Training Attempts

PLEASE READ BEFORE TRAINING. This file contains all of our attempts to re-train and improve accuracy scores of the model.

### **Attempt 1: Tried to train on our own 100+ images and then combine the weights (.h5 files).**

We trained on our 100+ custom images (with default configurations); once starting out with Coco’s pre-trained weights and once starting out with TACO’s pre-trained weights. Since those weights were only trained on 100+ images compared to TACO’s weights trained on 1500 images, we knew those weights alone wouldn’t be better than TACO’s. Thus, we decided to combine the two weight files. Doing such actually made the weights worse. This is because just training on 100+ images lowered the accuracy scores in the end, so adding it to the pre-trained weights just made it worse.

NOTE: this is not “fine tuning”, but it might be wise to look into.

### **Attempt 2: Tried to train on the official 1,500 images, our 100+ images and the 2,400 unofficial annotated images on the TACO website.**

After combining all of those annotated images, it totaled nearly 4,000 images to train on. We felt that increasing the size of the dataset would be a good way for the algorithm to train and prevent overfitting. As the training went on we noticed that it did not seem to be getting any better and it was still overfitting. This led us to do more research on training and what we can do to prevent overfitting from happening. After further research, we decided to adjust the hyperparameters that can be found in the config.py file (attempt 3).

### **Attempt 3: Narrow down the images added from the unofficial annotated images and change some hyperparameters.**

Something we realized through our previous training is that some of the class names were labeled incorrectly. That’s because the unofficial annotated images had a lot of objects that were annotated wrong and we were adding too many images of certain categories (like bottles) which made the categories unbalanced. To prevent this, we deleted over 400 annotated images to help balance out the categories. Next, we went through config.py and changed various hyperparameters that could potentially cause overfitting.

These hyperparameters were:

* BACKBONE: resnet50 >>> resnet101
* USE\_OBJECT\_ZOOM: True >>> False
* TRAIN\_ROIS\_PER\_IMAGE: 200 >>> 128
* LEARNING\_RATE: .001 >>> learning rate scheduler
* GRADIENT\_CLIP\_NORM: 5.0 >>> 10.0

Changing the backbone seemed to be the perfect solution at first. “As long as you have the computing power and time, this will help increase accuracy scores” or so they say. What we realized after training is that changing the backbone to one with increased layers (50 layers >>> 101 layers) is that increasing the amount of layers each image trains on would only be useful for a larger dataset and the training loss was decreasing at a much slower rate (We would have to nearly triple the amount of epochs trained on to get the loss where we wanted it).

Use\_object\_zoom we changed from true to false only because it was giving us errors. So this is something that could’ve caused overfitting.

Train\_rois\_per\_image was decreased since we found out that we wouldn’t need to train as many rois’ per image since we don’t have a lot of objects in each image for the training to detect. This should just help with total training time, but shouldn’t affect the accuracy scores at all.

Learning\_rate was put on a schedule. Basically, after 20 epochs, the learning rate would change. We started the learning rate at .01 and had it decrease to .0005 by epoch 80-100. This just made our loss scores staggered and was decreasing at too slow of a rate.

Gradient\_clip\_norm we still don’t fully understand. From what we have learned, Gradient\_clip\_norm helps accelerate the learning process. We noticed multiple websites saying to increase it as that helped with their loss, so we took their recommendation.

### **Attempt 4: Still trained on the same 3,600 images, changed all configurations back to default except learning rate.**

Our last attempt was still not where we wanted it to be. We noticed that the validation loss was staggered and our normal loss didn’t decrease enough. We knew this was because of the learning rate. To fix this, we used a tensorboard to constantly monitor the losses. This time around, we started the learning rate at .002, changed it to .001at epoch 50, then changed it to .0001 at epoch 70. This exponentially decreased our loss, but still made our validation loss staggered. This created overfitting which made the accuracy scores very good, but it wasn’t able to detect most of the categories correctly.

### **Possible solutions:**

The pre-trained model TACO is pretty good as is. Fine tuning their model might give it that last push it needs to be complete for only 1500 images. We would also suggest looking more into hyperparameter tuning as we did not grasp every different configuration for the time we had. So making small changes to those may help. There’s always the possibility to significantly increase the dataset. For that, I mean adding thousands of more images. Though that solution would be a lot more time consuming and less possible to pull off in a short period of time.