Adapting RetinaNet Architecture for Detection of Rib Fractures in Pediatric X-Rays

Nick Nagy

Introduction

The Complications with Rib Fracture Detection in Underage Patients

Rib fractures are often subtle: they are typically small in scale, and in a pediatric X-ray they can look less like a crack and more like a swelling along a rib. When searching for rib fractures, even a highly-skilled pediatric radiologist may experience difficulties locating every instance. It can especially pose a challenge for general doctors.

Unfortunately, it is a common occurrence for children raised by physically abusive adults to experience broken bones, without any verbal expressions of concern from anybody in the family: in fact, the positive predictive value of abuse given rib fractures is 95% for children under the age of three [1]. Most doctors who have familiarity with rib fractures are used to searching for them in adult X-Rays; child X-Rays are typically brought up in unrelated medical procedures, where a doctor might not even be searching for any signs of damaged bone. Consequently, a lot of fractured ribs go entirely undetected, especially in children.

This is the motivation behind completely automating the process of detecting rib fractures in any child or adult X-ray. Thanks to pediatricians we have been provided a database of anonymous X-rays from patients, each expertly labeled for any existing rib fractures. Our goal is to train a neural network to immediately identify new fractures with the same accuracy as any pediatric radiologist.

Other Examples of AI and Detection in the Medical Field

The usefulness of artificial intelligence has long been recognized with respect to medical application. A common field where this is observed is robotics: some surgical robots have shown to perform complicated operations with higher precision than highly trained surgeons [2].

Methods of recognition similar to our proposal have been applied to areas like disease detection. In 2017, Stanford published their work on a classification network trained to predict skin cancer in patients [3]. In 2018 the Australian Institute of Machine Learning from the University of Adelaide published their research on a device that rapidly scans each region in a breast exam until it finds an abnormality [4].

What is interesting about both examples is that the results showed the machines could perform their tasks faster than medical experts, and no less accurately. Achievements like these demonstrate how effective machines can be for handling complicated medical tasks.

Convolutional Neural Networks for Object Detection: YOLO Versus SSD

There are three common types of convolutional network problems. Classification networks are used to predict whether an object exists in a given image. Segmentation takes an image and isolates objects of interest from their background. Detection predicts where an object is by outputting coordinates defining a rectangular region that encompasses the object, as well as predicting what the object is: this is the type of network we use. For the purposes of our goal, we are only looking to identify fracture objects from the background.

One object detection network called You Only Look Once (YOLO) outputs a single prediction head, where each index in the first two dimensions represents a region of the input image. In turn, each of these indexes has associated bounding box and class predictions. Each box prediction has an associated scalar representing how confident the network is that that particular box encompasses any object at all. This approach of decomposing the image into regions helps minimize search space and maximize data, as any object could appear in any of these regions.

A trained YOLO network is impressively accurate on images with sparse object appearances; unfortunately, it tests poorly on images where many objects are in close proximity, especially cases where objects are overlapping. Intuitively, this trade off is not favorable for an X-ray where multiple small fractures appear close together.

A similar but more advanced approach is called Single Shot Detection (SSD): it is built off the structure of another pre-trained "backbone" convolutional network, where a set of convolutional layers at different pooling levels are used to form a feature pyramid network (FPN). Each of these pyramid layers has two outputs, one for box predictions and one for class predictions; these outputs are similar to the single output head in the YOLO architecture. Pyramid layers that are extracted earlier from the backbone network are higher resolution, but because they have been convolved very few times they hold relatively little semantic information. Inversely, low resolution pyramid layers have been convolved many times and represent complex features. The intuition behind FPNs is that a network can take advantage of the varying levels of semantic value and resolution to make many predictions.

