Recurrent CNNs for Bounding Box Stability in Object Detection

Donovan Fung
Stanford University
donovanf@stanford.edu

Prerna Dhareshwar
Stanford University
prernamd@stanford.edu

Sanjeev Suresh
Stanford University
sanjeev9@stanford.edu

Abstract

Most modern object detection algorithms (YOLO,SSD) are prone to bounding box jitter. Our project explores the feasibility of attaching a recurrent neural network at the end of a YOLO detector to de-noise/stabilize a jittery bounding box trajectory. Our network was tested on the MOT2015 dataset and improves both center position error and scale position error over existing architectures with initial training.

1 Introduction

Today's state-of-art object detection algorithms such as YOLO[1] and SSD[2] achieve near human level performance in object detection and do so at speeds which make them applicable for real-time applications. However, although these algorithms can produce detections at a high frame rate, the bounding boxes they produce often noticeably suffer a few artifacts (i) being inconsistent in aspect ratio, (ii) small random translational shifts about some center, (iii) bounding boxes not produced at all in one frame in the sequence. These three main factors characterize the stability of a bounding box trajectory. We informally denote the inconsistency in aspect ratio and the small random translational shifts as bounding box *jitter*.

Bounding box jitter in videos can often be distracting to the viewer and can also be problematic in applications where the behavior of an object is determined by the properties of the bounding box. For example, an autonomous vehicle may determine the velocity of an object by tracking its bounding box center – jitter would indicate false movement of the object.

Our project aims to explore the effectiveness of modelling object detection in a more human-like fashion – by utilizing knowledge of the detected object(s) from previous video frames. More specifically, we attach a recurrent neural network at the end of an object detection model that has been trained to predict a trajectory of stable bounding boxes. The general architecture of our model is shown in Fig. 1.

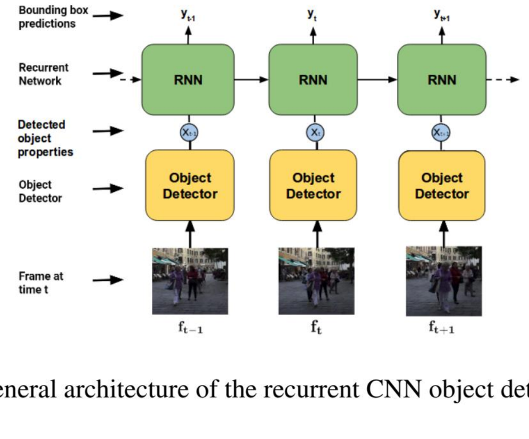

Figure 1: General architecture of the recurrent CNN object detection model.

1.1 Why does jitter occur?

To better develop a solution for eliminating jitter, we first explored why it even occurs. Our investigations ultimately lead to examining the object detection algorithms itself and overall characteristics (intensity, size, etc) of the images in

,

a sequence that produce unstable bounding boxes. As an edge case, we found that performing object detection on a video of a static object will also produce bounding box jitter, even when detected with >95% confidence, using both YOLO and SSD. We eventually gathered two primary factors that are the likely cause of bounding box jitter:

1. **Inherent pixel noise from the camera sensor**
   Noticeable jitter can be seen even on videos with static objects which leads us to believe that minute changes in pixel intensity frame by frame caused by the inherent camera noise resulted in small shifts in the proposed bounding box position

2. **Improper aggregation of the proposed bounding boxes**
   The aggregation method used in YOLO (non-max suppression) can lead to significant bounding box jitter in videos. In highly correlated detections (such as those in consecutive frames of a video), discarding low(er) probability bounding boxes is in one sense equivalent to losing valuable information from these detections.This can therefore cause jitter.

2 Related work

Traditionally, noisy signals from a signal processing standpoint are dealt with using a recursive filter such as a moving average filter or a Kalman filter. An RNN can be modelled very similarly to a recursive filter. In [3], an LSTM is used to learn the motion and noise model of the Kalman filter, which allows for learning of rich models from data. The *Simple Online and Realtime Tracking* (SORT) algorithm [4] estimates the inter-frame displacement of each bounding-box as a linear constant velocity model and uses a Kalman filter frame work for predicting the its next state.

