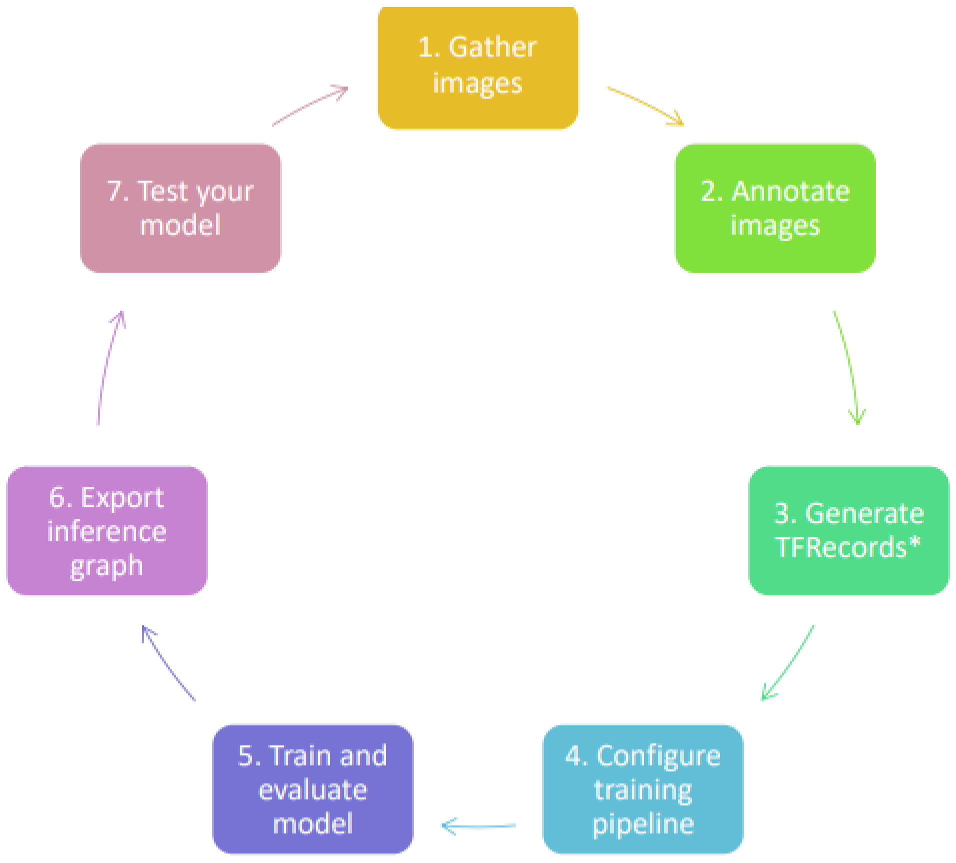
1. Gather images
2. Annotate images
3. Generate TFRecords*
4. Configure training pipeline
5. Train and evaluate model
6. Export inference graph
7. Test your model

- create dataset in Tensorflow Records format (TF Records)
- use a pretrained model to shorten training time required for object dection
- configure training pipeline of Tensorflow Object Detection API (TFOD API)
- train a custom object detector and monitor the training progress
- deploy the trained model for object detection

In the home directory, e.g. /home/ubuntu, create the following folder structure:

```
• raccoon_project
• -data
•  -images
•  -annotations
• -models
•  -model
•  -pretrained_model
```

The folders will be used for storing the files mentioned below:

- `data/images`: the .jpg files for training and validation
- `data/annotations`: the annotation files in .xml (Pascal VOC) format
- `data/`: label mapping file (e.g raccoon_label_map.pbtxt) and TFRecords
- `models/model`: all the training model checkpoints
- `models/pretrained_model`: the pretrained model checkpoints (e.g. ssd-mobilenet-v2 model checkpoints)
- `models`: training pipeline config file

- ## Create Label Map

- TFOD API requires a label map, which contains mapping of the used labels to an integer values. This label map is by both the training and detection processes.
- For example, if the dataset contains 2 labels, dogs and cats, then our label map file will have the following content:

```
item {
    id: 1
    name: 'cat'
}
item {
    id: 2
    name: 'dog'
}
```

- Since we are detecting only 1 object (raccoon), there is only 1 label required for our .pbtxt file. An example file is already provided for you called `raccoon_label_map.pbtxt`. Open the file and examine it. Copy the label map file from your lab folder to the `~/raccoon_project/data` directory.

