An Analysis of F-RCNN vs YOLO in Table Detection

An Honors Thesis

Presented by

Veer Singh

Completion Date:
May 2020

Approved By:

Subhransu Maji
Computer Science

Erik Learned-Miller
Computer Science

ABSTRACT

Title: **An Analysis of F-RCNN vs YOLO in Table Detection**
Author: **Veer Singh**
Thesis/Project Type: **Independent Honors Thesis**
Approved By: **Subhransu Maji, Computer Science**
Approved By: **Erik-Learned Miller, Computer Science**

This paper compares the performance between Faster-RCNN and YOLOv3 on a table-detection dataset. For detecting tables within large datasets, the tradeoff between accuracy and time becomes drastically important. By comparing the use of a granular model such as F-RCNN and a single-sample model as YOLOv3, this paper aims to provide an evaluation on the use of both in table-detection. The results from this work can be used and taken into consideration when attempting to detect tables on a large dataset.

1. Introduction

Table detection is one of the many problems that play an important part in analyzing and interpreting the structure of a document. Having an understanding of the document's structure allows us to not only extract the data on the document, but also preserve its organization. The task of identifying and localizing tables on a document is just one part of a wide domain of problems that fall under *object detection*.

There are many algorithms that are able to achieve high levels of accuracy in terms of properly detecting and localizing objects. The issue is that many of these algorithms can take significant amounts of time to train and test, which is not optimal for datasets with thousands of images. Fortunately, researchers have developed new neural-networks that can be used to detect objects in real-time with high accuracy. These neural networks look promising for detecting tables on large datasets both accurately and quickly.

Given the rapid digitalization of the world, this is important as large companies and institutions now find themselves attempting to take old paper data and finding some way to digitally transcribe them onto computers. For small organizations with relatively low amounts of old files and paperwork, this is a menial task that can be performed manually by a few counts of people. However, for larger organizations dealing with multiple variations of data, such as hospitals, banks, governments, and educational institutions, a computerized approach to quickly copying this data onto a virtual platform is preferred.

In this paper, we observe two different types of neural-networks that can be used to achieve this goal. We will be comparing two real-time detection networks; specifically, *Faster*

Regional Convolutional Neural Network, or F-RCNN, against *You Only Look Once*, or YOLO. This paper will provide an overview on steps taken to train the networks as well as metrics for speed and precision generated from running both networks on a real table-dataset. Sample images labelled by both networks will be provided, as well as training results over multiple epochs. Finally, a summarization and conclusion of the results of the research will be shown.

2. Review of Literature

2.1. Overview of history

During the early years following 2010, object-detection performance had slowly begun to plateau. Convolutional neural networks, or CNNs, that were being used at the time managed to achieve a low mAP, or *mean average precision*, of around 23.5% on the PASCAL VOC 2012 dataset (Girshik, 2014). This began to change when the RCNN, or *regional convolutional neural network*, was introduced by a team of researchers at UC Berkeley in 2014. The RCNN is a convolutional neural network that aims to use a combination of selective search and region proposals on an image to properly detect and label objects in an image. When it was tested on the PASCAL VOC dataset, it achieved an mAP of 53.3%, a significant improvement from previous results.

The development of RCNN was a major improvement over traditional CNNs not only in performance, but also computationally less expensive in terms of both time and memory usage. Rather than creating hundreds of thousands of potential regions to classify in an image, RCNN generates a relatively small, constant amount of areas to classify based on selective-search and image segmentation. Despite this improvement, RCNN still typically uses around 2000 image proposals, resulting in a detection time of approximately 47 seconds per image (Girshik, 2014). This makes it infeasible to train on large datasets of documents.

Since the development of RCNN, two more detection-based deep-learning networks have come out that have faster run-times.

The first algorithm that was developed is F-RCNN, which is a newer, modified version of RCNN. F-RCNN aims to skip the slower run-time of selective search by using a seperate CNN to predict regional proposals in the image. In addition, F-RCNN optimizes prior architecture, allowing the network to compute image features once as opposed to running each RoI through its own CNN. This resulted in a new testing time of approximately 37 milliseconds per image. Furthermore, these new modifications drove up accuracy, giving F-RCNN an mAP of 75.9% on the same PASCAL VOC 2012 dataset (Redmon, 2016).

