TKO_7096-3001 :Computer Vision & Sensor Fusion

Lecture 6-2

Fahimeh Farahnakian

University of Turku, Finland

https://users.utu.fi/fahfar/

Email: fahfar@utu.fi

TensorFlow Object Detection API

https://github.com/tensorflow/models/tree/master/research/object_detection

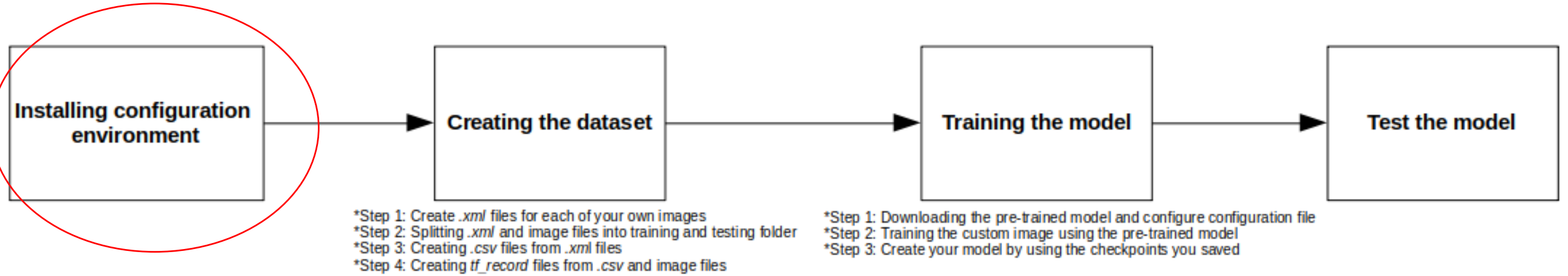
- **TensorFlow Detection API** is an **open source** framework built on top of TensorFlow that helps build, train and deploy object detection models.
- Instead of training your own model from scratch, you can build on existing models and fine-tune them for your own purpose without requiring as much computing power.

Detecting objets in images and videos

For whom?

Very powerful tool that can quickly enable anyone with less or no knowledge on machine learning to build and deploy powerful object detection model.

Tensorflow Object Detection API flowchart



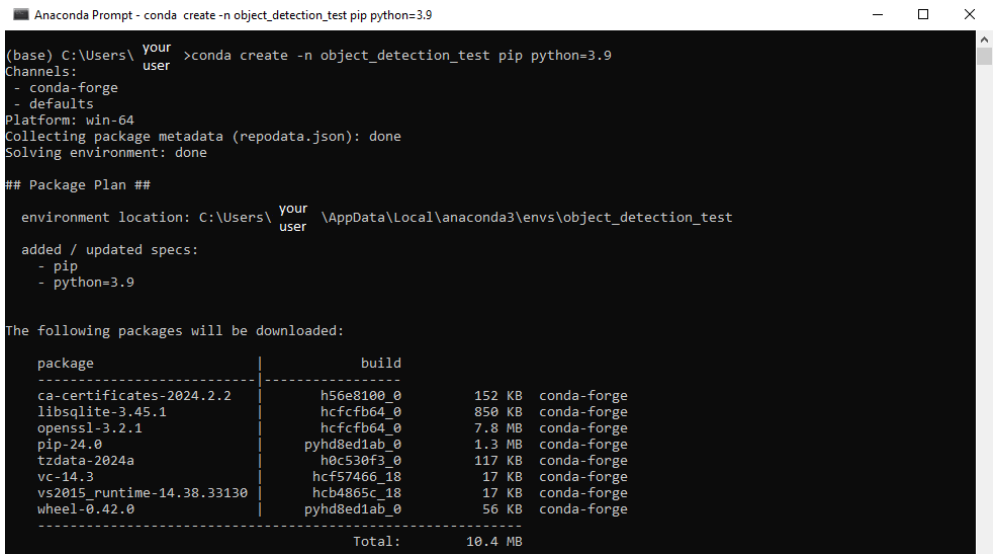
Step 1: Libraries Required

Tensorflow object detection API depends on the following libraries:

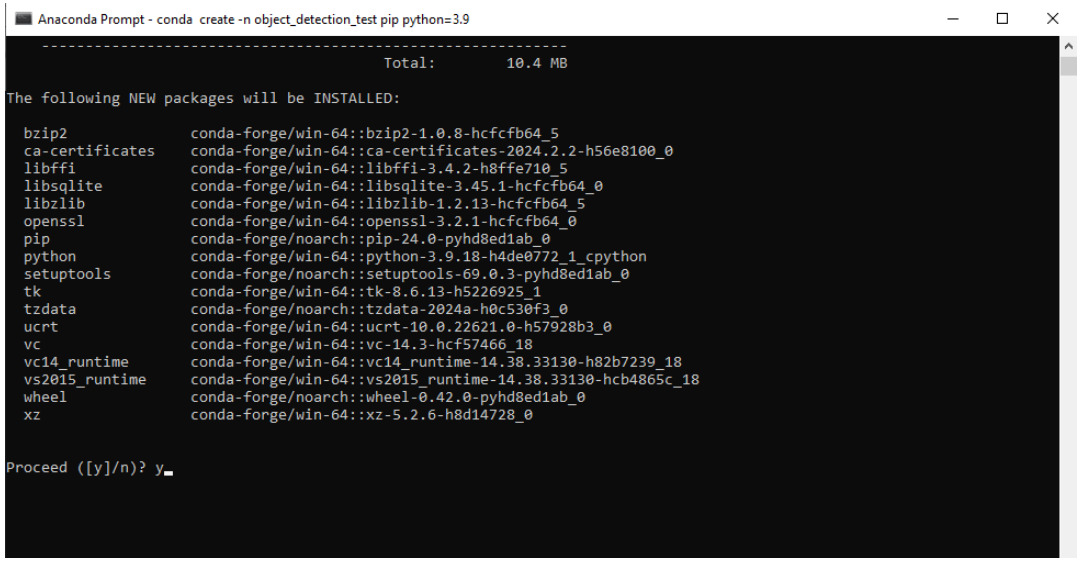
- ✓Protobuf
- ✓Python
- ✓Pillow
- ✓Lxml
- ✓Tf Slim
- ✓Matplotlib
- ✓ Tensorflow
- ✓Cocoapi
- ✓Jyputer Notebook
- ✓Cython

