A higher stride will allow for faster computation but may make some features harder to find.

A pooling layer is used to downsample the input along the spatial dimensions and extract dominant features. In fact,

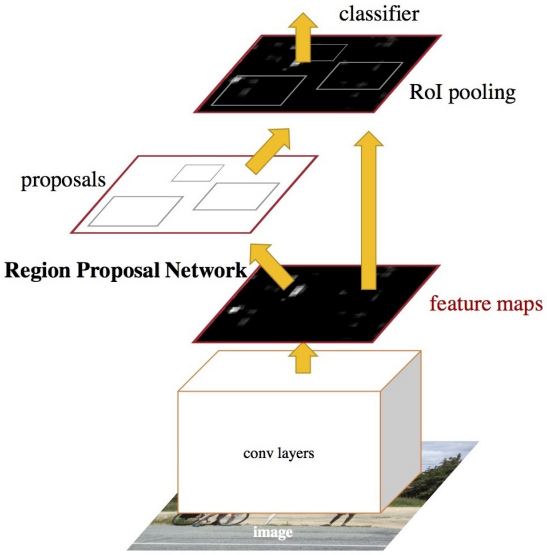


Fig. 4: Architecture of Faster R-CNN, image taken from [21].

since applying several convolutional filters increase the dimensionality of the output, it's easily observable how the amount of weights needed would be too massive to handle. The two most common pooling methods are max pooling and average pooling. In max pooling, the maximum value from the section of the image in question is selected, while in average pooling the average of all values of the kernel is chosen instead. Max pooling also acts as a better noise suppression than average pooling since noisy values are discarded right away.

A fully connected layer applies a linear transformation to the input such as $y = W \cdot x + b$ where x is the input, W are the weights and b is the bias. When a layer of this kind is the last layer in a CNN then it performs the actual discrimination and is able to learn a rule. A fully connected layer in this position will output a $1 \times 1 \times C$ vector where each value corresponds to a class score and where C is the number of classes we have in our problem. A fully connected layer expects a fixed size input to applies the weights to predict the correct label However, fully-connected layers are not so widespread in modern CNNs as they were before due to the high number of parameters they introduce.

2.3. Object detection using CNNs

2.3.1. Faster R-CNN

The first model we decided to employ is Faster R-CNN[21]. In its paper, as displayed in figure 4, it contains a VGG16[23] backbone, a region proposal network and a region of interest pooling layer. In the implementation we utilized, the backbone was replaced by a RenNet-50[41] CNN.

This CNN, as the names implies, presents fifty layers and can be split up into five stages:

- The first stage presents a single convolution with a 7x7 filter dimension and 64 different kernels with stride two. The output of this stage has a spatial dimension of 112x112 for an input size of 224x224.
- The second stage starts with a max pooling layer with 3x3 dimension and stride two. Then we have three convolutions. The first one has a 1x1 filter size and applies 64 different kernels. The second one has a 3x3 filter size and applies 64 different kernels. The last one has again a 1x1 filter size but applies 128 different kernels. These three convolutions are repeated three times in this order for a total of 9 layers. The output of the second stage has a spatial dimension of 56x56.
- The third stage consists of firstly a 1x1 filter dimension and applies 128 different kernels, followed by a convolution with a 3x3 filter dimension and again applies 128 different kernels and, lastly, a convolution with 1x1 filter size with 512 different kernels. These convolutions are repeated four times for a total of twelve layers. This stage output has a spatial dimension of 28x28.
- The fourth stage repeats the previous pattern. First a 1x1 filter size that applies 256 different kernels then a 3x3 filter size that again applies 256 different kernels and lastly a 1x1 filter size that applies 1024 different kernels. These convolutions are repeated six times for a total of eighteen iterations. This stage output has a spatial dimension of 14x14.
- The last stage consists of a first convolution with a 1x1 filter size and 512 different kernels then a convolution with 3x3 filter size and 512 different kernels and then a 1x1 convolution with 2048 different kernels. These convolutions are reiterated three times for a total of 9 layers.

At the end of this CNN we find an average pooling layer that precedes a fully connected layers with 1000 neurons. This fully connected layer forwards its output to a softmax function that transforms the output vector into a class probability vector, with values between 0 and 1.

If we count all the convolutions, the fully connected layer and the softmax function, they add up to fifty layers.

On top of the backbone, Faster R-CNN presents two modules: a fully convolutional layer that generates region proposals and a region of interest pooling layer that uses these proposals.

The region proposal network, or RPN, takes a feature map from the last convolutional layers of the backbone as an input and it outputs a list of proposals. To generate proposals, the RPN extracts features by feeding the feature map to a small network with a sliding window procedure. This network takes as an input a 3x3 spatial window, reduces its dimension and feeds the output to two sibling modules: a box regressor layer and a classifier layer, both implemented with a 1x1 convolutional layer. At each window location, the RPN predict multiple proposals up to a maximum defined value called k. Each proposal has $4 \times k$ outputs encoding the coordinates of the bounding box and $2 \times k$ score reporting the probability of an object being there or not being there. The value of k depends on the the number of anchors. In the paper the authors use 3 aspect ratios and 3 scales for a total of $k = 9$ anchors. They also state that this approach is translation invariant and presents a smaller model size.

The key improvement of this model is the speed that is able to achieve with regard to its predecessor, Fast R-CNN[22]. In fact, it is able to share convolutional features at a single scale, without needing to train the regressor for different scales. Their approach, which uses a method called "*pyramid of anchors*", it is able to identify region of interests at different scales by using different aspect ratios anchors while maintaining a feature map at a single scale.

The employed loss function is formulated as follows:

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

where i is the index of an anchor in a mini-batch, p_i is the predicted probability of an object being inside the anchor i, p_i^* is the ground truth labels and is 1 if the anchor is positive or 0 if the anchor is negative, t_i is a 4D vector with the coordinates of the predicted bounding box while t_i^* is a 4D vector with the ground truth coordinates. The L_{cls} is the loss w.r.t. the classes of the present objects vs the objects not present while $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$ where R is the smooth L_1 introduced in [22] which is

$$R(x) = \begin{cases} 0.5x^2 & \text{if } \mid x \mid < 1. \\ \mid x \mid -0.5 & \text{otherwise.} \end{cases}$$

The term $p_i^* L_{reg}$ means that the regression loss is activated only for positive anchors, i.e. $p_i^* > 1$.

The terms N_{cls} and N_{reg} represent respectively the mini-batch size and the number of anchor locations. They are used for normalization and an out-of-the-box Faster R-CNN presents $N_{cls} = 256$ and $N_{reg} = 2\,400$. The λ parameter is used for balancing and by default it is set to $\lambda = 10$.

For the bounding box regression the authors of the paper parameterizated the four coordinates as follows:

$$t_x = \frac{x - x_a}{w_a} \qquad t_y = \frac{y - y_a}{h_a}$$

$$t_w = log(\frac{w}{w_a}) \quad t_h = log(\frac{h}{h_a})$$

$$t_x^* = \frac{x^* - x_a}{w_a} \qquad t_y^* = \frac{y^* - y_a}{h_a}$$

$$t_w^* = log(\frac{w^*}{w_a}) \quad t_h^* = log(\frac{h^*}{h_a})$$

In the above equations x, y represent the bounding box center coordinates while w and h respectively represent the bounding box width and height. Variables with a as a subscript, such as x_a, refers to the anchor bounding box relative value and variables with a $*$ as a super script, such as x^*, represents the relative ground truth bounding box value.

This regressor predicts bounding box proposals differently than before. In fact, previous models [22] performed regression on features obtained from arbitrarly sized regions of interest and the weights were shared among different regions size. Here, different bounding box regressors are learned for different sizes and they do not share weights so that the network is able to correctly predict boxes of different sizes.

The region of interest, or RoI, pooling layer of Faster R-CNN is the same as Fast R-CNN. It accepts as input a feature map from the last convolutional layer and a set of object proposals from the RPN regressor. For each proposal it extracts a fixed-length feature vector from the corresponding section of the feature map.

In Faster R-CNN paper, the authors adopt a 4-step alternating training:

1. In the first step, the RPN is trained, initialized with an ImageNet[36] pre-trained model and then used to train the Fast R-CNN components. This last tuned network is then employed to initialize the RPN from now on.

2. In the second step a separate detection network is trained using Fast R-CNN and the previously generated proposals. The detection network is again initialized with the ImageNet pre-trained model.

3. In the third step the RPN is initialized with the above tuned detection network but the convolutional layers are fixed so that the training modifies only the layers unique to the RPN.

4. as the last step, the shared convolutional layers fixed are kept and the layers unique to the Fast R-CNN classifier are fine-tuned.

The RPN and the Fast R-CNN ROI pooling section share the same convolutional layers.

The network employed in DeepLesion[6] data set paper was modified to fit the needs of the task. The authors kept the VGG16 backbone but removed the last two pooling layers to allow smaller features to appear in the outputted feature map. Furthermore, they recomputed the anchor sizes and the anchor ratios by investigating the ground truth bounding boxes of the data set. In the end they used five anchor scale (16, 24, 32, 48, and 96) and three anchor ratios (1:2, 1:1 and 2:1). Lastly, the authors replaced the 4096D fully connected layer in VGG16 to decrease the size to 1/4 while keeping a comparable accuracy.

In our experiment we used the out-of-the-box PyTorch implementation of Faster R-CNN. More on our experimental setup in section 3.1.

2.3.2. YOLO

We decided to also pick YOLOv4 since it's a state of the art object detection system which is capable of both high detection speed and accuracy.

When YOLO was first released, it proposed a major boost in performances compared to other networks of the same period, such as Faster R-CNN. YOLO applies a single neural network to the whole image. This image is split into regions and the network, for each region, predicts a class and a bounding box. Non maxima suppression is later applied to combine bounding boxes of the same element. The downside of this approach is that each region can only detect one object. Since YOLO uses grids of 7x7, the maximum number of objects it can detect is 49.

In the second release of YOLO[31] the authors improved the accuracy of the predicted classes and bounding boxes. Besides batch normalization and training the classifier at a lower resolution, the most substantial change they did with YOLOv2, or YOLO9000, was to add a convolution with anchor boxes. Inspired bythe Faster R-CNN architecture, YOLOv2, with a fully connected layer, is able to share image features with the bounding box proposer. Detection speed and accuracy were substantially increased: the mAP went from 63.4% to 73.7% and the speed increased respectively from 40 fps to 81 fps.