On the other hand, YOLO attempts to look at the image in its entirety only once, and predicts both bounding boxes and class probabilities for objects using a single neural-network. Given that YOLO attempts to both classify and localize objects simultaneously, as opposed to F-RCNN's method of classifying many separate regions contained within the image, YOLO runs significantly faster, only needing 7 milliseconds per image to test.

2.2. Development of F-RCNN

The first paper detailing F-RCNN was published by Ren. et al. (2015) under the title *"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks."* In this paper, Girshick and his team describe that regardless of how the RCNN has evolved throughout its different versions, the basic process behind it remains the same:

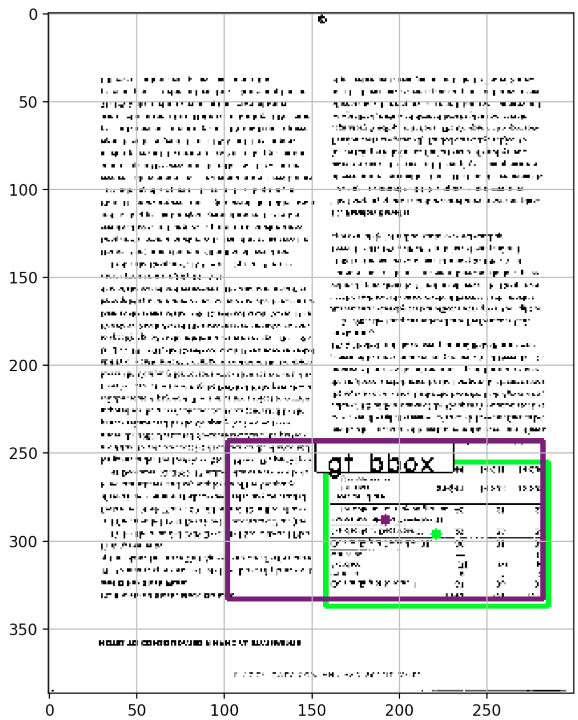
```
procedure regional-convolutional-neural-network(img i)
    RoI ← GenerateRegionProposals(i)
    FeatureMap ← GenerateFeatureMap(RoI)
    Features ← PerformMaxPooling(FeatureMap)
    for each pixel in Features:
        generate classification_probabilities
        generate object_bounding_boxes
```

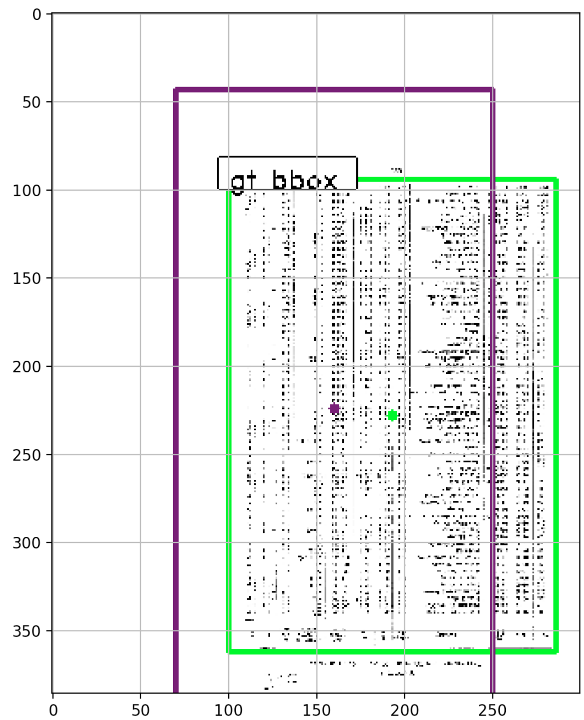
The idea is to create region proposals where actual objects might be, extract features from those proposals, then use those features to make classification probabilities and properly align the bounding boxes.

One bottleneck with traditional RCNN resides in the generation of region proposals. Traditional RCNN uses an algorithm called "selective search" to provide initial guesses as to where there's potential for an object to be within the image. This algorithm was particularly sluggish, as it cost about 2 seconds of execution time per image. The authors of the paper solve this problem by replacing "selective search" with a supplementary CNN they call the *Regional Proposal Network*, or RPN, which generates different region proposals.

In addition to being faster during execution, the RPN has the added benefit of being able to improve through a training process.

