

Overview:

The given problem was to train and use an object detection model for detecting the number of products present in a shelf image.

After running the trained model on all test images, I have modified and stored 50 test images in the predictions folder for manual review.

Reproduction instructions:

The complete process takes about ~2 hours and the results can be reproduced on a system with >8GB RAM and a decent GPU with the following steps:

- Open cmd and move to the project directory (which contains the scripts)
- Use ``python data_prep.py``
- Open the research directory with ``cd models/research``
- Now, use ``cp object_detection/packages/tf2/setup.py .`` and then ``pip install .`` to install the object detection library
- Install protoc on your OS
- Use ``protoc object_detection/protos/*.proto --python_out=.``
- ``cd ../../`` for heading back to the project directory
- Now, execute ``python train_inference.py`` to train the model and make predictions
- We can see the predicted test images (with bounding boxes) in the ``predictions`` folder.

Dataset preparation:

The dataset preparation as outlined in the ``data_prep.py`` includes:

- Downloading the required images, dataset and the necessary libraries
- Making Pandas csv data for training and test images with filename and annotation details
- Converting the csv files to TFRecord for training with Tensorflow model

Detection Network used:

For the given scenario, I used an EfficientDet D0 baseline model with a SSD (Single Shot Detector) framework with Tensorflow. The model performed pretty well with an mAP of 0.79.

Hyper-parameters used:

The complete config used for training the network can be found in the `training/effdet.config` file. The major hyper-parameters used included:

- IoU: An IoU of 0.25 was found to be the optimal value with this network
- Confidence Threshold: I used a threshold of 0.5 for filtering out any unwanted bounding boxes
- Num_iterations: 2000 epochs
- Batch size: 2
- Max_boxes: 100

Anchor Box tuning:

The model uses a multi-scale anchor generator, which can take into account the different shapes of products present in some images (some are close shot pictures, some are far away) and generate the anchor box accordingly.

After the model training, an average aspect ratio of 0.1/0.1 - 0.1/0.2 was found for the bounding boxes used by the model

The following results were achieved by the model after half the training (i.e. 1000 iterations) with an mAP of 0.513. Further training upto 2000-3000 iterations yields an mAP of around 0.79:

```
Evaluate annotation type: bbox
DONE (t=13.04s).
Accumulating evaluation results...
DONE (t=0.06s).
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.513
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.988
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.455
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.513
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.017
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.162
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.590
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.590
I0604 20:13:15.091218 139695411066688 model_lib v2.py:1006] Eval metrics at step 2000
I0604 20:13:15.103453 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Precision/mAP: 0.512919
I0604 20:13:15.104419 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Precision/mAP@.50IOU: 0.988452
I0604 20:13:15.105148 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Precision/mAP@.75IOU: 0.454673
I0604 20:13:15.105821 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Precision/mAP (small): -1.000000
I0604 20:13:15.106449 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Precision/mAP (medium): -1.000000
I0604 20:13:15.107059 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Precision/mAP (large): 0.512919
I0604 20:13:15.107694 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Recall/AR@1: 0.016503
I0604 20:13:15.108337 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Recall/AR@10: 0.161934
I0604 20:13:15.109012 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Recall/AR@100: 0.590181
I0604 20:13:15.109651 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Recall/AR@100 (small): -1.000000
I0604 20:13:15.110278 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Recall/AR@100 (medium): -1.000000
I0604 20:13:15.110959 139695411066688 model_lib v2.py:1009] + DetectionBoxes_Recall/AR@100 (large): 0.590181
I0604 20:13:15.111532 139695411066688 model_lib v2.py:1009] + Loss/localization_loss: 0.006147
I0604 20:13:15.112107 139695411066688 model_lib v2.py:1009] + Loss/classification_loss: 0.138923
I0604 20:13:15.112705 139695411066688 model_lib v2.py:1009] + Loss/regularization_loss: 0.030473
I0604 20:13:15.113337 139695411066688 model_lib v2.py:1009] + Loss/total_loss: 0.175544
I0604 20:16:23.039222 139695411066688 checkpoint_utils.py:140] Waiting for new checkpoint at object_detection/training/effdet/
I0604 20:16:32.059099 139695411066688 checkpoint_utils.py:203] Timed-out waiting for a checkpoint.
```