TensorFlow setup Documentation

Lyudmil Vladimirov

Sep 15, 2020

CONTENTS

Important: This tutorial is intended for TensorFlow 2.2, which (at the time of writing this tutorial) is the latest stable version of TensorFlow 2.x.

A version for TensorFlow 1.14 can be found here.

This is a step-by-step tutorial/guide to setting up and using TensorFlow's Object Detection API to perform, namely, object detection in images/video.

The software tools which we shall use throughout this tutorial are listed in the table below:

Target Software versions	
OS	Windows, Linux
Python	3.8
TensorFlow	2.2.0
CUDA Toolkit	10.1
CuDNN	7.6.5
Anaconda	Python 3.7 (Optional)

CHAPTER

ONE

INSTALLATION

1.1 General Remarks

- In contrast to TensorFlow 1.x, where different Python packages needed to be installed for one to run TensorFlow on either their CPU or GPU (namely `tensorflow` and `tensorflow-gpu`), TensorFlow 2.x only requires that the `tensorflow` package is installed and automatically checks to see if a GPU can be successfully registered.

1.2 Anaconda Python 3.7 (Optional)

Although having Anaconda is not a requirement in order to install and use TensorFlow, I suggest doing so, due to it's intuitive way of managing packages and setting up new virtual environments. Anaconda is a pretty useful tool, not only for working with TensorFlow, but in general for anyone working in Python, so if you haven't had a chance to work with it, now is a good chance.

1.2.1 Install Anaconda Python 3.7

Windows

- Go to https://www.anaconda.com/products/individual and click the "Download" button
- Download the Python 3.7 64-Bit Graphical Installer or the 32-Bit Graphical Installer installer, per your system requirements
- Run the downloaded executable (`.exe`) file to begin the installation. See here for more details
- (Optional) In the next step, check the box "Add Anaconda3 to my PATH environment variable". This will make Anaconda your default Python distribution, which should ensure that you have the same default Python distribution across all editors.

Linux

- Go to https://www.anaconda.com/products/individual and click the "Download" button
- Download the Python 3.7 64-Bit (x86) Installer
- Run the downloaded bash script (`.sh`) file to begin the installation. See here for more details.
- When prompted with the question "Do you wish the installer to prepend the Anaconda<2 or 3> install location to PATH in your /home/<user>/.bashrc ?", answer "Yes". If you enter "No", you must manually add the path to Anaconda or conda will not work.

1.2.2 Create a new Anaconda virtual environment

- Open a new *Terminal* window
- Type the following command:

```
conda create -n tensorflow pip python=3.8
```

- The above will create a new virtual environment with name `tensorflow`

Important: The term *Terminal* will be used to refer to the Terminal of your choice (e.g. Command Prompt, Powershell, etc.)

1.2.3 Activate the Anaconda virtual environment

- Activating the newly created virtual environment is achieved by running the following in the *Terminal* window:

```
conda activate tensorflow
```

- Once you have activated your virtual environment, the name of the environment should be displayed within brackets at the beggining of your cmd path specifier, e.g.:

```
(tensorflow) C:\Users\sglvladi>
```

Important: Throughout the rest of the tutorial, execution of any commands in a *Terminal* window should be done after the Anaconda virtual environment has been activated!

1.3 TensorFlow Installation

Getting setup with an installation of TensorFlow can be done in 3 simple steps.

1.3.1 Install the TensorFlow PIP package

- Run the following command in a *Terminal* window:

```
pip install --ignore-installed --upgrade tensorflow==2.2.0
```

1.3.2 Verify your Installation

- Run the following command in a *Terminal* window:

```
python -c "import tensorflow as tf;print(tf.reduce_sum(tf.random.
↪normal([1000, 1000])))"
```

- Once the above is run, you should see a print-out similar to the one bellow:

```
2020-06-22 19:20:32.614181: W tensorflow/stream_executor/platform/default/
↪dso_loader.cc:55] Could not load dynamic library 'cudart64_101.dll';␣
↪dlerror: cudart64_101.dll not found
2020-06-22 19:20:32.620571: I tensorflow/stream_executor/cuda/cudart_stub.
↪cc:29] Ignore above cudart dlerror if you do not have a GPU set up on␣
↪your machine.
2020-06-22 19:20:35.027232: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library nvcuda.dll
2020-06-22 19:20:35.060549: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1561] Found device 0 with properties:
pciBusID: 0000:02:00.0 name: GeForce GTX 1070 Ti computeCapability: 6.1
coreClock: 1.683GHz coreCount: 19 deviceMemorySize: 8.00GiB␣
↪deviceMemoryBandwidth: 238.66GiB/s
2020-06-22 19:20:35.074967: W tensorflow/stream_executor/platform/default/
↪dso_loader.cc:55] Could not load dynamic library 'cudart64_101.dll';␣
↪dlerror: cudart64_101.dll not found
2020-06-22 19:20:35.084458: W tensorflow/stream_executor/platform/default/
↪dso_loader.cc:55] Could not load dynamic library 'cublas64_10.dll';␣
↪dlerror: cublas64_10.dll not found
2020-06-22 19:20:35.094112: W tensorflow/stream_executor/platform/default/
↪dso_loader.cc:55] Could not load dynamic library 'cufft64_10.dll';␣
↪dlerror: cufft64_10.dll not found
2020-06-22 19:20:35.103571: W tensorflow/stream_executor/platform/default/
↪dso_loader.cc:55] Could not load dynamic library 'curand64_10.dll';␣
↪dlerror: curand64_10.dll not found
2020-06-22 19:20:35.113102: W tensorflow/stream_executor/platform/default/
↪dso_loader.cc:55] Could not load dynamic library 'cusolver64_10.dll';␣
↪dlerror: cusolver64_10.dll not found
2020-06-22 19:20:35.123242: W tensorflow/stream_executor/platform/default/
↪dso_loader.cc:55] Could not load dynamic library 'cusparse64_10.dll';␣
↪dlerror: cusparse64_10.dll not found
2020-06-22 19:20:35.140987: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-06-22 19:20:35.146285: W tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1598] Cannot dlopen some GPU libraries. Please make sure the␣
↪missing libraries mentioned above are installed properly if you would␣
↪like to use GPU. Follow the guide at https://www.tensorflow.org/install/
↪gpu for how to download and setup the required libraries for your␣
↪platform.
Skipping registering GPU devices...
2020-06-22 19:20:35.162173: I tensorflow/core/platform/cpu_feature_guard.
↪cc:143] Your CPU supports instructions that this TensorFlow binary was␣
↪not compiled to use: AVX2
2020-06-22 19:20:35.178588: I tensorflow/compiler/xla/service/service.
↪cc:168] XLA service 0x15140db6390 initialized for platform Host (this␣
↪does not guarantee that XLA will be used). Devices:
2020-06-22 19:20:35.185082: I tensorflow/compiler/xla/service/service.
↪cc:176]   StreamExecutor device (0): Host, Default Version
2020-06-22 19:20:35.191117: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1102] Device interconnect StreamExecutor with strength 1 edge␣
↪matrix:
2020-06-22 19:20:35.196815: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1108]
tf.Tensor(1620.5817, shape=(), dtype=float32)
```

1.3.3 GPU Support (Optional)

Although using a GPU to run TensorFlow is not necessary, the computational gains are substantial. Therefore, if your machine is equipped with a compatible CUDA-enabled GPU, it is recommended that you follow the steps listed below to install the relevant libraries necessary to enable TensorFlow to make use of your GPU.

By default, when TensorFlow is run it will attempt to register compatible GPU devices. If this fails, TensorFlow will resort to running on the platform's CPU. This can also be observed in the printout shown in the previous section, under the "Verify the install" bullet-point, where there are a number of messages which report missing library files (e.g. `Could not load dynamic library 'cudart64_101.dll'; dlerror: cudart64_101.dll not found`).

In order for TensorFlow to run on your GPU, the following requirements must be met:

Prerequisites
Nvidia GPU (GTX 650 or newer)
CUDA Toolkit v10.1
CuDNN 7.6.5

1.3.3.1 Install CUDA Toolkit

Windows

- Follow this link to download and install CUDA Toolkit 10.1
- Installation instructions can be found here

Linux

- Follow this link to download and install CUDA Toolkit 10.1 for your Linux distribution.
- Installation instructions can be found here

1.3.3.2 Install CUDNN

Windows

- Go to https://developer.nvidia.com/rdp/cudnn-download
- Create a user profile if needed and log in
- Select cuDNN v7.6.5 (Nov 5, 2019), for CUDA 10.1
- Download cuDNN v7.6.5 Library for Windows 10
- Extract the contents of the zip file (i.e. the folder named `cuda`) inside `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.1\`, where `<INSTALL_PATH>` points to the installation directory specified during the installation of the CUDA Toolkit. By default `<INSTALL_PATH>` = `C:\Program Files`.

Linux

- Go to https://developer.nvidia.com/rdp/cudnn-download
- Create a user profile if needed and log in
- Select cuDNN v7.6.5 (Nov 5, 2019), for CUDA 10.1
- Download cuDNN v7.6.5 Library for Linux
- Follow the instructions under Section 2.3.1 of the CuDNN Installation Guide to install CuDNN.

1.3.3.3 Environment Setup

Windows

- Go to *Start* and Search "environment variables"
- Click "Edit the system environment variables". This should open the "System Properties" window
- In the opened window, click the "Environment Variables…" button to open the "Environment Variables" window.
- Under "System variables", search for and click on the `Path` system variable, then click "Edit…"
- Add the following paths, then click "OK" to save the changes:
  - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.1\bin`
  - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.1\libnvvp`
  - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.1\extras\CUPTI\libx64`
  - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.1\cuda\bin`

Linux

As per Section 7.1.1 of the CUDA Installation Guide for Linux, append the following lines to `~/.bashrc`:

```
# CUDA related exports
export PATH=/usr/local/cuda-10.1/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda-10.1/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_
↪PATH}}
```

1.3.3.4 Update your GPU drivers (Optional)

If during the installation of the CUDA Toolkit (see *Install CUDA Toolkit*) you selected the *Express Installation* option, then your GPU drivers will have been overwritten by those that come bundled with the CUDA toolkit. These drivers are typically NOT the latest drivers and, thus, you may wish to update your drivers.

- Go to http://www.nvidia.com/Download/index.aspx
- Select your GPU version to download
- Install the driver for your chosen OS

1.3.3.5 Verify the installation

- Run the following command in a **NEW** *Terminal* window:

  ```
  python -c "import tensorflow as tf;print(tf.reduce_sum(tf.random.
  ↪normal([1000, 1000])))"
  ```

  Important: A new terminal window must be opened for the changes to the Environmental variables to take effect!!

- Once the above is run, you should see a print-out similar to the one bellow:

```
2020-06-22 20:24:31.355541: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cudart64_101.dll
2020-06-22 20:24:33.650692: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library nvcuda.dll
2020-06-22 20:24:33.686846: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1561] Found device 0 with properties:
pciBusID: 0000:02:00.0 name: GeForce GTX 1070 Ti computeCapability: 6.1
coreClock: 1.683GHz coreCount: 19 deviceMemorySize: 8.00GiB␣
↪deviceMemoryBandwidth: 238.66GiB/s
2020-06-22 20:24:33.697234: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cudart64_101.dll
2020-06-22 20:24:33.747540: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cublas64_10.dll
2020-06-22 20:24:33.787573: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cufft64_10.dll
2020-06-22 20:24:33.810063: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library curand64_10.dll
2020-06-22 20:24:33.841474: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cusolver64_10.dll
2020-06-22 20:24:33.862787: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cusparse64_10.dll
2020-06-22 20:24:33.907318: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-06-22 20:24:33.913612: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1703] Adding visible gpu devices: 0
2020-06-22 20:24:33.918093: I tensorflow/core/platform/cpu_feature_guard.
↪cc:143] Your CPU supports instructions that this TensorFlow binary was␣
↪not compiled to use: AVX2
2020-06-22 20:24:33.932784: I tensorflow/compiler/xla/service/service.
↪cc:168] XLA service 0x2382acc1c40 initialized for platform Host (this␣
↪does not guarantee that XLA will be used). Devices:
2020-06-22 20:24:33.939473: I tensorflow/compiler/xla/service/service.
↪cc:176]   StreamExecutor device (0): Host, Default Version
2020-06-22 20:24:33.944570: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1561] Found device 0 with properties:
pciBusID: 0000:02:00.0 name: GeForce GTX 1070 Ti computeCapability: 6.1
coreClock: 1.683GHz coreCount: 19 deviceMemorySize: 8.00GiB␣
↪deviceMemoryBandwidth: 238.66GiB/s
2020-06-22 20:24:33.953910: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cudart64_101.dll
2020-06-22 20:24:33.958772: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cublas64_10.dll
2020-06-22 20:24:33.963656: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cufft64_10.dll
2020-06-22 20:24:33.968210: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library curand64_10.dll
2020-06-22 20:24:33.973389: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cusolver64_10.dll
2020-06-22 20:24:33.978058: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cusparse64_10.dll
2020-06-22 20:24:33.983547: I tensorflow/stream_executor/platform/default/
↪dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-06-22 20:24:33.990380: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1703] Adding visible gpu devices: 0
2020-06-22 20:24:35.338596: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1102] Device interconnect StreamExecutor with strength 1 edge␣
↪matrix:
```

(continues on next page)

(continued from previous page)

```
2020-06-22 20:24:35.344643: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1108]      0
2020-06-22 20:24:35.348795: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1121] 0:   N
2020-06-22 20:24:35.353853: I tensorflow/core/common_runtime/gpu/gpu_
↪device.cc:1247] Created TensorFlow device (/job:localhost/replica:0/
↪task:0/device:GPU:0 with 6284 MB memory) -> physical GPU (device: 0,␣
↪name: GeForce GTX 1070 Ti, pci bus id: 0000:02:00.0, compute␣
↪capability: 6.1)
2020-06-22 20:24:35.369758: I tensorflow/compiler/xla/service/service.
↪cc:168] XLA service 0x2384aa9f820 initialized for platform CUDA (this␣
↪does not guarantee that XLA will be used). Devices:
2020-06-22 20:24:35.376320: I tensorflow/compiler/xla/service/service.
↪cc:176]   StreamExecutor device (0): GeForce GTX 1070 Ti, Compute␣
↪Capability 6.1
tf.Tensor(122.478485, shape=(), dtype=float32)
```

- Notice from the lines highlighted above that the library files are now `Successfully opened` and a debugging message is presented to confirm that TensorFlow has successfully `Created TensorFlow device`.