Step 1: Installation steps:

1. First step is to go to the following link: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html
2. Go the installation process step by step according to your environment



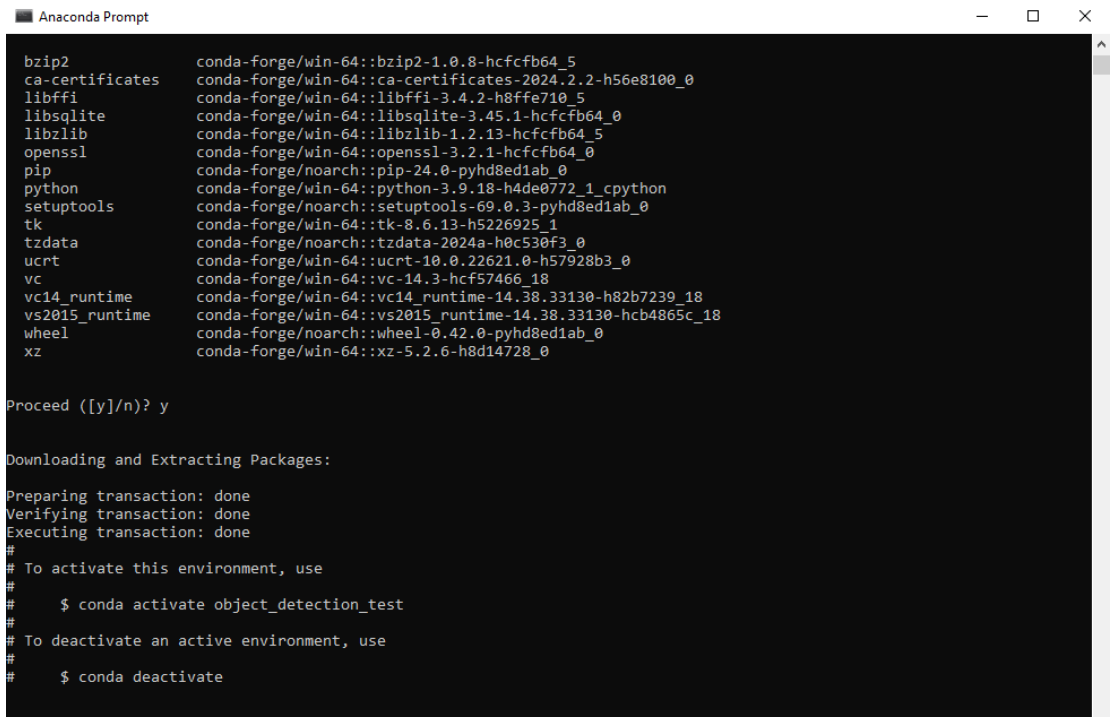
Step 1



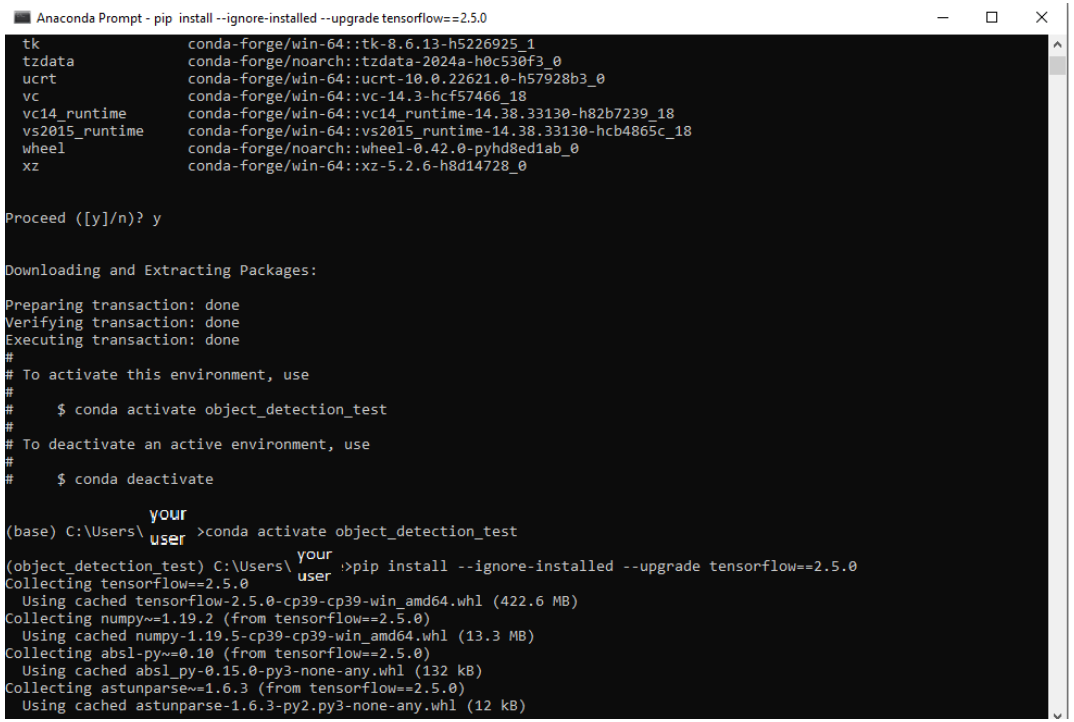
Step 2

Step 1: Installation steps:

Step 3



Step 4



Conda activate object_detection_env

pip install --ignore-installed –upgrade tensorflow==2.5.0

Step 1: Installation steps:

Downloading the TensorFlow Model Garden

Create a new folder under a path of your choice and name it `TensorFlow`. (e.g. `C:\Users\sglvladi\Documents\TensorFlow`).