- ## Download pretrained model

- Training a state of the art object detector from scratch can take days, even when using multiple GPUs! In order to speed up training, we'll take an object detector trained on a different dataset (COCO), and reuse some of it's parameters to initialize our new model. You can download the pre-trained model from Tensorflow model zoo.

```
## Download the pre-trained model to your home directory
cd ~
wget http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz

## unzip the model.tar.gz file
tar -xvf ssd_mobilenet_v2_coco_2018_03_29.tar.gz
```

- You will see a folder called 'ssd_mobilenet_v2_coco_2018_03_29' that contains several files including model.ckpt.*. These are checkpointed model that we will be using as pretrained model for our custom object detector.
- Copy all the contents inside the `ssd_mobilenet_v2_coco_2018_03_29` folder to `raccoon_project/models/pretrained_model` by:

```
## do this inside the ssd_mobilenet_v2_coco_2018_03_29 folder
cp -r * ~/raccoon_project/models/pretrained_model/
```

Configuring the Object Detection Pipeline

We will now configure the training pipeline for our object detector. Depending on which detector algorithm and the feature extractors you are using, the configuration will be different. To get started quickly, you can use the sample configuration available in the TFOD API directory: `<tfod_api_install_root>/models/research/object_detection/samples/configs`. Here you can see different combinations of detector algorithm and the feature extractor, e.g. ssd_mobilenet_v2_xx.config is for SSD using MobileNetV2 as feature extractor. An adapted config file has been made available in your lab folder called `train_pipeline.config`.
Open the `train_pipeline.config` file with an editor (e.g. `vi` or `nano` on Ubuntu) and modify those parameters that are marked: `<MODIFY THIS XXX>`

```
model {
    ssd {
        num_classes: 1
    ...
}


train_config: {
  ...
  fine_tune_checkpoint:
"/home/ubuntu/raccoon_project/models/pretrained_model/model.ckpt"
  ...
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "/home/ubuntu/raccoon_project/data/raccoon_train.record-00000-of-
00001"
  }
  label_map_path: "/home/ubuntu/raccoon_project/data/raccoon_label_map.pbtxt"
}
...

eval_input_reader: {
  tf_record_input_reader {
    input_path: "/home/ubuntu/raccoon_project/data/raccoon_val.record-00000-of-00001"
  }
  label_map_path: "/home/ubuntu/raccoon_project/data/raccoon_label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```
A sample file called `sample_train_pipeline.config` has been provided for you too. Use the sample if you do not want to modify the config file. Rename it as train_pipeline.config.
Copy the train_pipeline.config file to `raccoon_project/models`.
If you want to find out more information on configuring the pipeline, please refer to TFOD API documentation on pipeline for more information.

Start the training

You can start the training by running `model_main.py` from the directory `tensorflow/models/research directory`, and passing the various parameters such as config path, num of train steps, the directory to save the model to, etc. **Note**: on the server the tensorflow directory is located at `/home/ubuntu/git`)
```
# From the tensorflow/models/research/ directory
PIPELINE_CONFIG_PATH=/home/ubuntu/raccoon_project/models/train_pipeline.config
MODEL_DIR=/home/ubuntu/raccoon_project/models/model
NUM_TRAIN_STEPS=10000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1
python /home/ubuntu/git/tensorflow/models/research/object_detection/model_main.py \
    --pipeline_config_path=${PIPELINE_CONFIG_PATH} \
    --model_dir=${MODEL_DIR} \
    --num_train_steps=${NUM_TRAIN_STEPS} \
    --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES \
    --alsologtostderr
```
A script (`train.sh`) that contains the above has been created to avoid typing this repeatedly. Modify the script to change the PIPELINE_CONFIG and MODEL_DIR accordingly. Copy the modified `train.sh` script to `tensorflow/models/research` directory.
To make the script executable, you need to do the following:
```
chmod +x train.sh
```
You may see a lot of warning messages and you can safely ignore those (most of them are due to deprecation warning). If everything goes smoothly, you will start seeing the following training output:

Monitoring Training Progress with Tensorboard

You can monitor progress of the training and eval jobs by running Tensorboard on your local machine:

Open another terminal and run the tensorboard and specify the model directory as logdir.

