# **Home Objects detection using Mask RCNN**

[Mask RCNN](https://arxiv.org/pdf/1703.06870.pdf) is a supervised deep neural network to detect, localize and pixelwise segment the objects in an image by Kaiming He’s team at Facebook.

In this project, Matterport’s implementation [1] is used. I created a sample application called home\_object, which is stored in Root\_dir/samples/home\_object. The dataset and its annotation are stored in Root\_dir/datasets/home\_object.

**Implementation**

The python code for this project can be accessed from my Github repository [3].

Home\_object.py: containing the training and test script, over-written class of HomeObject. It is a modified version from balloon.

Inspect\_home\_object\_data.ipynb: checking dataset and annotations

Inspect\_homeobject\_model.py: checking trained model (for unknow reason, inspect\_homeobject\_model.ipynb does not work)

Check\_bounday.py: checking bounding box annotation from downloaded dataset

Rewrite\_images.py: read and re-create the jpeg images. Original jpeg files contain tags, such as ‘orientation’ used by the annotation tools. This cause the annotation tool to open the images and rotate the images according to orientation in the tag, which cause inconsistency of annotation data. Recreating the images will remove all the tags from original images.

**Annotation Tool**

In this project, we use VIA (VGG Image Annotator) [2] because of its simplicity.

**Loss Functions**

This implementation uses five loss functions:

rpn\_class\_loss: calculated as the categorical\_crossentropy of targe anchor class and RPN output.

Rpn\_bbox\_loss: smooth\_l1\_loss of target bounding box and RPN output bounding box

Mrcnn\_class\_loss: softmax\_cross\_entropy of target object classes and predicted classes

Mrcnn\_bbox\_loss: trucked smooth\_11\_loss of target bounding boxes and predicted bounding boxes.

Mrcnn\_mask\_loss: cumulated binary\_crossentropy loss of pixelwise target object mask and the predicted mask

The total loss is the sum of averaged loss in all the above categories. For the loss of classification, this implementation uses either binary cross entropy (for two class, such as object or background), or categorical cross entropy (for multi-classes). For regression loss, this implementation uses smoothed L1 loss instead of L2 loss. This has the advantage of less sensitive to outliers.

One improvement we can try is to incorporate the weights for different classes into the loss function. This can avoid the loss function being dominated by majority class (like ‘bottle’ class in our dataset).

**Optimizers**

The original implementation uses SGD optimizer in keras. We tried both SGD and adam optimizer. Both optimizes works similar. But when it is close to 30 epochs in training, Adam has more oscillation.

**Image Augmentation**

Because only a small training dataset (about 100 images) is available, it is not enough to train even the head layers. The best way to ameliorate the problem is to augment the dataset. Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips, etc.

For this project, I did not use off-line augmentation mainly because of time limit. It takes lots of time to annotate the augmented images.

During training, imgaug library is used to augment the training images on-the-fly, because the annotation data can be changed on-the-fly the same as the augmented images.

**Training**

We train the neural network by freezing all the layers except the head layers. We downloaded the pretrained network, which use backbone network of ResNet50. We are not sure about the quality of this trained model, because we found this in the comment of Matterport’s implementation. Matterport only provided a pretrained network with backbone of ResNet101.

In the first phase, SGD optimizer was used, and no image augmentation was used. We train the model for 30 epochs. The loss is decreased almost continuously, but the trained model can not detect objects. I am using my laptop to do the training, and each train takes about a day. For unknown reason, my laptop GPU does not work for this model, even though I reinstalled CUDA driver and cudnn library and downgrade my keras from latest version of 1.14.0 to 1.12, according to some guide found on web.

In second round, I used Adam optimizer instead of SGD, with imgaug library for on-the-fly data augmentation. The trained model can detect some objects. Figure 1 is an example.

I tried to start an AWS EC2 instance to the training, but later gave up because I need to install all the software myself.

In the third round, instead of retrain all the head layers from scratch, I start the training using the trained parameters of mrcc\_bbox layers from the trained model of mask\_rcnn\_coco\_2.h5. The training is still going on.

Figure 2-7 are the learning curve of the losses. From the curves, we can see the training loss is almost always decreased with with epochs. But training does not help validation loss in mrcc\_mask\_loss, rpn\_bbox\_loss, and rpn\_class\_loss. This means more training does not help to detect the object better and the region proposal better. These two factors are actually correlated. If you can not find the region of the object better, you will not be able to find the boundary of the object. Main reason for this is that the training dataset is mostly single object in each image. But in validation dataset, most images have several objects in them, and some parts of the object are often occuluded. This make the detection of the object mask much harder.

Overall, in current project, we don’t have enough training images to retrain the head layers.



Figure 1. Object detection. The reddish colored part is the prediction of the object mask

Figure 2. Learning curve of total loss

Figure 3 Learning curve of mrcnn\_bbox\_loss

Figure 4. Learning curve of mrcnn\_class\_loss

Figure 5. Learning curve mrcnn\_mask\_loss

Figure 6. Learning curve of rpn\_bbox\_loss

Figure 7. Learning curve of rpn\_class\_loss

**Comparison of Different Loss Functions**

Original paper uses smooth-L1-loss as the loss function in rpn\_bbox\_loss and mrcnn\_bbox\_loss, because its gradient is continuous and it is also robust to outliers. For comparison, we retrain the model using mean squared error as loss function for the above-mentioned losses. Figure 8-13 show the learning curves. In the following figures, run1 uses smooth\_l1\_loss, and run2 use mse loss.

Figure 8. Learning curve for total loss.

Figure 9. Learning curve of mrcnn\_bbox\_loss

Figure 10. Learning curve of mrcnn\_class\_loss

Figure 11. Learning curve mrcnn\_mask\_loss

Figure 12. Learning curve of rpn\_bbox\_loss

Figure 13. Learning curve of rpn\_class\_loss

From the graphs, we can see mse loss performs worse both in minimizing rpn\_bbox\_loss and rpn\_bbox\_loss, compared with smooth-l1-loss. I tried to get an accuracy metric for the rpn\_class, mrcnn\_class, and mrcnn\_mask, but keras did not output such metric, even though I specified such metric when compiling the model.

**Adjustment of LOSS\_WEIGHT**

In the original implementation [1], the loss weights for all the losses are set to 1. This causes the problem that optimizer does not pay much attention to rpn\_class\_loss, because it’s mostly 100 times smaller than other losses. So, I changed the LOSS\_WEIGHT for rpn\_class\_loss to 100. The following graph show the learning curves after the change.

Run1- without change of loss\_weight

Run3-with change

Figure 14. Learning curve for total loss.

Figure 15. Learning curve of mrcnn\_bbox\_loss

Figure 16. Learning curve of mrcnn\_class\_loss

Figure 17. Learning curve mrcnn\_mask\_loss

Figure 18. Learning curve of rpn\_bbox\_loss

Figure 19. Learning curve of rpn\_class\_loss

For similar reasons as above, we do not have good measurement of the effect of this change. Total loss is not a good indicator, because we make one component artificially bigger. Both training and validation loss is bigger for run3 than run1 for both total loss and rpn\_class\_loss. This is what we expected. Validation loss of run3 for rpn\_bbox\_loss is smaller than run1. That is probably an indication that the adjustment of loss\_weight might work.

[1] Matterrport, <https://github.com/matterport/Mask_RCNN>

[2] VIA annotation tool, <http://www.robots.ox.ac.uk/~vgg/software/via/>

[3] My implementation, https://github.com/pingdong2017/Mask\_RCNN-for-Home-object-detection