With YOLOv3[29] the authors tackled the multi label problem. Since an object, with the data set they were using, can be assigned multiple labels, it is incorrect to assign only one. An independent logistic classifier for each class was used to replace softmax. Furthermore, the authors also addressed the problems YOLO had with detecting small objects. They introduced shortcut connections that allowed smaller features from earlier future maps to *survive* until the later stages. This increased performances with small objects but also made larger objects harder to find.

The YOLOv3 head that predicts bounding boxes and performs classification, is similar to what we can find in YOLO9000. For generating the priors, they used the dimension cluster technique instead of hand picking anchors dimension. Running on the whole training data set, the au-

thors used k means clustering on the ground truth boxes to obtain k centroids to use as priors. To calculate the distance between a box and a centroid, they used $d(box, centroid) = 1 - IoU(box, centroid)$.

To predict the location of the bounding boxes they do something similar to YOLO where the coordinates of the boxes are predicted relative to the grid they fall in. In fact, for each box, the networks predicts four coordinates: t_x, t_y, t_w and t_h. The first two represent the box offset from the current cell top left corner while the last two represent the predicted width and height respectively. In the paper, to turn these values into usable bounding box coordinates, they used the following equations:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned}$$

where (c_x, c_y) is the offset from the top left corner of the images, p_w and p_h are respectively the width and height of the corresponding anchor.

YOLOv3 also assign an objectness score to each prediction. If the predicted bounding box overlaps an object more than any other bounding box prior then the objectness score is one, otherwise it gets ignored.

To calculate the loss for the bounding box, the authors used the sum of squared error $L(w, x) = \sum_{i=0}^{N} \frac{(t-\hat{t})^2}{N}$ where t is the predicted value while $\hat{t}$ is the ground truth.

Each box prediction also performs multi-label classification. Instead of using a softmax approach, the authors use independent logistic classifiers for each class. The loss employed in this phase is the binary cross-entropy loss.

For the training, they heavily used data augmentation, such as flip, rotation, crop, trained at different scales and batch normalization.

The change the authors of [28] proposed for YOLOv4 was to employ a different backbone. They proposed to use CSPDarknet53[29, 32] since it allowed for a larger receptive field, therefore a larger accuracy, while maintaining real-time performances. As a whole, the architecture of YOLOv4 is comprised of the said backbone, a spatial pyramid pooling[33] block to further increase the receptive field, PANet[34] path aggregation neck and YOLOv3[29] anchor based head.

The CSPDarknet53 backbone is comprised of two parts: a Darknet53[29] backbone and the cross stage partial network[32] technique applied to it.

DarkNet53, similarly to ResNet-50, contains in its name the number of convolutional layers it presents as also shown in figure 5. As reported in [29], this backbone performances are on par with ResNet-101[41] and ResNet-152[41] but with 36% less floating point operations. This backbone is enhanced with the cross stage partial network (CSPNet).

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig. 5: Architecture of DarkNet53 taken from [29]

The aim of CSPNet is to reduce the amount of computation during the back-propagation while still keeping an enough detailed gradient to perform precise calulations. The key insight is to partition the feature map of the base layer into two parts that are then merged at the end of the network. This also allows for the gradient flow to be split and propagated throughout different paths. The splitting of the gradient avoids the learning of redundant information. As we can observe in figure 6, the output of the base layer is being initially split and then merged back together in a partial transition layer. In fact, CSPNet optimizes the function

$$y = M([x'_0, T(F(x''_0))])$$

where F is a mapping function, the input is split into two parts $x_0 = [x'_0, x''_0]$, T is the function used to truncate the gradient flow and M is the transition function used to reunite the two parts.

CNNs accept input of a certain size, such as 240x240, or they do resize the input images to allow compatibility with fully-connected layers. This limits the accuracy of the network as a whole[33]. To avoid this, the authors of [33] suggest to replace the last pooling layer with a spatial pyramid pooling block to allow a variable sized feature map to be turn into a fixed length vector. A SPP improves the Bag of Words[42] approach. It maintains spatial information by pooling together local spatial bins. Since the size of said spatial bins is proportional to the image size, the number of bins is fixed regardless

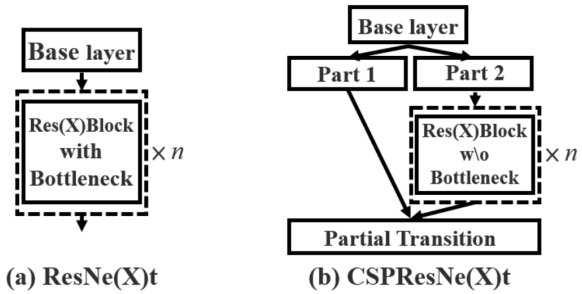


Fig. 6: Example of CSPNet applied to a ResNe(X)t[32]

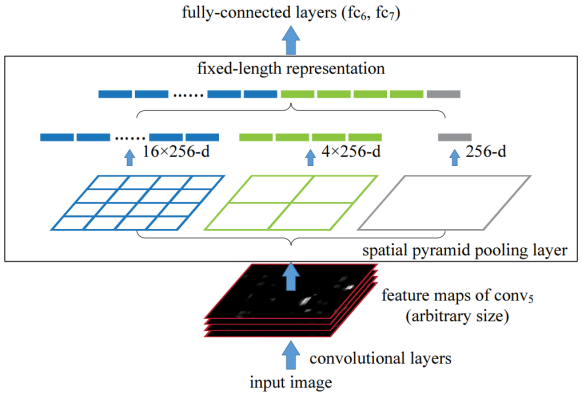


Fig. 7: Example of spatial pyramid pooling taken from [33]

of the image size. In figure 7 we can observe how the SPP works. Bins of different sizes are used to pool the response of each filters of the last convolutional layer, 256 in this example. The output of this pooling layer is kM dimensional, where M is the number of bins and k is the number of filters. In the paper, max pooling is used to pool the responses for each bin throughout all the filters.

The path aggregation network, or PANet, aims to improve the information flow in proposal based instance segmentation network. PANet, as shown in figure 8, consists of five components. To obtain the performance improvements, we have to look closely at these three modules:

- Bottom-up path augmentation
- Adaptive feature pooling
- Fully-connected fusion

The bottom-up path augmentation allows strong low-level features to be propagated to higher levels through a shortcut, the dashed green line in figure 8. This path consists of 10 layers instead of the 100+ layers along the default path. Propagating the strong response of repeating patterns and local textures allows the network to improve the localization performance.

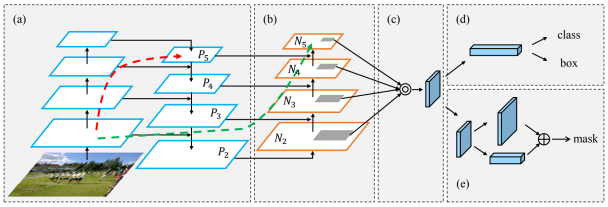


Fig. 8: Architecture of PANet taken from [34]. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) Box branch. (e) Fully-connected fusion.

In the feature pyramid network[43], bounding box proposals are assigned to different feature levels based on their size. Small proposals are assigned to low feature levels while larger proposals to higher levels. We can see how a small difference of a handful of pixels can result in two similar proposals to be assigned to different feature levels. To solve this, the authors of [34] suggest to employ adaptive feature pooling. This means to pool features from all level and fuse them together for generating the prediction.

To increase the quality of the final prediction, the authors combine a fully convolutional network, or FCN, and the fully-connected layer predictions. FCNs are able to predict pixel-wise class instances in mask segmentation and the authors of Mask R-CNN[44] also put a small FCN on the feature grid to predict instances without class competition. On the other hand, fully-connected layers provide different results based on the location of the proposal and can therefore adapt to different spatial locations. Fully-connected layers also use global information for each proposal prediction.

YOLOv4, to have a faster and more accurate regression in case of both overlap and inclusion, the authors decided to employ the Complete IoU Loss[45]. To achieve this, the loss must consider three geometric factors: overlap area, central point distance and aspect ratio. The loss equation is:

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt}}{c^2} + \alpha v$$

where ρ is the euclidean distance between the two boundig box centers, b is the predicted bounding box, b^{gt} is the ground truth bounding box, c is the diagonal lenght of the smallest box enclosing the two boundig boxes, the predicted and ground truth ones, v measures the consistency of aspect ratios

$$v = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2,$$

and α is a positive trade off parameter defined as

$$\alpha = \frac{v}{(1 - Iou) + v}.$$

For our experiments we select YOLOv4 since this latest release brought such improvement to both speed and accuracy over the previous versions as well as other state of the art object detection neural networks.

3. EXPERIMENTAL FRAMEWORK

In this section, we explain the way we set the experimental framework and the evaluation procedure we adopted.

3.1. Faster R-CNN

To run experiments with Faster R-CNN[21], we employed the PyTorch[46] implementation of the model. Differently from the original Faster R-CNN paper, this implementation used a ResNet-50[47] backbone instead of VGG16[23]. The remaining sections of the network, the RPN module and the RoI pooling layers were the same as the ones described in the paper.

The only change we needed to make was to set the number of classes inside the RPN module to the number of different lesions in our data set plus one. Faster R-CNN also contemplates a background class, referred to as class zero, which is discarded during classification.

To experiment, we decided not to modify the internal structure of the backbone, as done in the data set paper, but we only tweaked some hyperparameters: since the PyTorch library allowed us to import the model with a pre-trained or untrained backbone, we decided to test with both to see if a difference could be observed. Also, the library allowed us to pick our optimizers. The first optimizer we decided to pick was stochastic gradient descent[48], due to its simplicity but also because it's the same optimizer used in the data set paper with the authors' customized Faster R-CNN model. The second optimizer we selected was Adam[49] due to its adaptive gradient effectiveness. Both optimizers are briefly discussed in section 3.1.1.

With this in mind, we combinations of hyperparameters are displayed in table 2. The pretrained weights of PyTorch[46] Faster R-CNN were obtained by training the model on COCO Train2017[40]. Looking at our configurations, we firstly run an experiment with the pretrained backbone and the adam optimizer without batch balancing. With this experiment, we wanted to observe the baseline results for the model. Since we did not perform batch balancing, we expected the model to overfit on the most predominant class of the dataset.

The next two experiments, 2 and 3, both had balanced batches and the Adam optimizer but here we wanted to observe how much of an impact the pretrained backbone can have with said optimizer. The same happens for experiments 4 and 5 but, instead of Adam, we are using SGD.

