

# Configuration Manual

MSc Research Project  
Data Analytics

Abhinav Bhardwaj  
Student ID: x20100906

School of Computing  
National College of Ireland

Supervisor: Prof Aaloka Anant

National College of Ireland  
Project Submission Sheet  
School of Computing



<b>Student Name:</b>	Abhinav Bhardwaj
<b>Student ID:</b>	x20100906
<b>Programme:</b>	MSc Data Analytics
<b>Year:</b>	2021-22
<b>Module:</b>	MSc Research Project
<b>Supervisor:</b>	Prof Aaloka Anant
<b>Submission Due Date:</b>	31/01/2022
<b>Project Title:</b>	Configuration Manual
<b>Word Count:</b>	1409
<b>Page Count:</b>	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

<b>Signature:</b>	Abhinav Bhardwaj
<b>Date:</b>	31st January 2022

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project</b> , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Abhinav Bhardwaj  
x20100906

## 1 Introduction

The actions taken to carry out this research's implementation are described in full in this configuration manual. Data collection and processing, feature extraction, and model creation are all part of this process. In order to assure reproducibility, the code samples, screenshots, and step-by-step instructions are also included.

## 2 Hardware and Software configurations

<b>Host machine/Operating System:</b>	MacBook Pro/MacOS Catalina And Windows Machine (AMD Ryzen 9 5900HS with Radeon Graphics 3.30 GHz)/Windows 10 Home
<b>RAM</b>	8 GB, M1 chip processor And 16 GB (For Windows).
<b>Hard Disk</b>	256GB And 1TB SSD (For Windows)
<b>Cloud compute (GPU)</b>	Free GPU Tesla K80 offered by Colab with 2496 CUDA cores and 12GB RAM.

Table 1: Hardware specifications

<b>Programming language</b>	Python (Anaconda distribution)
<b>IDE</b>	Jupyter notebook.
<b>Cloud environment</b>	Google Collaboratory
<b>Browser</b>	Google chrome

Table 2: Software specifications

Data is first processed on the local workstation and saved as .tar.gz files before being transferred to Google Colab for modeling.

## 3 Data Preparation

The author has used self-created data. The file, post the split in training and testing, has been saved on Google Drive. Please follow the link to download the same.

Weapon dataset link : <https://drive.google.com/drive/folders/1zLBJ099QElaai0tSVLBwpdvfL8yG>

### 3.1 Creating Environment

Open Terminal/Comman Window

- Set up a new environment with name tfod using the following command:
- `!conda create --name tfod python=3.8` ; when asked for proceed : press Y
- `!conda activate tfod`
- `!python -m pip install --upgrade pip`
- `!pip install ipykernel`
- `python -m ipykernel install --user --name=tfodj`

Steps to install Tensorflow :

- Kindly refer the foot note for Tensorflow Github repository<sup>1</sup>
- `!git clone https://github.com/tensorflow/models.git`
- `!pip install --ignore-installed --upgrade tensorflow==2.5.0`
- verify your installation :
- `python -c "import tensorflow as tf;`
- `print(tf.reduce_sum(tf.random.normal([1000, 1000])))"`

You can skip steps 15 to 21 if you only want to run the second section of the code on google colab. Disclaimer: The last piece of the code, where one have to detect an object using a live feed from one's webcam. Code only run on one's local machine. Google Colab does not have any solutions in which one may attach a webcam and execute object detection on a live feed.