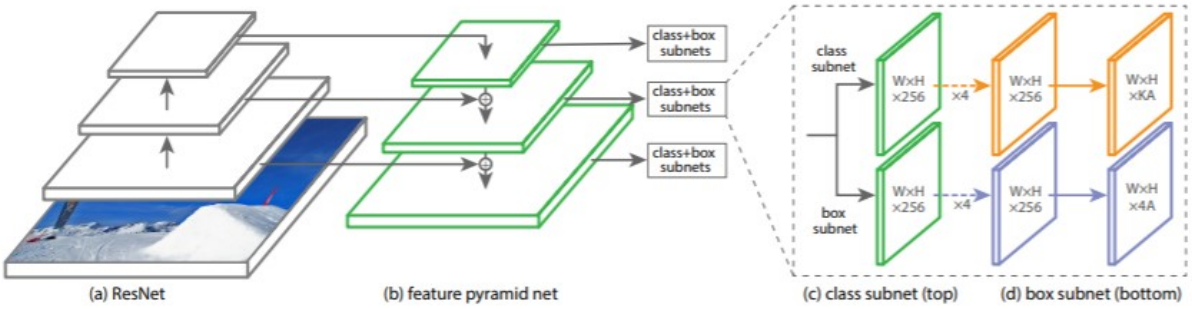

Fig. 1. A visual demonstration of single shot detection [5]. Feature pyramids are created by convolving and summing layers of a backbone network with each other (a and b). Each pyramid layer has its own regression and classification subnetwork and head(s) similar to the singular output of YOLO (c).

Box predictions will be made on each pyramid layer but, obviously, the pyramids are all different sizes from each other and the input image. Box prediction coordinates are normalized to match the dimensions of the layer they are predicted in, such that a box that takes up its entire image would have a width and height both of 1.0. This is to avoid the problem of exploding gradients.

Additionally, SSD uses the idea of "anchors", or fixed points in the images that dictate the size and placement of initialized anchor boxes. The coordinates of the anchors and anchor boxes are not updated during training – they just serve as a starting point for prediction boxes.

By splitting pyramid layers by different sets of anchors, we get many different anchor boxes throughout the image space, of varying areas and width-height ratios. Most early prediction boxes will overlap with each one another. SSD uses non-maximum suppression where, as the name suggests, for a series of overlapping prediction boxes it trains on the prediction that most overlaps a truth box; all other prediction boxes are discarded.

The regression loss is calculated by applying smooth-L1 loss,

$$L_{1,smooth}(\alpha) = \begin{cases} \frac{(\sigma\alpha)^2}{2}, |\alpha| < \frac{1}{\sigma^2} \\ |\alpha| - \frac{1}{2\sigma^2}, \quad otherwise \end{cases}$$

on the difference between prediction and truth box coordinates. Traditionally, the classification loss used by SSD was softmax-cross entropy:

$$C(p, y) = -\sum_{i}^{N} y_i \ln(p_i)$$

RetinaNet

RetinaNet is an improved-upon implementation of a single-shot detection network. It is called such because it introduces a classification loss function dubbed "focal" loss:

$$C(p, y) = -\sum_{i}^{N} y_i \ln(1 - p_i)^{\gamma} \ln(p_i)$$

It is used to reduce the adverse effects of class imbalances in data, not just between different classes, but between classes and background (the absence of a class). It reduces the amount by which a high-confidence prediction p_i attributes for the total loss, and boosts the accountability of low-confidence predictions.

Methods

Adapting RetinaNet for Rib Fracture Detection

We use a Keras implementation of RetinaNet [6]. The repository includes a selection of pre-trained backbone networks for which the FPN can be built off from. For the time being, we use the default choice, the 50-layer Residual Net [7].

Between each training iteration, the classification and regression loss, as well as the sum of both losses, are returned. A snapshot of the training model is saved after every epoch: this makes restarting from checkpoints in training runs very flexible. By default, a validation step returns a value of mean average precision (mAP) after every training epoch. We compare the change in mAP over epochs, as well as the results in some of our own implemented metrics (see the section "Metrics" below).

As a default, the publishers of this implementation use $\sigma = 3.0$ for calculating smooth-L1 loss. For the time being, we leave this hyperparameter unchanged.

Data

As the majority part of our data we have 114 X-rays with fractures labeled by pediatricians. The images are stored in MatLab files with a table listing the fractures by coordinates [x, y, box width, box height], where x and y define the top left corner of the bounding box.

The Keras implementation of RetinaNet uses CSV files for pre-processing data. Its default expected format for each line of the CSV is [path to the image, x1, y1, x2, y2, class], where (x1, y1) and (x2, y2) define the top left and bottom right coordinates of the bounding box, respectively. We use a custom Python script to iterate through each fracture folder; save the MatLab image as a .jpg in the same folder; convert each set of [x, y, width, height] coordinates in the MatLab table to [x1, y1, x2, y2]; and write the coordinates and path of the image to a new line in a CSV. Each image is pseudo-randomly assigned to either a training, validation or testing CSV, where its probability of being saved to each are 0.7, 0.15 and 0.15. To help confirm the network's true positive accuracy, 75 X-rays with no fractures present are added to the validation set.