In the architecture of an F-RCNN, the RPN begins by taking the input image and generating a feature map. Once each feature map is generated, an *n x n* sliding window is run over these feature maps. Everytime the sliding window moves to a new position, the network generates a set of *anchor points*. Anchor points can be thought of as bounding boxes that are defined by their center *(x, y)* and some width and height (*w, h*).





Each anchor point for that sliding window has the same center; however, they all differ in that they might all be different sizes. In the above image, the anchor points would be the purple bounding boxes. Furthermore, a value, p^*, is computed which indicates how much these anchor points overlap with the actual ground truth bounding boxes. Threshold values of .3 and .7 are used respectively for calculating p^*:

$$p^* = \begin{cases} 1 & if \quad IoU > 0.7 \\ -1 & if \quad IoU < 0.3 \\ 0 & \quad otherwise \end{cases}$$

This process creates approximately 20k anchors in total. The goal of assigning each of the anchor points a specific score is to setup the ability to run *non-maxima suppression,* or NMS, which effectively removes anchor points that have a low p^* score, and hence a low probability of being associated with an object. When the network runs NMS on the anchors, it reduces the total count to approximately 6k anchors per image. Lastly, the RPN performs both regression by predicting a bounding box for the true object.

Next, F-RCNN trains its RPN in mini-batches by sampling a total of 128 positively scored anchors and 128 negatively scored anchors. The calculation for loss is achieved by summing the average classification loss, $L_{cls}(p_i, p_i^*)$, with the average regression loss, $L_{re}(t_i, t_i^*)$ alongside a balancing factor of λ. The *classification loss* penalizes the network for misclassifying an object while the *regression loss* penalizes the network for drawing an incorrect bounding box. It's important to note that the regression loss is activated only when the associated anchor has a p^* score of 1, meaning there is an object in the anchor point.

Finally, the network does not only calculate a classification loss and regression loss for the RPN, but also calculates one for the convolutional layers past the initial RPN in the beginning. The total loss becomes the sum of all the classification losses and regression losses:

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

Here, i is the index of an anchor in a mini-batch, whereas p_i* is the assigned p* value to that anchor box. In the regression loss, t_i is a four-dimensional vector representing the top left, top right, bottom left, and bottom right corners of the bounding box associated with that anchor.

The paper notes that using an RPN over selective-search not only improved speed, but accuracy increased as well. The mAP for Fast RCNN on the MS COCO dataset was 35.9% and trained at approximately 200ms per image. At the same time, F-RCNN achieved an mAP of 42.7% in approximately a tenth of the time for Fast RCNN. The author of the paper concludes that the addition of the RPN improves region proposal quality and thus the overall object detection accuracy.

2.3. Development of YOLO

The first paper detailing YOLO was published in 2016 by Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi under the title *"You Only Look Once: Unified, Real-Time Object Detection"*

YOLO does not use a pipeline like F-RCNN. Instead, YOLO performs its detection using a single neural network. The authors of the paper claim they simply treated detection as a regression problem, taking in the entire image and creating bounding boxes and class probabilities that way.

The process behind YOLO is to first encode the entire image into a vector and feed that vector into its own underlying CNN. The process follows a series of incredibly simple steps:

procedure you-only-look-once(*img i)*
    cells = DivideImage(S, S)
    for each cell **in** cells

```
        generate classification_probabilities
        generate object_bounding_boxes
        save results -> vector v
    run-rcnn (v)
```

YOLO's loss function consists of two primary parts: *localization loss* and *classification loss*. The *localization loss* is similar to F-RCNN's *regression loss*, where the network is penalized for having largely offset bounding boxes from the ground truth. The *classification loss* is calculated by giving an object an incorrect conditional class probability. The authors provide suggested values for regularization constants for both these losses in their paper.

According to the authors, due to the spatial constraints on bounding boxes, they believe YOLO is incredibly limited in how many *nearby* objects it can predict. An example they provide is that YOLO would struggle to find flocks of birds all tightly clustered together. In fact, these types of small localization errors account for most of YOLO's errors, rather than classification errors.

In terms of performance on large datasets, YOLO achieved an mAP of 66.4% on the VASCAL 2007 + 2012 dataset. This is a little less than it's F-RCNN counterpart that achieved a mAP of 73.2% on the same dataset. However, YOLO was superior in terms of speed, processing images at 21 frames per second while F-RCNN lagged behind at 7 frames per second. This goes to show that YOLO processes images much faster than F-RCNN, sacrificing a small amount of accuracy to do so. One interesting point the authors bring up is that YOLO performed significantly better on identifying artwork in images in comparison to RCNN, since RCNN is more suited towards *natural* images. This could imply that something static, such as a table, perhaps might not be so easily localized by F-RCNN if the RPN is not able to properly capture the portrait in any region proposal.