Lastly, we are running one experiment with pretrained backbone, balanced batches and Adam optimizer but, this time, we also are adding a data augmentation pipeline to measure the possible increase or decrease in detection accuracy.

Concerning the learning rate, we discovered that 0.0001 was one of the optimal value that allowed a trade-off between speed and accuracy while using Adam. Higher learning rates, such as 0.001, resulted in an exponential increase in the loss after the first epochs, while smaller learning rates, combined with the learning rate decay, didn't allow for the model to keep learning at a reasonable rate after a handful of epochs.

For SGD, we found that the same learning rate wasn't allowing the loss to decrease throughout iterations so we kept lowering the learning rate until it reached 0.00002. At this point, we saw that lowering the learning rate further wasn't going to outperform the Adam optimizer. More on this in section 5.

We trained all the Faster R-CNN experiments for 110 epochs. For the experiments with the balanced batches, the images from the least predominant classes were presented cyclically, until the network encountered once all the images from the most abundant class.

We didn't use any learning rate scheduler although Adam optimizer does decrease the learning step on its own, as explained in the relative sub-subsection.

Momentum for SGD was set to 0.9.

3.1.1. Optimizers

SGD is different from batch gradient descent or mini-batch gradient descent. In the stochastic approach we only use one example for calculating the cost instead of calculating the cost of the whole batch. The formula for updating the weights is

$$w_{t+1} = w_t - \eta \cdot \nabla_{w_t} C(x, y)$$

where w_t are the weights at iteration t, η is the learning rate, ∇_{w_t} is the gradient with regard to w_t and $L(x, y)$ is the cost function with the current input.

SGD presents the benefits to be fast, since it only calculates the cost of one example at a time, and also easy to be understood given its simplicity. However, due to its limited vision, the algorithm is slower to converge and has more probability to be stuck in a local minimum. To avoid this, the concept of momentum has been introduced.

Momentum helps the optimization move faster towards the (local) minimum. The way it works is to add a fraction of the last updates of the weights. The formula become

$$w_{t+1} = w_t - \eta \cdot \nabla_{w_t} C(x, y) + \rho v_t$$

where ρ is the momentum value, usually set at 0.9 and v_t is the change of the weights in iteration t. As a result, if our optimization process would *fall* in a local minimum while descending, the momentum would allow it to keep moving and exit this minimum.

Adam, differently from stochastic gradient descent, does not maintain a single learning rate for weights updates but it is adapted as the training proceeds. The authors of [49] state that Adam combines the advantages of both AdaGrad[50] and RMSProp[51].

Adam is efficient with sparse gradient problems, as AdaGrad, and it adapts the learning parameter on the average first

#	Pretrained backbone	Optimizer	Balanced batches	Data augmentation
1	✓	Adam		
2		Adam	✓	
3	✓	Adam	✓	
4		SGD	✓	
5	✓	SGD	✓	
6	✓	Adam	✓	✓

Table 2: Table summarising all the experiment we ran with Faster R-CNN

moment, as in RMSProp, but it also employs the average of the second moment. The algorithm uses the moving average of the gradient and the square gradient whose decay rates are controlled by variables β_1 and β_2.

The algorithm for Adam needs the following inputs:

- step size α
- β_1, β_2 exponential decay rates for momentum estimates
- stochastic objective function $f(w)$
- w_0 initial parameter vector

Meanwhile the algorithm proceeds as following:

1. $m_0 \leftarrow 0$ initialize first moment vector
2. $v_0 \leftarrow 0$ initialize second moment vector
3. $t \leftarrow 0$ initialize timestep
4. while w_t not converged do
   (a) $t \leftarrow t + 1$
   (b) $g_t \leftarrow \nabla_w f_t(w_{t-1})$ get gradient w.r.t. time step t
   (c) $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ update biased first moment estimate
   (d) $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ update biased second raw moment estimate
   (e) $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$ compute bias-corrected first moment estimate
   (f) $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ compute bias-corrected second raw moment estimate
   (g) $w_t \leftarrow w_{t-1} - \frac{\alpha \cdot \hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon}$ update parameters

3.1.2. Data balancing

When we did not perform data balancing, examples from the data set were progressively selected. This mean that batches were mostly containing images from the same class. To improve this, we used a custom function to provide balanced batches. First, the function divide all the images in different bins based on their class. By default, each batch contains 32 elements so the algorithm divides this number by the number of classes it has and picks an equal amount of images from each. Since the data set is heavily imbalanced, it has classes that are way more copious than other implying that smaller classes are going to run out of images to offer. To make up for this, the algorithm is cyclically going to select pictures once a class has already been entirely seen.

3.1.3. Data augmentation

The augmentation pipeline utilized in the last experiments consisted of several transforms: we first add $50px$ pad on each side of the image in a way that the added information is consistent with what was already present in the image; next, we perform a random rotation between $-30°$ and $30°$, a random centered crop between $0px$ and $64px$ on each side and a rescale operation to a 240x240px resolution. Using this pipeline, we can be sure to use slightly different images each time data augmentation is used. When the batch balancer would select an image from a very small class, instead of seeing the same image over and over, it would study a slightly different version of it, possibly improving the generalization process.

3.2. YOLO

To train YOLOv4[28], after we compiled the darknet framework on our system, we followed the instructions available at the project repository: `https://github.com/alexeyab/darknet`

For a first experiment, we followed the repository guidelines on modifying the configuration file. To set the config file to work with our data set, we tweaked the following parameters:

- the batch dimension parameter was set to 64 `batch=64` and the subdivision parameter, to avoid incurring in an out of memory error, was set to 32.
- the network dimension, width and height, was set to 448x448

- the upper limit of iteration was set to 16000
- the flip data augmentation was disabled since some types of lesions are not always observed on both side of the body
- the number of classes was set to be eight in all the three `[yolo]` layers

- the filters parameter in the three convolutional layers before each `[yolo]` layer was set to 39, accordingly to the number of class and following the formula `filters=(classes+5)` $\times 3$ as provided in the repository instructions.

In a second YOLO experiment we decided to try to increase detection and classification accuracy on smaller object. On top of the previous modifications, we also tweaked the following hyperparameters:

- we calculated the anchors for this specific data set and replaced the default values in each `[yolo]` layers with `anchors = 15, 15, 21, 21, 29, 26, 34, 38, 50, 37, 46, 58, 69, 60, 97, 94, 156,133`
- the stride was increased to four and layers parameter in the different `[yolo]` layers was set to 23 to allow the network to detect objects smaller than 16x16 pixels
- the network dimensions was increased to 869x869 for training. During inference it was further increased to 1024x1024 to increase the accuracy of the model without the need to re-train the network.

We did not feel the need to change the default learning rate, which was 0.001, as it seemed to be compatible with the data set in use. Also, the momentum was left at its default setting which was, again, 0.9. The optimization process and learning rate scheduling were handled by the framework itself as they were *hidden* from the user. The YOLO model trained for 16 000 iterations which correspond to 100 epochs. However, since the network saved weights every 1 000 iterations and we evaluated the model using these backup weights, we reported the results for each one of this evaluation. So, the YOLO results chart displays 16 evaluation steps.

3.3. Evaluation protocol

To evaluate the model we trained, we mainly used mean average precision and intersection over union.

3.3.1. Mean average precision

Mean average precision is the average of the AP or average precision. To explain this, we have to introduce three elements: precision, recall and the precision-recall curve. Precision is the measurement of how accurate is our classifier. It reports how many true positives we found in our results and is calculated as

$$precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

while recall is the measurement of how successful is our model in detecting all the positives in the image and is calculated as

$$recall = \frac{True\ Positive}{True\ Positive + False\ Negatives}$$

We used these two metrics to generate a precision-recall curve which is used to observe the trade-off between precision and recall for different confidence level in a classifier. In our implementation, we used threshold values ranging from 0 to 1.0 with a 0.1 step for a total of 11 points. For each one of this threshold values, we evaluated all the predictions with a confidence level equal to or greater than the threshold. For each threshold value then, we obtained a precision and a recall value which represented one point on the precision-recall curve, as shown in example figure 9.

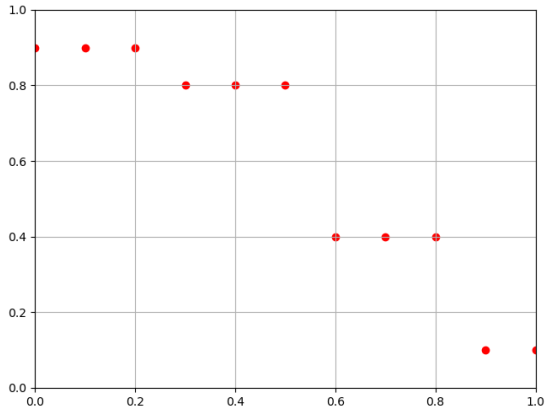


Fig. 9: An example of precision-recall points

Having obtained this curve, we used these points to calculate the area under the curve with the formula

$$\sum_{i=1}^{n} precision_i \times (recall_i - recall_{i-1})$$

where $precision_i$ and $recall_i$ are respectively the precision and recall values of the i^{th} point on the curve. With the above-mentioned formula, we would obtain an approximation of the area under the curve like the one displayed in figure 10. This approximation is what is used in this filed to calculate the precision of a classifier.

The area under the curve we just obtained is the AP. If we repeat this process for each class then we will obtain, in our specific case, 8 average precisions, one per class. The average of all of the AP is the mAP.

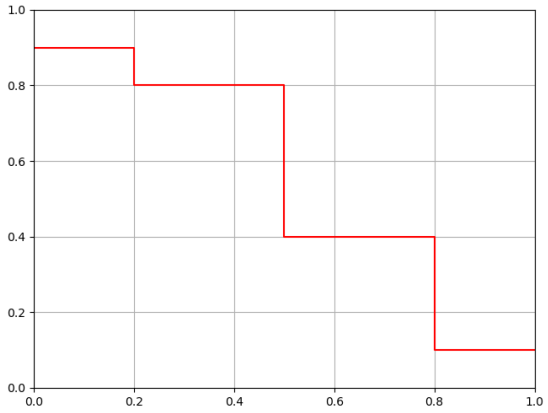


Fig. 10: Example of area under the precision-recall curve

3.3.2. Intersection over union

Intersection over union, or IoU, is a metric used to evaluate the accuracy of a detector. Specifically, it represents how accurate is the predicted object location versus the ground truth.