- To install the CUDA Toolkit as per the local machine's requirement and built follow the link: [https://developer.nvidia.co-11.2.2-download-archive?target\\_os=Linux&target\\_arch=x86\\_64](https://developer.nvidia.co-11.2.2-download-archive?target_os=Linux&target_arch=x86_64)
- To install the CUDNN follow : <https://developer.nvidia.com/rdp/cudnn-download>
- Create a user profile if needed and log in to select archive file for Cudnn: <https://developer.nvidia.com/archive-cudnn-11.2.2>
- 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 CDrive :Program Files.
- Download the latest protoc-\*.zip release from <https://github.com/protocolbuffers/protobuf/releases>
- Extract the contents of the downloaded protoc-\*.zip in a directory PATH.TO\_PB of your choice (e.g. C drive Program FilesProtobuf)

---

<sup>1</sup>Tensorflow Github repository : <https://github.com/tensorflow/models>

- Add PATH\_TO\_PB bin to your Path environment variable.
- In a new Terminal 1, cd into TensorFlow/models/research/ directory and run the following command: `!protoc object_detection/protos/*.proto --python_out=.`

We have to download Tensorflow 2 Object Detection API. For the same please follow the step under the same terminal/ command prompt:

- Kindly refer the foot note for Tensorflow API<sup>2</sup>
- Download the COCO API : `!git clone https://github.com/cocodataset/cocoapi.git`
- `cd cocoapi/PythonAPI`
- `!cp -r pycocotools PATH_TO_TF/models/research/`
- From within TensorFlow/models/research/
- `cp object_detection/packages/tf2/setup.py .`
- `python -m pip install --use-feature=2020-resolver .`
- Test the installation : From within TensorFlow/models/research/
- `!python object_detection/builders/model_builder_tf2_test.py`
- If everything goes fine run : jupyter notebook
- Post that run the first files Image\_Collection.

## 4 Project Development

PYTHON programming was used exclusively in the implementation. This research project is divided into three stages: data preparation, modeling, and evaluation. The first stage consists of data preprocessing and data selection, followed by the modeling stage, which consists of model implementation using TensorFlow, Keras, and Tensorflow Zoo model2, and finally, model evaluation using performance metrics such as average precision, average recall, and localization loss.

### 4.1 Data collection

- First, we will start importing the required libraries such as OpenCV2 and Time, which will help us capture the live images using our webcam. Kindly refer the figure 1
- In the following section, we will be setting up folders on our local machine. Where all our images, pre-trained models, and trained models will be saved. Kindly refer the figure 2