3. Experiments and Results

Both F-RCNN and YOLO were implemented in a Python-3 environment using Keras with a TensorFlow backend. The standard baseline used for all inferencing and testing was a GTX 1060 with 8GB of memory. The baseline processor used was an Intel Quad-Core i7 8th Generation processor.

During training, a series of checkpoints were created. These checkpoints are made when the loss at a particular epoch is less than the loss of the previous epoch. This way, the best set of weights at each particular epoch are stored and are later used for testing on a validation split of the dataset.

3.1. Discussion

We utilized two models for object detection and localization for the task of detecting tables from scanned documents. The first model is F-RCNN and the second model is YOLOv3. In subsequent subsections, we discuss the results for each model on the tables dataset. Both F-RCNN and YOLOv3 were trained and tested on an annotated dataset of 512 documents that contain tables. Given the relatively low amount of data available in our dataset, a small batch size of 10 was used. The number of epochs chosen was the recommended amount as suggested by the authors of the papers for both models. Hyperparameters such as learning rate and decay were also kept in alignment with values provided by the authors.

3.2. Data Annotation

For our dataset, a sample of 10 different documents were taken from the U.S. Department of Energy's Technical Information website and analyzed for tables. From these 10 documents, we were able to extract 512 different table variations. To annotate these documents, a Python program was developed that allows the user to draw bounding boxes over a select area of the image and store them in a separate CSV file. These bounding boxes were stored in the *x_min, y_min, x_max, y_max*. This CSV file is used by both YOLOv3 and F-RCNN to parse the images in our dataset.

3.3. Faster-RCNN

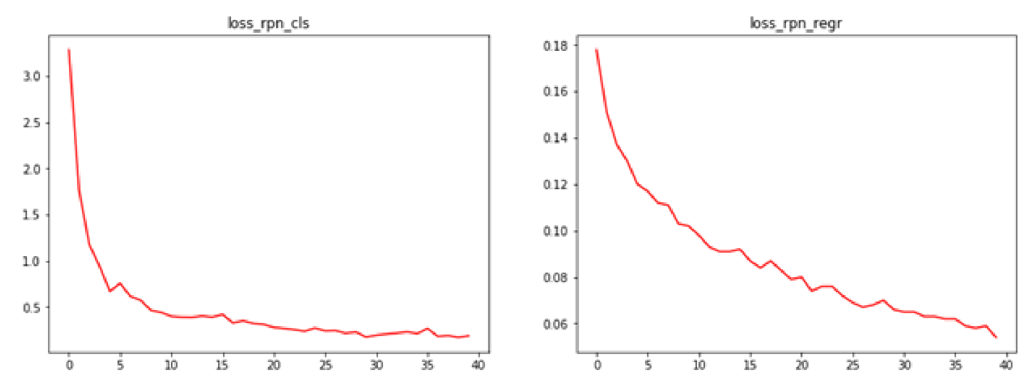
3.3.1. Recap

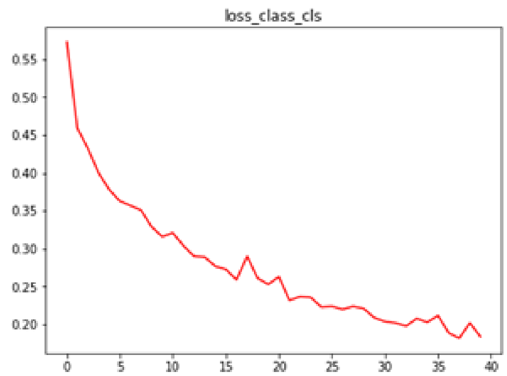
As mentioned before, the first model is a relatively older model that converts the input images to convolution-based feature-maps. These maps are then forwarded to a network that predicts the region proposals, i.e., the regions that might have the object. These region proposals are further processed to designate a set of bounding boxes to localize the objects we are detecting. A classification layer is used to predict the class of the bounding box while a regression layer is utilized to predict the continuous values of the coordinates of the objects. This way, there are multiple networks inside Faster-RCNN and the overall loss is a combined loss for a multi-task problem with four components i.e. region proposal classifier, region proposal regression, class detector classifier, and class detector regression.

