Installation

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.

Anaconda Python 3.8 (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.

Install Anaconda Python 3.8

Windows | Linux

- Go to https://www.anaconda.com/products/individual and click the "Download" button
- Download the Python 3.8 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.

Create a new Anaconda virtual environment

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

```
conda create -n tensorflow pip python=3.9
```

- 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.)

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!

TensorFlow Installation

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

Install the TensorFlow PIP package

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

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

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)
```

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 v11.2
CuDNN 8.1.0

Install CUDA Toolkit

Windows | Linux

- Follow this link to download and install CUDA Toolkit 11.2
- Installation instructions can be found here

Install CUDNN

Windows | Linux

- Go to https://developer.nvidia.com/rdp/cudnn-download
- Create a user profile if needed and log in
- Select Download cuDNN v8.1.0 (January 26th, 2021), for CUDA 11.0,11.1 and 11.2
- Download cuDNN Library for Windows (x86)

- Extract the contents of the zip file (i.e. the folder named `cuda`) inside `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\`, where `<INSTALL_PATH>` points to the installation directory specified during the installation of the CUDA Toolkit. By default `<INSTALL_PATH>` = `C:\Program Files`.

Environment Setup

Windows | Linux

- 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\v11.2\bin`
    - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\libnvvp`
    - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\include`
    - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\extras\CUPTI\lib64`
    - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\cuda\bin`

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

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:

```
2021-06-08 18:28:38.452128: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library cudart64_110.dll
2021-06-08 18:28:40.948968: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library nvcuda.dll
2021-06-08 18:28:40.973992: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] 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
2021-06-08 18:28:40.974115: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library cudart64_110.dll
2021-06-08 18:28:40.982483: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library cublas64_11.dll
2021-06-08 18:28:40.982588: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library cublasLt64_11.dll
2021-06-08 18:28:40.986795: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library cufft64_10.dll
2021-06-08 18:28:40.988451: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library curand64_10.dll
2021-06-08 18:28:40.994115: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library cusolver64_11.dll
2021-06-08 18:28:40.998408: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library cusparse64_11.dll
2021-06-08 18:28:41.000573: I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Successfully opened dynamic library cudnn64_8.dll
2021-06-08 18:28:41.001094: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding
visible gpu devices: 0
2021-06-08 18:28:41.001651: I tensorflow/core/platform/cpu_feature_guard.cc:142] This
TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-06-08 18:28:41.003095: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] 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
2021-06-08 18:28:41.003244: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1871] Adding
visible gpu devices: 0
2021-06-08 18:28:42.072538: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device
interconnect StreamExecutor with strength 1 edge matrix:
2021-06-08 18:28:42.072630: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]      0
2021-06-08 18:28:42.072886: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1277] 0:   N
2021-06-08 18:28:42.075566: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1418]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 6613 MB
memory) -> physical GPU (device: 0, name: GeForce GTX 1070 Ti, pci bus id: 0000:02:00.0,
compute capability: 6.1)
tf.Tensor(641.5694, 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`.

TensorFlow Object Detection API Installation

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

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/
   └─ ...
```

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>\bin` 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 | Command Prompt

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

1

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

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 | Linux

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.

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.

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 --use-feature=2020-resolver .
```

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
    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.

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
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update): 0.0s
I0608 18:49:13.183754 29296 test_util.py:2102]
time(__main__.ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update
[ RUN      ] ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold):
0.0s
I0608 18:49:13.186750 29296 test_util.py:2102]
time(__main__.ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold
[ RUN      ] ModelBuilderTF2Test.test_invalid_model_config_proto
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_model_config_proto): 0.0s
I0608 18:49:13.188250 29296 test_util.py:2102]
time(__main__.ModelBuilderTF2Test.test_invalid_model_config_proto): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_model_config_proto
[ RUN      ] ModelBuilderTF2Test.test_invalid_second_stage_batch_size
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_second_stage_batch_size): 0.0s
I0608 18:49:13.190746 29296 test_util.py:2102]
time(__main__.ModelBuilderTF2Test.test_invalid_second_stage_batch_size): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_second_stage_batch_size
[ RUN      ] ModelBuilderTF2Test.test_session
[  SKIPPED ] ModelBuilderTF2Test.test_session
[ RUN      ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor):
0.0s
I0608 18:49:13.193742 29296 test_util.py:2102]
time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
[ RUN      ] ModelBuilderTF2Test.test_unknown_meta_architecture
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
I0608 18:49:13.195241 29296 test_util.py:2102]
time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_meta_architecture
[ RUN      ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
I0608 18:49:13.197239 29296 test_util.py:2102]
time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
----------------------------------------------------------------------
Ran 24 tests in 29.980s

OK (skipped=1)
```

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.