1.4 TensorFlow Object Detection API Installation

Now that you have installed TensorFlow, it is time to install the TensorFlow Object Detection API.

1.4.1 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/
   └── ...
```

1.4.2 Protobuf Installation/Compilation

The Tensorflow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries must be downloaded and compiled.

This should be done as follows:

- Head to the protoc releases page
- Download the latest `protoc-*-*.zip` release (e.g. `protoc-3.12.3-win64.zip` for 64-bit Windows)
- Extract the contents of the downloaded `protoc-*-*.zip` in a directory `<PATH_TO_PB>` of your choice (e.g. `C:\Program Files\Google Protobuf`)
- Add `<PATH_TO_PB>` to your `Path` environment variable (see *Environment Setup*)
- In a new *Terminal*[1], `cd` into `TensorFlow/models/research/` directory and run the following command:

```
# From within TensorFlow/models/research/
protoc object_detection/protos/*.proto --python_out=.
```

Important: If you are on Windows and using Protobuf 3.5 or later, the multi-file selection wildcard (i.e `*.proto`) may not work but you can do one of the following:

Windows Powershell

```
# From within TensorFlow/models/research/
Get-ChildItem object_detection/protos/*.proto | foreach {protoc "object_detection/
↪protos/$($_.Name)" --python_out=.}
```

Command Prompt

```
# From within TensorFlow/models/research/
for /f %i in ('dir /b object_detection\protos\*.proto') do protoc object_detection\
↪protos\%i --python_out=.
```

1.4.3 COCO API installation

As of TensorFlow 2.x, the `pycocotools` package is listed as a dependency of the Object Detection API. Ideally, this package should get installed when installing the Object Detection API as documented in the *Install the Object Detection API* section below, however the installation can fail for various reasons and therefore it is simpler to just install the package beforehand, in which case later installation will be skipped.

Windows

Run the following command to install `pycocotools` with Windows support:

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

Note that, according to the package's instructions, Visual C++ 2015 build tools must be installed and on your path. If they are not, make sure to install them from here.

Linux

[1] NOTE: You MUST open a new *Terminal* for the changes in the environment variables to take effect.

Download cocoapi to a directory of your choice, then `make` and copy the pycocotools subfolder to the `Tensorflow/models/research` directory, as such:

```
git clone https://github.com/cocodataset/cocoapi.git
cd cocoapi/PythonAPI
make
cp -r pycocotools <PATH_TO_TF>/TensorFlow/models/research/
```

Note: The default metrics are based on those used in Pascal VOC evaluation.

- To use the COCO object detection metrics add `metrics_set: "coco_detection_metrics"` to the `eval_config` message in the config file.
- To use the COCO instance segmentation metrics add `metrics_set: "coco_mask_metrics"` to the `eval_config` message in the config file.

1.4.4 Install the Object Detection API

Installation of the Object Detection API is achieved by installing the `object_detection` package. This is done by running the following commands from within `Tensorflow\models\research`:

```
# From within TensorFlow/models/research/
cp object_detection/packages/tf2/setup.py .
python -m pip install .
```

Note: During the above installation, you may observe the following error:

```
ERROR: Command errored out with exit status 1:
     command: 'C:\Users\sglvladi\Anaconda3\envs\tf2\python.exe' -u -c
↪'import sys, setuptools, tokenize; sys.argv[0] = '"'"'C:\\Users\\sglvladi\\
↪AppData\\Local\\Temp\\pip-install-yn46ecei\\pycocotools\\setup.py'"'"'; __
↪file__='"'"'C:\\Users\\sglvladi\\AppData\\Local\\Temp\\pip-install-
↪yn46ecei\\pycocotools\\setup.py'"'"';f=getattr(tokenize, '"'"'open'"'"',␣
↪open)(__file__);code=f.read().replace('"'"'\r\n'"'"', '"'"'\n'"'"');f.
↪close();exec(compile(code, __file__, '"'"'exec'"'"'))' install --record
↪'C:\Users\sglvladi\AppData\Local\Temp\pip-record-wpn7b6qo\install-record.
↪txt' --single-version-externally-managed --compile --install-headers 'C:\
↪Users\sglvladi\Anaconda3\envs\tf2\Include\pycocotools'
         cwd: C:\Users\sglvladi\AppData\Local\Temp\pip-install-yn46ecei\
↪pycocotools\
    Complete output (14 lines):
    running install
    running build
    running build_py
    creating build
    creating build\lib.win-amd64-3.8
    creating build\lib.win-amd64-3.8\pycocotools
    copying pycocotools\coco.py -> build\lib.win-amd64-3.8\pycocotools
    copying pycocotools\cocoeval.py -> build\lib.win-amd64-3.8\pycocotools
    copying pycocotools\mask.py -> build\lib.win-amd64-3.8\pycocotools
    copying pycocotools\__init__.py -> build\lib.win-amd64-3.8\pycocotools
    running build_ext
    skipping 'pycocotools\_mask.c' Cython extension (up-to-date)
    building 'pycocotools._mask' extension
```

(continues on next page)

(continued from previous page)

```
    error: Microsoft Visual C++ 14.0 is required. Get it with "Build Tools␣
↪for Visual Studio": https://visualstudio.microsoft.com/downloads/
    ----------------------------------------
ERROR: Command errored out with exit status 1: 'C:\Users\sglvladi\Anaconda3\
↪envs\tf2\python.exe' -u -c 'import sys, setuptools, tokenize; sys.argv[0]␣
↪= '"'"'C:\\Users\\sglvladi\\AppData\\Local\\Temp\\pip-install-yn46ecei\\
↪pycocotools\\setup.py'"'"'; __file__='"'"'C:\\Users\\sglvladi\\AppData\\
↪Local\\Temp\\pip-install-yn46ecei\\pycocotools\\setup.py'"'"';
↪f=getattr(tokenize, '"'"'open'"'"', open)(__file__);code=f.read().replace('
↪"'"'\r\n'"'"', '"'"'\n'"'"');f.close();exec(compile(code, __file__, '"'"
↪'exec'"'"'))' install --record 'C:\Users\sglvladi\AppData\Local\Temp\pip-
↪record-wpn7b6qo\install-record.txt' --single-version-externally-managed --
↪compile --install-headers 'C:\Users\sglvladi\Anaconda3\envs\tf2\Include\
↪pycocotools' Check the logs for full command output.
```

This is caused because installation of the `pycocotools` package has failed. To fix this have a look at the *COCO API installation* section and rerun the above commands.

1.4.5 Test your Installation

To test the installation, run the following command from within `Tensorflow\models\research`:

```
# From within TensorFlow/models/research/
python object_detection/builders/model_builder_tf2_test.py
```

Once the above is run, allow some time for the test to complete and once done you should observe a printout similar to the one below:

```
...
[       OK ] ModelBuilderTF2Test.test_create_ssd_models_from_config
[ RUN      ] ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update
[       OK ] ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update
[ RUN      ] ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold
[       OK ] ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold
[ RUN      ] ModelBuilderTF2Test.test_invalid_model_config_proto
[       OK ] ModelBuilderTF2Test.test_invalid_model_config_proto
[ RUN      ] ModelBuilderTF2Test.test_invalid_second_stage_batch_size
[       OK ] ModelBuilderTF2Test.test_invalid_second_stage_batch_size
[ RUN      ] ModelBuilderTF2Test.test_session
[  SKIPPED ] ModelBuilderTF2Test.test_session
[ RUN      ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
[       OK ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
[ RUN      ] ModelBuilderTF2Test.test_unknown_meta_architecture
[       OK ] ModelBuilderTF2Test.test_unknown_meta_architecture
[ RUN      ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
[       OK ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
----------------------------------------------------------------------
Ran 20 tests in 68.510s

OK (skipped=1)
```

1.4.6 Try out the examples

If the previous step completed successfully it means you have successfully installed all the components necessary to perform object detection using pre-trained models.

If you want to play around with some examples to see how this can be done, now would be a good time to have a look at the *Examples* section.

CHAPTER

TWO

TRAINING CUSTOM OBJECT DETECTOR

So, up to now you should have done the following:

- Installed TensorFlow (See *TensorFlow Installation*)
- Installed TensorFlow Object Detection API (See *TensorFlow Object Detection API Installation*)

Now that we have done all the above, we can start doing some cool stuff. Here we will see how you can train your own object detector, and since it is not as simple as it sounds, we will have a look at:

1. How to organise your workspace/training files
2. How to prepare/annotate image datasets
3. How to generate tf records from such datasets
4. How to configure a simple training pipeline
5. How to train a model and monitor it's progress
6. How to export the resulting model and use it to detect objects.

2.1 Preparing the Workspace

1. If you have followed the tutorial, you should by now have a folder `Tensorflow`, placed under `<PATH_TO_TF>` (e.g. `C:/Users/sglvladi/Documents`), with the following directory tree:

```
TensorFlow/
├─ addons/ (Optional)
│  └─ labelImg/
└─ models/
   ├─ community/
   ├─ official/
   ├─ orbit/
   ├─ research/
   └─ ...
```

2. Now create a new folder under `TensorFlow` and call it `workspace`. It is within the `workspace` that we will store all our training set-ups. Now let's go under workspace and create another folder named `training_demo`. Now our directory structure should be as so:

```
TensorFlow/
├─ addons/ (Optional)
│  └─ labelImg/
├─ models/
│  ├─ community/
```

(continues on next page)

(continued from previous page)

```
│   ├─ official/
│   ├─ orbit/
│   ├─ research/
│   └─ ...
└─ workspace/
    └─ training_demo/
```

3. The `training_demo` folder shall be our *training folder*, which will contain all files related to our model training. It is advisable to create a separate training folder each time we wish to train on a different dataset. The typical structure for training folders is shown below.

```
training_demo/
├─ annotations/
├─ exported-models/
├─ images/
│   ├─ test/
│   └─ train/
├─ models/
├─ pre-trained-models/
└─ README.md
```

Here's an explanation for each of the folders/filer shown in the above tree:

- `annotations`: This folder will be used to store all `*.csv` files and the respective TensorFlow `*.record` files, which contain the list of annotations for our dataset images.
- `exported-models`: This folder will be used to store exported versions of our trained model(s).
- `images`: This folder contains a copy of all the images in our dataset, as well as the respective `*.xml` files produced for each one, once `labelImg` is used to annotate objects.
  - `images/train`: This folder contains a copy of all images, and the respective `*.xml` files, which will be used to train our model.
  - `images/test`: This folder contains a copy of all images, and the respective `*.xml` files, which will be used to test our model.
- `models`: This folder will contain a sub-folder for each of training job. Each subfolder will contain the training pipeline configuration file `*.config`, as well as all files generated during the training and evaluation of our model.
- `pre-trained-models`: This folder will contain the downloaded pre-trained models, which shall be used as a starting checkpoint for our training jobs.
- `README.md`: This is an optional file which provides some general information regarding the training conditions of our model. It is not used by TensorFlow in any way, but it generally helps when you have a few training folders and/or you are revisiting a trained model after some time.

If you do not understand most of the things mentioned above, no need to worry, as we'll see how all the files are generated further down.

2.2 Preparing the Dataset

2.2.1 Annotate the Dataset

2.2.1.1 Install LabelImg

There exist several ways to install `labelImg`. Below are 3 of the most common.

Using PIP (Recommended)

1. Open a new *Terminal* window and activate the *tensorflow_gpu* environment (if you have not done so already)
2. Run the following command to install `labelImg`:

```
pip install labelImg
```

3. `labelImg` can then be run as follows:

```
labelImg
# or
labelImg [IMAGE_PATH] [PRE-DEFINED CLASS FILE]
```

Use precompiled binaries (Easy)

Precompiled binaries for both Windows and Linux can be found here .

Installation is the done in three simple steps:

1. Inside you `TensorFlow` folder, create a new directory, name it `addons` and then `cd` into it.
2. Download the latest binary for your OS from here. and extract its contents under `Tensorflow/addons/labelImg`.
3. You should now have a single folder named `addons/labelImg` under your `TensorFlow` folder, which contains another 4 folders as such:

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

4. `labelImg` can then be run as follows:

```
# From within Tensorflow/addons/labelImg
labelImg
# or
labelImg [IMAGE_PATH] [PRE-DEFINED CLASS FILE]
```

Build from source (Hard)

The steps for installing from source follow below.

1. Download labelImg

- Inside you `TensorFlow` folder, create a new directory, name it `addons` and then `cd` into it.
- To download the package you can either use Git to clone the labelImg repo inside the `TensorFlow\addons` folder, or you can simply download it as a ZIP and extract it's contents inside the `TensorFlow\addons` folder. To keep things consistent, in the latter case you will have to rename the extracted folder `labelImg-master` to `labelImg`.[1]
- You should now have a single folder named `addons\labelImg` under your `TensorFlow` folder, which contains another 4 folders as such:

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

2. Install dependencies and compiling package

- Open a new *Terminal* window and activate the *tensorflow_gpu* environment (if you have not done so already)
- `cd` into `TensorFlow/addons/labelImg` and run the following commands:

  Windows

  ```
  conda install pyqt=5
  pyrcc5 -o libs/resources.py resources.qrc
  ```

  Linux