3.3.2. Results

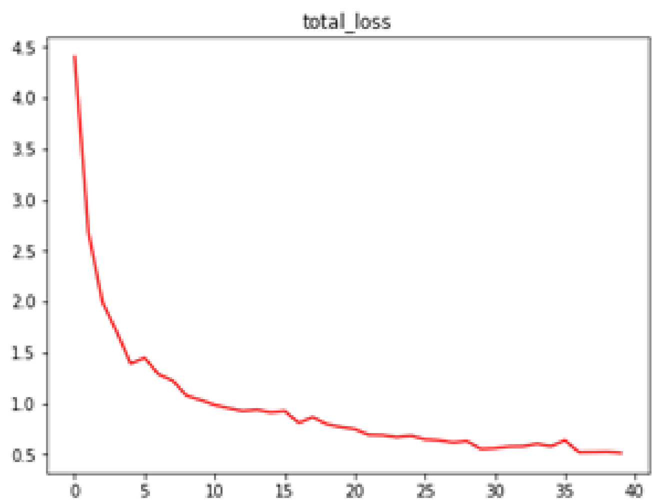
In terms of training time, the model took approximately 290 seconds per epoch to learn the dataset. For inferencing, the model was able to make predictions at around 0.37 seconds per image. This performance speed is near real-time, showing the F-RCNN is localizing and detecting tables at an incredibly fast pace. On a complete batch of a testing and validation split, the F-RCNN takes 42.8 seconds to complete. The mAP on the testing and validation split is approximately 94%.

Recall that loss for F-RCNN has four components: the classification losses for the RPN as well as the network itself. We show the metrics for each of these components over the training epochs in the graphs below.

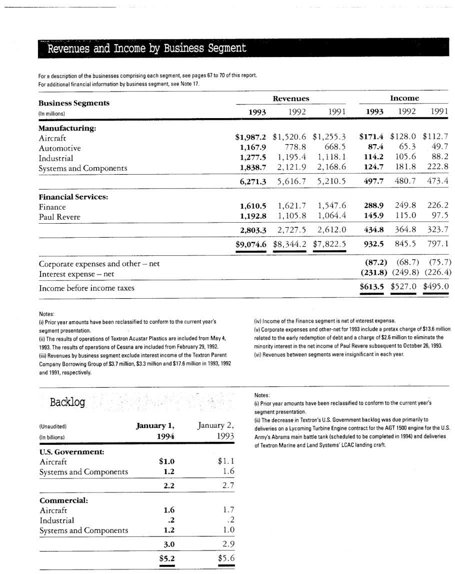




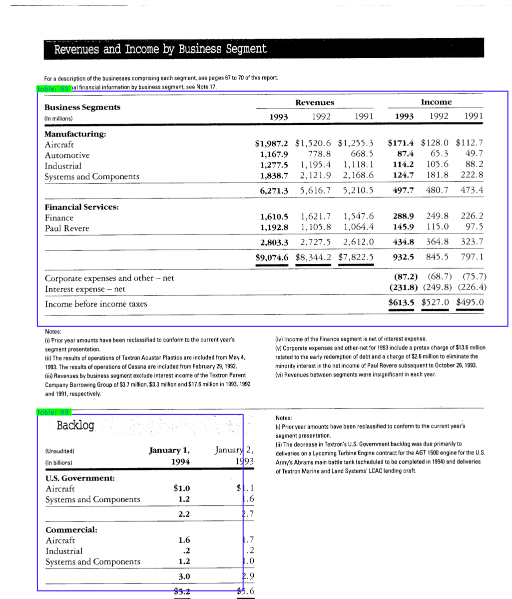
The total loss is based on the summation of all these prior losses as shown earlier. This total loss for the entire network can be observed in the following graph.