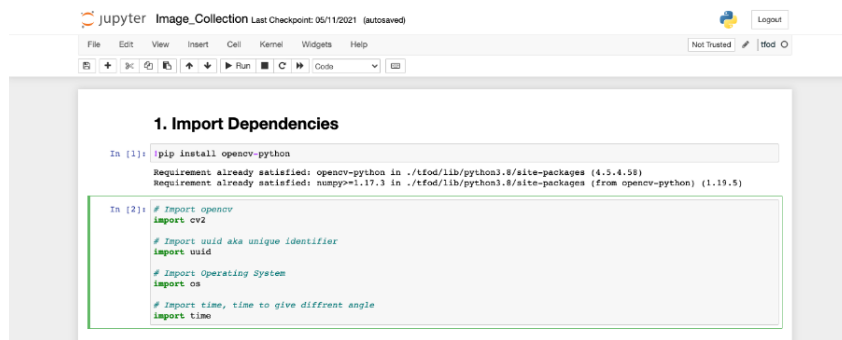


Figure 1: Import Lib

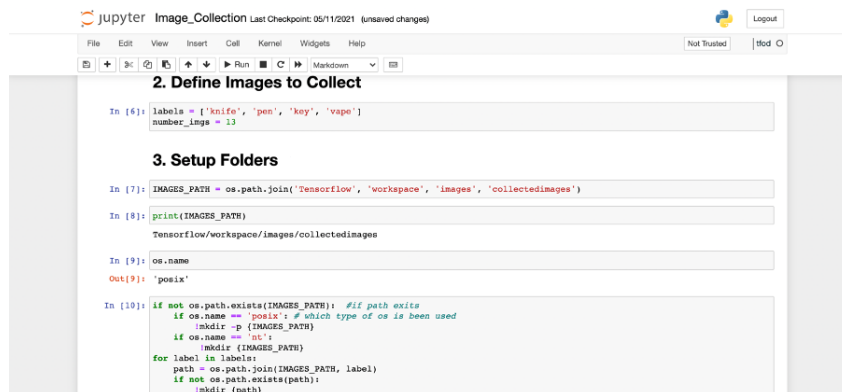


Figure 2: Setting up folders

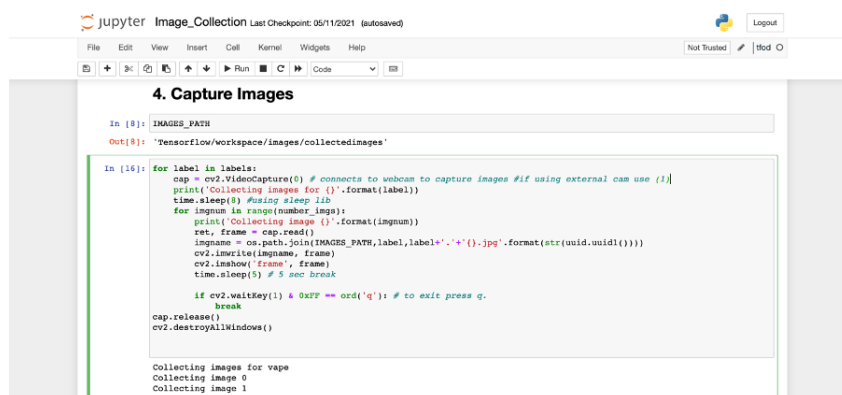


Figure 3: Image capture for Data collection

```

Jupyter Image_Collection Last Checkpoint: 05/11/2021 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help
Run Code

5. Image Labelling

In [9]: !pip install --upgrade pyqt5 lxml
Requirement already satisfied: pyqt5 in ./tfod/lib/python3.8/site-packages (5.15.6)
Requirement already satisfied: lxml in ./tfod/lib/python3.8/site-packages (4.6.4)
Collecting lxml
  Downloading lxml-4.7.1-cp38-cp38-macosx_10_14_x86_64.whl (4.5 MB)
    [REDACTED] 4.5 MB 1.9 MB/s
Requirement already satisfied: PyQT5=5.15.2 in ./tfod/lib/python3.8/site-packages (from pyqt5) (5.15.2)
Requirement already satisfied: PyQT5=5.15.2 in ./tfod/lib/python3.8/site-packages (from pyqt5) (5.15.2)
Installing collected packages: lxml
  Attempting uninstall: lxml
    Found existing installation: lxml 4.6.4
    Uninstalling lxml-4.6.4:
      Successfully uninstalled lxml-4.6.4
  Successfully installed lxml-4.7.1

In [10]: LABELING_PATH = os.path.join('Tensorflow', 'labeling')

In [11]: if not os.path.exists(LABELING_PATH):
         !mkdir (LABELING_PATH)
         !git clone https://github.com/tzutalin/labelImg (LABELING_PATH)

In [13]: if os.name == 'posix':
         !cd (LABELING_PATH) && make qt5py2
         if os.name == 'nt':
             !cd (LABELING_PATH) && pyrc5 -o libs/resources.py resources.qrc
         pyrc5 -o libs/resources.py resources.qrc