  ```
  sudo apt-get install pyqt5-dev-tools
  sudo pip install -r requirements/requirements-linux-python3.txt
  make qt5py3
  ```

3. Test your installation

- Open a new *Terminal* window and activate the *tensorflow_gpu* environment (if you have not done so already)
- `cd` into `TensorFlow/addons/labelImg` and run the following command:

  ```
  # From within Tensorflow/addons/labelImg
  python labelImg.py
  # or
  python labelImg.py [IMAGE_PATH] [PRE-DEFINED CLASS FILE]
  ```

[1] The latest repo commit when writing this tutorial is 8d1bd68.

2.2.1.2 Annotate Images

- Once you have collected all the images to be used to test your model (ideally more than 100 per class), place them inside the folder `training_demo/images`.
- Open a new *Terminal* window.
- Next go ahead and start `labelImg`, pointing it to your `training_demo/images` folder.
  - If you installed `labelImg` *Using PIP (Recommended)*:

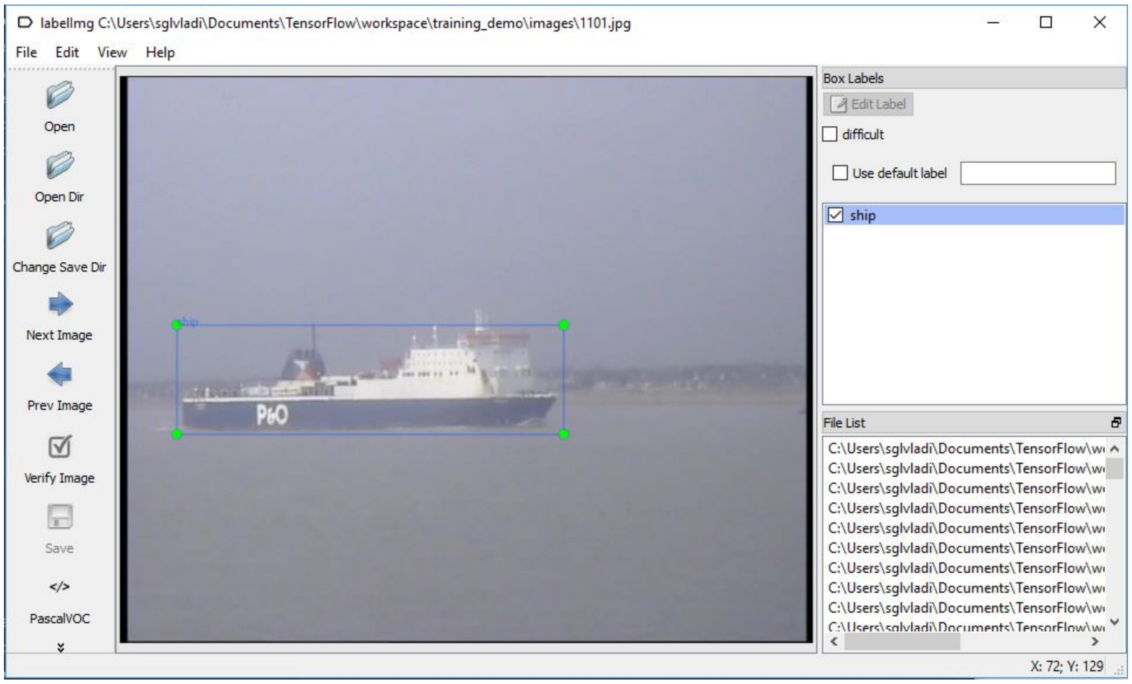
    ```
    labelImg <PATH_TO_TF>/TensorFlow/workspace/training_demo/images
    ```

  - Othewise, `cd` into `Tensorflow/addons/labelImg` and run:

    ```
    # From within Tensorflow/addons/labelImg
    python labelImg.py ../../workspace/training_demo/images
    ```

- A File Explorer Dialog windows should open, which points to the `training_demo/images` folder.
- Press the "Select Folder" button, to start annotating your images.

Once open, you should see a window similar to the one below:



I won't be covering a tutorial on how to use `labelImg`, but you can have a look at labelImg's repo for more details. A nice Youtube video demonstrating how to use `labelImg` is also available here. What is important is that once you annotate all your images, a set of new `*.xml` files, one for each image, should be generated inside your `training_demo/images` folder.

2.2.2 Partition the Dataset

Once you have finished annotating your image dataset, it is a general convention to use only part of it for training, and the rest is used for evaluation purposes (e.g. as discussed in *Evaluating the Model (Optional)*).

Typically, the ratio is 9:1, i.e. 90% of the images are used for training and the rest 10% is maintained for testing, but you can chose whatever ratio suits your needs.

Once you have decided how you will be splitting your dataset, copy all training images, together with their corresponding `*.xml` files, and place them inside the `training_demo/images/train` folder. Similarly, copy all testing images, with their `*.xml` files, and paste them inside `training_demo/images/test`.

For lazy people like myself, who cannot be bothered to do the above, I have put together a simple script that automates the above process:

```
""" usage: partition_dataset.py [-h] [-i IMAGEDIR] [-o OUTPUTDIR] [-r RATIO] [-x]

Partition dataset of images into training and testing sets

optional arguments:
  -h, --help            show this help message and exit
  -i IMAGEDIR, --imageDir IMAGEDIR
                        Path to the folder where the image dataset is stored. If not␣
↪specified, the CWD will be used.
  -o OUTPUTDIR, --outputDir OUTPUTDIR
                        Path to the output folder where the train and test dirs␣
↪should be created. Defaults to the same directory as IMAGEDIR.
  -r RATIO, --ratio RATIO
                        The ratio of the number of test images over the total number␣
↪of images. The default is 0.1.
  -x, --xml             Set this flag if you want the xml annotation files to be␣
↪processed and copied over.
"""
import os
import re
from shutil import copyfile
import argparse
import math
import random


def iterate_dir(source, dest, ratio, copy_xml):
    source = source.replace('\\', '/')
    dest = dest.replace('\\', '/')
    train_dir = os.path.join(dest, 'train')
    test_dir = os.path.join(dest, 'test')

    if not os.path.exists(train_dir):
        os.makedirs(train_dir)
    if not os.path.exists(test_dir):
        os.makedirs(test_dir)

    images = [f for f in os.listdir(source)
              if re.search(r'([a-zA-Z0-9\s_\\.\-\(\):])+(.jpg|.jpeg|.png)$', f)]

    num_images = len(images)
    num_test_images = math.ceil(ratio*num_images)
```

(continues on next page)

(continued from previous page)

```
    for i in range(num_test_images):
        idx = random.randint(0, len(images)-1)
        filename = images[idx]
        copyfile(os.path.join(source, filename),
                 os.path.join(test_dir, filename))
        if copy_xml:
            xml_filename = os.path.splitext(filename)[0]+'.xml'
            copyfile(os.path.join(source, xml_filename),
                     os.path.join(test_dir,xml_filename))
        images.remove(images[idx])

    for filename in images:
        copyfile(os.path.join(source, filename),
                 os.path.join(train_dir, filename))
        if copy_xml:
            xml_filename = os.path.splitext(filename)[0]+'.xml'
            copyfile(os.path.join(source, xml_filename),
                     os.path.join(train_dir, xml_filename))


def main():

    # Initiate argument parser
    parser = argparse.ArgumentParser(description="Partition dataset of images into␣
↪training and testing sets",
                                     formatter_class=argparse.RawTextHelpFormatter)
    parser.add_argument(
        '-i', '--imageDir',
        help='Path to the folder where the image dataset is stored. If not specified,␣
↪the CWD will be used.',
        type=str,
        default=os.getcwd()
    )
    parser.add_argument(
        '-o', '--outputDir',
        help='Path to the output folder where the train and test dirs should be␣
↪created. '
             'Defaults to the same directory as IMAGEDIR.',
        type=str,
        default=None
    )
    parser.add_argument(
        '-r', '--ratio',
        help='The ratio of the number of test images over the total number of images.␣
↪The default is 0.1.',
        default=0.1,
        type=float)
    parser.add_argument(
        '-x', '--xml',
        help='Set this flag if you want the xml annotation files to be processed and␣
↪copied over.',
        action='store_true'
    )
    args = parser.parse_args()

    if args.outputDir is None:
        args.outputDir = args.imageDir
```

(continues on next page)

(continued from previous page)

```
    # Now we are ready to start the iteration
    iterate_dir(args.imageDir, args.outputDir, args.ratio, args.xml)


if __name__ == '__main__':
    main()
```

- Under the `TensorFlow` folder, create a new folder `TensorFlow/scripts`, which we can use to store some useful scripts.
- To make things even tidier, let's create a new folder `TensorFlow/scripts/preprocessing`, where we shall store scripts that we can use to preprocess our training inputs. Below is out `TensorFlow` directory tree structure, up to now:

```
TensorFlow/
├─ addons/ (Optional)
│  └─ labelImg/
├─ models/
│  ├─ community/
│  ├─ official/
│  ├─ orbit/
│  ├─ research/
│  └─ ...
├─ scripts/
│  └─ preprocessing/
└─ workspace/
   └─ training_demo/
```

- Click `here` to download the above script and save it inside `TensorFlow/scripts/preprocessing`.
- Then, `cd` into `TensorFlow/scripts/preprocessing` and run:

```
python partition_dataset.py -x -i [PATH_TO_IMAGES_FOLDER] -r 0.1

# For example
# python partition_dataset.py -x -i C:/Users/sglvladi/Documents/
↪Tensorflow/workspace/training_demo/images -r 0.1
```

Once the script has finished, two new folders should have been created under `training_demo/images`, namely `training_demo/images/train` and `training_demo/images/test`, containing 90% and 10% of the images (and `*.xml` files), respectively. To avoid loss of any files, the script will not delete the images under `training_demo/images`. Once you have checked that your images have been safely copied over, you can delete the images under `training_demo/images` manually.

2.2.3 Create Label Map

TensorFlow requires a label map, which namely maps each of the used labels to an integer values. This label map is used both by the training and detection processes.

Below we show an example label map (e.g `label_map.pbtxt`), assuming that our dataset containes 2 labels, `dogs` and `cats`:

```
item {
    id: 1
```

(continues on next page)

(continued from previous page)

```
    name: 'cat'
}

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

Label map files have the extention `.pbtxt` and should be placed inside the `training_demo/annotations` folder.

2.2.4 Create TensorFlow Records

Now that we have generated our annotations and split our dataset into the desired training and testing subsets, it is time to convert our annotations into the so called `TFRecord` format.

2.2.4.1 Convert `*.xml` to `*.record`

To do this we can write a simple script that iterates through all `*.xml` files in the `training_demo/images/train` and `training_demo/images/test` folders, and generates a `*.record` file for each of the two. Here is an example script that allows us to do just that:

```
""" Sample TensorFlow XML-to-TFRecord converter

usage: generate_tfrecord.py [-h] [-x XML_DIR] [-l LABELS_PATH] [-o OUTPUT_PATH] [-i␣
↪IMAGE_DIR] [-c CSV_PATH]

optional arguments:
  -h, --help            show this help message and exit
  -x XML_DIR, --xml_dir XML_DIR
                        Path to the folder where the input .xml files are stored.
  -l LABELS_PATH, --labels_path LABELS_PATH
                        Path to the labels (.pbtxt) file.
  -o OUTPUT_PATH, --output_path OUTPUT_PATH
                        Path of output TFRecord (.record) file.
  -i IMAGE_DIR, --image_dir IMAGE_DIR
                        Path to the folder where the input image files are stored.␣
↪Defaults to the same directory as XML_DIR.
  -c CSV_PATH, --csv_path CSV_PATH
                        Path of output .csv file. If none provided, then no file will␣
↪be written.
"""

import os
import glob
import pandas as pd
import io
import xml.etree.ElementTree as ET
import argparse

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging (1)
import tensorflow.compat.v1 as tf
from PIL import Image
from object_detection.utils import dataset_util, label_map_util
```

(continues on next page)

(continued from previous page)

```
from collections import namedtuple

# Initiate argument parser
parser = argparse.ArgumentParser(
    description="Sample TensorFlow XML-to-TFRecord converter")
parser.add_argument("-x",
                    "--xml_dir",
                    help="Path to the folder where the input .xml files are stored.",
                    type=str)
parser.add_argument("-l",
                    "--labels_path",
                    help="Path to the labels (.pbtxt) file.", type=str)
parser.add_argument("-o",
                    "--output_path",
                    help="Path of output TFRecord (.record) file.", type=str)
parser.add_argument("-i",
                    "--image_dir",
                    help="Path to the folder where the input image files are stored. "
                         "Defaults to the same directory as XML_DIR.",
                    type=str, default=None)
parser.add_argument("-c",
                    "--csv_path",
                    help="Path of output .csv file. If none provided, then no file␣
↪will be "
                         "written.",
                    type=str, default=None)

args = parser.parse_args()

if args.image_dir is None:
    args.image_dir = args.xml_dir

label_map = label_map_util.load_labelmap(args.labels_path)
label_map_dict = label_map_util.get_label_map_dict(label_map)


def xml_to_csv(path):
    """Iterates through all .xml files (generated by labelImg) in a given directory␣
↪and combines
    them in a single Pandas dataframe.

    Parameters:
    ----------
    path : str
        The path containing the .xml files
    Returns
    -------
    Pandas DataFrame
        The produced dataframe
    """

    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
```

(continues on next page)

(continued from previous page)

```
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height',
                   'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def class_text_to_int(row_label):
    return label_map_dict[row_label]


def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(),␣
↪gb.groups)]


def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
```

(continues on next page)

(continued from previous page)

```
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example


def main(_):

    writer = tf.python_io.TFRecordWriter(args.output_path)
    path = os.path.join(args.image_dir)
    examples = xml_to_csv(args.xml_dir)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())
    writer.close()
    print('Successfully created the TFRecord file: {}'.format(args.output_path))
    if args.csv_path is not None:
        examples.to_csv(args.csv_path, index=None)
        print('Successfully created the CSV file: {}'.format(args.csv_path))


if __name__ == '__main__':
    tf.app.run()
```

- Click here to download the above script and save it inside `TensorFlow/scripts/preprocessing`.
- Install the `pandas` package:

```
conda install pandas # Anaconda
                     # or
pip install pandas   # pip
```

- Finally, `cd` into `TensorFlow/scripts/preprocessing` and run:

```
# Create train data:
python generate_tfrecord.py -x [PATH_TO_IMAGES_FOLDER]/train -l [PATH_TO_
↪ANNOTATIONS_FOLDER]/label_map.pbtxt -o [PATH_TO_ANNOTATIONS_FOLDER]/
↪train.record

# Create test data:
python generate_tfrecord.py -x [PATH_TO_IMAGES_FOLDER]/test -l [PATH_TO_
↪ANNOTATIONS_FOLDER]/label_map.pbtxt -o [PATH_TO_ANNOTATIONS_FOLDER]/
↪test.record

# For example
# python generate_tfrecord.py -x C:/Users/sglvladi/Documents/Tensorflow/
↪workspace/training_demo/images/train -l C:/Users/sglvladi/Documents/
↪Tensorflow/workspace/training_demo/annotations/label_map.pbtxt -o C:/
↪Users/sglvladi/Documents/Tensorflow/workspace/training_demo/annotations/
↪train.record
# python generate_tfrecord.py -x C:/Users/sglvladi/Documents/Tensorflow/
↪workspace/training_demo/images/test -l C:/Users/sglvladi/Documents/
↪Tensorflow2/workspace/training_demo/annotations/label_map.pbtxt -o C:/
↪Users/sglvladi/Documents/Tensorflow/workspace/training_demo/annotations/
↪test.record
```

(continues on next page)

(continued from previous page)

Once the above is done, there should be 2 new files under the `training_demo/annotations` folder, named `test.record` and `train.record`, respectively.