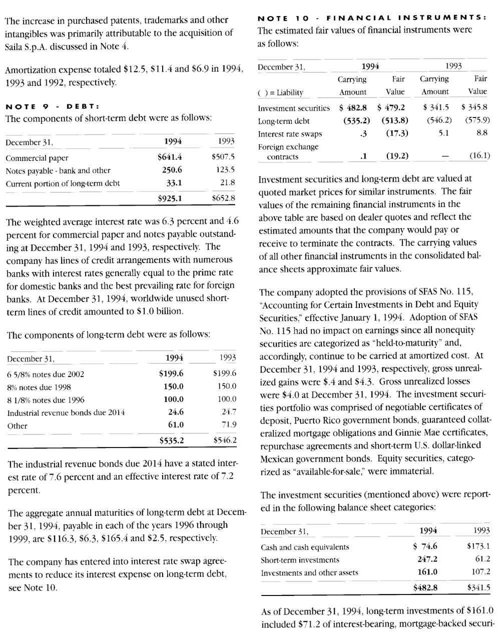
It's notable that after the 35th epoch, the loss begins to converge to a final value. As a result of the loss reaching a singular constant, this is where we decided to stop training the neural network. After training the F-RCNN, it seems to perform exceptionally well on our dataset. The model tends to make consistent predictions across the entire dataset.