One of the limitations we face is the size of our dataset. The Stanford team used nearly 130,000 training images in their research. To accommodate for the lack of raw data, we also apply randomly generated data augmentations during training, such as cropping or small-angle rotations. The Keras implementation includes methods for applying random transformations to an image and its truth boxes during training. We limited ourselves to the following random transformations: a maximum rotation of 15 degrees; a minimum and maximum scaling factor of 0.9 and 1.1, respectively; and a 0.5 probability of flipping the image about the x-axis. Additionally, we wrote our own method for cropping an image to 0.9 its original size. The coordinates of the truth boxes are also updated after cropping.

The image could be cropped in nine different ways, each with equal probability, based on a randomly generated integer. These regions are defined with respect to the top left corner, middle-left region, bottom left corner, top-middle region, central region, bottom-middle region, top right corner, middle-right region and bottom-right corner. There is also an equal probability that the image will not be cropped at all. During training, we regulate our algorithm to retain 0.9 of the original image post-cropping.

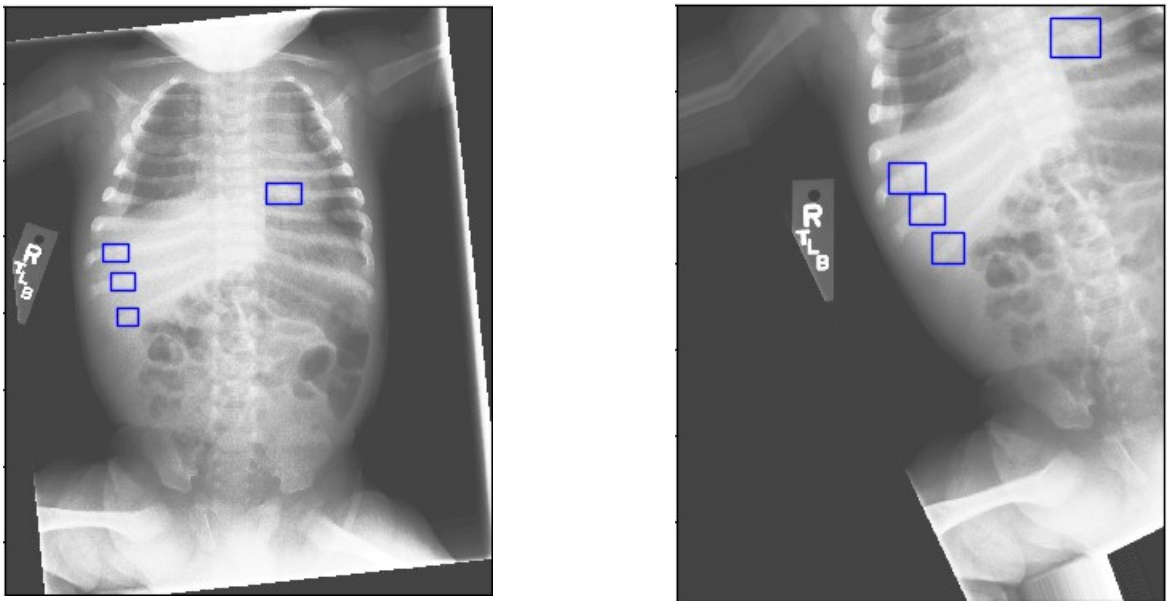

Fig. 2. (left) a raw training image with its truth boxes plotted in blue. (right) augmented image with adjusted box coordinates. The augmentations applied are in order: an initial counter-clockwise rotation of 20 degrees, followed by a scaling of 1.1, and a cropping about all edges retaining 0.9 of the image.

Metrics

Considering we want our prediction boxes to perfectly overlap with our truth boxes, an important metric is intersection-over-union. It is defined as the area of intersection between both boxes divided by the area of the union, such that no overlap returns 0 and complete overlap returns 1. This metric is also used to calculate the model's regression loss.

We choose to evaluate the success of our model by mean recall and mean precision, which represent how frequently truth boxes are found by the network and how frequently predictions are correct, respectively. More specifically, recall is equal to the total number of true positives over all true positives and false negatives; precision is the total true positives over all positives.

