Stapler Detection and Installation Report

1. Algorithm

   Object Detection via Yolov4, and applying COCO data set.

   >Training of 2D detection of the stapler at 360 degree of camera angles, with various stapler state (open/close), and different environment (white wooden desk/brown wooden desk).

   >Based on a rigid reference, measure and tell whether installed staples is appropriate and how guiding message, and training of the 2D detection at all common operating angles with various stapler state, and different environment (similar as above).

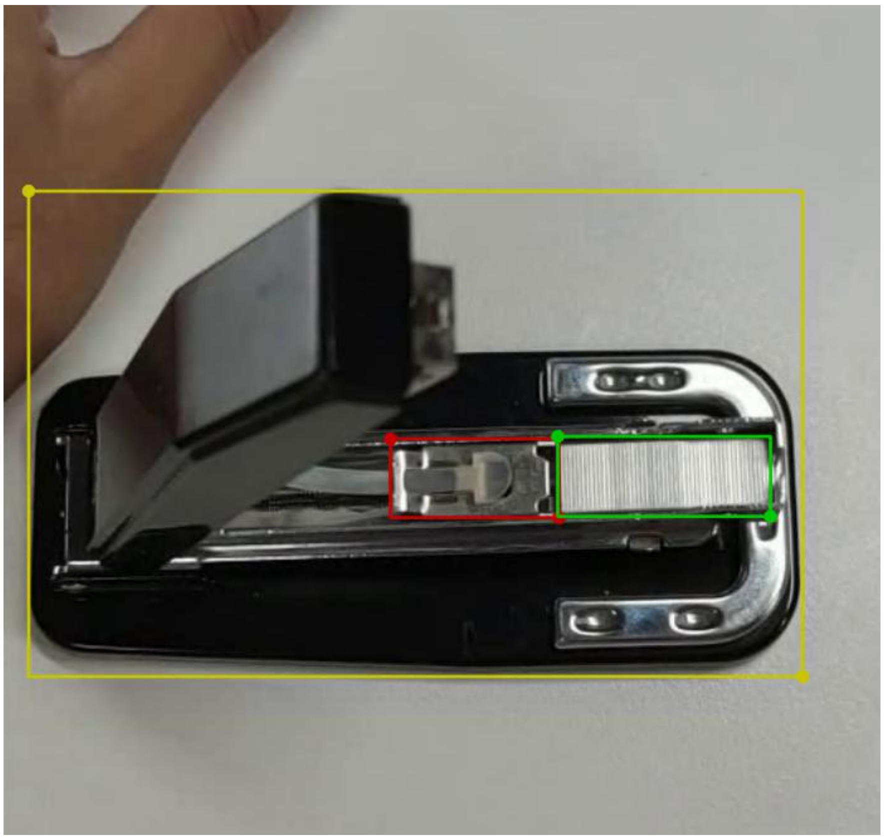
2. Training data set

   >We take photos of the target, here is the stapler of course, we take them at common-used angles, with different state of the stapler and different length of nails.