2.3 Configuring a Training Job

For the purposes of this tutorial we will not be creating a training job from scratch, but rather we will reuse one of the pre-trained models provided by TensorFlow. If you would like to train an entirely new model, you can have a look at TensorFlow's tutorial.

The model we shall be using in our examples is the SSD ResNet50 V1 FPN 640x640 model, since it provides a relatively good trade-off between performance and speed. However, there exist a number of other models you can use, all of which are listed in TensorFlow 2 Detection Model Zoo.

2.3.1 Download Pre-Trained Model

To begin with, we need to download the latest pre-trained network for the model we wish to use. This can be done by simply clicking on the name of the desired model in the table found in TensorFlow 2 Detection Model Zoo. Clicking on the name of your model should initiate a download for a `*.tar.gz` file.

Once the `*.tar.gz` file has been downloaded, open it using a decompression program of your choice (e.g. 7zip, WinZIP, etc.). Next, open the `*.tar` folder that you see when the compressed folder is opened, and extract its contents inside the folder `training_demo/pre-trained-models`. Since we downloaded the SSD ResNet50 V1 FPN 640x640 model, our `training_demo` directory should now look as follows:

```
training_demo/
├─ ...
├─ pre-trained-models/
│  └─ ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/
│     ├─ checkpoint/
│     ├─ saved_model/
│     └─ pipeline.config
└─ ...
```

Note that the above process can be repeated for all other pre-trained models you wish to experiment with. For example, if you wanted to also configure a training job for the EfficientDet D1 640x640 model, you can download the model and after extracting its context the demo directory will be:

```
training_demo/
├─ ...
├─ pre-trained-models/
│  ├─ efficientdet_d1_coco17_tpu-32/
│  │  ├─ checkpoint/
│  │  ├─ saved_model/
│  │  └─ pipeline.config
│  └─ ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/
│     ├─ checkpoint/
│     ├─ saved_model/
│     └─ pipeline.config
└─ ...
```

2.3.2 Configure the Training Pipeline

Now that we have downloaded and extracted our pre-trained model, let's create a directory for our training job. Under the `training_demo/models` create a new directory named `my_ssd_resnet50_v1_fpn` and copy the `training_demo/pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/pipeline.config` file inside the newly created directory. Our `training_demo/models` directory should now look like this:

```
training_demo/
├─ ...
├─ models/
│  └─ my_ssd_resnet50_v1_fpn/
│     └─ pipeline.config
└─ ...
```

Now, let's have a look at the changes that we shall need to apply to the `pipeline.config` file (highlighted in yellow):

```
model {
  ssd {
    num_classes: 1 # Set this to the number of different label classes
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_resnet50_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 0.00039999998989515007
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.029999999329447746
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
      }
    }
    box_coder {
```

(continues on next page)

(continued from previous page)

```
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    box_predictor {
      weight_shared_convolutional_box_predictor {
        conv_hyperparams {
          regularizer {
            l2_regularizer {
              weight: 0.00039999998989515007
            }
          }
          initializer {
            random_normal_initializer {
              mean: 0.0
              stddev: 0.009999999776482582
            }
          }
          activation: RELU_6
          batch_norm {
            decay: 0.996999979019165
            scale: true
            epsilon: 0.0010000000474974513
          }
        }
        depth: 256
        num_layers_before_predictor: 4
        kernel_size: 3
        class_prediction_bias_init: -4.599999904632568
      }
    }
    anchor_generator {
      multiscale_anchor_generator {
        min_level: 3
        max_level: 7
        anchor_scale: 4.0
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        scales_per_octave: 2
```

(continues on next page)

(continued from previous page)

```
      }
    }
    post_processing {
      batch_non_max_suppression {
        score_threshold: 9.99999993922529e-09
        iou_threshold: 0.6000000238418579
        max_detections_per_class: 100
        max_total_detections: 100
        use_static_shapes: false
      }
      score_converter: SIGMOID
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 8 # Increase/Decrease this value depending on the available memory (Higher values require more memory and vice-versa)
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.03999999910593033
          total_steps: 25000
```

(continues on next page)

(continued from previous page)

```
153           warmup_learning_rate: 0.013333000242710114
154           warmup_steps: 2000
155         }
156       }
157       momentum_optimizer_value: 0.8999999761581421
158     }
159     use_moving_average: false
160   }
161   fine_tune_checkpoint: "pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/
    ↪checkpoint/ckpt-0" # Path to checkpoint of pre-trained model
162   num_steps: 25000
163   startup_delay_steps: 0.0
164   replicas_to_aggregate: 8
165   max_number_of_boxes: 100
166   unpad_groundtruth_tensors: false
167   fine_tune_checkpoint_type: "detection" # Set this to "detection" since we want to␣
    ↪be training the full detection model
168   use_bfloat16: false # Set this to false if you are not training on a TPU
169   fine_tune_checkpoint_version: V2
170 }
171 train_input_reader {
172   label_map_path: "annotations/label_map.pbtxt" # Path to label map file
173   tf_record_input_reader {
174     input_path: "annotations/train.record" # Path to training TFRecord file
175   }
176 }
177 eval_config {
178   metrics_set: "coco_detection_metrics"
179   use_moving_averages: false
180 }
181 eval_input_reader {
182   label_map_path: "annotations/label_map.pbtxt" # Path to label map file
183   shuffle: false
184   num_epochs: 1
185   tf_record_input_reader {
186     input_path: "annotations/test.record" # Path to testing TFRecord
187   }
188 }
```

It is worth noting here that the changes to lines `178` to `179` above are optional. These should only be used if you installed the COCO evaluation tools, as outlined in the *COCO API installation* section, and you intend to run evaluation (see *Evaluating the Model (Optional)*).

Once the above changes have been applied to our config file, go ahead and save it.

2.4 Training the Model

Before we begin training our model, let's go and copy the `TensorFlow/models/research/object_detection/model_main_tf2.py` script and paste it straight into our `training_demo` folder. We will need this script in order to train our model.

Now, to initiate a new training job, open a new *Terminal*, `cd` inside the `training_demo` folder and run the following command:

```
python model_main_tf2.py --model_dir=models/my_ssd_resnet50_v1_fpn --pipeline_config_
↪path=models/my_ssd_resnet50_v1_fpn/pipeline.config
```

Once the training process has been initiated, you should see a series of print outs similar to the one below (plus/minus some warnings):

```
...
WARNING:tensorflow:Unresolved object in checkpoint: (root).model._box_predictor._base_
↪tower_layers_for_heads.class_predictions_with_background.4.10.gamma
W0716 05:24:19.105542  1364 util.py:143] Unresolved object in checkpoint: (root).
↪model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.
↪4.10.gamma
WARNING:tensorflow:Unresolved object in checkpoint: (root).model._box_predictor._base_
↪tower_layers_for_heads.class_predictions_with_background.4.10.beta
W0716 05:24:19.106541  1364 util.py:143] Unresolved object in checkpoint: (root).
↪model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.
↪4.10.beta
WARNING:tensorflow:Unresolved object in checkpoint: (root).model._box_predictor._base_
↪tower_layers_for_heads.class_predictions_with_background.4.10.moving_mean
W0716 05:24:19.107540  1364 util.py:143] Unresolved object in checkpoint: (root).
↪model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.
↪4.10.moving_mean
WARNING:tensorflow:Unresolved object in checkpoint: (root).model._box_predictor._base_
↪tower_layers_for_heads.class_predictions_with_background.4.10.moving_variance
W0716 05:24:19.108539  1364 util.py:143] Unresolved object in checkpoint: (root).
↪model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.
↪4.10.moving_variance
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore or tf.
↪keras.Model.load_weights) but not all checkpointed values were used. See above for␣
↪specific issues. Use expect_partial() on the load status object, e.g. tf.train.
↪Checkpoint.restore(...).expect_partial(), to silence these warnings, or use assert_
↪consumed() to make the check explicit. See https://www.tensorflow.org/guide/
↪checkpoint#loading_mechanics for details.
W0716 05:24:19.108539  1364 util.py:151] A checkpoint was restored (e.g. tf.train.
↪Checkpoint.restore or tf.keras.Model.load_weights) but not all checkpointed values␣
↪were used. See above for specific issues. Use expect_partial() on the load status␣
↪object, e.g. tf.train.Checkpoint.restore(...).expect_partial(), to silence these␣
↪warnings, or use assert_consumed() to make the check explicit. See https://www.
↪tensorflow.org/guide/checkpoint#loading_mechanics for details.
WARNING:tensorflow:num_readers has been reduced to 1 to match input file shards.
INFO:tensorflow:Step 100 per-step time 1.153s loss=0.761
I0716 05:26:55.879558  1364 model_lib_v2.py:632] Step 100 per-step time 1.153s loss=0.
↪761
...
```

Important: The output will normally look like it has "frozen", but DO NOT rush to cancel the process. The training outputs logs only every 100 steps by default, therefore if you wait for a while, you should see a log for the loss at step 100.

The time you should wait can vary greatly, depending on whether you are using a GPU and the chosen value for `batch_size` in the config file, so be patient.

If you ARE observing a similar output to the above, then CONGRATULATIONS, you have successfully started your first training job. Now you may very well treat yourself to a cold beer, as waiting on the training to finish is likely to take a while. Following what people have said online, it seems that it is advisable to allow you model to reach a `TotalLoss` of at least 2 (ideally 1 and lower) if you want to achieve "fair" detection results. Obviously, lower

TotalLoss is better, however very low TotalLoss should be avoided, as the model may end up overfitting the dataset, meaning that it will perform poorly when applied to images outside the dataset. To monitor TotalLoss, as well as a number of other metrics, while your model is training, have a look at *Monitor Training Job Progress using TensorBoard*.

If you ARE NOT seeing a print-out similar to that shown above, and/or the training job crashes after a few seconds, then have a look at the issues and proposed solutions, under the *Common issues* section, to see if you can find a solution. Alternatively, you can try the issues section of the official Tensorflow Models repo.

Note: Training times can be affected by a number of factors such as:

- The computational power of you hardware (either CPU or GPU): Obviously, the more powerful your PC is, the faster the training process.
- Whether you are using the TensorFlow CPU or GPU variant: In general, even when compared to the best CPUs, almost any GPU graphics card will yield much faster training and detection speeds. As a matter of fact, when I first started I was running TensorFlow on my *Intel i7-5930k* (6/12 cores @ 4GHz, 32GB RAM) and was getting step times of around *12 sec/step*, after which I installed TensorFlow GPU and training the very same model -using the same dataset and config files- on a *EVGA GTX-770* (1536 CUDA-cores @ 1GHz, 2GB VRAM) I was down to *0.9 sec/step*!!! A 12-fold increase in speed, using a "low/mid-end" graphics card, when compared to a "mid/high-end" CPU.
- The complexity of the objects you are trying to detect: Obviously, if your objective is to track a black ball over a white background, the model will converge to satisfactory levels of detection pretty quickly. If on the other hand, for example, you wish to detect ships in ports, using Pan-Tilt-Zoom cameras, then training will be a much more challenging and time-consuming process, due to the high variability of the shape and size of ships, combined with a highly dynamic background.
- And many, many, many, more. . . .

2.5 Evaluating the Model (Optional)

By default, the training process logs some basic measures of training performance. These seem to change depending on the installed version of Tensorflow.

As you will have seen in various parts of this tutorial, we have mentioned a few times the optional utilisation of the COCO evaluation metrics. Also, under section *Partition the Dataset* we partitioned our dataset in two parts, where one was to be used for training and the other for evaluation. In this section we will look at how we can use these metrics, along with the test images, to get a sense of the performance achieved by our model as it is being trained.

Firstly, let's start with a brief explanation of what the evaluation process does. While the training process runs, it will occasionally create checkpoint files inside the `training_demo/training` folder, which correspond to snapshots of the model at given steps. When a set of such new checkpoint files is generated, the evaluation process uses these files and evaluates how well the model performs in detecting objects in the test dataset. The results of this evaluation are summarised in the form of some metrics, which can be examined over time.