To calculate the IoU, as shown in figure 11, we need two bounding boxes: a predicted one and the actual bounding box. We first obtain the area of the intersection between these two and, if it is greater than zero, we divide by the area of the union of the two boxes.

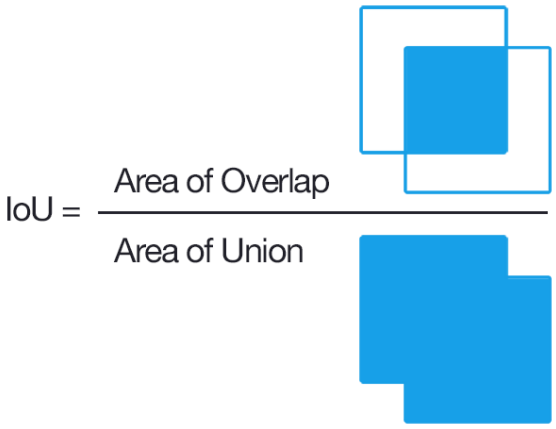


Fig. 11: IoU equation with graphic representation of meaning. Image taken from *here*, credit to Adrian Rosebrock

3.3.3. Evaluation procedure

For Faster R-CNN we had to write our evaluation algorithm. The algorithm consists of these steps:

1. for several threshold values do:
   (a) Make inference on a batch of images
   (b) For each image, get the predicted bounding box, label and confidence level
   (c) If the confidence level is greater than the current threshold does:
      - If the overlap between the predicted bounding box and the ground truth one is greater than a certain IoU threshold, add this value to the list of IoU values. Then check whether the prediction and ground truth were a true positive or the prediction was a false positive and the ground truth a false negative. If the overlap is less than the IoU threshold, mark the prediction as a false positive and the ground truth as a false negative.

   (d) Repeat these steps until there are no more batches
   (e) Obtain, using the number of true positive, false positive and true negative, precision and recall values for each class to be put on the precision-recall curve.
2. After having obtained all the precision-recall curves, calculate the AP for each class.
3. Get the average of all the APs
4. Return the mAP and the average IoU

   For YOLO, since it runs inside the darknet framework, we didn't have to write our algorithm but we could use command-line arguments that would calculate the mAP, IoU, false negatives and AP per class on their own.

   For both models, we set the IoU threshold at 21%.

4. RESULTS

4.1. Faster R-CNN

In table 3 we report the results of the Faster R-CNN experiments.

In figures 12(a) and 12(b) we can find the results of the first experiment we ran. We just fed all the images, once per epoch, to the model and collected the results while using Adam[49]. If we look at the results evaluated on the train set We can observe how the loss decreases before stabilizing in the last thirty epochs. Furthermore, we can also see how the mean average precision, or mAP, and the intersect over union, IoU, are increasing but at a very slow rate except, again, for the last epochs, where they find a steep increase. However, if we look at the results on the test set, we can see how the mAP and IoU are behaving slightly worse than previously shown. If we look at the last epoch, which is the best one, the performances on the test set is around half the performances on the train set.

In the next experiments, we stick with Adam as an optimizer and we used the same learning rate while providing balanced batches to the network. The only difference between

Augmentation	Pre-trained	Batches	Optimizer	Best epoch	mAP train	mAP test	IoU train	IoU test
No	COCO[40] 2017	unbalanced	Adam[49]	110	38%	12%	43%	21%
No	No	balanced	Adam	60	65%	15%	43%	17%
No	COCO 2017	balanced	Adam	110	79%	19%	77%	22%
Yes	COCO 2017	balanced	Adam	110	49%	19%	38%	18%
No	No	balanced	SGD[48]	20	1%	0%	2%	0%
No	COCO 2017	balanced	SGD	10	0%	0%	1%	0%

Table 3: Table reporting all the Faster R-CNN results we obtained

the two images is that in figures 13(a) and 13(b) we used an untrained backbone while in figures 14(a) and 14(b) we used a pre-trained backbone. As we can observe, the model with a pre-trained backbone, in the end, outperformed the untrained one. In fact, if we look at the evaluation on the test set, we can see how the experiment with the pre-trained backbone achieved around 20% for both mAP and IoU, outperforming the untrained one that was able to reach just around 10% for both metrics. However, these results will not be able to correctly classify all the lesions in the dataset. If we compare the evaluation on the train set, we can see how in figure 13(a) at exactly epoch 60 the model is able to achieve over 50% for IoU and over 60% for mAP and this is also visible on the test set in figure 13(b) with, however, a much smaller magnitude. These values are lost after a handful of epochs as is, again, also visible on the test set evaluation. If we focus on the experiment with the pre-trained backbone, we can see that the model was able to learn to differentiate the lesions on the train set at a more constant and steadier pace. However, as already mentioned, the results on the test set are sub-optimal and cannot be employed in a real-life scenario. In fact, when we look at the last epoch, which is again the most promising one, the performances on the test set are around four times weaker than the performances on the train set.

The results showed in figures 15(a) and 15(b) are the ones where we resized the images to 240x240 pixels before training the network. Here we can observe some similarities between this results and the ones in figures 12(b) and 12(a). Both models had troubles learning a rule for most of their run-time and only started learning towards the end. Although the mAP and IoU evaluated on the train set doubled in the last 20 epochs, we can only see a small increase in the same epochs on the test set. Here again, the best epoch is the last one but looking at the results on the train set, both mAP and IoU were lower than 50% while the performances on the test set were around the 10% mark.

At this point we swapped Adam with SGD[48] and tried training the network as before, with both pre-trained and untrained backbones. In figures 16(a) and 16(b) we can find the results of the untrained backbone while in figures 17(a) and 17(b) the results of the pre-trained one.

In all these figures it's easy to observe how the loss was, overall, stationary. It's observable from the IoU how the model wasn't able to learn and detect adequate bounding boxes for both the train and test set. We noted how the SGD optimizer was not suited for this problem and these experimental settings.

In the previous experiment, we were able to notice how all of the results obtained with Adam showed signs of overfitting while the results obtained with SGD weren't able to learn anything.

4.2. YOLO

In table 4, we sum up the results of the yolo experiments.

In figures 18(a) and 18(b) we reported the results of the first experiment with this model. We can observe the same patterns observed before with the Faster R-CNN experiments: The loss ha a decreasing trend while the model is able to learn only to detect and classify the train set. Moreover, we can see that the oscillation of the mAP and IoU on the test set presents the same oscillations as the train set but on a much smaller scale. Here, the loss is met with a more steady descend after just five epochs with just a small spike at epoch seventy. Also, the mAP on the test set seems to be three times weaker than the one on the train set while the IoU seems to be just above the halfway mark.

In figures, 19(a) and 19(b) we reported the evaluation of YOLO trained with a resolution of 869x869 but, during inference, we increased the resolution to 1024x1024. Here we can observe a slight improvement over the previous results. While the mAP is overall the same with only an increase of 2%, the IoU was met with a 10% increase. Although, we have to notice how this last metric was very much fluctuating between 20% and 40% before epoch 70. After this point, the fluctuation persisted but with a much smaller magnitude. On the other hand, the loss stabilized itself almost immediately after only twenty epochs. In this experiment, the mAP on the test set it's four times smaller than the one on the train set while the IoU is half the magnitude of the IoU on the train set.

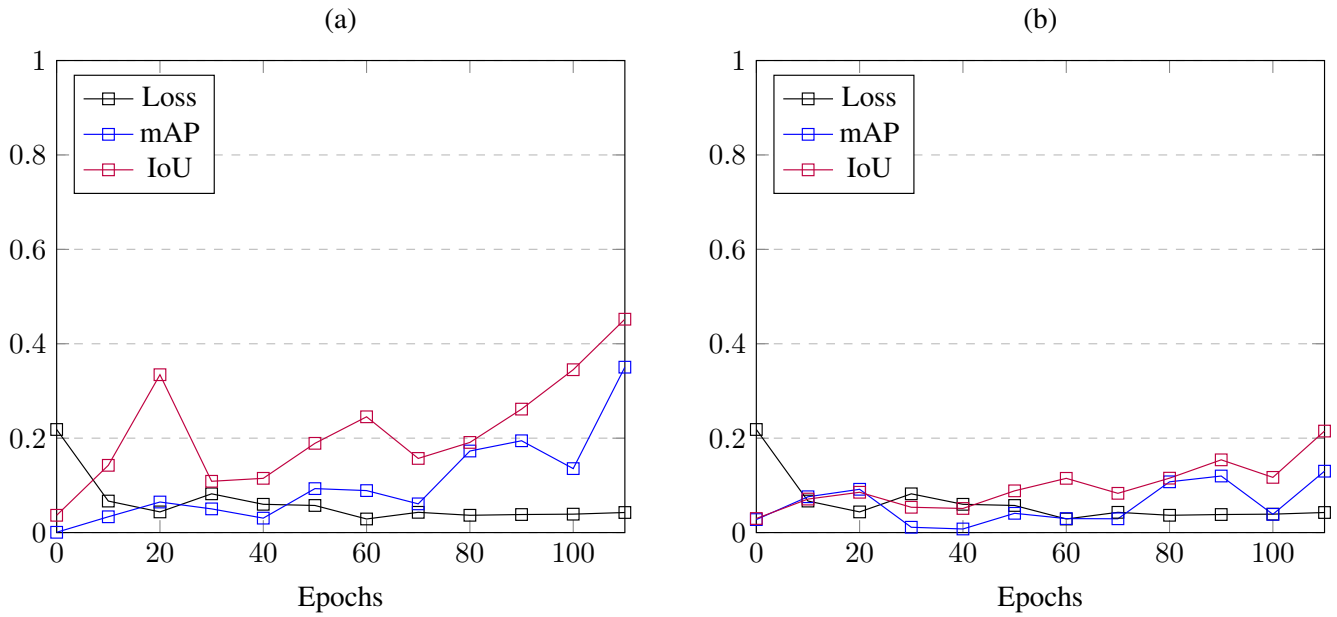


Fig. 12: Faster R-CNN: Results of training using Adam with a learning rate of 0.0001 and a pre-trained backbone. Training was performed on unbalanced batches. (a) is evaluated on the train set while (b) is evaluated on the test set.

Pre-trained	Batches	Training resolution	Testing resolution	Best epoch	mAP train	mAP test	IoU train	IoU test
COCO[40]	balanced	416x416	416x416	140	63%	18%	38%	22%
COCO	balanced	816x816	1024x1024	140	80%	20%	59%	32%

Table 4: Table reporting the results of the two YOLO experiment we ran.