```

Figure 4: Image labelling

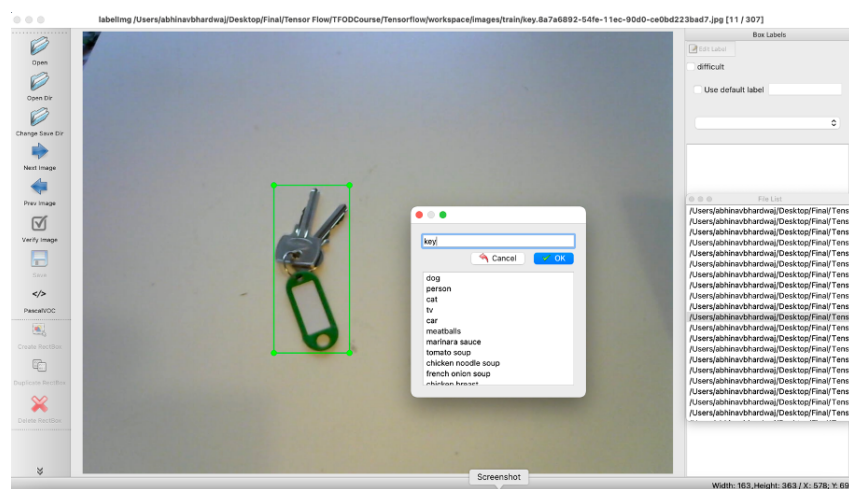


Figure 5: Labelling tool

- In this step, we are ready to capture the live images from our webcam using the open CV2 library. We can set up the timer as per our need for capturing the images. Kindly refer the figure 3
- We have to label the captured image using a graphical image annotation tool created by Darren Tzutalin from his public GitHub repository named LabelImg. On saving the each labbeled image it will also save the xml file with the same name as of file. The file will contains the x and y coordinates of the labelled image. Kindly refer the figure 4. Kindly refer the footnote for git hub repo <sup>3</sup>.
- Referring the figure 5, is an example of how one can access the local directory and start labelling the images.
- Finally we have images in our said folders of Knife, Key, Pen, Vape. We have to move them into the testing and training folders Manually along with their XML

<sup>2</sup>Tensorflow API installation : <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html>

<sup>3</sup>Image labelling: <https://github.com/tzutalin/labelImg>



Figure 6: Moving to Test and train

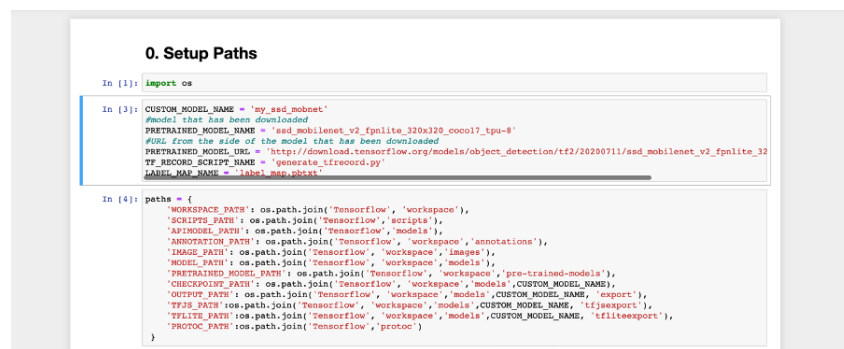


Figure 7: Setting up paths for Pre-Trained models

files. For this project there was a split of 75 training and 25 testing data split.

Using the step 7 is only for those who are running the code on Google Colab. This will create an archive folder for your test and train dataset. Which needs to be uploaded on Colab. Kindly refer the figure 6

## 4.2 Training and Detection

- Starting with importing the required library and the pre-trained model from the TensorFlow zoo GitHub repository. Paths have been defined for all the pre-trained models. Kindly refer the figure 7 and figure 8
- Downloading the pre-trained models from and setting up on the paths and location for extraction. Kindly refer the figure 9
- Running the verification script for TensorFlow, will give us go ahead of rest of the code. We need OK from post we execute the script. Kindly refer the figure 10
- Finally one can extract the pre-trained models it will look like the one in the referred figure 11
- This step required to create the label maps for all the object. please note that this is case-sensitive and make sure one should use the same name as it has been



```

'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace', 'pre-trained-models'),
'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME),
'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME, 'export'),
'TFJS_PATH': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME, 'tfjsexport'),
'TFLITE_PATH': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME, 'tfliteexport'),
'PROTOC_PATH': os.path.join('Tensorflow', 'protoc')
}

In [5]: files = {
        'PIPELINE_CONFIG': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME, 'pipeline.config'),
        'TF_RECORD_SCRIPT': os.path.join(paths['ANNOTATION_PATH'], 'TF_RECORD_SCRIPT_NAME'),
        'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], 'LABELMAP_NAME')
    }

In [6]: for path in paths.values():
        if not os.path.exists(path):
            if os.name == 'posix':
                mkdir -p {path}
            if os.name == 'nt':
                mkdir {path}

In [7]: paths['CHECKPOINT_PATH']
Out[7]: 'Tensorflow/workspace/models/my_ssd_mobnet'

In [8]: files
Out[8]: {'PIPELINE_CONFIG': 'Tensorflow/workspace/models/my_ssd_mobnet/pipeline.config',
        'TF_RECORD_SCRIPT': 'Tensorflow/scripts/generate_tfrecord.py',
        'LABELMAP': 'Tensorflow/workspace/annotations/label_map.pbtxt'}