The steps to run the evaluation are outlined below:

1. Firstly we need to download and install the metrics we want to use.
   - For a description of the supported object detection evaluation metrics, see here.
   - The process of installing the COCO evaluation metrics is described in *COCO API installation*.
2. Secondly, we must modify the configuration pipeline (`*.config` script).
   - See lines 178-179 of the script in *Configure the Training Pipeline*.

3. The third step is to actually run the evaluation. To do so, open a new *Terminal*, `cd` inside the `training_demo` folder and run the following command:

```
python model_main_tf2.py --model_dir=models/my_ssd_resnet50_v1_fpn --
↪pipeline_config_path=models/my_ssd_resnet50_v1_fpn/pipeline.config --
↪checkpoint_dir=models/my_ssd_resnet50_v1_fpn
```

   Once the above is run, you should see a checkpoint similar to the one below (plus/minus some warnings):

```
...
WARNING:tensorflow:From C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-
↪packages\object_detection\inputs.py:79: sparse_to_dense (from␣
↪tensorflow.python.ops.sparse_ops) is deprecated and will be removed in␣
↪a future version.
Instructions for updating:
Create a `tf.sparse.SparseTensor` and use `tf.sparse.to_dense` instead.
W0716 05:44:10.059399 17144 deprecation.py:317] From C:\Users\sglvladi\
↪Anaconda3\envs\tf2\lib\site-packages\object_detection\inputs.py:79:␣
↪sparse_to_dense (from tensorflow.python.ops.sparse_ops) is deprecated␣
↪and will be removed in a future version.
Instructions for updating:
Create a `tf.sparse.SparseTensor` and use `tf.sparse.to_dense` instead.
WARNING:tensorflow:From C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-
↪packages\object_detection\inputs.py:259: to_float (from tensorflow.
↪python.ops.math_ops) is deprecated and will be removed in a future␣
↪version.
Instructions for updating:
Use `tf.cast` instead.
W0716 05:44:12.383937 17144 deprecation.py:317] From C:\Users\sglvladi\
↪Anaconda3\envs\tf2\lib\site-packages\object_detection\inputs.py:259: to_
↪float (from tensorflow.python.ops.math_ops) is deprecated and will be␣
↪removed in a future version.
Instructions for updating:
Use `tf.cast` instead.
INFO:tensorflow:Waiting for new checkpoint at models/my_ssd_resnet50_v1_
↪fpn
I0716 05:44:22.779590 17144 checkpoint_utils.py:125] Waiting for new␣
↪checkpoint at models/my_ssd_resnet50_v1_fpn
INFO:tensorflow:Found new checkpoint at models/my_ssd_resnet50_v1_fpn\
↪ckpt-2
I0716 05:44:22.882485 17144 checkpoint_utils.py:134] Found new checkpoint␣
↪at models/my_ssd_resnet50_v1_fpn\ckpt-2
```

While the evaluation process is running, it will periodically check (every 300 sec by default) and use the latest `models/my_ssd_resnet50_v1_fpn/ckpt-*` checkpoint files to evaluate the performance of the model. The results are stored in the form of tf event files (`events.out.tfevents.*`) inside `models/my_ssd_resnet50_v1_fpn/eval_0`. These files can then be used to monitor the computed metrics, using the process described by the next section.

2.6 Monitor Training Job Progress using TensorBoard

A very nice feature of TensorFlow, is that it allows you to coninuously monitor and visualise a number of different training/evaluation metrics, while your model is being trained. The specific tool that allows us to do all that is Tensorboard.

To start a new TensorBoard server, we follow the following steps:

- Open a new *Anaconda/Command Prompt*
- Activate your TensorFlow conda environment (if you have one), e.g.:

  ```
  activate tensorflow_gpu
  ```

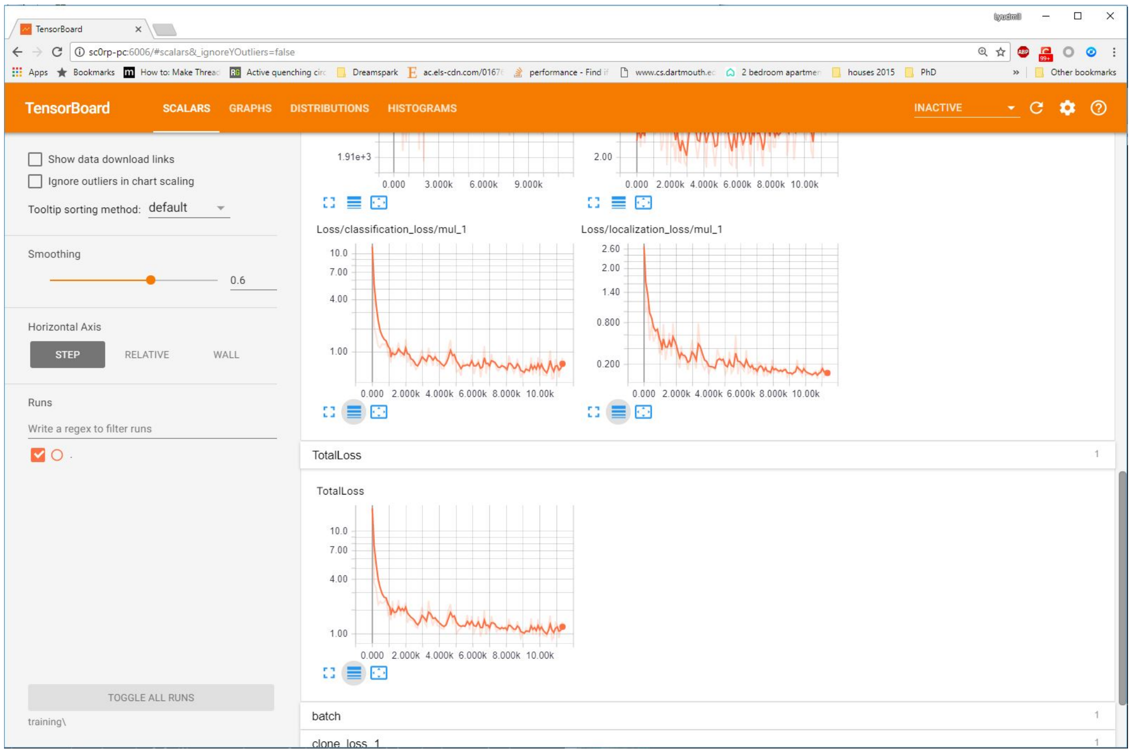
- `cd` into the `training_demo` folder.
- Run the following command:

  ```
  tensorboard --logdir=models/my_ssd_resnet50_v1_fpn
  ```

The above command will start a new TensorBoard server, which (by default) listens to port 6006 of your machine. Assuming that everything went well, you should see a print-out similar to the one below (plus/minus some warnings):

```
...
TensorBoard 2.2.2 at http://localhost:6006/ (Press CTRL+C to quit)
```

Once this is done, go to your browser and type `http://localhost:6006/` in your address bar, following which you should be presented with a dashboard similar to the one shown below (maybe less populated if your model has just started training):



2.7 Exporting a Trained Model

Once your training job is complete, you need to extract the newly trained inference graph, which will be later used to perform the object detection. This can be done as follows:

- Copy the `TensorFlow/models/research/object_detection/exporter_main_v2.py` script and paste it straight into your `training_demo` folder.
- Now, open a *Terminal*, `cd` inside your `training_demo` folder, and run the following command:

```
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path .\
↪models\my_efficientdet_d1\pipeline.config --trained_checkpoint_dir .\models\my_
↪efficientdet_d1\ --output_directory .\exported-models\my_model
```

After the above process has completed, you should find a new folder `my_model` under the `training_demo/exported-models`, that has the following structure:

```
training_demo/
├─ ...
├─ exported-models/
│  └─ my_model/
│     ├─ checkpoint/
│     ├─ saved_model/
│     └─ pipeline.config
└─ ...
```

This model can then be used to perform inference.

Note: You may get the following error when trying to export your model:

```
Traceback (most recent call last):
  File ".\exporter_main_v2.py", line 126, in <module>
    app.run(main)
  File "C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-packages\absl\app.py", line 299,
↪ in run
    _run_main(main, args)
  ...
  File "C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-packages\tensorflow\python\
↪keras\engine\base_layer.py", line 1627, in get_losses_for
    reachable = tf_utils.get_reachable_from_inputs(inputs, losses)
  File "C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-packages\tensorflow\python\
↪keras\utils\tf_utils.py", line 140, in get_reachable_from_inputs
    raise TypeError('Expected Operation, Variable, or Tensor, got ' + str(x))
TypeError: Expected Operation, Variable, or Tensor, got level_5
```

If this happens, have a look at the *"TypeError: Expected Operation, Variable, or Tensor, got level_5"* issue section for a potential solution.

CHAPTER

THREE

EXAMPLES

Below is a gallery of examples

3.1 Detect Objects Using Your Webcam

This demo will take you through the steps of running an "out-of-the-box" detection model to detect objects in the video stream extracted from your camera.

3.1.1 Create the data directory

The snippet shown below will create the `data` directory where all our data will be stored. The code will create a directory structure as shown bellow:

```
data
└── models
```

where the `models` folder will will contain the downloaded models.

```
import os

DATA_DIR = os.path.join(os.getcwd(), 'data')
MODELS_DIR = os.path.join(DATA_DIR, 'models')
for dir in [DATA_DIR, MODELS_DIR]:
    if not os.path.exists(dir):
        os.mkdir(dir)
```

3.1.2 Download the model

The code snippet shown below is used to download the object detection model checkpoint file, as well as the labels file (.pbtxt) which contains a list of strings used to add the correct label to each detection (e.g. person).

The particular detection algorithm we will use is the *SSD ResNet101 V1 FPN 640x640*. More models can be found in the TensorFlow 2 Detection Model Zoo. To use a different model you will need the URL name of the specific model. This can be done as follows:

1. Right click on the *Model name* of the model you would like to use;
2. Click on *Copy link address* to copy the download link of the model;
3. Paste the link in a text editor of your choice. You should observe a link similar to `download.tensorflow.org/models/object_detection/tf2/YYYYYYYY/XXXXXXXXX.tar.gz`;

4. Copy the XXXXXXXXX part of the link and use it to replace the value of the MODEL_NAME variable in the code shown below;
5. Copy the YYYYYYYY part of the link and use it to replace the value of the MODEL_DATE variable in the code shown below.

For example, the download link for the model used below is: download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet101_v1_fpn_640x640_coco17_tpu-8.tar.gz

```
import tarfile
import urllib.request

# Download and extract model
MODEL_DATE = '20200711'
MODEL_NAME = 'ssd_resnet101_v1_fpn_640x640_coco17_tpu-8'
MODEL_TAR_FILENAME = MODEL_NAME + '.tar.gz'
MODELS_DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/tf2/'
MODEL_DOWNLOAD_LINK = MODELS_DOWNLOAD_BASE + MODEL_DATE + '/' + MODEL_TAR_FILENAME
PATH_TO_MODEL_TAR = os.path.join(MODELS_DIR, MODEL_TAR_FILENAME)
PATH_TO_CKPT = os.path.join(MODELS_DIR, os.path.join(MODEL_NAME, 'checkpoint/'))
PATH_TO_CFG = os.path.join(MODELS_DIR, os.path.join(MODEL_NAME, 'pipeline.config'))
if not os.path.exists(PATH_TO_CKPT):
    print('Downloading model. This may take a while... ', end='')
    urllib.request.urlretrieve(MODEL_DOWNLOAD_LINK, PATH_TO_MODEL_TAR)
    tar_file = tarfile.open(PATH_TO_MODEL_TAR)
    tar_file.extractall(MODELS_DIR)
    tar_file.close()
    os.remove(PATH_TO_MODEL_TAR)
    print('Done')

# Download labels file
LABEL_FILENAME = 'mscoco_label_map.pbtxt'
LABELS_DOWNLOAD_BASE = \
    'https://raw.githubusercontent.com/tensorflow/models/master/research/object_
↪detection/data/'
PATH_TO_LABELS = os.path.join(MODELS_DIR, os.path.join(MODEL_NAME, LABEL_FILENAME))
if not os.path.exists(PATH_TO_LABELS):
    print('Downloading label file... ', end='')
    urllib.request.urlretrieve(LABELS_DOWNLOAD_BASE + LABEL_FILENAME, PATH_TO_LABELS)
    print('Done')
```

3.1.3 Load the model

Next we load the downloaded model

```
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder

tf.get_logger().setLevel('ERROR')           # Suppress TensorFlow logging (2)

# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
```

(continues on next page)

(continued from previous page)

```
    tf.config.experimental.set_memory_growth(gpu, True)

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(PATH_TO_CFG)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(PATH_TO_CKPT, 'ckpt-0')).expect_partial()

@tf.function
def detect_fn(image):
    """Detect objects in image."""

    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)

    return detections, prediction_dict, tf.reshape(shapes, [-1])
```

3.1.4 Load label map data (for plotting)

Label maps correspond index numbers to category names, so that when our convolution network predicts *5*, we know that this corresponds to *airplane*. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine.

```
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                    use_display_
↪name=True)
```

3.1.5 Define the video stream

We will use OpenCV to capture the video stream generated by our webcam. For more information you can refer to the OpenCV-Python Tutorials

```
import cv2

cap = cv2.VideoCapture(0)
```

3.1.6 Putting everything together

The code shown below loads an image, runs it through the detection model and visualizes the detection results, including the keypoints.

Note that this will take a long time (several minutes) the first time you run this code due to tf.function's trace-compilation — on subsequent runs (e.g. on new images), things will be faster.

Here are some simple things to try out if you are curious:

- Modify some of the input images and see if detection still works. Some simple things to try out here (just uncomment the relevant portions of code) include flipping the image horizontally, or converting to grayscale (note that we still expect the input image to have 3 channels).