Method	mAP@50	mAP@75
Faster R-CNN[21] + DeepLesion[6]	57%	48%
Faster R-CNN (ours)	15.2%	6.9%
YOLOv4[28] + COCO	65.7%	47.3
YOLOv4 (ours)	16.5%	6.3%

Table 5: Table showing the best results of our experiments and the one reported in the paper

4.3. Comparison

As we can observe from table 5, our best experiments of both Faster R-CNN and YOLO are not able to outperform the mean average precision values reported in their respective papers. Moreover, it seems that our best experiment with YOLO provided better results than our best experiment with Faster R-CNN. We have to note, however, that YOLO trained for a number of images equivalent to 140 epochs while Faster R-CNN to just 110 epochs. If we look at the reported mAP and IoU of epoch 110 of the YOLO experiment in figure 19 (b), we can observe comparable numbers. Our Faster R-CNN achieved 15.2% mAP@50 and 6.9% mAP@75 while our YOLO obtained 16.5% mAP@50 and 6.3% mAP@75. If we look, instead, at the loss of each we can observe how the trend is similar. They both have a constant decreasing trend with minor spikes upwards.

Looking at the precision of the two models we employed, we can see that they have comparable results. YOLO seems to be better at classifying the found object with a 1.4% advantage versus Faster R-CNN if we use an IoU threshold of 50% while this last model performed better with a higher IoU threshold of 75% resulting in a 0.6% increase in precision. This may be due to two opposite factors: firstly, when spatially locating an object, the corresponding feature on the feature map may be too small to survive the locating process; secondly, YOLO uses a grid system to divide the images into cells where the model performs object detection and location. Due to this, the object we want to detect may be smaller than the cell itself and the anchors and therefore the model's head may be overestimating the bounding boxes.

5. DISCUSSION

As we observed in the previous sections, all of the experiments weren't able to provide employable results. Some of

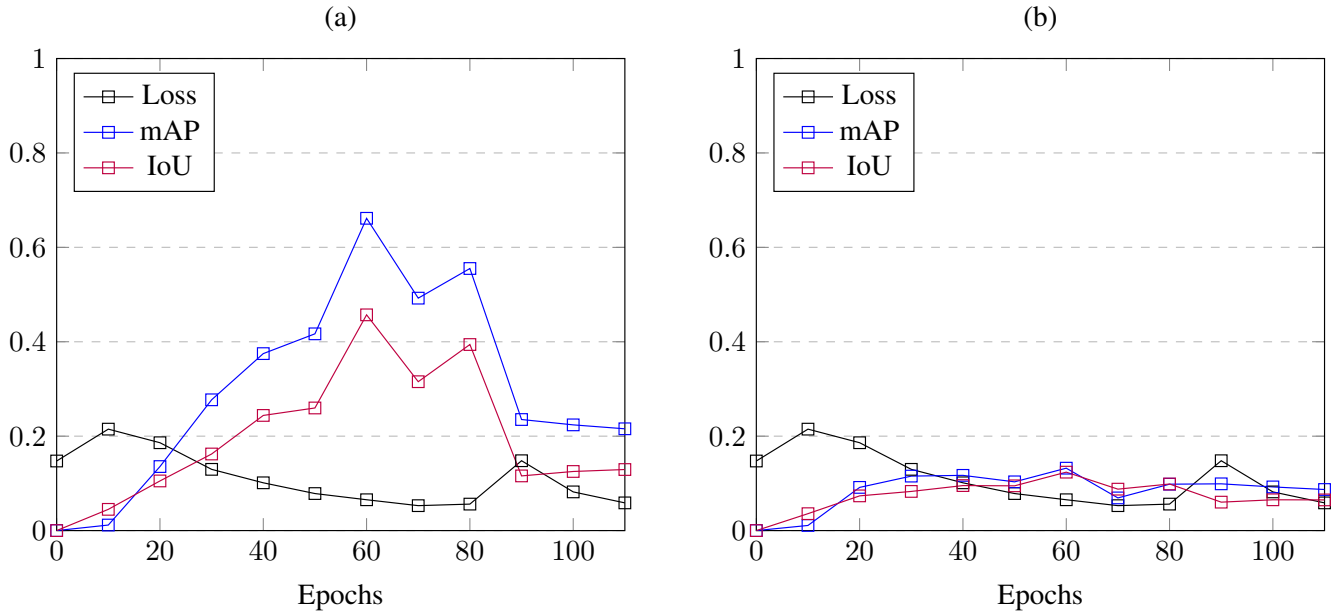


Fig. 13: Faster R-CNN: Result of training using Adam with a learning rate of 0.0001 and an untrained backbone. Training was performed on balanced batches and (a) was evaluated on the train set while (b) on the test set.

the models were able to learn a rule on the train set but failed applying the same rule on the data set. To understand why, we mainly looked at two things: the Universal Lesion Detector presented in the data set paper and the detailed results we obtained,comprising of number of false negatives and test predictions.

First of all, with both Faster R-CNN and YOLO, we noticed that there were a staggering amount of false negatives, around 40% where the model wasn't able to detect any lesion at all. With Faster R-CNN we also observed how in some of the examples the estimation of bounding boxes was problematic: sometimes, as in figure 20, the network was able to correctly locate the lesion while slightly overestimating the bounding box size while other times, as in figure 21, the network failed to locate the issue while predicting a bounding box double the size of the ground truth one.

Although, this was not always the case: in figures 22 and 23 we can observe how the model is able to correctly locate the lesions while predicting pretty tight bounding boxes. These are, however, images from the train set, so this kind of results was expected with these images.

If we look, instead, at inference on images taken from the test set in figures 24 and 25 we can see an example of success and one of missed detection. These images looks normal and, at a first glance, they don't give any insight. If we combine this information with the number of false negatives the model detected, we can provide an explanation. Throughout all the epochs, the inference on the test set was able to only generate bounding box predictions on 40% of the images in the last epochs. After several epochs, if the model is still unable to detect certain features, we could claim it could be an issue with the feature map that gets generated since some features seems to disappear before the later stages suggesting that the size of the lesion is strongly influencing detection accuracy.

Secondly, the model presented in the paper is able to achieve 80% accuracy with a 0.5 IoU threshold. Since the actual neural network they used was a custom Faster R-CNN implementation, we had to look at the modification made to understand the huge gap between our results. One of the changes that we think was fundamental is the removal of the last two pooling layers of the VGG16 backbone. This allowed for smaller features to be identified more easily since a bigger feature map allowed smaller feature to not being pooled with a bigger or stronger feature.

Furthermore, the substandard performances on the test set may also suggest that the train set isn't representative of the data set. This is probably due to the high number of non-detections given the small size of the different lesions. This thesis is also supported by figure 19(b). In fact, as we increased the network size during both training and detection, therefor allowing smaller features to be brought forward with shortcut connections , we observed how the mAP and the IoU were able to achieve a more constant increase and better performances. The IoU, however, also presented a more noisy behaviour since it was moving around between 20% and 40%.

If we dwell into more detailed results, we can study the confusion matrix relative to the last epoch of our best experiment, the one with Adam and the pre-trained backbone. In fact, in figure 26(a) we can observe the confusion matrix w.r.t. the last epoch generated on the train set, while in figure 26(b)

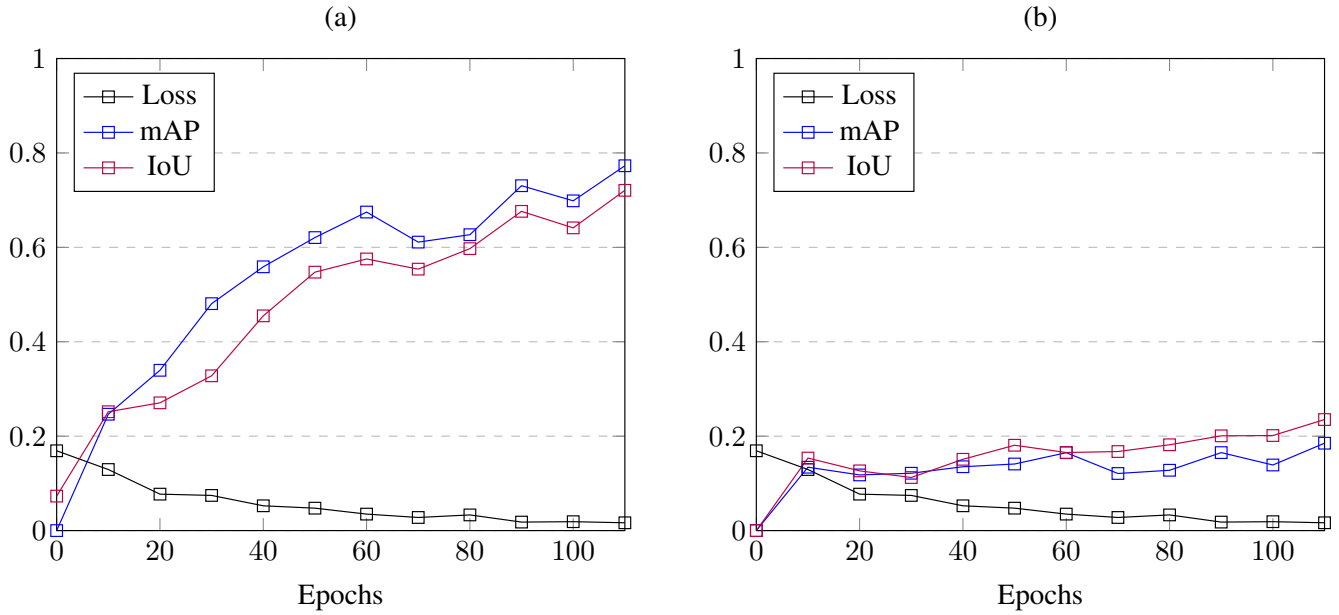


Fig. 14: Faster R-CNN: Results of training using Adam with a learning rate of 0.0001 and a pre-trained backbone. Training was performed with balanced batches and (a) was evaluated on the train set while (b) was evaluated on the test set.