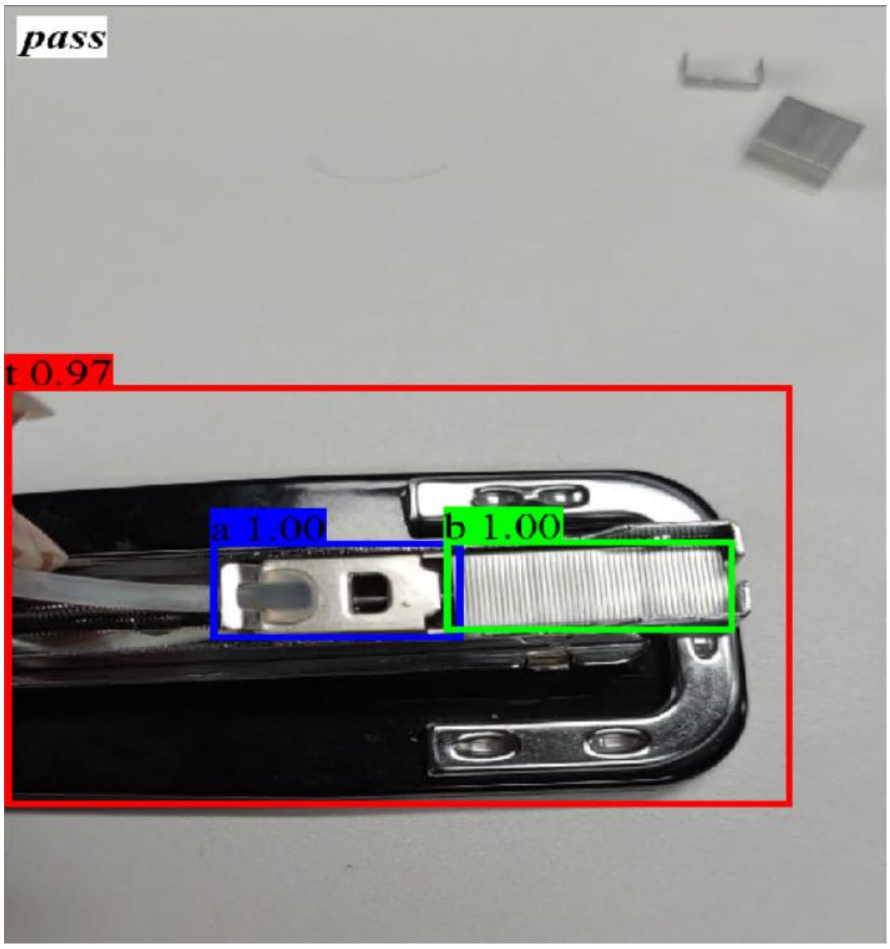
   >We need to make our model after training recognize the stapler and nails and whether the length of nails is appropriate, we have to have three kinds of labels here which are the stapler, we named it with ‘t’ in the detection, the nails, we named it with ‘b’, and a reference, we named it with ‘a’.

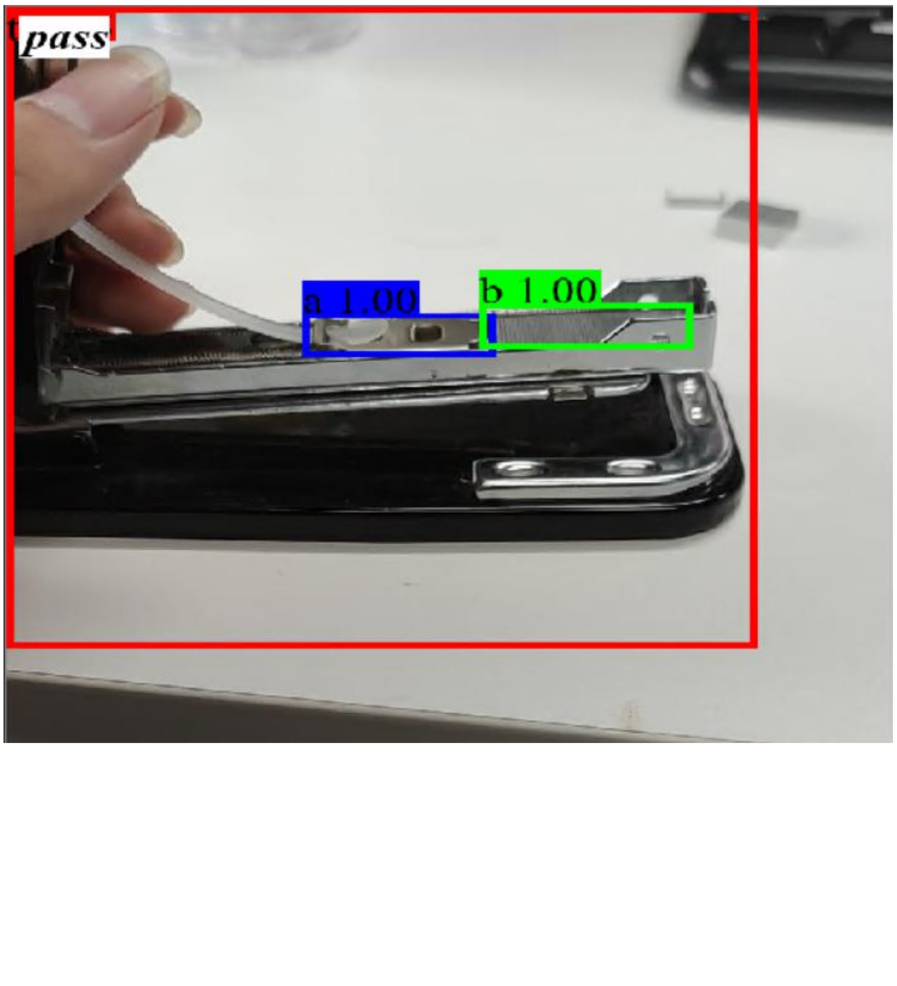
>After a series of translate, we get a data set which can be used in the yolov4 structure to train, we first put in photos with stapler to train a model just for detect the stapler. Next, put in photos with nails and reference to train a model just for nails detection. While the model is not as what we expected, we take more photos and let the computer do reinforcement learning. After two models are ready, we program to use the two models to automatically label photos for us, and program to integrate the labels for nails and the labels for stapler.