- Print out *detections['detection_boxes']* and try to match the box locations to the boxes in the image. Notice that coordinates are given in normalized form (i.e., in the interval [0, 1]).
- Set `min_score_thresh` to other values (between 0 and 1) to allow more detections in or to filter out more detections.

```
import numpy as np

while True:
    # Read frame from camera
    ret, image_np = cap.read()

    # Expand dimensions since the model expects images to have shape: [1, None, None,␣
↪3]
    image_np_expanded = np.expand_dims(image_np, axis=0)

    # Things to try:
    # Flip horizontally
    # image_np = np.fliplr(image_np).copy()

    # Convert image to grayscale
    # image_np = np.tile(
    #     np.mean(image_np, 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections, predictions_dict, shapes = detect_fn(input_tensor)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
          image_np_with_detections,
          detections['detection_boxes'][0].numpy(),
          (detections['detection_classes'][0].numpy() + label_id_offset).astype(int),
          detections['detection_scores'][0].numpy(),
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw=200,
          min_score_thresh=.30,
          agnostic_mode=False)

    # Display output
    cv2.imshow('object detection', cv2.resize(image_np_with_detections, (800, 600)))

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Total running time of the script: (0 minutes 0.000 seconds)

3.2 Object Detection From TF1 Saved Model

This demo will take you through the steps of running an "out-of-the-box" TensorFlow 1 compatible detection model on a collection of images. More specifically, in this example we will be using the Saved Model Format to load the model.

3.2.1 Download the test images

First we will download the images that we will use throughout this tutorial. The code snippet shown bellow will download the test images from the TensorFlow Model Garden and save them inside the `data/images` folder.

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging (1)
import pathlib
import tensorflow as tf

tf.get_logger().setLevel('ERROR')           # Suppress TensorFlow logging (2)

# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

def download_images():
    base_url = 'https://raw.githubusercontent.com/tensorflow/models/master/research/
↪object_detection/test_images/'
    filenames = ['image1.jpg', 'image2.jpg']
    image_paths = []
    for filename in filenames:
        image_path = tf.keras.utils.get_file(fname=filename,
                                            origin=base_url + filename,
                                            untar=False)
        image_path = pathlib.Path(image_path)
        image_paths.append(str(image_path))
    return image_paths

IMAGE_PATHS = download_images()
```

3.2.2 Download the model

The code snippet shown below is used to download the pre-trained object detection model we shall use to perform inference. The particular detection algorithm we will use is the *SSD MobileNet v2*. More models can be found in the TensorFlow 1 Detection Model Zoo. To use a different model you will need the URL name of the specific model. This can be done as follows:

1. Right click on the *Model name* of the model you would like to use;
2. Click on *Copy link address* to copy the download link of the model;
3. Paste the link in a text editor of your choice. You should observe a link similar to `download.tensorflow.org/models/object_detection/XXXXXXXXX.tar.gz`;
4. Copy the `XXXXXXXXX` part of the link and use it to replace the value of the `MODEL_NAME` variable in the code shown below;

For example, the download link for the model used below is: `download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz`

```
# Download and extract model
def download_model(model_name):
    base_url = 'http://download.tensorflow.org/models/object_detection/'
    model_file = model_name + '.tar.gz'
    model_dir = tf.keras.utils.get_file(fname=model_name,
                                        origin=base_url + model_file,
                                        untar=True)
    return str(model_dir)

MODEL_NAME = 'ssd_mobilenet_v2_coco_2018_03_29'
PATH_TO_MODEL_DIR = download_model(MODEL_NAME)
```

3.2.3 Download the labels

The coode snippet shown below is used to download the labels file (.pbtxt) which contains a list of strings used to add the correct label to each detection (e.g. person). Since the pre-trained model we will use has been trained on the COCO dataset, we will need to download the labels file corresponding to this dataset, named `mscoco_label_map.pbtxt`. A full list of the labels files included in the TensorFlow Models Garden can be found here.

```
# Download labels file
def download_labels(filename):
    base_url = 'https://raw.githubusercontent.com/tensorflow/models/master/research/
↪object_detection/data/'
    label_dir = tf.keras.utils.get_file(fname=filename,
                                        origin=base_url + filename,
                                        untar=False)
    label_dir = pathlib.Path(label_dir)
    return str(label_dir)

LABEL_FILENAME = 'mscoco_label_map.pbtxt'
PATH_TO_LABELS = download_labels(LABEL_FILENAME)
```

3.2.4 Load the model

Next we load the downloaded model

```
import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

PATH_TO_SAVED_MODEL = PATH_TO_MODEL_DIR + "/saved_model"

print('Loading model...', end='')
start_time = time.time()

# Load saved model and build the detection function
model = tf.saved_model.load(PATH_TO_SAVED_MODEL)
detect_fn = model.signatures['serving_default']

end_time = time.time()
elapsed_time = end_time - start_time
print('Done! Took {} seconds'.format(elapsed_time))
```

Out:

```
Loading model...Done! Took 9.374149322509766 seconds
```

3.2.5 Load label map data (for plotting)

Label maps correspond index numbers to category names, so that when our convolution network predicts *5*, we know that this corresponds to *airplane*. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine.

```
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                    use_display_
↪name=True)
```

3.2.6 Putting everything together

The code shown below loads an image, runs it through the detection model and visualizes the detection results, including the keypoints.

Note that this will take a long time (several minutes) the first time you run this code due to tf.function's trace-compilation — on subsequent runs (e.g. on new images), things will be faster.

Here are some simple things to try out if you are curious:

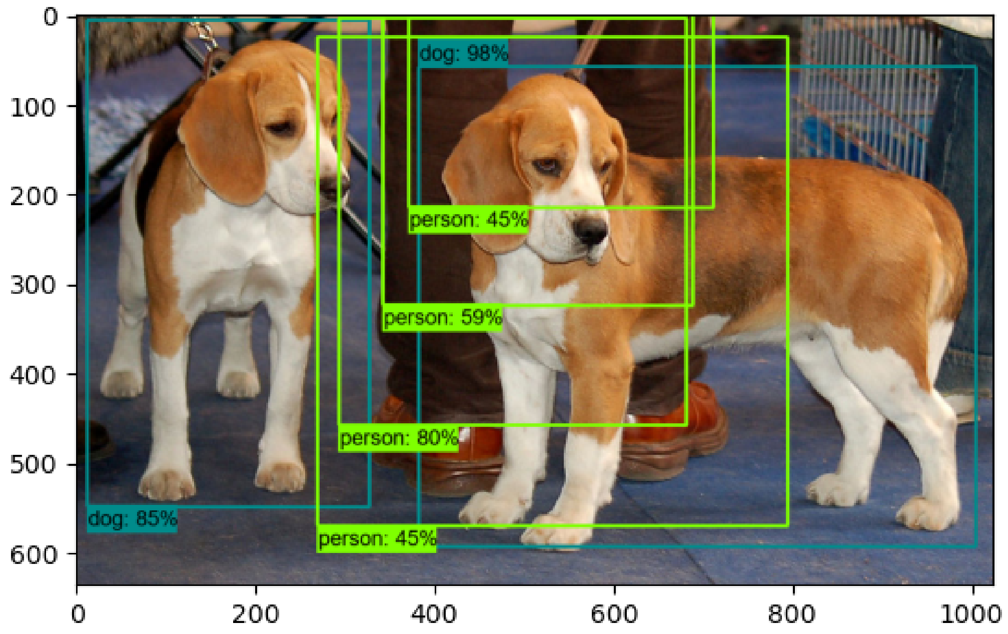
- Modify some of the input images and see if detection still works. Some simple things to try out here (just uncomment the relevant portions of code) include flipping the image horizontally, or converting to grayscale (note that we still expect the input image to have 3 channels).
- Print out *detections['detection_boxes']* and try to match the box locations to the boxes in the image. Notice that coordinates are given in normalized form (i.e., in the interval [0, 1]).
- Set `min_score_thresh` to other values (between 0 and 1) to allow more detections in or to filter out more detections.

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')   # Suppress Matplotlib warnings

def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.

    Puts image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
    (height, width, channels), where channels=3 for RGB.

    Args:
      path: the file path to the image

    Returns:
      uint8 numpy array with shape (img_height, img_width, 3)
    """
    return np.array(Image.open(path))


```

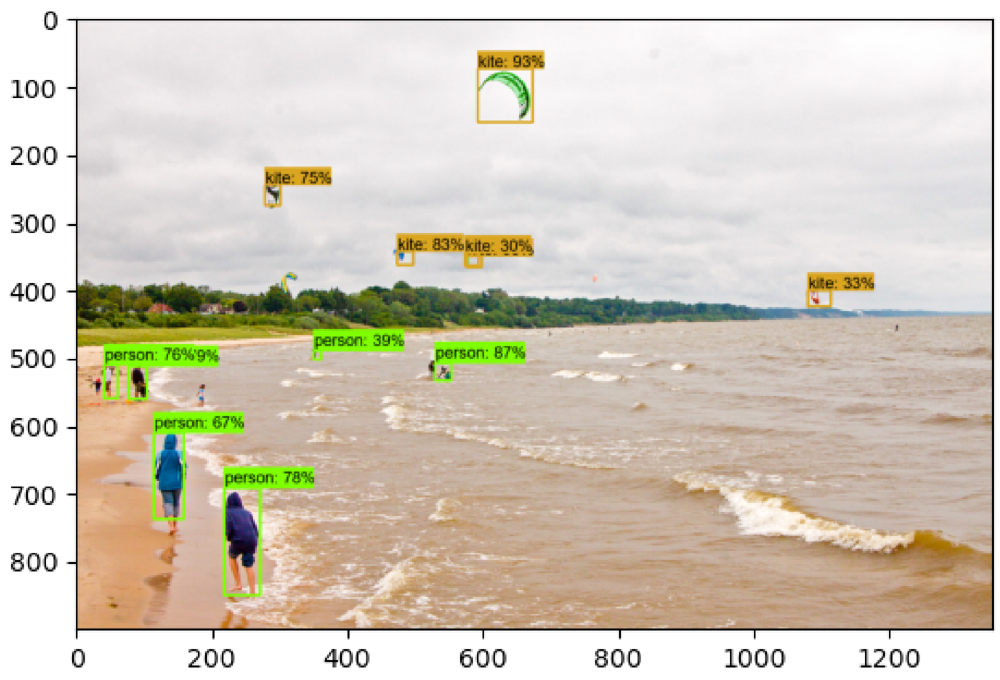
(continues on next page)

(continued from previous page)

```
for image_path in IMAGE_PATHS:

    print('Running inference for {}... '.format(image_path), end='')

    image_np = load_image_into_numpy_array(image_path)

    # Things to try:
    # Flip horizontally
    # image_np = np.fliplr(image_np).copy()

    # Convert image to grayscale
    # image_np = np.tile(
    #     np.mean(image_np, 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

    # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
    input_tensor = tf.convert_to_tensor(image_np)
    # The model expects a batch of images, so add an axis with `tf.newaxis`.
    input_tensor = input_tensor[tf.newaxis, ...]

    detections = detect_fn(input_tensor)

    # All outputs are batches tensors.
    # Convert to numpy arrays, and take index [0] to remove the batch dimension.
    # We're only interested in the first num_detections.
    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
          image_np_with_detections,
          detections['detection_boxes'],
          detections['detection_classes'],
          detections['detection_scores'],
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw=200,
          min_score_thresh=.30,
          agnostic_mode=False)

    plt.figure()
    plt.imshow(image_np_with_detections)
    print('Done')
plt.show()

# sphinx_gallery_thumbnail_number = 2
```


dog: 98%
dog: 85%
person: 45%
person: 59%
person: 80%
person: 45%
0
100
200
300
400
500
600
0
200
400
600
800
1000

-



•

Out:

```
Running inference for C:\Users\sglvladi\.keras\datasets\image1.jpg... Done
Running inference for C:\Users\sglvladi\.keras\datasets\image2.jpg... Done
```

Total running time of the script: (0 minutes 16.843 seconds)

3.3 Object Detection From TF2 Saved Model

This demo will take you through the steps of running an "out-of-the-box" TensorFlow 2 compatible detection model on a collection of images. More specifically, in this example we will be using the Saved Model Format to load the model.

3.3.1 Download the test images

First we will download the images that we will use throughout this tutorial. The code snippet shown bellow will download the test images from the TensorFlow Model Garden and save them inside the `data/images` folder.

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging (1)
import pathlib
import tensorflow as tf
```

(continues on next page)

(continued from previous page)

```
tf.get_logger().setLevel('ERROR')           # Suppress TensorFlow logging (2)

# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

def download_images():
    base_url = 'https://raw.githubusercontent.com/tensorflow/models/master/research/
↪object_detection/test_images/'
    filenames = ['image1.jpg', 'image2.jpg']
    image_paths = []
    for filename in filenames:
        image_path = tf.keras.utils.get_file(fname=filename,
                                            origin=base_url + filename,
                                            untar=False)
        image_path = pathlib.Path(image_path)
        image_paths.append(str(image_path))
    return image_paths

IMAGE_PATHS = download_images()
```

3.3.2 Download the model

The code snippet shown below is used to download the pre-trained object detection model we shall use to perform inference. The particular detection algorithm we will use is the *CenterNet HourGlass104 1024x1024*. More models can be found in the TensorFlow 2 Detection Model Zoo. To use a different model you will need the URL name of the specific model. This can be done as follows:

1. Right click on the *Model name* of the model you would like to use;
2. Click on *Copy link address* to copy the download link of the model;
3. Paste the link in a text editor of your choice. You should observe a link similar to `download.tensorflow.org/models/object_detection/tf2/YYYYYYYY/XXXXXXXXX.tar.gz`;
4. Copy the `XXXXXXXXX` part of the link and use it to replace the value of the `MODEL_NAME` variable in the code shown below;
5. Copy the `YYYYYYYY` part of the link and use it to replace the value of the `MODEL_DATE` variable in the code shown below.