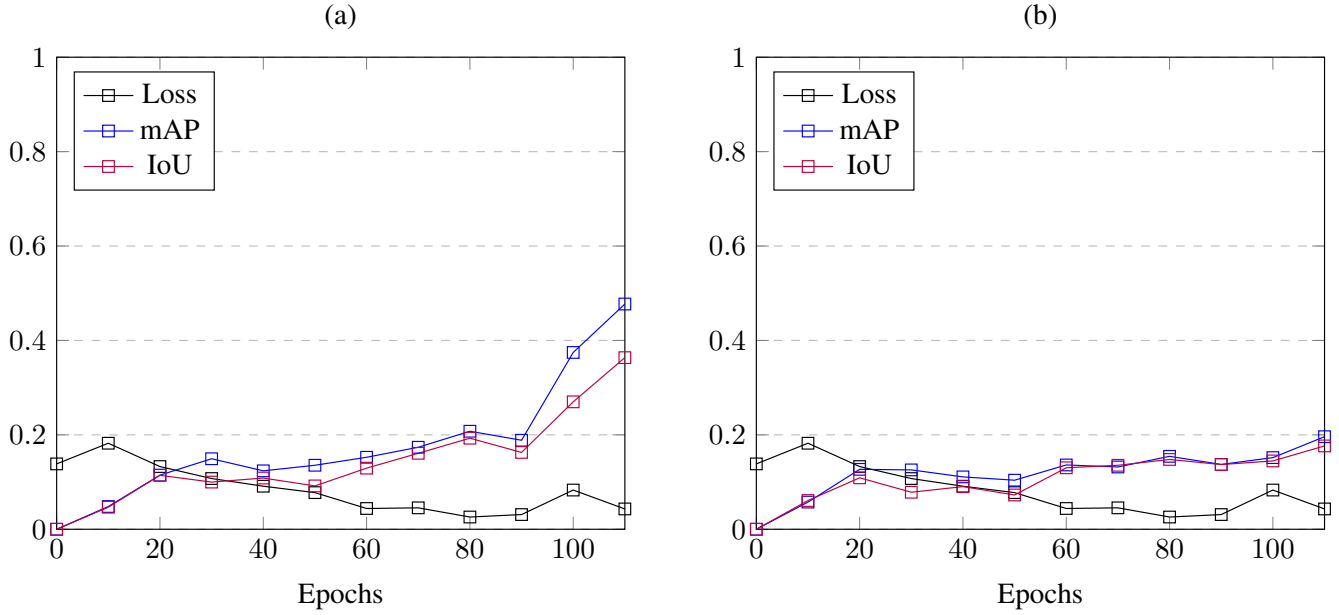


Fig. 15: Faster R-CNN: Results of training using Adam with a learning rate of 0.0001 and a pre-trained backbone. During training, images were resized to 240x240px and data augmentation was used. Trained on balanced batched. (a) was evaluated on the train set while (b) was evaluated on the test set.

we report the confusion matrix generated on the test set. We can clearly notice how the diagonal on the train set looks more sharp. Moreover, If we compare the distribution on the diagonal with the distribution of classes in figure 2 we can see a resemblance: the classes displayed as 1 and 4 in figure 26(a) are the ones with the most elements followed by class 2 and three. This is not also present in figure 26(b) where class 1 has around 50% of the elements of class 4 instead of being fairly similar. On the other hand, the confusion matrix evaluated on the test set doesn't look that sharp at all. The classes

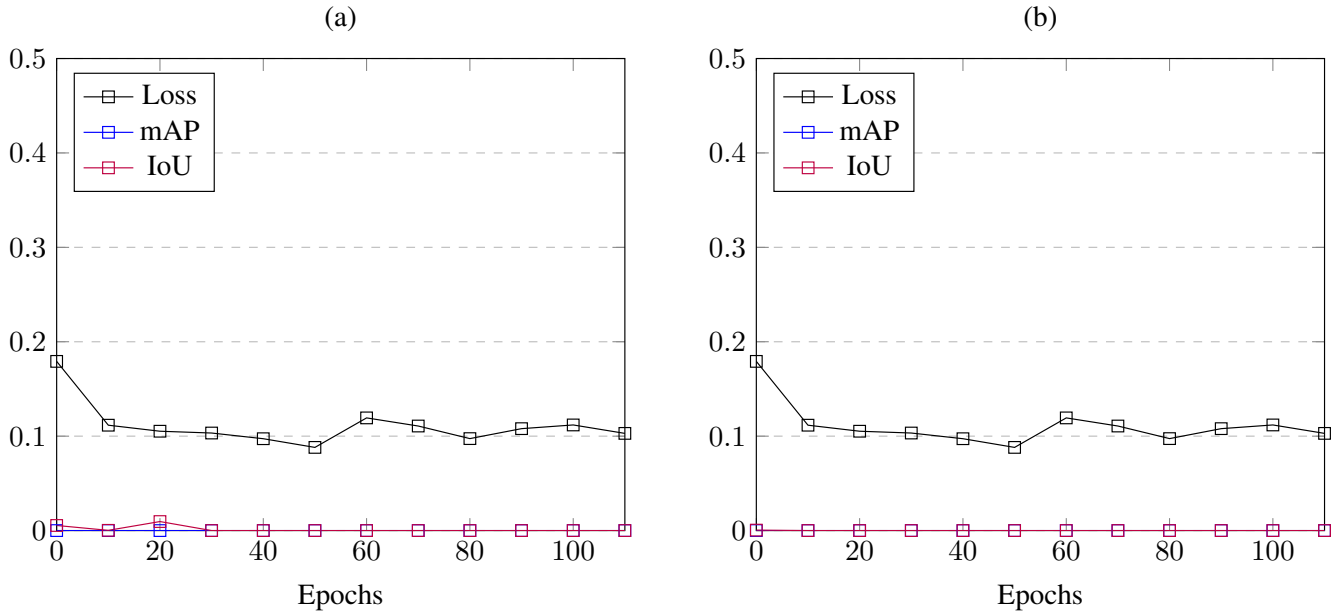


Fig. 16: Faster R-CNN: Result of training using SGD with a learning rate of 0.00002 and an untrained backbone. Training was performed on balanced batched and (a) was evaluated on the train set while (b) was evaluated on the test set.

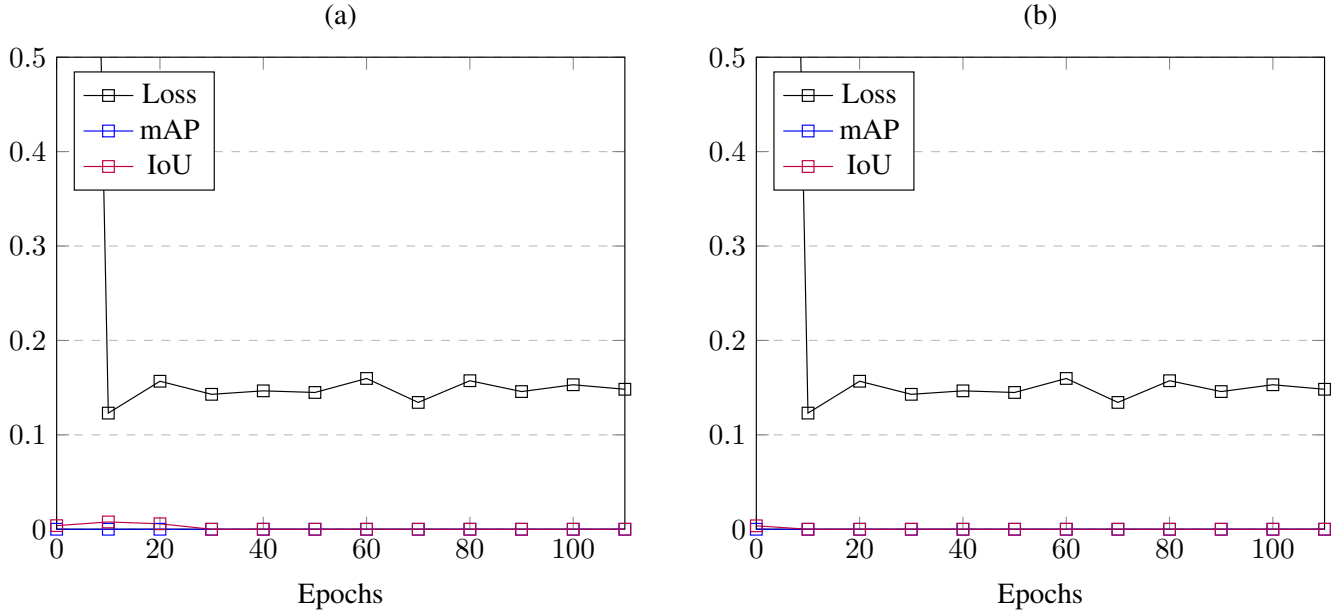


Fig. 17: Faster R-CNN: Result of training using SGD with a learning rate of 0.00002 and a pre-trained backbone. Training was performed on balanced batches. (a) was evaluated on the train set while (b) was evaluated on the test set.

that are more difficult to detect are 0 and 5 since those are the ones with the fewest elements and less elements imply less data to utilize for learning. In fact, if we look closely we can see how class 6 commits very few errors and therefore we can say that the model has troubles finding this class at all. On the other hand, if we look at class 0 we can see that half that class gets correctly recognized but the other half gets mistaken as class 4, which is a clear sign of overfitting on one of the most predominant class. We can also understand why the model overfits on this class instead of the other most predominant classes: class 4 contains lung lesions while the other classes represents mediastinum, liver and abdomen lesions. All three