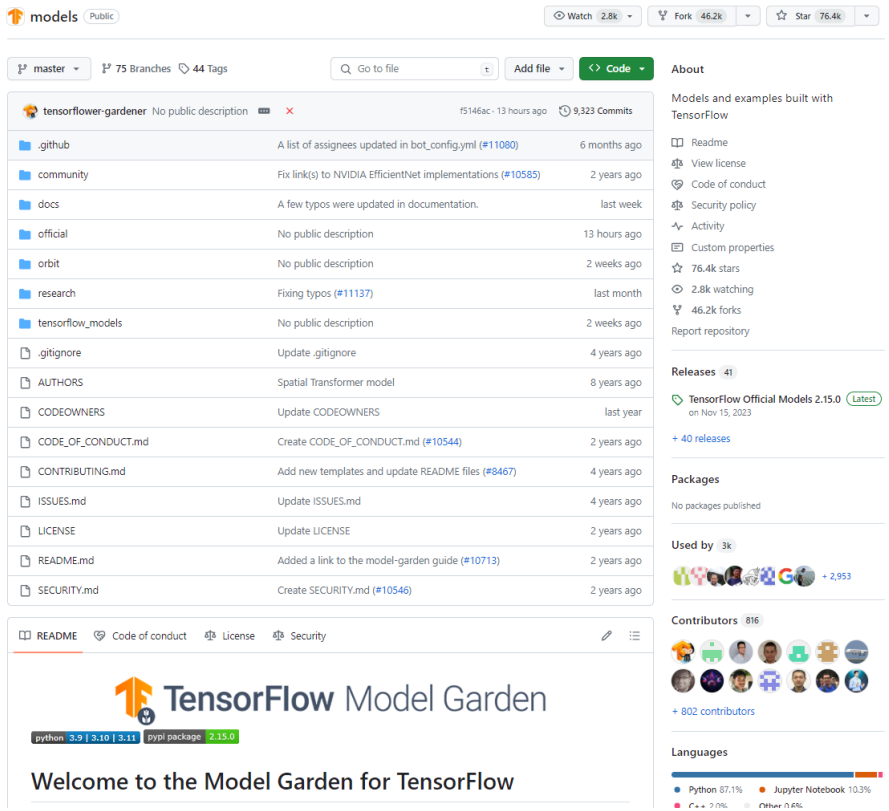
From your *Terminal* `cd` into the `TensorFlow` directory.

To download the models you can either use Git to clone the TensorFlow Models repository inside the `TensorFlow` folder, or you can simply download it as a ZIP and extract its contents inside the `TensorFlow` folder. To keep things consistent, in the latter case you will have to rename the extracted folder `models-master` to `models`.

You should now have a single folder named `models` under your `TensorFlow` folder, which contains another 4 folders as such:

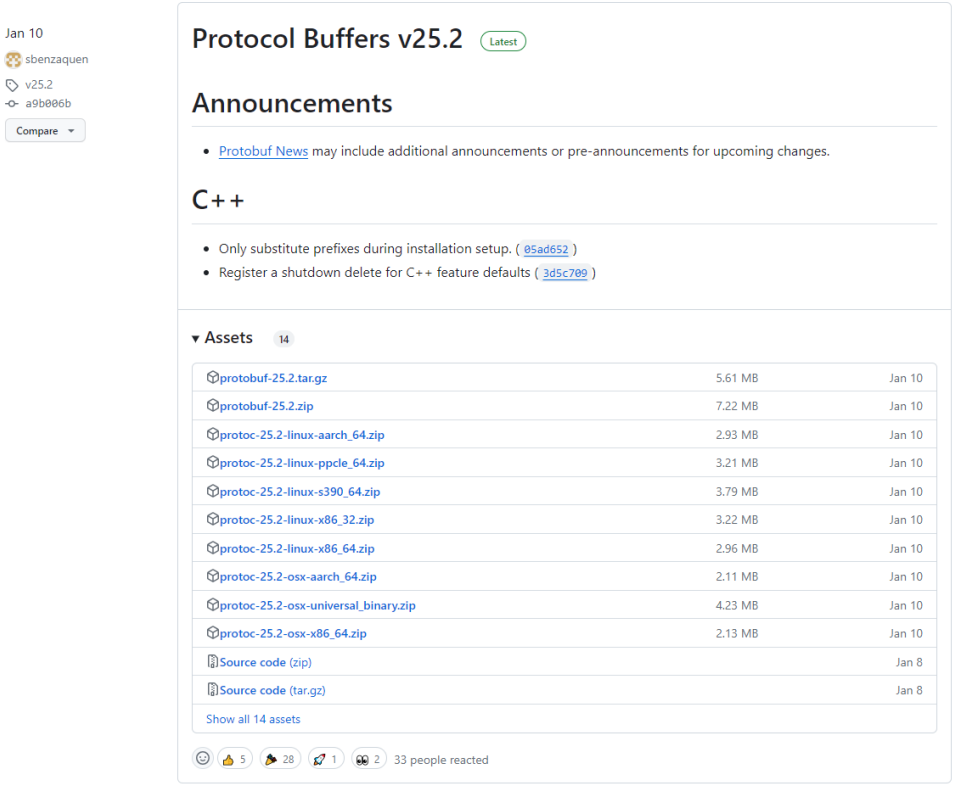
```
TensorFlow/
└─ models/
   ├─ community/
   ├─ official/
   ├─ orbit/
   ├─ research/
   └─ ...
```

Step 5

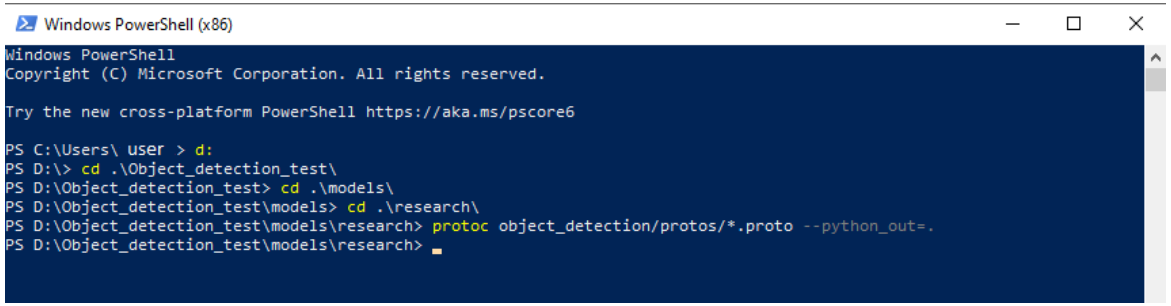


Link:
https://github.com/tensorflow/models

Step 1: Installation steps:



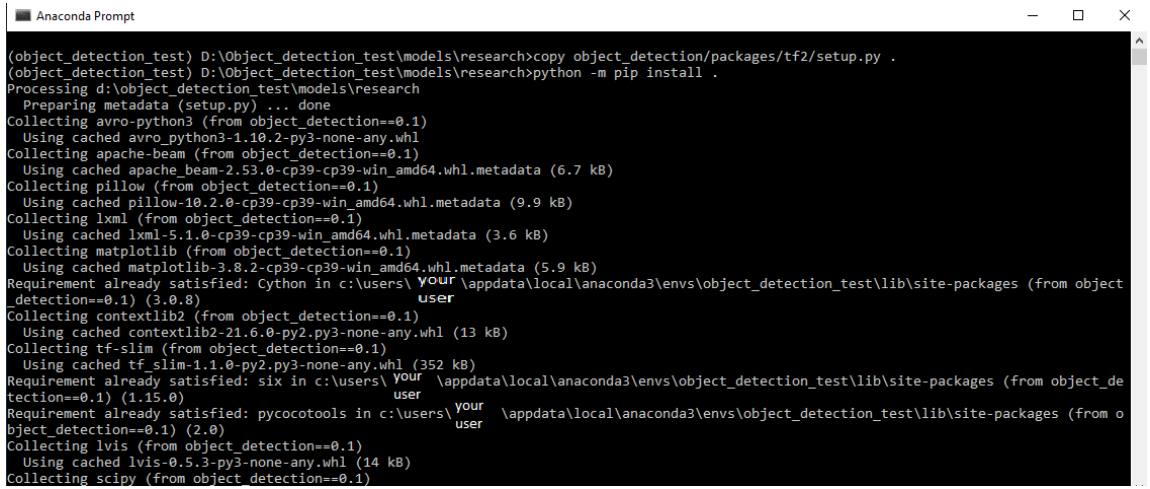
Step 6
Link:
https://github.com/protocolbuffers/protobuf/releases