For example, the download link for the model used below is: `download.tensorflow.org/models/object_detection/tf2/20200711/centernet_hg104_1024x1024_coco17_tpu-32.tar.gz`

```
# Download and extract model
def download_model(model_name, model_date):
    base_url = 'http://download.tensorflow.org/models/object_detection/tf2/'
    model_file = model_name + '.tar.gz'
    model_dir = tf.keras.utils.get_file(fname=model_name,
                                        origin=base_url + model_date + '/' + model_
↪file,
                                        untar=True)
    return str(model_dir)

MODEL_DATE = '20200711'
```

(continues on next page)

(continued from previous page)

```
MODEL_NAME = 'centernet_hg104_1024x1024_coco17_tpu-32'
PATH_TO_MODEL_DIR = download_model(MODEL_NAME, MODEL_DATE)
```

3.3.3 Download the labels

The coode snippet shown below is used to download the labels file (.pbtxt) which contains a list of strings used to add the correct label to each detection (e.g. person). Since the pre-trained model we will use has been trained on the COCO dataset, we will need to download the labels file corresponding to this dataset, named `mscoco_label_map.pbtxt`. A full list of the labels files included in the TensorFlow Models Garden can be found here.

```
# Download labels file
def download_labels(filename):
    base_url = 'https://raw.githubusercontent.com/tensorflow/models/master/research/
↪object_detection/data/'
    label_dir = tf.keras.utils.get_file(fname=filename,
                                        origin=base_url + filename,
                                        untar=False)
    label_dir = pathlib.Path(label_dir)
    return str(label_dir)

LABEL_FILENAME = 'mscoco_label_map.pbtxt'
PATH_TO_LABELS = download_labels(LABEL_FILENAME)
```

3.3.4 Load the model

Next we load the downloaded model

```
import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

PATH_TO_SAVED_MODEL = PATH_TO_MODEL_DIR + "/saved_model"

print('Loading model...', end='')
start_time = time.time()

# Load saved model and build the detection function
detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)

end_time = time.time()
elapsed_time = end_time - start_time
print('Done! Took {} seconds'.format(elapsed_time))
```

Out:

```
Loading model...Done! Took 42.73200988769531 seconds
```

3.3.5 Load label map data (for plotting)

Label maps correspond index numbers to category names, so that when our convolution network predicts *5*, we know that this corresponds to *airplane*. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine.

```
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                    use_display_
↪name=True)
```

3.3.6 Putting everything together

The code shown below loads an image, runs it through the detection model and visualizes the detection results, including the keypoints.

Note that this will take a long time (several minutes) the first time you run this code due to tf.function's trace-compilation — on subsequent runs (e.g. on new images), things will be faster.

Here are some simple things to try out if you are curious:

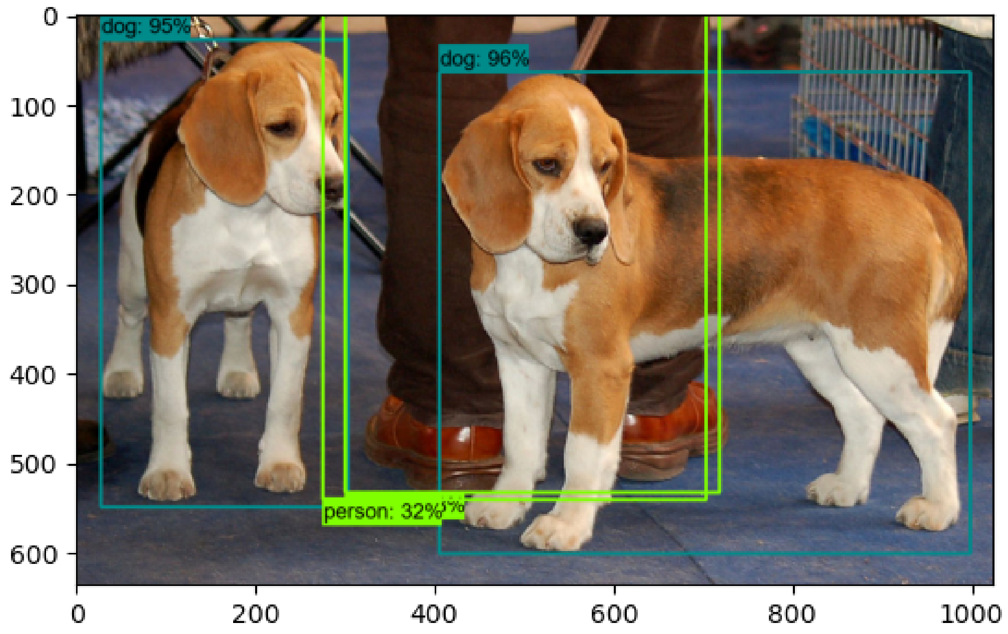
- Modify some of the input images and see if detection still works. Some simple things to try out here (just uncomment the relevant portions of code) include flipping the image horizontally, or converting to grayscale (note that we still expect the input image to have 3 channels).
- Print out *detections['detection_boxes']* and try to match the box locations to the boxes in the image. Notice that coordinates are given in normalized form (i.e., in the interval [0, 1]).
- Set `min_score_thresh` to other values (between 0 and 1) to allow more detections in or to filter out more detections.

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')   # Suppress Matplotlib warnings

def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.

    Puts image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
    (height, width, channels), where channels=3 for RGB.

    Args:
      path: the file path to the image

    Returns:
      uint8 numpy array with shape (img_height, img_width, 3)
    """
    return np.array(Image.open(path))


for image_path in IMAGE_PATHS:

    print('Running inference for {}... '.format(image_path), end='')

    image_np = load_image_into_numpy_array(image_path)
```

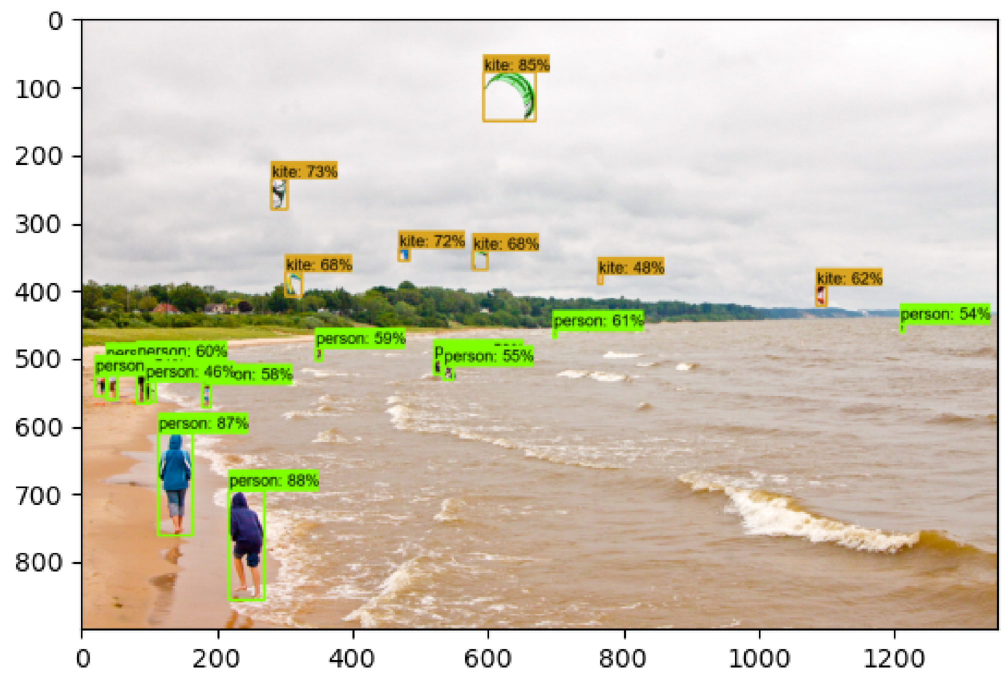
(continues on next page)

(continued from previous page)

```
    # Things to try:
    # Flip horizontally
    # image_np = np.fliplr(image_np).copy()

    # Convert image to grayscale
    # image_np = np.tile(
    #     np.mean(image_np, 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

    # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
    input_tensor = tf.convert_to_tensor(image_np)
    # The model expects a batch of images, so add an axis with `tf.newaxis`.
    input_tensor = input_tensor[tf.newaxis, ...]

    # input_tensor = np.expand_dims(image_np, 0)
    detections = detect_fn(input_tensor)

    # All outputs are batches tensors.
    # Convert to numpy arrays, and take index [0] to remove the batch dimension.
    # We're only interested in the first num_detections.
    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
          image_np_with_detections,
          detections['detection_boxes'],
          detections['detection_classes'],
          detections['detection_scores'],
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw=200,
          min_score_thresh=.30,
          agnostic_mode=False)

    plt.figure()
    plt.imshow(image_np_with_detections)
    print('Done')
plt.show()

# sphinx_gallery_thumbnail_number = 2
```


dog: 95%
dog: 96%
person: 32%
0
100
200
300
400
500
600
0
200
400
600
800
1000

•



-

Out:

```
Running inference for C:\Users\sglvladi\.keras\datasets\image1.jpg... Done
Running inference for C:\Users\sglvladi\.keras\datasets\image2.jpg... Done
```

Total running time of the script: (0 minutes 48.355 seconds)

3.4 Object Detection From TF2 Checkpoint

This demo will take you through the steps of running an "out-of-the-box" TensorFlow 2 compatible detection model on a collection of images. More specifically, in this example we will be using the Checkpoint Format to load the model.

3.4.1 Download the test images

First we will download the images that we will use throughout this tutorial. The code snippet shown bellow will download the test images from the TensorFlow Model Garden and save them inside the `data/images` folder.

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging (1)
import pathlib
import tensorflow as tf

```

(continues on next page)

(continued from previous page)

```
tf.get_logger().setLevel('ERROR')           # Suppress TensorFlow logging (2)

# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

def download_images():
    base_url = 'https://raw.githubusercontent.com/tensorflow/models/master/research/
↪object_detection/test_images/'
    filenames = ['image1.jpg', 'image2.jpg']
    image_paths = []
    for filename in filenames:
        image_path = tf.keras.utils.get_file(fname=filename,
                                            origin=base_url + filename,
                                            untar=False)
        image_path = pathlib.Path(image_path)
        image_paths.append(str(image_path))
    return image_paths

IMAGE_PATHS = download_images()
```

3.4.2 Download the model

The code snippet shown below is used to download the pre-trained object detection model we shall use to perform inference. The particular detection algorithm we will use is the *CenterNet HourGlass104 1024x1024*. More models can be found in the TensorFlow 2 Detection Model Zoo. To use a different model you will need the URL name of the specific model. This can be done as follows:

1. Right click on the *Model name* of the model you would like to use;
2. Click on *Copy link address* to copy the download link of the model;
3. Paste the link in a text editor of your choice. You should observe a link similar to `download.tensorflow.org/models/object_detection/tf2/YYYYYYYY/XXXXXXXXX.tar.gz`;
4. Copy the `XXXXXXXXX` part of the link and use it to replace the value of the `MODEL_NAME` variable in the code shown below;
5. Copy the `YYYYYYYY` part of the link and use it to replace the value of the `MODEL_DATE` variable in the code shown below.

For example, the download link for the model used below is: `download.tensorflow.org/models/object_detection/tf2/20200711/centernet_hg104_1024x1024_coco17_tpu-32.tar.gz`

```
# Download and extract model
def download_model(model_name, model_date):
    base_url = 'http://download.tensorflow.org/models/object_detection/tf2/'
    model_file = model_name + '.tar.gz'
    model_dir = tf.keras.utils.get_file(fname=model_name,
                                        origin=base_url + model_date + '/' + model_
↪file,
                                        untar=True)
    return str(model_dir)

MODEL_DATE = '20200711'
```

(continues on next page)

(continued from previous page)

```
MODEL_NAME = 'centernet_hg104_1024x1024_coco17_tpu-32'
PATH_TO_MODEL_DIR = download_model(MODEL_NAME, MODEL_DATE)
```

3.4.3 Download the labels

The coode snippet shown below is used to download the labels file (.pbtxt) which contains a list of strings used to add the correct label to each detection (e.g. person). Since the pre-trained model we will use has been trained on the COCO dataset, we will need to download the labels file corresponding to this dataset, named `mscoco_label_map.pbtxt`. A full list of the labels files included in the TensorFlow Models Garden can be found here.

```
# Download labels file
def download_labels(filename):
    base_url = 'https://raw.githubusercontent.com/tensorflow/models/master/research/
↪object_detection/data/'
    label_dir = tf.keras.utils.get_file(fname=filename,
                                        origin=base_url + filename,
                                        untar=False)
    label_dir = pathlib.Path(label_dir)
    return str(label_dir)

LABEL_FILENAME = 'mscoco_label_map.pbtxt'
PATH_TO_LABELS = download_labels(LABEL_FILENAME)
```

3.4.4 Load the model

Next we load the downloaded model

```
import time
from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder

PATH_TO_CFG = PATH_TO_MODEL_DIR + "/pipeline.config"
PATH_TO_CKPT = PATH_TO_MODEL_DIR + "/checkpoint"

print('Loading model... ', end='')
start_time = time.time()

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(PATH_TO_CFG)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(PATH_TO_CKPT, 'ckpt-0')).expect_partial()

@tf.function
def detect_fn(image):
    """Detect objects in image."""

    image, shapes = detection_model.preprocess(image)
```

(continues on next page)

(continued from previous page)

```
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)

    return detections

end_time = time.time()
elapsed_time = end_time - start_time
print('Done! Took {} seconds'.format(elapsed_time))
```

Out:

```
Loading model... Done! Took 0.7255048751831055 seconds
```

3.4.5 Load label map data (for plotting)

Label maps correspond index numbers to category names, so that when our convolution network predicts *5*, we know that this corresponds to *airplane*. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine.

```
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                    use_display_
↪name=True)
```

3.4.6 Putting everything together

The code shown below loads an image, runs it through the detection model and visualizes the detection results, including the keypoints.