We define our true positives to be the set of truth boxes that have an intersection-over-union (above a threshold), with a prediction box above a confidence score threshold. Once a prediction box that has been matched with a truth box is found, that truth box is suppressed so that we do not count for multiples of the same true positives. As we vary the score threshold, the relationship between the change in both metrics is approximately inverse on our dataset, as can be observed in the figure below.

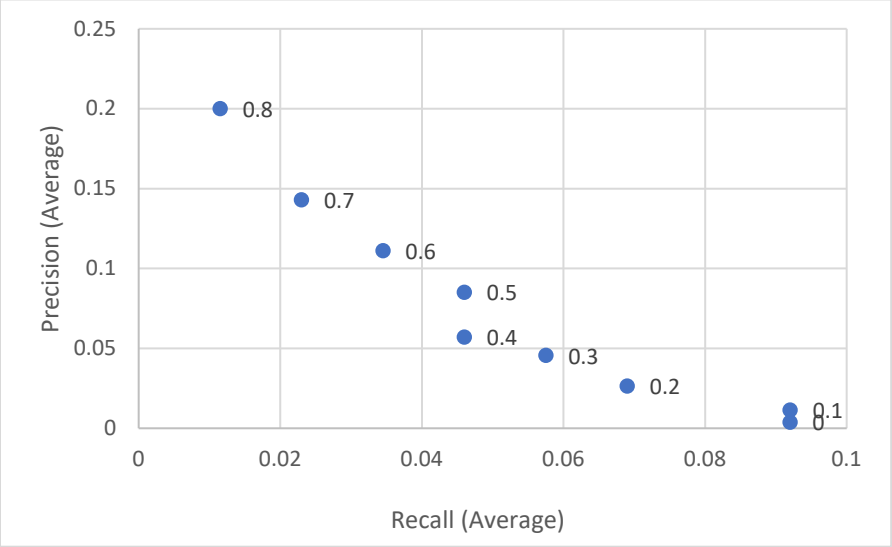

Fig. 3. Positive sensitivity vs positive predictive value on our validation dataset after 10 epochs of training with a learning rate of 0.0001. Each point is labeled with the confidence score threshold. The values were computed by summing across all data before division.

During training, a measure of mean average precision (mAP) on the validation dataset is also returned. Mean average precision is closely related to the area under the cumulative recall vs precision curve when traversing the prediction data in descending order of confidence.

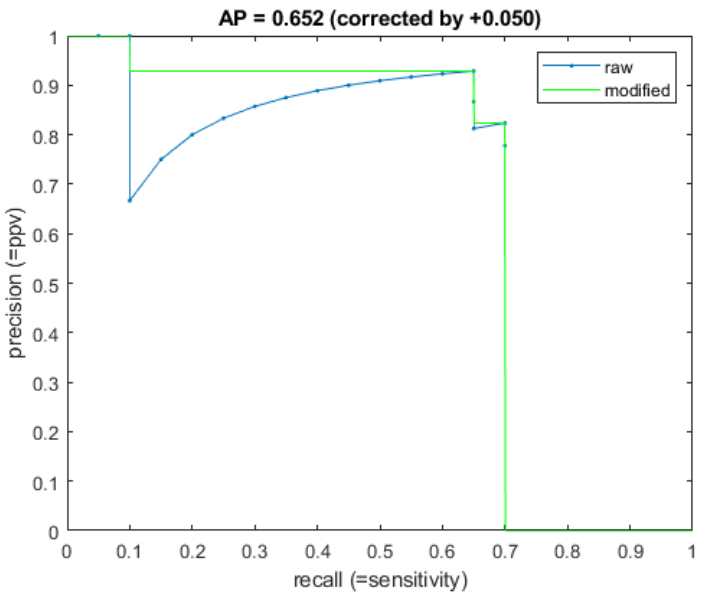

Fig. 4. example of mAP calculated from a cumulative precision vs recall curve. Vertical drops along the blue curve represent the introduction of a false positive; all other steps are related to the addition of a true positive. mAP is the area under the green approximation. The figure was generated by Mike Bindschadler.