```

Figure 8: Part 2 of setting up paths for Pre-Trained models

### 1. Download TF Models Pretrained Models from Tensorflow Model Zoo and Install TFOD

```

In [9]: # https://www.tensorflow.org/install/source_windows

In [10]: if os.name == 'nt':
        !pip install wget
        !import wget

In [11]: if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
        !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}

In [12]: # Install Tensorflow Object Detection
        if os.name == 'posix':
            !apt-get install protobuf-compiler
            !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && cp object_detection/pack
        if os.name == 'nt':
            url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
            wget.download(url)
            !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
            !cd {paths['PROTOC_PATH']} && tar -xzf protoc-3.15.6-win64.zip
            os.environ['PATH'] = os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
            !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_detection\g
            !cd Tensorflow/models/research\slim && pip install -e .

```

Figure 9: setting up paths for Pre-Trained models

```

In [10]: VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_t
        # Verify Installation
        !python {VERIFICATION_SCRIPT}

Running tests under Python 3.8.12: C:\Users\cheta\anaconda3\envs\labhinav_2\python.exe
[ RUN      ] ModelBuilderTF2Test.test_create_center_net_deepmac
2021-12-09 22:44:04.455131: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized
with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operat
ions: AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-12-09 22:44:04.952792: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/re
plica:0/task:0/device:GPU:0 with 5486 MB memory:  -> device: 0, name: NVIDIA GeForce RTX 3070 Laptop GPU, pci bus id:
0000:01:00:0, compute capability: 8.6
C:\Users\cheta\anaconda3\envs\labhinav_2\lib\site-packages\object_detection-0.1-py3.8.egg\object_detection\builders\mo
del_builder.py:1100: DeprecationWarning: The 'warn' function is deprecated, use 'warning' instead
logging.warn('Building experimental DeepMAC meta-arch.')
W1209 22:44:05.615393 4512 model_builder.py:1100] Building experimental DeepMAC meta-arch. Some features may be omit
ted.
INFO:tensorflow:time(_main_.ModelBuilderTF2Test.test_create_center_net_deepmac): 1.4s
I1209 22:44:05.848446 4512 test_util.py:2308] time(_main_.ModelBuilderTF2Test.test_create_center_net_deepmac): 1.4
s
[ OK      ] ModelBuilderTF2Test.test_create_center_net_deepmac
[ RUN      ] ModelBuilderTF2Test.test_create_center_net_model0 (customize_head_params=True)
INFO:tensorflow:time(_main_.ModelBuilderTF2Test.test_create_center_net_model0 (customize_head_params=True)): 1.12s
I1209 22:44:06.970698 4512 test_util.py:2308] time(_main_.ModelBuilderTF2Test.test_create_center_net_model0 (custo
mize_head_params=True)): 1.12s
[ OK      ] ModelBuilderTF2Test.test_create_center_net_model0 (customize_head_params=True)
[ RUN      ] ModelBuilderTF2Test.test_create_center_net_model1 (customize_head_params=False)
INFO:tensorflow:time(_main_.ModelBuilderTF2Test.test_create_center_net_model1 (customize_head_params=False)): 0.24s
I1209 22:44:07.211781 4512 test_util.py:2308] time(_main_.ModelBuilderTF2Test.test_create_center_net_model1 (custo
mize_head_params=False)): 0.24s
[ OK      ] ModelBuilderTF2Test.test_create_center_net_model1 (customize_head_params=False)
[ RUN      ] ModelBuilderTF2Test.test_create_center_net_model_from_Screenshot
INFO:tensorflow:time(_main_.ModelBuilderTF2Test.test_create_center_net_model_from_keypoints): 0.33s