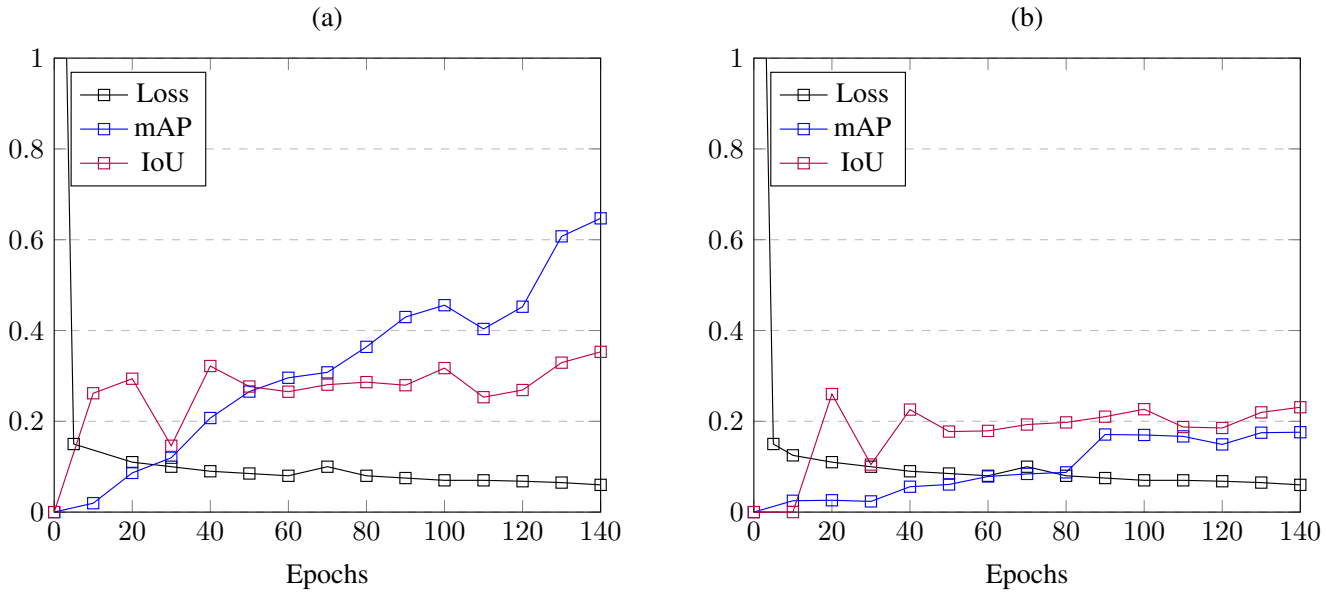


Fig. 18: YOLOv4: 416x416 with standard anchors. Result of training with a learning rate of 0.001 on balanced batches. (a) was evaluated on the train set while (b) was evaluated on the test set.

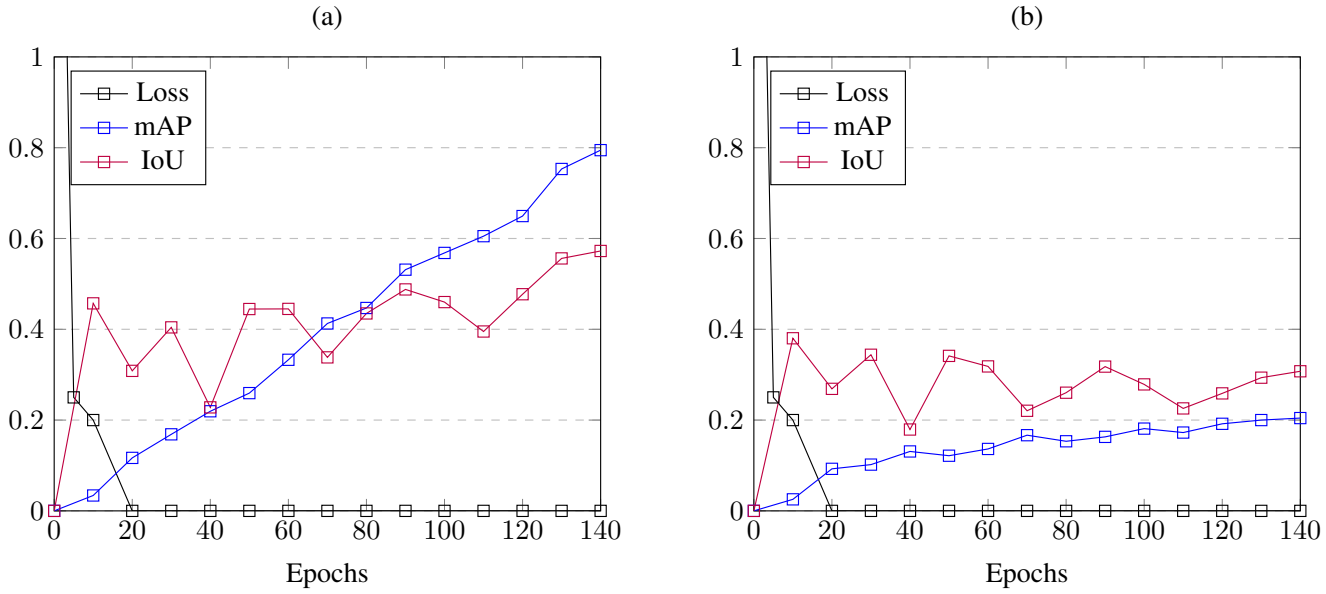


Fig. 19: YOLOv4: 869x869 with custom anchors. Result of training with a learning rate of 0.001 on balanced batches. (a) was evaluated on the train set while (b) was evaluated on the test set.

kind of lesions are smaller than lung lesions since the mediastinum and liver are organs way smaller than the lungs and abdomen lesions are small and sparse lesions. The difference in size of the lesions, combined with the issue our model has with small annotations, makes smaller lesions harder to detect and learn showing the signs of overfitting.

We also investigated the size of the bounding boxes that were predicted as displayed in table 6. As we can observe, the model mostly predicted smaller bounding boxes with a 53.92% rate versus the 46.78% of bigger bounding box predictions. However, smaller bounding box were just 33% smaller on average meanwhile the bigger bounding boxes were two times as big with the biggest being forty-one times bigger while the smallest was only around twenty times smaller.

If we look at the predictions obtained with Faster R-

	Percentage of predictions	Mean ratio	Median ratio	Minimum ratio	Maximum ratio
Bigger	46.78%	2.09	1.31	1.0003	41.48
Smaller	53.92%	0.67	0.74	0.056	0.99

Table 6: Table reporting information on the size of the bounding box predicted with YOLO[28] w.r.t. the ground truth boxes. These values were obtained by first diving the area of each predicted bounding box with the corresponding ground truth box.

	Percentage of predictions	Mean ratio	Median ratio	Minimum ratio	Maximum ratio
Bigger	60.84%	3.68	1.77	1.002	54.09
Smaller	39.15%	0.58	0.61	0.007	0.99

Table 7: Table reporting information on the size of the predicted bounding box with Faster R-CNN[21] w.r.t. the ground truth boxes. These values were obtained by first diving the area of each predicted bounding box with the corresponding ground truth box.

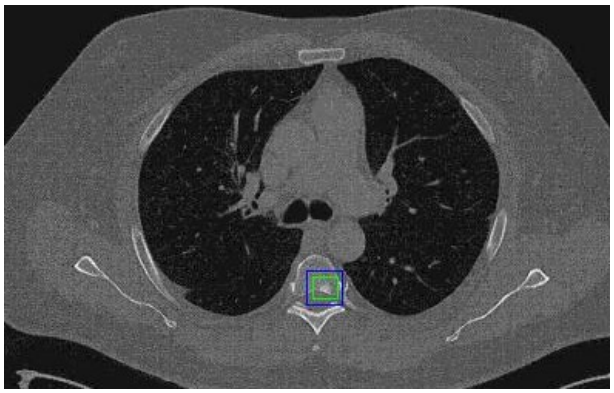


Fig. 20: Example of larger bounding box generation on a train set image. The model in question was Faster R-CNN with Adam as optimizer and a pre-trained backbone.

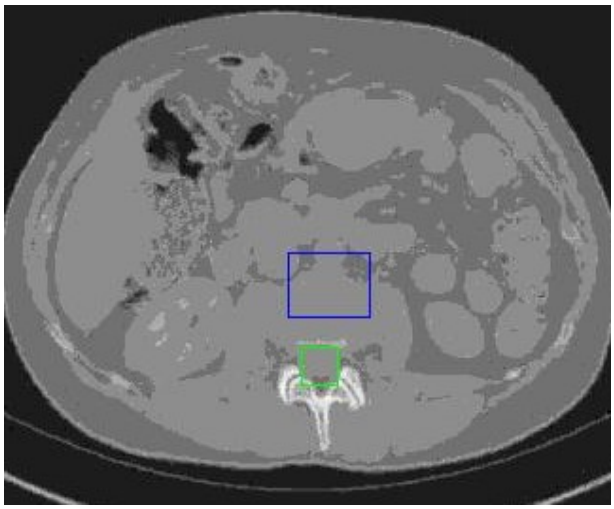


Fig. 21: Example of larger bound box generation on a test set image. The model in question was Faster R-CNN with Adam as optimizer and a pre-trained backbone.

CNN[21] we can observe a small decrease in performances. Faster R-CNN was able to obtain more prediction but a vast majority consisted of false negatives. This is also evident in table 7. In fact, of all the predictions made by Faster R-CNN, 60.84% were bigger than the ground truth one. We encounter also an increase in the average bounding box that results to be bigger more than three times and a half while the biggest prediction is fifty-four times bigger. The values relatives to the proposals that were smaller than the ground truth are mostly comparable except for the smallest prediction which is now around ten times smaller than the smallest YOLO prediction.

We can suppose that YOLO obtained better results due to the changes the authors made with YOLOv3[29]. As discussed also previously, the shortcut connections allows smaller object, and therefore their ground truth boxes, to appear in a later stages feature map. Since this is not present in our Faster R-CNN implementation, the smallest object our models find are, therefore, bigger features that present bigger bounding boxes.

6. CONCLUSION

We discussed the technologies chosen for this task, Faster R-CNN and YOLO, and the DeepLesion data set that has been the focus of a series of experiments. We have tried to get closer to the results obtained in [6] with their Universal Lesion Detector without modifying the internal structure of the network but only by adjusting some parameters.

We have observed how this data set contains its own unique challenges, such as presenting very small lesions and being heavily imbalanced with bone lesions accounting for slightly above 1% of total lesion while lung lesions were around 25% of the whole data set.