Mean average precision is aversely affected by large numbers of false positives, especially false positives with high confidence scores. It is also bounded by the overall mean recall of the data (as can be observed in the Fig. 4), such that it cannot be higher than the recall. This makes it a useful metric for quickly summarizing the network's ability to *confidently* predict bounding boxes in correct locations during run time.

Our validation dataset is comprised of both X-Rays with fractures present and X-Rays with no fractures present. Usually, these metrics would each be calculated across all data. It is important to note that when we calculate our recall and precision, we are only calculating them on images with fractures – this does not affect the recall, but does mean our calculated precision score is higher than the true precision.

Confirming Results Using Image Plots

Because we are able to retrieve the prediction boxes and associated confidence scores from an inference model after feeding an image, we can easily plot the truth and prediction boxes directly on the original X-Ray image to verify the model's accuracy. During our metric evaluations, we save each image with plotted truth boxes, and all prediction boxes that are above the assigned confidence threshold. Truth boxes are displayed in blue. Prediction boxes above our IoU threshold are displayed in green, and below are displayed in red.

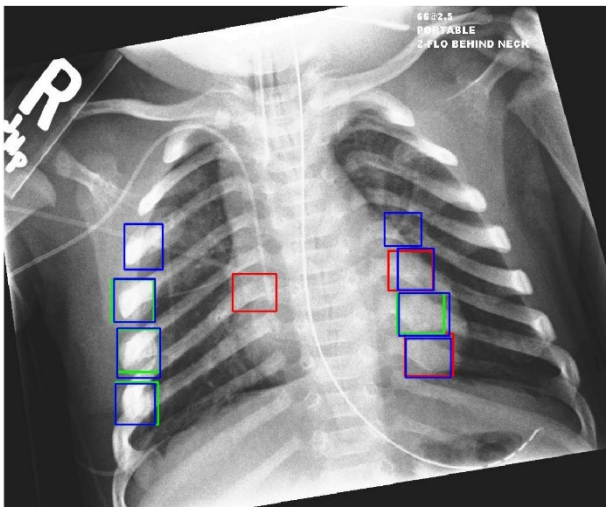

Fig. 5. A visual evaluation of our model on an image. Truth boxes are displayed in blue. Prediction boxes are displayed if they are above assigned a confidence score above 0.6: they are shown in green if their IoU with a truth box is above 0.8, and in red otherwise.

Performance on Individual Augmented Images During Training

Because we apply random data augmentation during training, we are also interested in having visual confirmation of how the network performs on augmented images, as well as a quantitative measurement of the performance. For example, we may discover with further probing that our network can generalize well to X-Rays of various dimensions, but may predict poorly on ribcages captured at skewed angles.

Whereas with most of our evaluations, this is checked on our training dataset rather than our validation set. Unfortunately, Keras only returns mean losses per epoch, rather than loss metrics per image. Alternatively, we have access to the methods the authors used to apply augmentations to each image and its boxes. Though batch losses are not returned, they are printed to the console, and we can quickly observe a particular batch that ran poorly. By leaving our training data un-shuffled, we then have an implicit direct mapping between each image and its transformation, after which we can re-apply single loss or accuracy captures on the transformed result from a saved snapshot.

Our proposed approach to this mapping is for each epoch, at every iteration we write the transformation matrix to a new line of a CSV. Saving the transformed image and boxes themselves take up a lot of unnecessary space. We can instead re-load a line from the epoch CSV and a raw image and its coordinates from our training CSV, and then re-compute the transformation and return a desired metric.

A potentially minor consequence of this implementation is that by traversing the training data in a linear fashion each epoch, we lose some of the benefit of flexible error landscapes that come from continually shuffling data. Our intuition is that we are accommodating for this by randomly transforming the data in new ways with each epoch.

Hyperparameter Optimization

The original authors of RetinaNet began training with a learning rate of 0.01. To determine our ideal choice of learning rate for our problem, we perform a learning rate sweep, training the network for 10 epochs each run with starting with a learning rate of 0.1, then dropping by a factor of 10 until promising results diminish. We judge the performance of each learning rate by converting the training model to an inference model, and running the evaluation metrics on each image in the training and validation sets.

Although while training we have random augmentations applied to the training data, we use our raw dataset to calculate precision and recall scores while selecting a learning rate.

Because precision and recall measurements have no effect on the loss function and therefore no impact on the learning process itself, we chose fairly restrictive thresholds for our metrics. For all results we assign our IoU threshold and confidence score threshold to be 0.8 and 0.6, respectively.