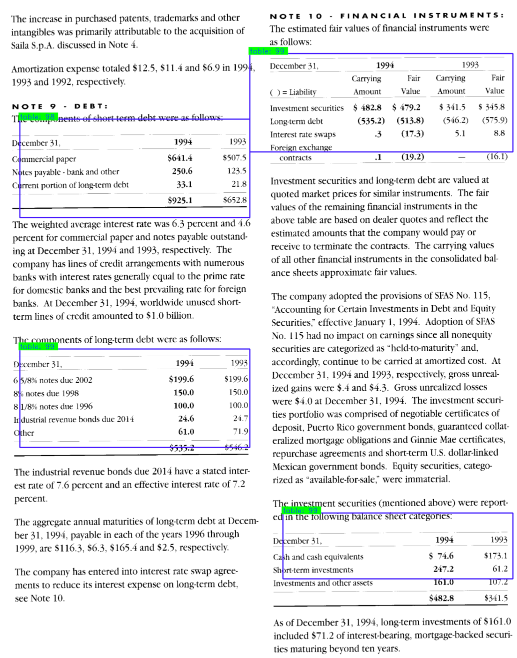
→



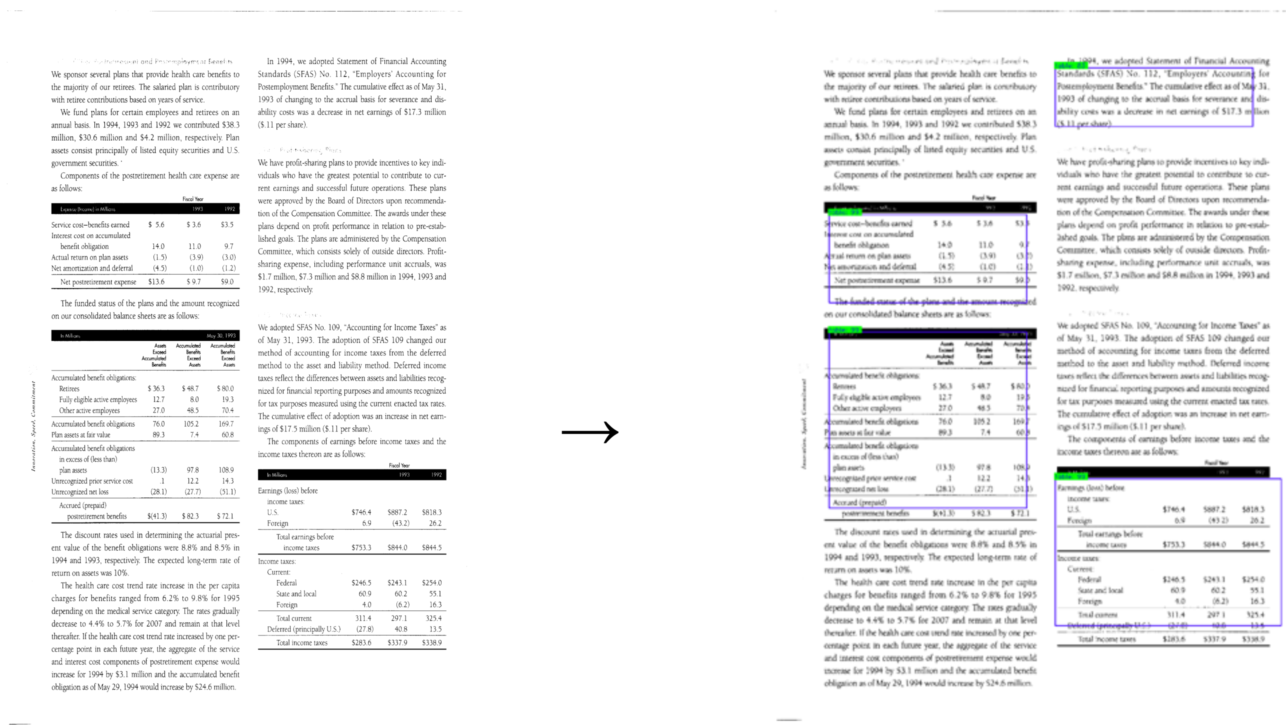
before **detection** after



→



before **detection** after



before → detection → after

To reiterate, the mAP achieved by the model was 94%. The model does present us with a few amounts of false positives as well. In the example below, one false positive is found within the top-right bounding-box. The model has labeled this paragraph as a table with an 83% confidence rating. False positives such as this one can be ultimately reduced by tweaking the confidence threshold for the model. By default, the F-RCNN threshold is set at 75%. A higher confidence requirement, such as 90%, would prevent many false positives from occurring, but reduce mAP.

These false positives can be attributed to the table formatting in the training dataset. The tables seen in this image have delinations that distinctly separate both the rows and the columns. However, because some tables in the dataset lack any clear border between the row and columns, the model tends to misclassify normal paragraphs of text in the document as tables.

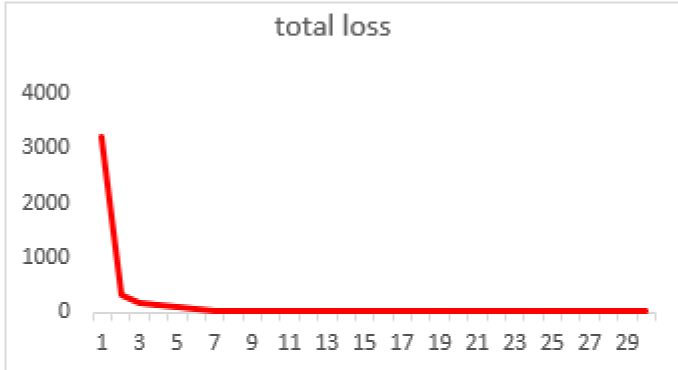
3.4. Development of YOLO

3.4.1. Recap

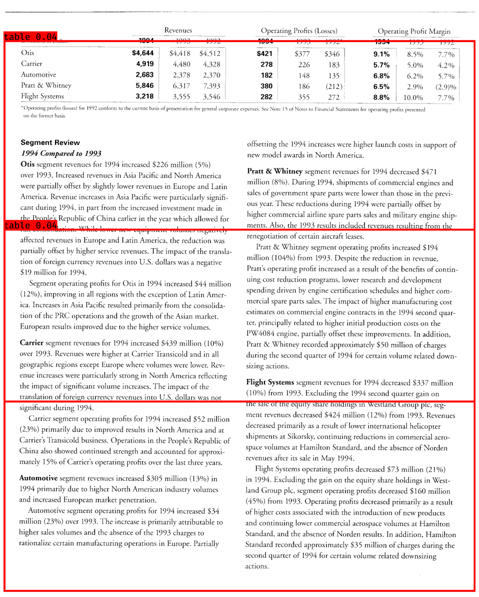
F-RCNN utilizes regions for localizations of the objects and does not look at the whole image at once. This approach leads to multiple networks within F-RCNN where each network is treated as a separate component and is responsible for a specific task. YOLO, however, utilizes a single CNN to predict both the bounding boxes and class probabilities for objects. This prevents it from localizing smaller clustered objects within an image. We compare the performance of F-RCNN with YOLOv3 below.