The issue of improper aggregation is addressed in [5], where a *weighted* non-max suppression is used to propose the bounding box which greatly improved the bounding box stability. Another method in [5] was the *Motion Guided Propagation* method which used optical flow to reduce inter-frame bounding box variance.

In [6], a spatially supervised recurrent CNN is used to learn the features in the spatiotemporal domain. This model specifically uses YOLO and an LSTM to improve object tracking in cases such as major occlusion and severe motion blur. This method however still suffers from bounding box jitter.

3 Data-set and Features

For training our model, we decided to follow the approach used by most papers on object tracking and detection and use the MOT dataset [6]. The MOT dataset provides us 11 training sequences of various resolutions from 1920x1080 to 640x480. It also provides us with 11 validation sequences which we used in the milestone phase of the project to evaluate the performance of our experiments.

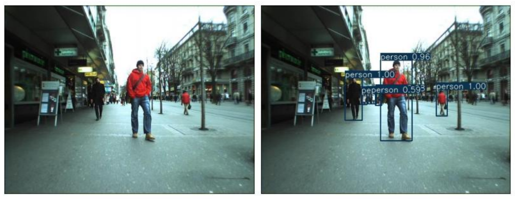

Figure 2: (a) MOT2015 Sample (b) Detections

4 Methods

4.1 RNN/LSTM

We model our recurrent network as a time-series prediction problem where it predicts a bounding box using information from past frames. We feed the LSTM a sequence of two main features that are concatenated:

1. Bounding box outputs from YOLO, specifically only the x, y location, width, and height
2. Flattened feature maps extracted from the object detector's convolutional network higher layers

Feeding higher layer feature maps into the LSTM allows the prediction of bounding boxes to also be dependent on the history of convolutional activations, which results in a more powerful bounding box prediction.

Since our LSTM also takes as input the activations (average-pooled over 16 different filters each of size 52×52) from a higher layer of the CNN, we chose to have linear layers as an encoder prior to the recurrent layer. The recurrent layer output is then fed into a final linear layer which produces the bounding box predictions. After multiple iterations of hyper-parameter tweaking, we decided that our network have the following structure:

1. Linear layer with input size of 2708 ($52^2 + 4$, feature maps plus bounding box) and output size of 100.
2. Linear layer with input size of 100 and output size of 10.
3. LSTM layer with input size of 10, hidden size of 4, and a layer size of 3.
4. Linear layer with input size of 4 and output size of 4.

A detailed breakdown of our network is depicted in Fig. 3.

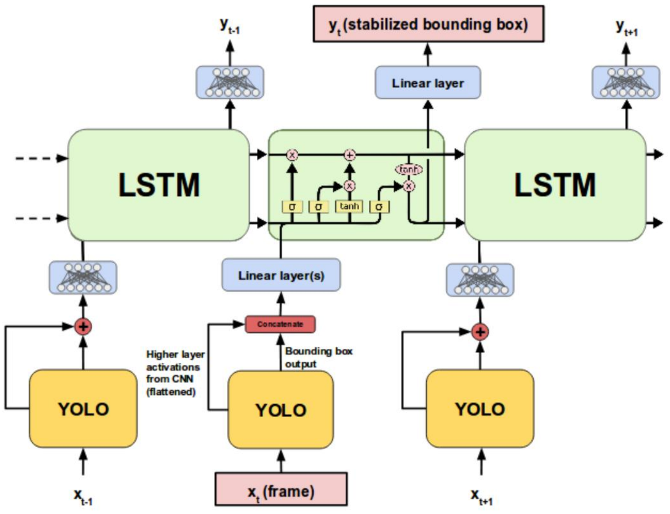

Figure 3: Breakdown of the architecture.