Results

Learning Rate Sweep

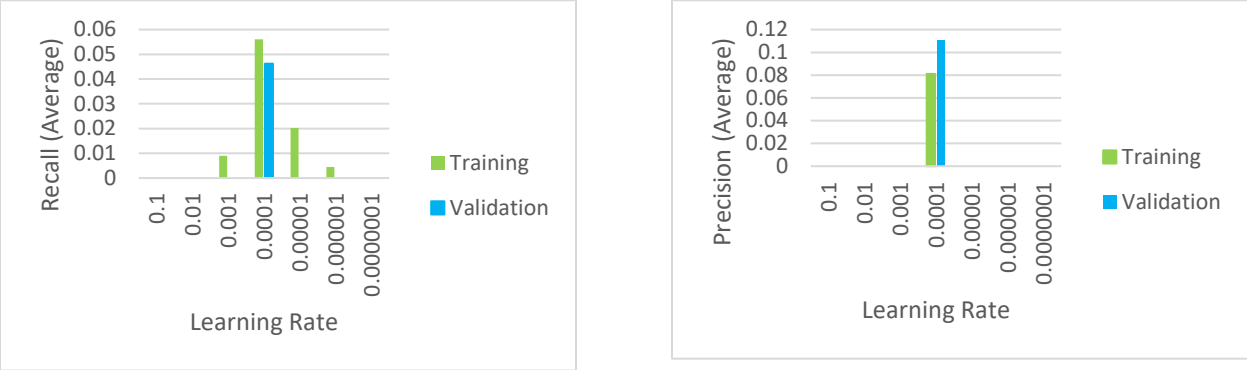

Fig. 6. Results of a ten-epoch learning rate sweep from 0.001 to 0.000001. (Left): Average positive sensitivity with an IoU threshold of 0.8. (Right): Average positive predictive value with a confidence threshold above 0.6.

For our learning rate sweep, we measured scores of 0.0 for both precision and recall at relatively high learning rates. Recall demonstrated a somewhat bell-shaped curve from learning rates 0.01 to 10^{-6} on the training data, with a maximum at 0.0001. We stopped generating runs after a learning rate of 10^{-7}, at which point neither metric returned positive scores on either dataset. In fact, for our validation dataset, the only non-zero scores returned were after a run with a learning rate 0.0001.

On all accounts of training performance and generalizability, this learning rate proved to be the most optimal out of our test runs, and we adopted it for the rest of our experiments.