Note that this will take a long time (several minutes) the first time you run this code due to tf.function's trace-compilation — on subsequent runs (e.g. on new images), things will be faster.

Here are some simple things to try out if you are curious:

- Modify some of the input images and see if detection still works. Some simple things to try out here (just uncomment the relevant portions of code) include flipping the image horizontally, or converting to grayscale (note that we still expect the input image to have 3 channels).
- Print out *detections['detection_boxes']* and try to match the box locations to the boxes in the image. Notice that coordinates are given in normalized form (i.e., in the interval [0, 1]).
- Set `min_score_thresh` to other values (between 0 and 1) to allow more detections in or to filter out more detections.

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')   # Suppress Matplotlib warnings

def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.

    Puts image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
```

(continues on next page)

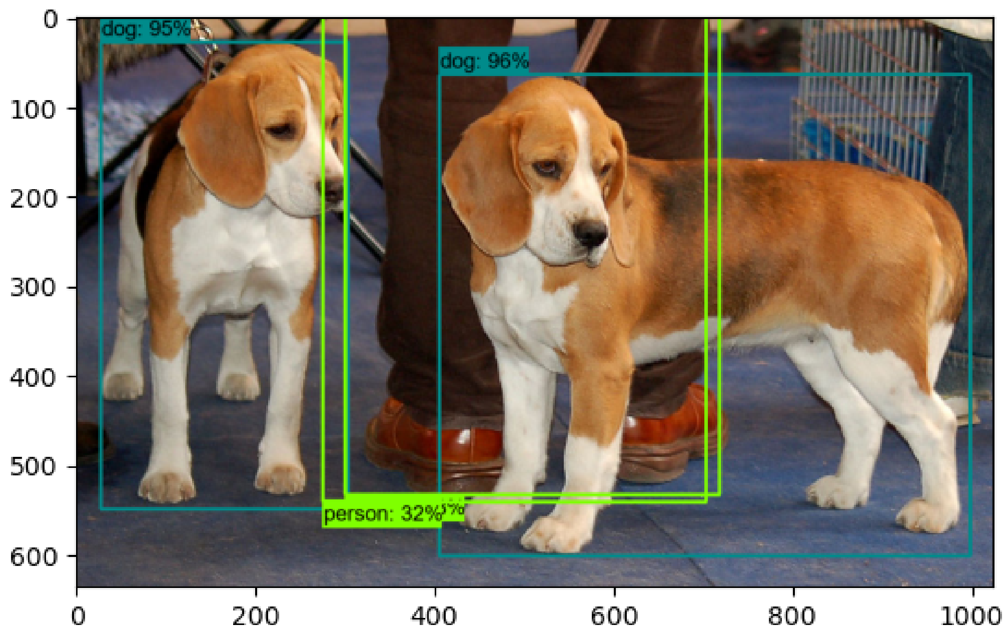
(continued from previous page)

```
    (height, width, channels), where channels=3 for RGB.

    Args:
      path: the file path to the image

    Returns:
      uint8 numpy array with shape (img_height, img_width, 3)
    """
    return np.array(Image.open(path))


for image_path in IMAGE_PATHS:

    print('Running inference for {}... '.format(image_path), end='')

    image_np = load_image_into_numpy_array(image_path)

    # Things to try:
    # Flip horizontally
    # image_np = np.fliplr(image_np).copy()

    # Convert image to grayscale
    # image_np = np.tile(
    #     np.mean(image_np, 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)

    detections = detect_fn(input_tensor)

    # All outputs are batches tensors.
    # Convert to numpy arrays, and take index [0] to remove the batch dimension.
    # We're only interested in the first num_detections.
    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
            detections['detection_boxes'],
            detections['detection_classes']+label_id_offset,
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=200,
            min_score_thresh=.30,
            agnostic_mode=False)

    plt.figure()
    plt.imshow(image_np_with_detections)
    print('Done')
```

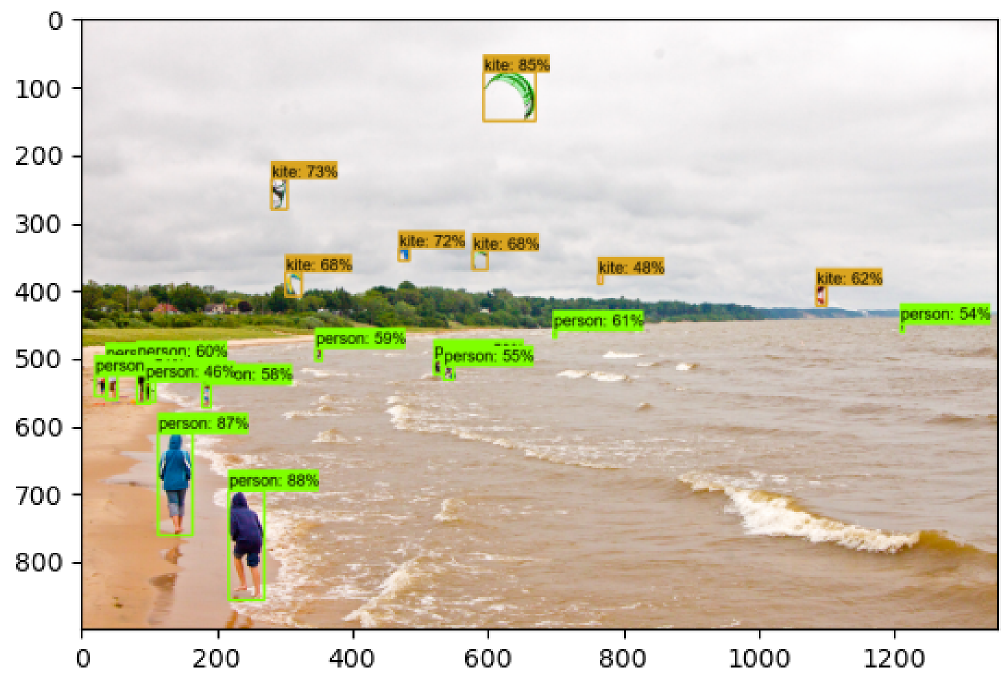
(continues on next page)

(continued from previous page)

```
plt.show()

# sphinx_gallery_thumbnail_number = 2
```



•



•

Out:

```
Running inference for C:\Users\sglvladi\.keras\datasets\image1.jpg... Done
Running inference for C:\Users\sglvladi\.keras\datasets\image2.jpg... Done
```

Total running time of the script: (0 minutes 20.082 seconds)

CHAPTER

FOUR

COMMON ISSUES

Below is a list of common issues encountered while using TensorFlow for objects detection.

4.1 Python crashes - TensorFlow GPU

If you are using *GPU Support (Optional)* and when you try to run some Python object detection script (e.g. *Test your Installation*), after a few seconds, Windows reports that Python has crashed then have a look at the *Anaconda/Command Prompt* window you used to run the script and check for a line similar (maybe identical) to the one below:

```
2018-03-22 03:07:54.623130: E C:\tf_jenkins\workspace\rel-win\M\windows-gpu\
↪PY\36\tensorflow\stream_executor\cuda\cuda_dnn.cc:378] Loaded runtime␣
↪CuDNN library: 7101 (compatibility version 7100) but source was compiled␣
↪with 7003 (compatibility version 7000).  If using a binary install,␣
↪upgrade your CuDNN library to match.  If building from sources, make sure␣
↪the library loaded at runtime matches a compatible version specified␣
↪during compile configuration.
```

If the above line is present in the printed debugging, it means that you have not installed the correct version of the cuDNN libraries. In this case make sure you re-do the *Install CUDNN* step, making sure you instal cuDNN v7.0.5.

4.2 Cleaning up Nvidia containers (TensorFlow GPU)

Sometimes, when terminating a TensorFlow training process, the Nvidia containers associated to the process are not cleanly terminated. This can lead to bogus errors when we try to run a new TensorFlow process.

Some known issues caused by the above are presented below:

- Failure to restart training of a model. Look for the following errors in the debugging:

```
2018-03-23 03:03:10.326902: E C:\tf_jenkins\workspace\rel-win\M\windows-
↪gpu\PY\36\tensorflow\stream_executor\cuda\cuda_dnn.cc:385] could not␣
↪create cudnn handle: CUDNN_STATUS_ALLOC_FAILED
2018-03-23 03:03:10.330475: E C:\tf_jenkins\workspace\rel-win\M\windows-
↪gpu\PY\36\tensorflow\stream_executor\cuda\cuda_dnn.cc:352] could not␣
↪destroy cudnn handle: CUDNN_STATUS_BAD_PARAM
2018-03-23 03:03:10.333797: W C:\tf_jenkins\workspace\rel-win\M\windows-
↪gpu\PY\36\tensorflow/stream_executor/stream.h:1983] attempting to␣
↪perform DNN operation using StreamExecutor without DNN support
2018-03-23 03:03:10.333807: I C:\tf_jenkins\workspace\rel-win\M\windows-
↪gpu\PY\36\tensorflow\stream_executor\stream.cc:1851] stream␣
↪00000216F05CB660 did not wait for stream: 00000216F05CA6E0
```

(continues on next page)

(continued from previous page)

```
2018-03-23 03:03:10.340765: I C:\tf_jenkins\workspace\rel-win\M\windows-
↪gpu\PY\36\tensorflow\stream_executor\stream.cc:4637] stream␣
↪00000216F05CB660 did not memcpy host-to-device; source: 000000020DB37B00
2018-03-23 03:03:10.343752: F C:\tf_jenkins\workspace\rel-win\M\windows-
↪gpu\PY\36\tensorflow\core\common_runtime\gpu\gpu_util.cc:343] CPU->GPU␣
↪Memcpy failed
```

To solve such issues in Windows, open a *Task Manager* windows, look for Tasks with name `NVIDIA Container` and kill them by selecting them and clicking the *End Task* button at the bottom left corner of the window.

If the issue persists, then you're probably running out of memory. Try closing down anything else that might be eating up your GPU memory (e.g. Youtube videos, webpages etc.)

4.3 "WARNING:tensorflow:Entity `<bound method X of <Y>>` could not be transformed . . . "

In some versions of Tensorflow, you may see errors that look similar to the ones below:

```
...
WARNING:tensorflow:Entity <bound method Conv.call of <tensorflow.python.layers.
↪convolutional.Conv2D object at 0x000001E92103EDD8>> could not be transformed and␣
↪will be executed as-is. Please report this to the AutgoGraph team. When filing the␣
↪bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach␣
↪the full output. Cause: converting <bound method Conv.call of <tensorflow.python.
↪layers.convolutional.Conv2D object at 0x000001E92103EDD8>>: AssertionError: Bad␣
↪argument number for Name: 3, expecting 4
WARNING:tensorflow:Entity <bound method BatchNormalization.call of <tensorflow.python.
↪layers.normalization.BatchNormalization object at 0x000001E9225EBA90>> could not be␣
↪transformed and will be executed as-is. Please report this to the AutgoGraph team.␣
↪When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_
↪VERBOSITY=10`) and attach the full output. Cause: converting <bound method␣
↪BatchNormalization.call of <tensorflow.python.layers.normalization.
↪BatchNormalization object at 0x000001E9225EBA90>>: AssertionError: Bad argument␣
↪number for Name: 3, expecting 4
...
```

These warnings appear to be harmless form my experience, however they can saturate the console with unnecessary messages, which makes it hard to scroll through the output of the training/evaluation process.

As reported here, this issue seems to be caused by a mismatched version of gast. Simply downgrading gast to version `0.2.2` seems to remove the warnings. This can be done by running:

```
pip install gast==0.2.2
```

4.4 "AttributeError: module 'google.protobuf.descriptor' has no attribute '_internal_create_key"

It is possible that when executing `from object_detection.utils import label_map_util` you may get the above error. As per the discussion is in this Stack Overflow thread, upgrading the Python protobuf version seems to solve this issue:

```
pip install --upgrade protobuf
```

4.5 "TypeError: Expected Operation, Variable, or Tensor, got level_5"

When trying to export oyu trained model using the `exporter_main_v2.py` script, you may come across an error that looks like this:

```
1  Traceback (most recent call last):
2    File ".\exporter_main_v2.py", line 126, in <module>
3      app.run(main)
4    File "C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-packages\absl\app.py", line 299,
   ↪ in run
5      _run_main(main, args)
6    ...
7    File "C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-packages\tensorflow\python\
   ↪keras\engine\base_layer.py", line 1627, in get_losses_for
8      reachable = tf_utils.get_reachable_from_inputs(inputs, losses)
9    File "C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-packages\tensorflow\python\
   ↪keras\utils\tf_utils.py", line 140, in get_reachable_from_inputs
10     raise TypeError('Expected Operation, Variable, or Tensor, got ' + str(x))
11 TypeError: Expected Operation, Variable, or Tensor, got level_5
```

This error seems to come from TensorFlow itself and a discussion on the issue can be found here. As discussed there, a fix to the above issue can be achieved by opening the `tf_utils.py` file and adding a line of code. Below is a summary of how this can be done:

- Look at the line that corresponds to line 9 (highlighted) in the above error print out.
- Copy the path to the `tf_utils.py` file; in my case this was `C:\Users\sglvladi\Anaconda3\envs\tf2\lib\site-packages\tensorflow\python\keras\utils\tf_utils.py`
- Open the file and replace line 140 of the file as follows:
  - Change:

    ```
    raise TypeError('Expected Operation, Variable, or Tensor, got ' + str(x))
    ```

    to:

    ```
    if not isinstance(x, str):
        raise TypeError('Expected Operation, Variable, or Tensor, got ' + str(x))
    ```

At the time of writting this tutorial, a fix to the issue had not been implemented in the version of TensorFlow installed using `pip`. It is possible that this will get incorporated at some later point.