Step 7

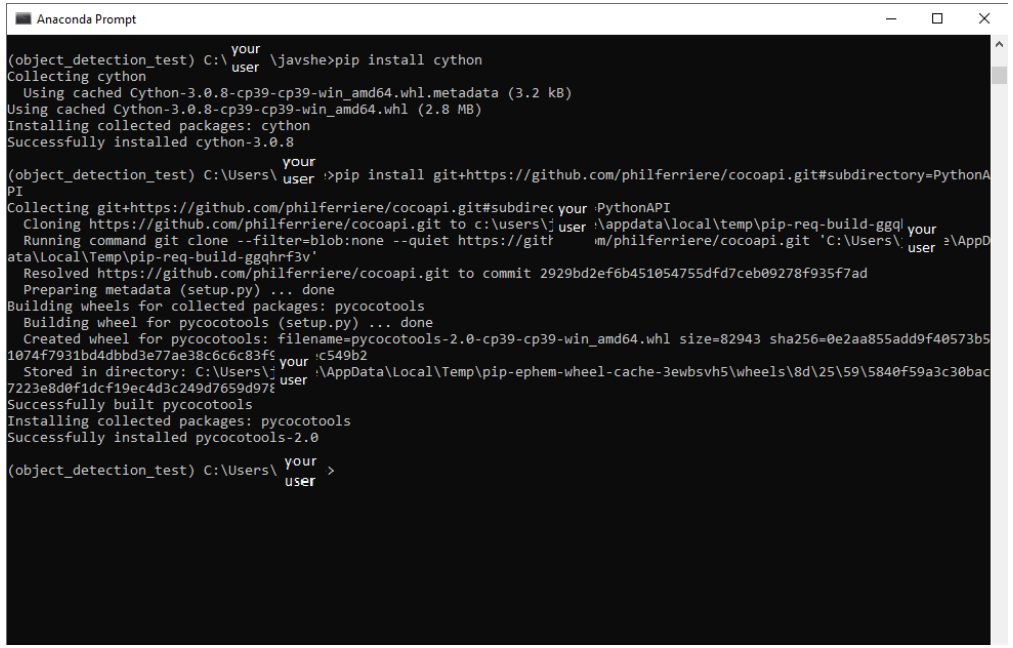
Step 1: Installation steps:

Step 8



copy object_detection/packages/tf2/setup.py .
python –m pip install .

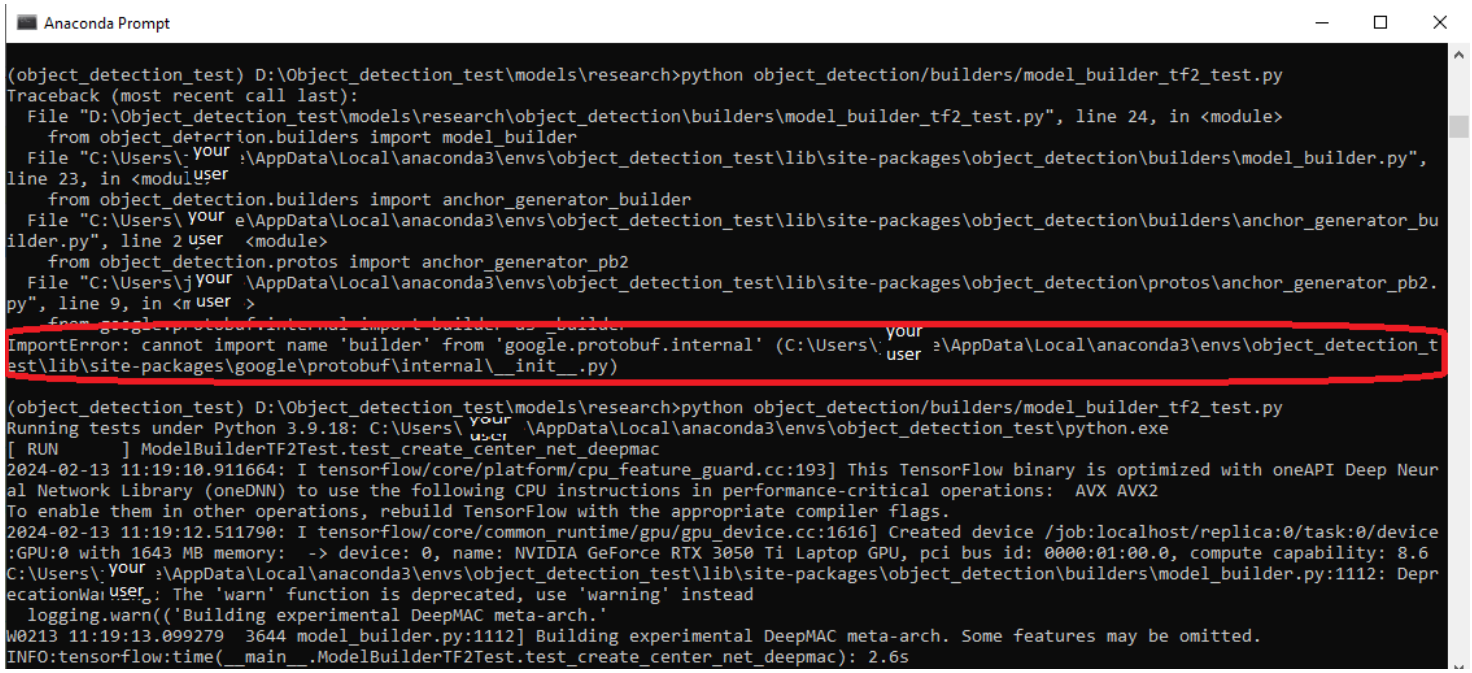
Step 9



pip install cython
pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI

Step 1: Installation steps:

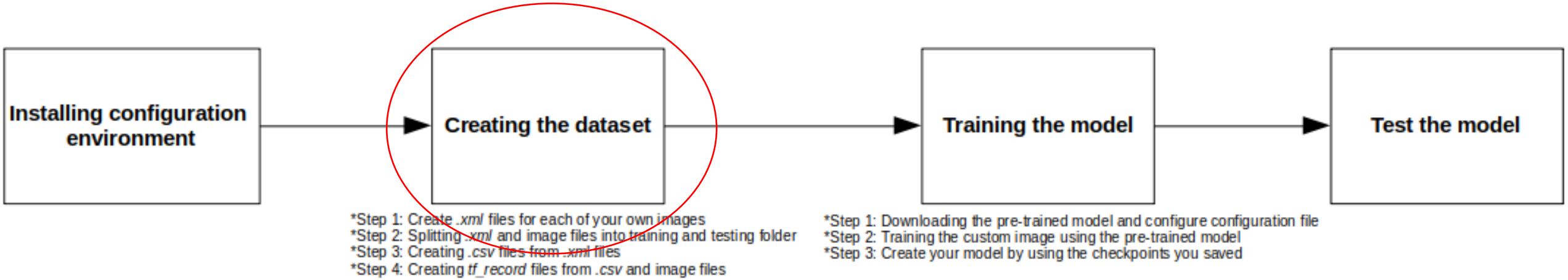
Possible complication



Solution:

- Copy the builder.py file from moodle to the location inside the red box

Tensorflow Object Detection API flowchart

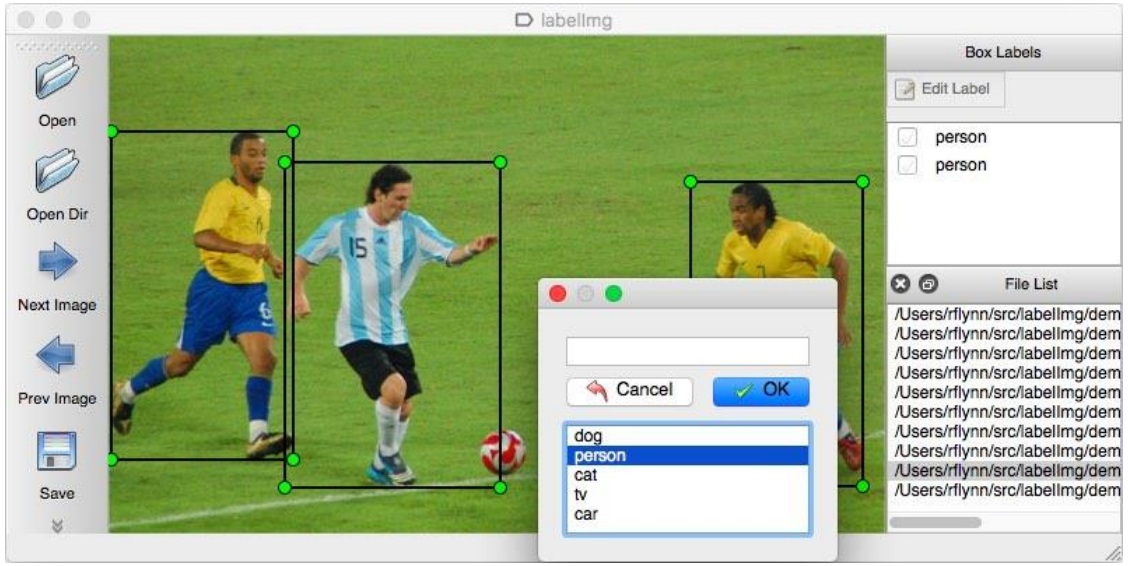


Step2: Creating the dataset (1/3)

1. Gather a dataset: split data into training and test dataset

2. Label your images: You need the height, width and class of each object to train your object detection model. This includes the associated xmin, xmax, ymin, and ymax bounding boxes.

- To help you create these labels, you can use an image labelling software like LabelImg, an open source program that saves an XML label for each image. You can then convert them into a CSV table for training.
- LabelImg provides a user-friendly GUI.



Step2: Creating the dataset (2/3)

3. Create Label Map (.pbtxt)

Classes need to be listed in the label map. The label map should contain only two items like the following for dog and cat dataset:

item {
id: 1
name: 'Dog'

id: 2
name: 'Cat'
}

- Note that id must start from 1, because 0 is a reserved id.

4. Convert XML to CSV file(.csv)

- We have all images and their bounding boxes are in XML format when they are labelled with LabelImg tool.
- You can use this link to create XML files to CSV which contains all the XML files and their bounding box co-ordinates to single CSV file which is input for creating TFrecords.

Step2: Creating the dataset (3/3)

5. Create TFRecord (.record)

- TFRecord is an important data format designed for Tensorflow. (Read more about it here). Before you can train your custom object detector, you must convert your data into the TFRecord format.
- Since we need to train as well as validate our model, the data set will be split into training (train.record) and validation sets (val.record). The purpose of training set is straight forward - it is the set of examples the model learns from. The validation set is a set of examples used DURING TRAINING to iteratively assess model accuracy.
- You can use create_tf_record.py to convert our data set into train.record and val.record. Download here and save it to models/research/object_detection/dataset_tools/.