Training Beyond 10 Epochs

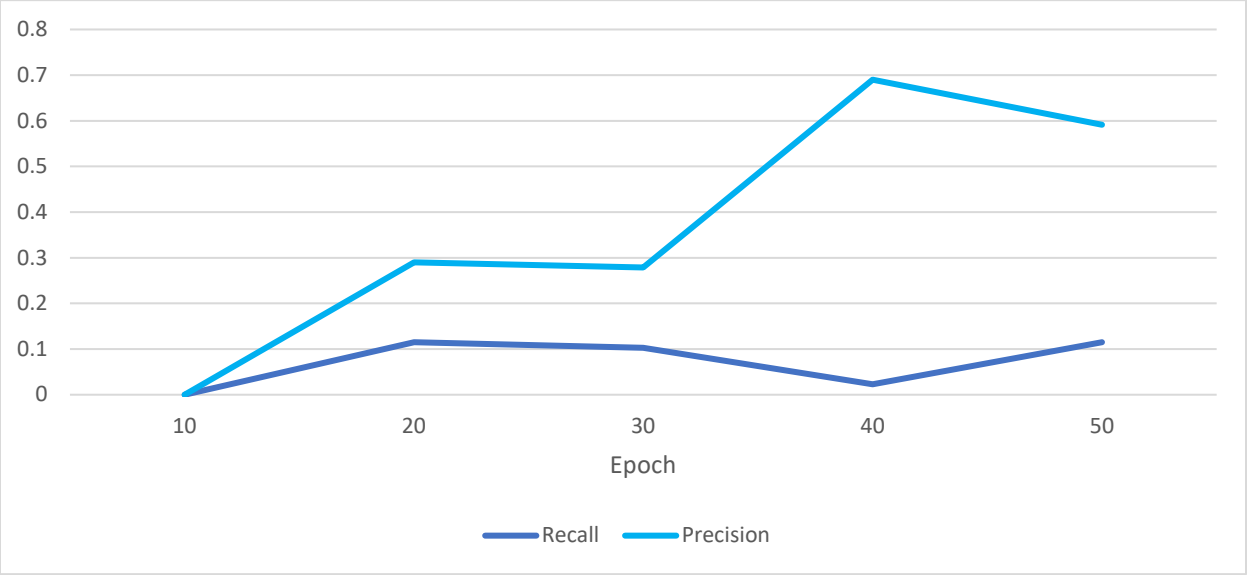

Fig. 7. Plot of recall and precision on raw validation data between 10 epoch steps from 10 to 50 epochs.

We can observe from the plot above a general trend of increasing precision over time, while recall remains generally more constant across the entire run. We can match these scores to the plotted image prediction boxes at each inference point.

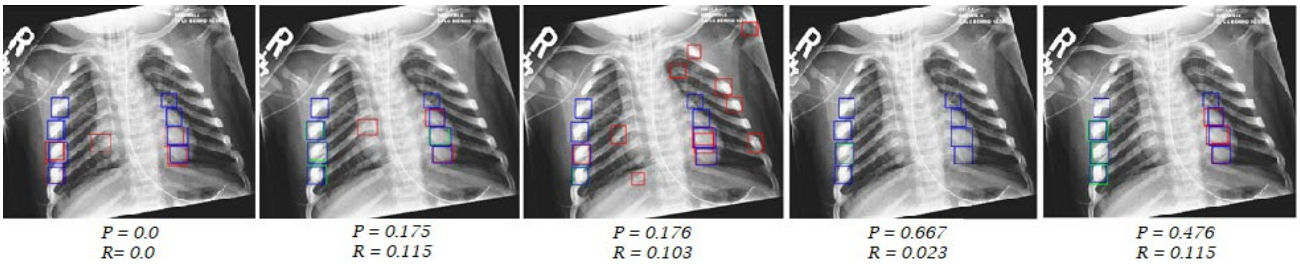

Fig. 8. Evaluation of precision (P) and recall (R) visualized through an image from the validation dataset. Images from left to right are generated after 10, 20, 30, 40, and 50 epochs.

Note that the precision and recall scores are mean scores across the entire validation set, while the image plotted above is only one of the twelve raw images from our validation set that contain rib fractures. The image serves as a pseudo-representation of our model's performance over time. Also note how the prediction boxes are most appropriately assigned when both precision and recall are high.

It would initially seem that because precision generally improves across these 5 samples that our model is also improving at a steady rate. However, a plot of the model's mean losses after each training epoch reveals the network's poor ability to learn. The default implementation for mAP confidence and IoU thresholds were 0.05 and 0.5, respectively. We did not change these values until later. We therefore show the loss and mAP results for the epochs where the score thresholds matched the strict thresholds we used for evaluation.

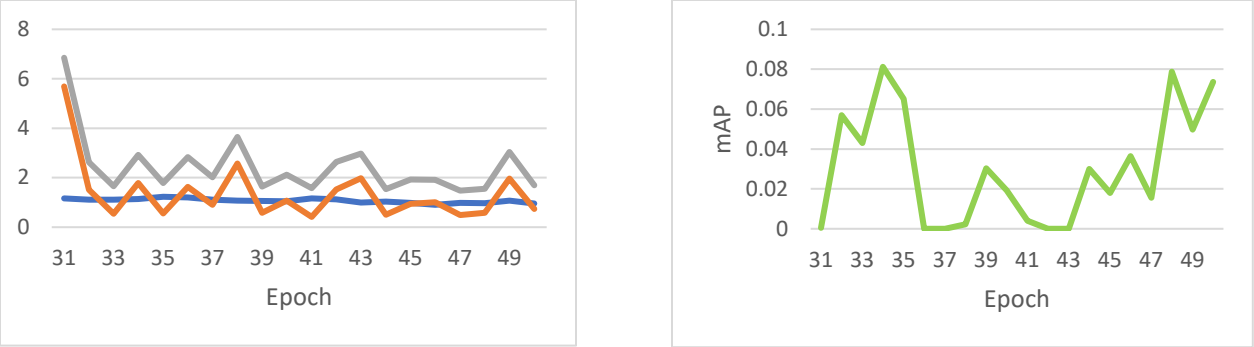

Fig. 9. (left) A plot of the mean training regression (blue), classification (orange) and total (gray) loss from epochs 31 to 50. (right) The mean average precision score on the raw validation dataset returned after each epoch from 31 to 50.