4.2 Evaluation Metrics for Stability

We adopt a novel bounding box stability metric proposed in [1], but modify it for single object tracking rather than multiple objects. This metric quantifies bounding box stability along a trajectory with the following evaluations:

1. Center position error, E_C
2. Scale and ratio error, E_R

Center position error
Center position error evaluates the stability of the center positions of the detections along a trajectory in both the x and y direction. The predicted bounding box in the f^{th} frame is defined as $B_p^f = (x_p^f, y_p^f, w_p^f, h_p^f)$, which is the center of horizontal axis, vertical axis, height, and height, respectively. Similarly, the corresponding ground-truth bounding box is defined as $B_g^f = (x_g^f, y_g^f, w_g^f, h_g^f)$. Then, the center position error is taken as the standard deviation of all errors in the trajectory.

$$e_x^f = \frac{x_p^f - x_g^f}{w_g^f}, \quad \sigma_x = std(e_x), \quad e_y^f = \frac{y_p^f - y_g^f}{h_g^f}, \quad \sigma_y = std(e_y)$$
$$E_C = \sigma_x + \sigma_y \tag{1}$$

Scale and ratio error
Similar to center position error, scale and ratio error evaluates the stability of the scale and aspect ratio of the bounding boxes along a trajectory. A square root of the area ratio is used to represent the scale deviation, and the ratio of two aspect ratios is defined as the aspect ratio deviation. The total scale and ratio error is defined as the standard deviation of the errors along the whole trajectory.

$$e_s^f = \sqrt{\frac{w_p^f h_p^f}{w_g^f h_g^f}}, \quad \sigma_s = std(e_s), \quad e_r^f = (\frac{w_p^f}{h_p^f})/(\frac{w_g^f}{h_g^f}), \quad \sigma_r = std(e_r)$$
$$E_R = \sigma_s + \sigma_r \tag{2}$$

These errors are summed up to produce the overall stability evaluation.

$$\Phi = E_C + E_R \tag{3}$$

5 Training

5.1 YOLO Training

We retrained YOLO and refashioned it as a single-class detector for person detection to better fit our dataset. To do this, we modified readily available code from [7] based on a PyTorch implementation of the training routine used in Darknet. The training was done on 5500 images from the MOT2015 dataset over 100 epochs at a learning rate of 0.001. We did data augmentation also by using image flip, saturation and affine transformations.The training used a pretrained weights file from Darknet and simply retrained the final layers using transfer learning. We also modified our detector to extract higher level features from the 100th layer of YOLO. This was then modified and used by the LSTM as input features.

5.2 LSTM Training

We trained our LSTM model on 76 bounding box trajectories taken from the MOT dataset, each trajectory . We trained the LSTM with a learning rate of 0.0001 with an exponential decay of 0.99 every 20 epochs. We used the Adam optimizer and trained for 15000 epochs. We trained on a custom loss which we called the stability MSE loss. It consisted of 3 terms - the Mean square error between predictions and ground truth (which ensured that we got close to the ground truth values), the center position error (a variance which reduced variations due to noise in the center position) and the scale ratio error (a variance that reduced the variations in aspect ratio of the bounding box due to noise.) The results obtained are discussed in the following section.

6 Experiments/Results/Discussion

6.1 Evaluation on stability metrics

Evaluating our outputs on the stability metrics defined in section 4.2, we observed that we did indeed do better on both center position error and scale ratio error when compared to pure YOLO. However SORT did better than our model on center position tracking, but did poorly on aspect ratio error. The following tabular column compares the average errors for each of the models on the validation set -

	Pure YOLO	YOLO + SORT	YOLO + RNN
Center Position Error	93.01	71.11	89.89
Scale Ratio Error	163.27	323.979	161.44

The plots of the two errors comparing pure YOLO and our YOLO + RNN on training examples is shown in figure 4 below.