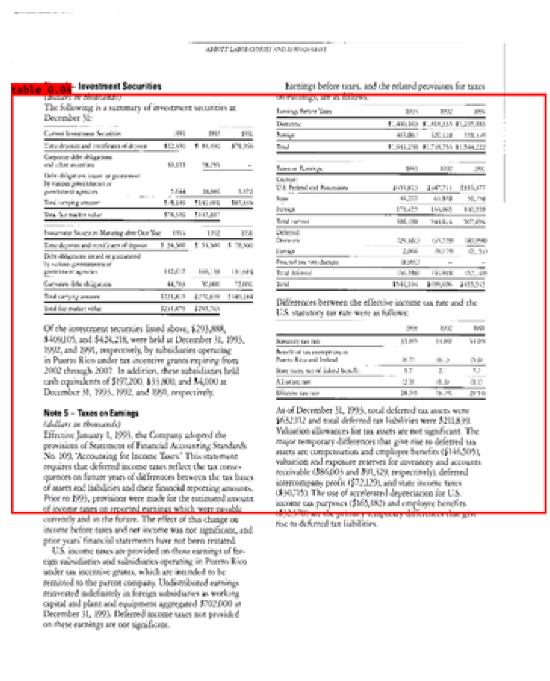
3.4.2. Results

In our experiment, YOLO turned out to be significantly faster than F-RCNN in terms of both training and inferencing. During the training phase, the amount of time per epoch was drastically less, averaging around 26.22 seconds. This is significantly less than F-RCNN's training time of 42.8 seconds. Furthermore, inferencing with YOLO was drastically faster than F-RCNN. The amount of time per image it took for YOLO to attempt to predict and localize tables came in at 74 milliseconds. This is drastically less than F-RCNN's inferencing time of 370 milliseconds, per image. This fast inferencing time matches our expectations that YOLO would perform faster thanks to its single unified loss. We show this loss over the training epochs in the following graph.



It is observed that YOLO converges faster than F-RCNN. It only takes 10 epochs to converge as compared to 36 epochs of F-RCNN. However, these faster times come at a cost of detection accuracy on our dataset. YOLO achieves a low mAP of 2% across our entire table dataset.





In contrast to F-RCNN, YOLO captures significantly less tables on our documents than F-RCNN. The amount of tables YOLO detects varies when thresholding the model's confidence score. When thresholding with a confidence score of .25 for tables, YOLO is unable to capture any tables at all. This confidence score was fine-tuned to .01 to capture the most amount of tables, if any. Given that YOLO's maximum confidence on detecting any table in our dataset comes down to 4%, it severely lacks in recognizing the granular differences between tables and text. It's important to note that in these cases, YOLO does not detect any false positives as well.

3.4.3. Discussion

The reason behind such degradation of classification accuracy can be traced back to how both models work. Because F-RCNN is able to generate smaller, more fine-tuned RoI's to convolve through the network, it's making both large and small observations about the image. YOLO's single sample method may not allow the model to examine the image on a finer scale, causing it to abstractions that hurt the network in the end. This leads us to infer that model is underfitting to the data. An observation of the training loss shows that despite the drastic improvements the model makes to reduce loss during the very first epochs, the loss still converges at an incredibly high loss-value.

4. Final Remarks

In the end, for our dataset of 512 elements, YOLO provided an incredibly low mAP of 2%. At the same time, F-RCNN produced an mAP of 94% on the entire dataset. The total amount of time for inferencing taken for YOLO averaged around 26 seconds while F-RCNN took nearly twice as long, at 42 seconds.

Our results lead us to conclude that for smaller datasets, it is viable to use F-RCNN rather than YOLO. Although YOLO produces faster training and testing times, better performance would require a significantly larger dataset for training. In summary, it’s a trade-off between accuracy and time complexity. For any task which demands higher localization accuracy, it is plausible to use F-RCNN. Furthermore, if the dataset of images is sizable within the hundreds range, F-RCNN resulted in preferable training and inferencing results. However, if there is a large-scale annotated dataset available and time complexity has to be considered, YOLO could potentially be a better candidate if YOLO is able to increase its accuracy with a larger dataset.

References

Girshick, et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” *ArXiv.org*, 6 Jan. 2016, arxiv.org/abs/1506.01497.

Redmon, et al. “You Only Look Once: Unified, Real-Time Object Detection.” *ArXiv.org*, 9 May 2016, arxiv.org/abs/1506.02640.

Girshick, et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.” *ArXiv.org*, 22 Oct. 2014, arxiv.org/abs/1311.2524.

Huang, J., Rathod, V., Sun, C., & Anoop, Z. (2017, April 25). Speed/accuracy trade-offs for modern convolutional object detectors. Retrieved May 16, 2020, from https://arxiv.org/abs/1611.10012

Y. Zhao, B. Zheng and H. Li, "FRCNN-Based DL Model for Multiview Object Recognition and Pose Estimation," 2018 37th Chinese Control Conference (CCC), Wuhan, 2018, pp. 9487-9494, doi: 10.23919/ChiCC.2018.8483556.