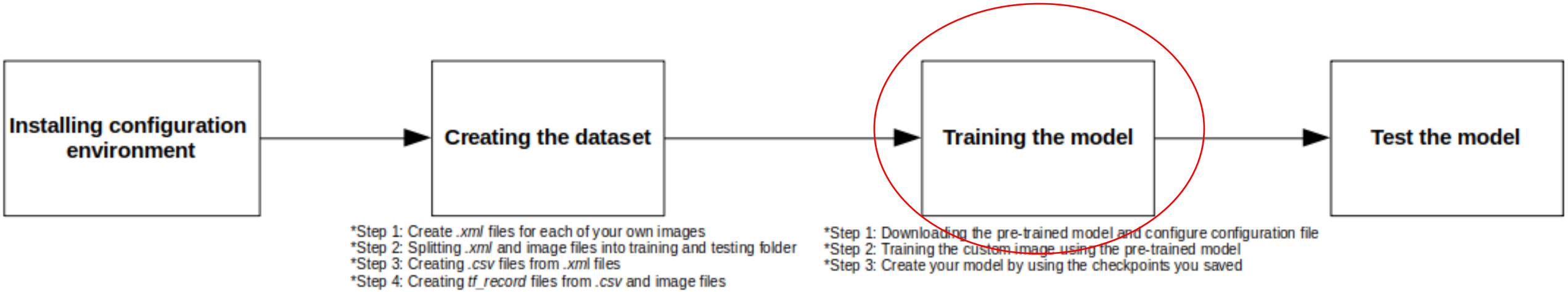
Just change the label name in if row_label == 'Label1': as per your classificatications.

From the models directory

$ python research/object_detection/dataset_tools/create_tf_record.py

- If the script is executed successfully, train.record and val.record

Tensorflow Object Detection API flowchart



Step3: Training the model (1/4)

1. Download a pre-trained model and configure configuration file

- There are many pre-trained object detection models available in the model zoo. In order to train them using our custom data set, the models need to be *restored* in Tensorflow using their checkpoints (.ckpt files), which are records of previous model states. (model.ckpt.meta, model.ckpt.index, model.ckpt.data-00000-of-00001)

- Each of the pretrained models has a config file that contains details about the model. To detect our custom class, the config file needs to be modified accordingly.
- The config files are included in the models directory you cloned in the very beginning. You can find them in:

  models/research/object_detection/samples/configs

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
CenterNet MobileNetV2 FPN 512x512	6	23.4	Boxes
CenterNet MobileNetV2 FPN Keypoints 512x512	6	41.7	Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes
EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes
EfficientDet D7 1536x1536	325	51.2	Boxes
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3	Boxes
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38.3	Boxes
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29.3	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31.0	Boxes
Faster R-CNN ResNet50 V1 800x1333	65	31.6	Boxes
Faster R-CNN ResNet101 V1 640x640	55	31.8	Boxes
Faster R-CNN ResNet101 V1 1024x1024	72	37.1	Boxes
Faster R-CNN ResNet101 V1 800x1333	77	36.6	Boxes
Faster R-CNN ResNet152 V1 640x640	64	32.4	Boxes
Faster R-CNN ResNet152 V1 1024x1024	85	37.6	Boxes
Faster R-CNN ResNet152 V1 800x1333	101	37.4	Boxes
Faster R-CNN Inception ResNet V2 640x640	206	37.7	Boxes
Faster R-CNN Inception ResNet V2 1024x1024	236	38.7	Boxes
Mask R-CNN Inception ResNet V2 1024x1024	301	39.0/34.6	Boxes/Masks
ExtremeNet (deprecated)	--	--	Boxes
ExtremeNet	--	--	Boxes

Step3: Training the model (2/4)

Here are the items we need to change:

- Since we're only trying to detect cats and dogs, change **num_classes** to **2** (Dogs & Cats).
- fine_tune_checkpoint tells the model which checkpoint file to use.
- The model also needs to know where the TFRecord files and label maps are for both training and validation sets. Since our train.record and val.record are saved in tf_record folder, our config should reflect that:

train_input_reader: {
tf_record_input_reader {
input_path: "tf_record/train.record"
}
label_map_path: "annotations/label_map.pbtxt"
}eval_input_reader: {
tf_record_input_reader {
input_path: "tf_record/val.record"
}
label_map_path: "annotations/label_map.pbtxt"
shuffle: false
num_readers: 1
}

Step3: Training the model (3/4)

At this point, your models directory should look like this:

```
models
├── annotations
│   ├── label_map.pbtxt
│   ├── trainval.txt
│   └── xmls
│   ├── 1.xml
│   ├── 2.xml
│   ├── ...
│
├── images
│   ├── 1.jpg
│   ├── 2.jpg
│   ├── ...
│
├── checkpoints
│   ├── model.ckpt.data-00000-of-00001
│   ├── model.ckpt.index
│   └── model.ckpt.meta
│
├── tf_record
│   ├── train.record
│   └── val.record
│
├── research
│   ├── ...
...
```