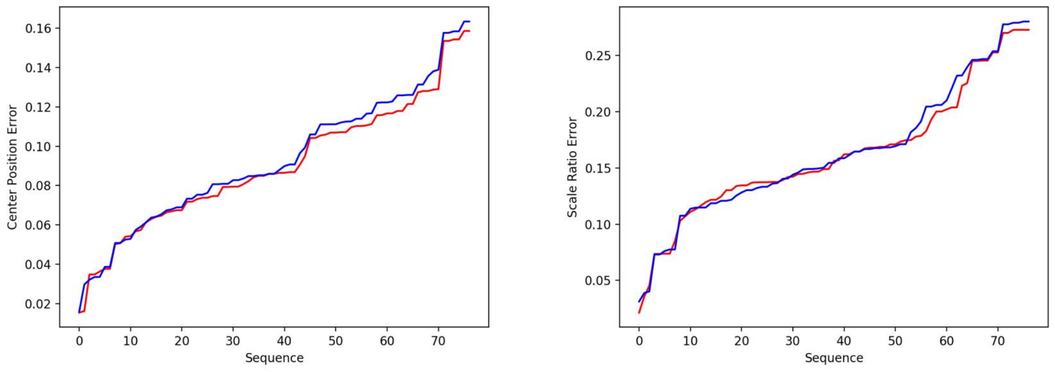

Figure 4: (a) Center Position error (b) Scale Ratio error - The red line indicates YOLO + RNN and the blue line indicates pure YOLO. The x-axis denotes the test sequence number.

Though the difference between the two seems small, we are positive that if trained for longer, the improvement would have been much more. Some challenges we encountered during training were related to the loss either reducing too slowly or not at all. Changing to learning to improve speed of training merely made the loss stagnant. Hence with more hyper-parameter tuning and longer training, we are sure the improvement would be stark. The plot in figure 5 shows the training loss, and how slowly the values decrease.

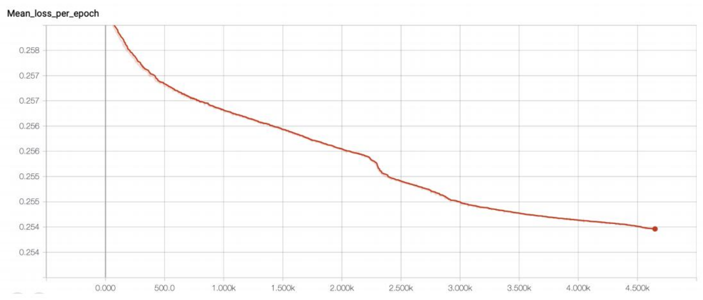

Figure 5: Training loss with number of epochs

We also observed an average additional inference time per frame of 0.057 milliseconds which produces an almost negligible effect for real time applications.

7 Conclusion/Future Work

While we are confident that an RNN block after the object detection block can reduce the jitter in comparison to the SORT algorithm, there are some finer details such as hyperparameter tuning that have to be performed before the improvement is significant. If given more time, we would increase the number of epochs that we trained for and try another RNN like the GRU instead of the LSTM. We would also train the CNN and RNN in tandem instead of training each individually, hence constructing an end to end model.

Another consideration we didn't take into account was that of multiple object tracking. Tracking itself is a big issue today and we tried to stabilize bounding boxes on top of tracking. So to simplify it we should have considered a single object tracking dataset instead of MOT.

8 Contributions

Sanjeev trained the object detector and explored the similarities of the Kalman filter. Donovan implemented the connection between YOLO and the LSTM. Prerna trained the LSTM. Everyone contributed equally in gathering and processing the training data, error analysis, and discussing the hyperparameters and overall training procedure.

References

[1] Redmon, Joseph, et al. "You Only Look Once: Unified, Real-Time Object Detection".

[2] Liu, Wei, et al. "SSD: Single Shot MultiBox Detector." European conference on computer vision. Springer, Cham, 2016.

[3] Huseyin Coskun, et al. "Long Short-Term Memory Kalman Filters: Recurrent Neural Estimators for Pose Regularization", CoRR 2017.

[4] Bewley, Alex, et al. "Simple online and realtime tracking." 2016 IEEE International Conference on Image Processing (ICIP). IEEE, 2016.

[5] Zhang, Hong, and Naiyan Wang. "On The Stability of Video Detection and Tracking." arXiv preprint arXiv:1611.06467 (2016).

[6] Ning, Guanghan, et al. "Spatially supervised recurrent convolutional neural networks for visual object tracking.". IEEE, 2017.

[7] Leal-Taixé, Laura, et al. "MOT Challenge 2015: Towards a benchmark for multi-target tracking." arXiv preprint arXiv:1504.01942 (2015).

[8] Jocher, Glenn. "YOLOv3 in PyTorch", GitHub Repository, https://github.com/ultralytics/yolov3

[9] Olah, Chris. "Understanding LSTMs", http://colah.github.io/posts/2015-08-Understanding-LSTMs/