While we were not able to recreate the same results the authors obtained in the paper due to the uniqueness of this

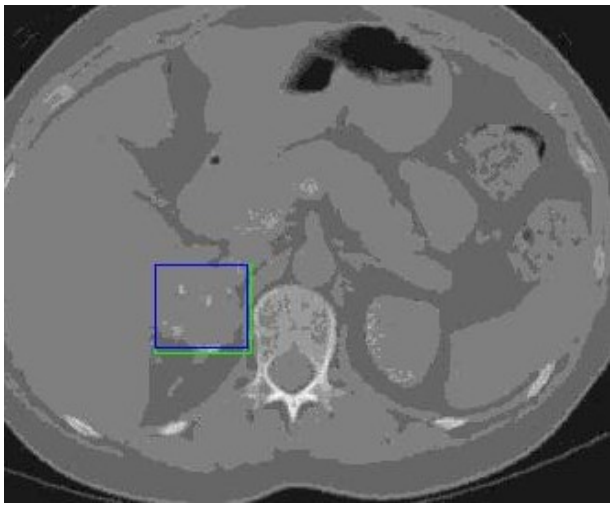


Fig. 22: Example of correct detection and classification on a train set image. Referenced experiment was Faster R-CNN with Adam optimizer and pre-trained backbone

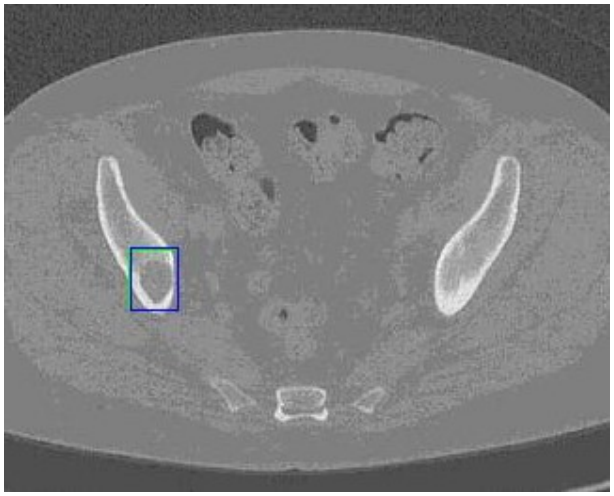


Fig. 23: Example of correct detection and classification on a train set image. Referenced experiment was Faster R-CNN with Adam optimizer and pre-trained backbone

data set, we were able to provide a plausible explanation of what were the main issue met during this research experiment.

If we had more time on our hands, we would have tried running another set of experiments with YOLO increasing ever more the network resolution and observing if in the results the amount of false negatives would decrease or not. Also, we also wanted to try and swap the Faster R-CNN backbone from ResNet50 to the original VGG16 without the last two pooling layers, as mentioned in the data set paper. With this last experiment, we would have wanted to observe what impact the pooling layers have on the lesions detection.

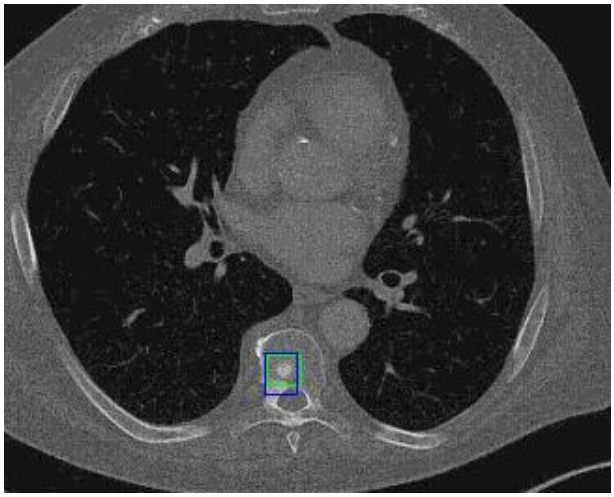


Fig. 24: Example of correct detection and classification on a test set image. Referenced experiment was Faster R-CNN with Adam optimizer and pre-trained backbone

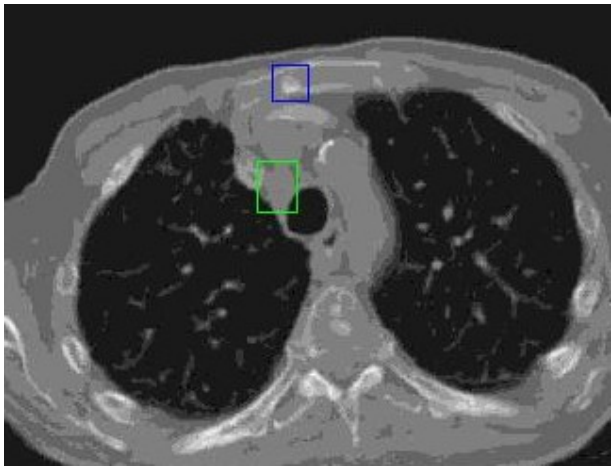


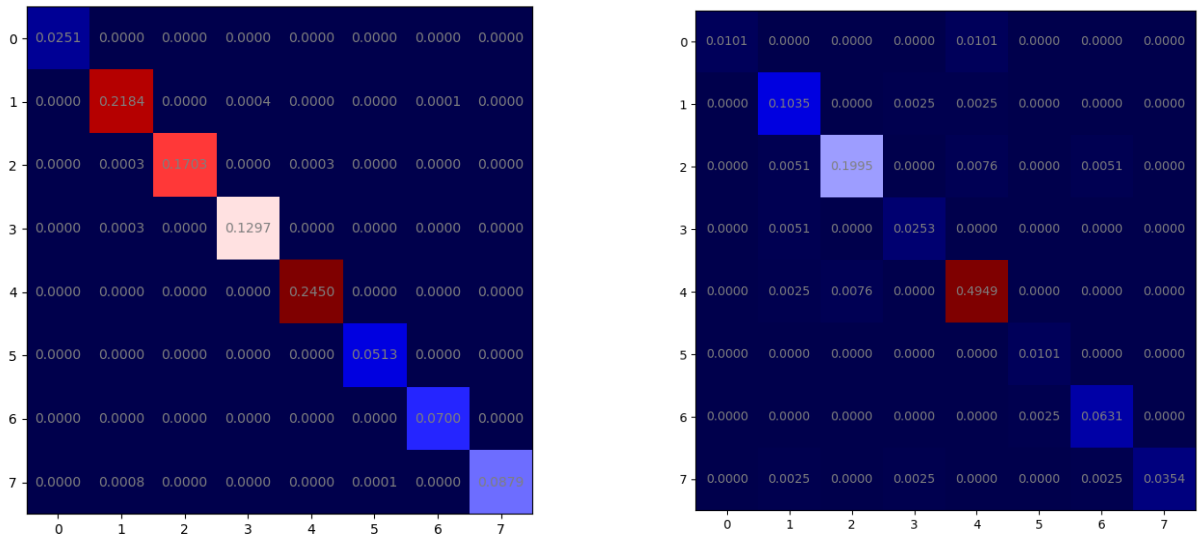
Fig. 25: Example of correct detection and classification on a test set image. Referenced experiment was Faster R-CNN with Adam optimizer and pre-trained backbone

7. REFERENCES

[1] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

[2] Stamatia Dasiopoulou, Vasileios Mezaris, Ioannis Kompatsiaris, V-K Papastathis, and Michael G Strintzis, "Knowledge-assisted semantic video object detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 10, pp. 1210–1224, 2005.

[3] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat, "A



(a) Train set confusion

(b) Test set confusion

Fig. 26: Confusion matrix w.r.t Faster R-CNN trained with a pre-trained backbone, Adam and balanced batches.

survey on instance segmentation: state of the art," *International Journal of Multimedia Information Retrieval*, pp. 1–19, 2020.

[4] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[5] Zhuoling Li, Minghui Dong, Shiping Wen, Xiang Hu, Pan Zhou, and Zhigang Zeng, "Clu-cnns: Object detection for medical images," *Neurocomputing*, vol. 350, pp. 53 – 59, 2019.

[6] Ke Yan, Xiaosong Wang, Le Lu, and Ronald M. Summers, "DeepLesion: automated mining of large-scale lesion annotations and universal lesion detection with deep learning," *Journal of Medical Imaging*, vol. 5, no. 3, pp. 1 – 11, 2018.

[7] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2097–2106.

[8] Yoav Freund and Robert E Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119 – 139, 1997.

[9] Yoav Freund, "Data filtering and distribution modeling algorithms for maching learning (ph. d. thesis)," 1993.

[10] Robert E Schapire, "The strength of weak learnability," *Machine learning*, vol. 5, no. 2, pp. 197–227, 1990.

[11] Paul Viola and Michael Jones, "Rapid object detection using a boosted cascade of simple features," 2001.

[12] David G Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*. Ieee, 1999, vol. 2, pp. 1150–1157.

[13] B. Sirmacek and C. Unsalan, "Urban-area and building detection using sift keypoints and graph theory," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 4, pp. 1156–1167, 2009.

[14] Robert K. McConnell, "Us4567610a," `https://patents.google.com/patent/US4567610`, 1986.

[15] Navneet Dalal and Bill Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. IEEE, 2005, vol. 1, pp. 886–893.

[16] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[17] G. Azzopardi and N. Petkov, "Trainable cosfire filters for keypoint detection and pattern recognition," *IEEE*

Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 2, pp. 490–503, 2013.

[18] John G Daugman, "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters," *JOSA A*, vol. 2, no. 7, pp. 1160–1169, 1985.

[19] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[20] Koen EA Van de Sande, Jasper RR Uijlings, Theo Gevers, and Arnold WM Smeulders, "Segmentation as selective search for object recognition," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 1879–1886.

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.

[22] Ross Girshick, "Fast r-cnn," 2015.

[23] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[24] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[25] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *Advances in neural information processing systems*, 2012, pp. 2843–2851.

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016.

[28] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[29] Joseph Redmon and Ali Farhadi, "Yolov3: An incremental improvement," 2018.

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," 2015.

[31] Joseph Redmon and Ali Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[32] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 390–391.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[34] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia, "Path aggregation network for instance segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[35] Marc D Kohli, Ronald M Summers, and J Raymond Geis, "Medical image data and datasets in the era of machine learning—whitepaper from the 2016 c-mimi meeting dataset session," *Journal of digital imaging*, vol. 30, no. 4, pp. 392–399, 2017.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "Imagenet large scale visual recognition challenge," 2014.

[37] "Drive: Digital retinal images for vessel extraction," .

[38] Xingyi Yang, Xuehai He, Jinyu Zhao, Yichen Zhang, Shanghang Zhang, and Pengtao Xie, "Covid-ct-dataset: A ct scan dataset about covid-19," 2020.

[39] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman, "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, pp. 303–338, June 2010.

[40] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár, "Microsoft coco: Common objects in context," 2014.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[42] Josef Sivic and Andrew Zisserman, "Video google: A text retrieval approach to object matching in videos," in *null*. IEEE, 2003, p. 1470.

[43] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[44] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[45] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren, "Distance-iou loss: Faster and better learning for bounding box regression.," in *AAAI*, 2020, pp. 12993–13000.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds., pp. 8024–8035. Curran Associates, Inc., 2019.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[48] Herbert E. Robbins, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 2007.

[49] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," 2014.

[50] John Duchi, Elad Hazan, and Yoram Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011.

[51] Alex Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.