```
tensorboard --logdir=/home/ubuntu/raccoon_project/models/model
```

Once Tensorboard is running, navigate to `<serverIP>:6006` from your favourite web browser. (if you are running this in the cloud VM, and accessing it from school computer, port 6006 is blocked. You should access it by using `https://<serverIP>/tensorboard`, using the reverse proxy we have setup on the cloud VM)
You should be able see various charts such as following (after training it for 5 to 10 minutes):

Here you can see the plots for the varios mAP metrics over the training steps.

You can also see the evaluation results on the images by selecting the Images tab. Here you can see a comparison of the ground truth bounding boxes (right) and the predicted bounding boxes (left). Initially you will see that the bounding boxes were not very accurately drawn and may have multiple detections. But as training progressed, the detection should get better and better.

Stop Training

Your training can take quite a while (1 to 2 hours). You can determine if you can stop the training by looking at the validation loss or mAP. If the validation loss has plateued out for a few epochs, you can probably stop the training.

Exporting the Tensorflow Graph

After your model has been trained, you should export it to a Tensorflow graph proto. First, you need to identify a candidate checkpoint to export. The checkpoint is stored in your model directory and will typically consist of three files:

```
model.ckpt-${CHECKPOINT_NUMBER}.data-00000-of-00001
model.ckpt-${CHECKPOINT_NUMBER}.index
model.ckpt-${CHECKPOINT_NUMBER}.meta
```

${CHECKPOINT_NUMBER} corresponds to the training step. Pick a checkpoint that has the lowest validation loss or best mAP (or whatever metrics you choose). Run the following command from tensorflow/models/research/:

Assuming we have identified checkpoint-1000 to export, and assuming we used the above folder structure:

```
# From tensorflow/models/research/
python object_detection/export_inference_graph.py \
    --input_type image_tensor \
    --pipeline_config_path  /home/ubuntu/raccoon_project/models/train_pipeline.config 
\
    --trained_checkpoint_prefix /home/ubuntu/raccoon_project/models/model/model.ckpt-
10000 \
    --output_directory  /home/ubuntu/raccoon_project/models/exported_graphs
```

Afterwards, you should see a directory named exported_graphs containing the SavedModel and frozen graph.

Using our trained model

We have already trained the raccoon detector on a GPU server. You can try out our trained model. Change directory to your home directory and download our models using:

```
wget https://sdaaidata.s3-ap-southeast-1.amazonaws.com/pretrained-
weights/iti107/session-5/raccoon_trained_models.tar.gz
tar -xvf raccoon_trained_models.tar.gz
```

You will see a directory called `model` and inside the model directory, you should see some checkpoints, e.g. model.chkpt-26972.*. Assume the best model is the model.chkpt-26972,
Before running the following script, make sure you have stopped the training process on the GPU (on a CPU, there is no need to stop), as it will make use of the same GPU memory that you are running your training process, and it will throw a out-of-memory error.

```
# From tensorflow/models/research/
python object_detection/export_inference_graph.py \
    --input_type image_tensor \
    --pipeline_config_path  /home/ubuntu/raccoon_project/models/train_pipeline.config 
\
    --trained_checkpoint_prefix /home/ubuntu/model/model.ckpt-26972 \
    --output_directory  /home/ubuntu/raccoon_project/models/exported_graphs
```

Test your custom model

Now you are ready to test your trained model. Run the provided notebook `object_detection_using_tfod_api.ipynb` to run your raccoon detector!