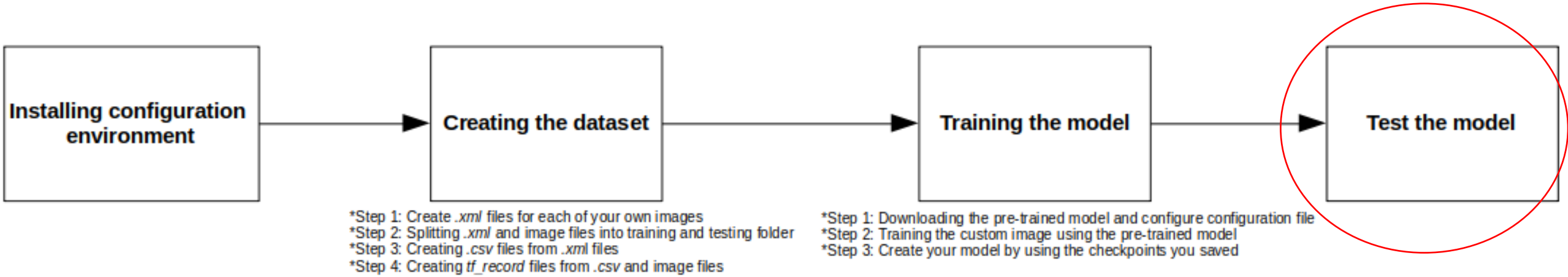
Step3: Training the model (4/4)

If you have successfully completed all previous steps, you're ready to start training!

Follow the steps below:

- **Change into the models directory**
  $ cd tensorflow/models#
- **Make directory for storing training progress**
  $ mkdir train
- **Make directory for storing validation results**
  $ mkdir eval
- **Begin training with TensorFlow 2 Object Detector**
  $ python /content/models/research/object_detection/model_main_tf2.py
  --pipeline_config_path={pipeline_file} \
  --model_dir={model_dir} \
  --alsologtostderr \
  --num_train_steps={num_steps} \
  --sample_1_of_n_eval_examples=1 \
  --num_eval_steps={num_eval_steps}

Tensorflow Object Detection API flowchart



Step4: Test the model (1/3)

- **Evaluation**

Evaluation can be run in parallel with training. The *eval.py* script checks the train directory for progress and evaluate the model

```
# From the models directory$ python research/object_detection/legacy/eval.py \
--logtostderr \
--pipeline_config_path=ssd_mobilenet_v2_coco.config \
--checkpoint_dir=train \
--eval_dir=eval
```

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.457
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.729
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.502
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.122
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.297
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.659
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.398
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.559
 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 ] = 0.236
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.486
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.746
INFO:tensorflow:Writing metrics to tf summary.
INFO:tensorflow:DetectionBoxes_Precision/mAP: 0.456758
INFO:tensorflow:DetectionBoxes_Precision/mAP (large): 0.659280
INFO:tensorflow:DetectionBoxes_Precision/mAP (medium): 0.296693
INFO:tensorflow:DetectionBoxes_Precision/mAP (small): 0.122108
INFO:tensorflow:DetectionBoxes_Precision/mAP@.50IOU: 0.728587
INFO:tensorflow:DetectionBoxes_Precision/mAP@.75IOU: 0.502194
INFO:tensorflow:DetectionBoxes_Recall/AR@1: 0.397509
INFO:tensorflow:DetectionBoxes_Recall/AR@10: 0.558966
INFO:tensorflow:DetectionBoxes_Recall/AR@100: 0.590182
INFO:tensorflow:DetectionBoxes_Recall/AR@100 (large): 0.745691
INFO:tensorflow:DetectionBoxes_Recall/AR@100 (medium): 0.485964
INFO:tensorflow:DetectionBoxes_Recall/AR@100 (small): 0.236275
INFO:tensorflow:Losses/Loss/BoxClassifierLoss/classification_loss: 0.234645
INFO:tensorflow:Losses/Loss/BoxClassifierLoss/localization_loss: 0.139109
INFO:tensorflow:Losses/Loss/RPNLoss/localization_loss: 0.603733
INFO:tensorflow:Losses/Loss/RPNLoss/objectness_loss: 0.206419
```

Step4: Test the model (2/3)

- **Model export**

Once you finish training your model, you can export your model to be used for inference. If you've been following the folder structure, use the following command:

 # From the models directory

$ mkdir fine_tuned_model

$ python research/object_detection/export_inference_graph.py \
--input_type image_tensor \
--pipeline_config_path ssd_mobilenet_v2_coco.config \
--trained_checkpoint_prefix train/model.ckpt-<the_highest_checkpoint_number> \
--output_directory fine_tuned_model

Step4: Test the model (3/2)

- **Classify images**

Now that you have a model, you can use it to detect in test pictures and videos!

The models directory came with a notebook file (.ipynb) that we can use to get inference. It is located at models/research/object_detection/colab_tutorials/ object_detection_tutorial.ipynb. Follow the steps below to tweak the notebook:

1. MODEL_NAME = 'ssd_mobilenet_v2_coco_2018_03_29'
2. PATH_TO_CKPT = 'path/to/your/frozen_inference_graph.pb'
3. PATH_TO_LABELS = 'models/annotations/label_map.pbtxt'
4. NUM_CLASSES = 2
5. Comment out cell #5 completely (just below Download Model)
6. Since we're only testing on one image, comment out PATH_TO_TEST_IMAGES_DIR and TEST_IMAGE_PATHS in cell #9 (just below Detection)
7. In cell #11 (the last cell), remove the for-loop, unindent its content, and add path to your test image:

imagepath = 'path/to/image_you_want_to_test.jpg

After following through the steps, run the notebook and you should see your object in your test image highlighted by a bounding box!

How to train your own Object Detector with TensorFlow's Object Detector API

- https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9