```

Figure 10: Verification Script

```

In [16]: if os.name == 'posix':
        wget {PRETRAINED_MODEL_URL}
        mv {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
        cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}
    if os.name == 'nt':
        wget.download(PRETRAINED_MODEL_URL)
        move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
        cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}

14 [..] 679936 / 44635028
34 [..] 1564672 / 44635028
54 [...] 2441216 / 44635028
74 [.....] 3334144 / 44635028
94 [.....] 4145152 / 44635028
114 [.....] 5029888 / 44635028
134 [.....] 5906432 / 44635028
154 [.....] 6791168 / 44635028
174 [.....] 7675904 / 44635028
194 [.....] 8560640 / 44635028
214 [.....] 9437184 / 44635028
234 [.....] 10321920 / 44635028
254 [.....] 11206656 / 44635028
274 [.....] 12091392 / 44635028

```

Figure 11: setting up paths for Pre-Trained models

```

2. Create Label Map

In [17]: labels = [{'name': 'key', 'id': 1}, {'name': 'knife', 'id': 2}, {'name': 'pen', 'id': 3}, {'name': 'vape', 'id': 4}]

        with open(files['LABELMAP'], 'w') as f:
            for label in labels:
                f.write('name: {}\n'.format(label['name']))
                f.write('id: {}\n'.format(label['id']))
                f.write('\n')

3. Create TF records

In [23]: # OPTIONAL IF RUNNING ON CLOUD
        ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
        if os.path.exists(ARCHIVE_FILES):
            !tar -zxvf {ARCHIVE_FILES}

In [24]: if not os.path.exists(files['TF_RECORD_SCRIPT']):
        !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}

        Cloning into 'Tensorflow/scripts'...
        remote: Enumerating objects: 3, done.
        remote: Counting objects: 100% (3/3), done.
        remote: Compressing objects: 100% (2/2), done.
        remote: Total 3 (delta 0), reused 1 (delta 0), pack-reused 0
        Receiving objects: 100% (3/3), done.

In [25]: !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.
        !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.

```

Figure 12: Creating label maps and TF records

used while creating the folders and labelling them. WE are also creating the TF records and setting up tf training and testing scripts for the model. Kindly refer the figure 12

- once we have the tf scripts for testing and training data. we need to now configure the pre-trained models to the refereed paths. Kindly refer the figure 13
- Training of model will require the command prompt to see the process of model training. Once we have the command printed paste it on the Tensflow terminal. Kindly refer the figure 14 and figure 15
- Under the evualtion part we have to follow the procecess of copying the command to the tensorflow terminal which will generate the following optput. Kindly refer the figure 16 and figure 17
- To visualize TensorBoard from Train folder kindly follow the path: Tensorflow workspace models my\_ssd\_mobnet train; tensorboard --logdir=.
- To visualize TensorBoard from Eval folder kindly follow the path: Tensorflow workspace models my\_ssd\_mobnet eval tensorboard --logdir=.

```

In [15]: config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])

In [16]: config

Out[16]: {'model': ssd {
  num_classes: 4
  image_resizer {
    fixed_shape_resizer {
      height: 320
      width: 320
    }
  }
  feature_extractor {
    type: 'ssd_mobilenet_v2_fpn_keras'
    depth_multiplier: 1.0
    min_depth: 16
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 3.9999998989515007e-05
        }
      }
      initializer {
        ...
      }
    }
  }
}

In [10]: pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], 'r') as f:
  proto_str = f.read()
  text_format.Merge(proto_str, pipeline_config)

In [13]: pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(p
Screenshot RED_MODEL_PATH', PRETRAINED_MODEL_NAME,
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"