We observe a promising drop in classification loss right after 30 epochs, but then the function almost immediately begins fluctuating between 1.5 and 3.5. The magnitude of the slope of the linear trendline for regression loss from epochs 31 to 50 is just above 0.01. All functions are decreasing on average, but none with the speed or consistency to confidently predict long-term success.

A plot of mAP from epochs 31 to 50 also diminishes the promise of the less frequently plotted precision-recall graph. Although the metric starts relatively high near the beginning and end of capture, it never rises above 0.04 between epochs 35 and 48. As discussed in our methods, mAP is bounded by data's recall score – given our results in recall measurement, it isn't terribly surprising that our mAP score never gets too high. An additional 51st training epoch was run and the returned mAP had dropped back to 0.005.

Conclusion

Summary

When tuning our learning rate, we tested our model's performance on a training set with a 0.5 probability of raw images being flipped about the x-axis. Using precision and recall on the validation data to confirm the accuracy and generalizability of each run, we decided a learning rate of 0.0001 was optimal for future training. With subsequent runs we introduced new randomly generated transformations in our training data, and our model showed no promising trends of learning after 40 additional epochs.

Next Steps

Due to timing constraints, we were unable to finalize our script for re-applying transformations to data, post-training. It is likely not an immediate concern. The only new augmentation we introduced was cropping, which we previously confirmed to work correctly on sample data, and therefore we are confident that truth boxes are appropriately updated after each augmentation during runtime. However, as addressed in our methods, we anticipate that contrasting augmentations that were poorly assessed with ones that were accurately predicted on may lead to other revelations about our network's generalizability. We fully intend to finish and test our script as we proceed with training.

It is not impossible that by introducing new random transformations only after selecting our model's learning rate, we changed the effectiveness of that learning rate. Alternatively, weights were updated

too quickly before the introduction of new data variations for proper re-generalizability. Were we to restart our training from epoch 0, we may discover a different choice of hyperparameter that is more appropriate, or at the very least we will be starting the training process with a better generalized dataset.

Future training runs may reveal that our loss still plateaus even with fixed parameters and data. A possible resolution is to slowly decrease the learning rate over time, as the variability in the loss terrain becomes less obvious. If this proves ineffective, we will have to start adjusting different model hyperparameters, such as replacing network's backbone model or modifying the number of feature pyramid layers. We acknowledged that our plotted regression loss updated particularly infrequently. Experimenting with changing values of σ in the smooth-L1 calculation may boost minimization. At worst, we will need a larger set of data to properly achieve our goal.

As our network improves, obtaining correct measurements of precision will bear more significance for evaluation. We intend to revise our methods of metric calculations such that they are performed across all validation images.

References

[1] Adam A. (2018). Automatic Rib Fracture Detection in Pediatric Radiography to Identify Non Accidental Trauma. (Grant Proposal). University of Washington Imaging Research Laboratory. [PDF]

[2] Strickland, E. (2016, May 04). Autonomous Robot Surgeon Bests Humans in World First. Retrieved from https://spectrum.ieee.org/the-human-os/robotics/medical-robots/autonomous-robot surgeon-bests-human-surgeons-in-world-first

[3] Stanford University. (2018, May 03). Artificial intelligence used to identify skin cancer. Retrieved from https://news.stanford.edu/2017/01/25/artificial-intelligence-used-identify-skin-cancer/

[4] Tetris-Like Program Could Speed Breast Cancer Detection. (n.d.). Retrieved from https://www.adelaide.edu.au/news/news102302.html

[5] Lin, T. (2018). Focal Loss for Dense Object Detection. Facebook AI Research. Retrieved from https://arxiv.org/pdf/1708.02002.pdf.

[6] (n.d.). Retrieved from https://github.com/fizyr/keras-retinanet/tree/master/keras_retinanet.

[7] Kaiming H. (2015). Deep Residual Learning for Image Recognition. Microsoft Research. Retrieved from https://arxiv.org/pdf/1512.03385.pdf