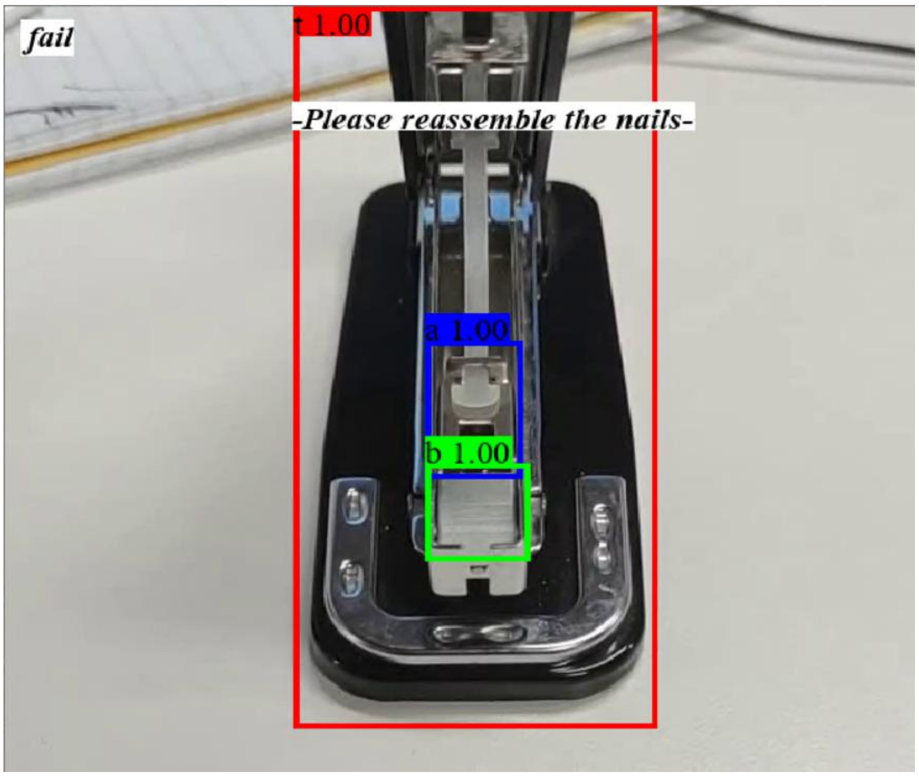
>We make a pre-process before training, we utilize the coco data set which is far more complicate and huge than our data set. We import the coco data set, and apply the transfer learning so that we could lower the risk of gradient explosion or gradient disappear, and also, a larger data set could effectively rise the learning efficiency, and lower the loss of detection at a higher speed. Cuz a training process could usually last for dozens of hours or even several days, a lower risk would prevent us from waiting for a long time, and it collapsed, and we need to retrain and redo the whole process again, and a higher speed could apparently get us a good result earlier.



3. Effect of the algorithm on testing data


pass
t 0.97
a 1.00
b 1.00


t
pass
a 1.00
b 1.00



Since the outcome is limited by the camera resolution, suggest use the same camera with both training set and testing set.

4. Task time estimation

* Development time: about 2 to 3 weeks, 1 developer

* Label images: 1285 images, 1 to 2 weeks, 1 developer, using labelme tool installed with 'pip install labelme'

* Training process: NVIDIA 2080-Ti, 70 hours per training for 1285 images with 3 annotations: 'reference', 'staple', 'stapler' (500 epochs per training)