```

Figure 13: Configuring Pre-trained models

```

In [10]: pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], 'r') as f:
  proto_str = f.read()
  text_format.Merge(proto_str, pipeline_config)

In [13]: pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME,
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path = files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'train')
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'te

In [14]: config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], 'wb') as f:
  f.write(config_text)

6. Train the model

In [17]: TRAINING_SCRIPT = os.path.join(paths['AFIMODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')

In [18]: command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=2000".format(TRAINING_SCRIPT, paths['CH

In [19]: print(command)

python Tensorflow/models/research/object_detection/model_main_tf2.py --model_dir=Tensorflow/workspace/models/smy_ssd_m
obnet --pipeline_config_path=Tensorflow/workspace/models/smy_ssd_mobnet/pipeline.config --num_train_steps=2000

In [ ]: !{command}

```

Figure 14: Model Training

```

Anaconda Prompt (anaconda3)
INFO:tensorflow:Step 14700 per-step time 0.297s
11215 10:46:55.389793 32114 model_lib_v2.py:698] Step 14700 per-step time 0.297s
INFO:tensorflow:('loss/classification_loss': 0.07078673,
'loss/localization_loss': 0.015473858,
'loss/regularization_loss': 0.101359315,
'loss/total_loss': 0.1876191,
'learning_rate': 0.00533646)
11215 10:46:59.310799 32114 model_lib_v2.py:701] ('loss/classification_loss': 0.07078673,
'loss/localization_loss': 0.015473858,
'loss/regularization_loss': 0.101359315,
'loss/total_loss': 0.1876191,
'learning_rate': 0.00533646)
INFO:tensorflow:Step 14800 per-step time 0.296s
11215 10:47:24.948480 32114 model_lib_v2.py:698] Step 14800 per-step time 0.296s
INFO:tensorflow:('loss/classification_loss': 0.071862996,
'loss/localization_loss': 0.029359762,
'loss/regularization_loss': 0.100948055,
'loss/total_loss': 0.20216334,
'learning_rate': 0.00533646)
11215 10:47:24.948480 32114 model_lib_v2.py:701] ('loss/classification_loss': 0.071862996,
'loss/localization_loss': 0.029359762,
'loss/regularization_loss': 0.100948055,
'loss/total_loss': 0.20216334,
'learning_rate': 0.00533646)
INFO:tensorflow:Step 14900 per-step time 0.298s
11215 10:47:54.783230 32114 model_lib_v2.py:698] Step 14900 per-step time 0.298s
INFO:tensorflow:('loss/classification_loss': 0.09342567,
'loss/localization_loss': 0.01109097,
'loss/regularization_loss': 0.10068545,
'loss/total_loss': 0.20537021,
'learning_rate': 0.00533646)
11215 10:47:54.783230 32114 model_lib_v2.py:701] ('loss/classification_loss': 0.09342567,
'loss/localization_loss': 0.01109097,
'loss/regularization_loss': 0.10068545,
'loss/total_loss': 0.20537021,
'learning_rate': 0.00533646)
INFO:tensorflow:Step 15000 per-step time 0.299s
11215 10:48:24.680906 32114 model_lib_v2.py:698] Step 15000 per-step time 0.299s
INFO:tensorflow:('loss/classification_loss': 0.10080235,
'loss/localization_loss': 0.0343809,
'loss/regularization_loss': 0.10099121,
'loss/total_loss': 0.23609936,
'learning_rate': 0.004939596)
11215 10:48:24.680906 32114 model_lib_v2.py:701] ('loss/classification_loss': 0.10080235,
'loss/localization_loss': 0.0343809,
'loss/regularization_loss': 0.10099121,
'loss/total_loss': 0.23609936,
'learning_rate': 0.004939596)

(abhinav_2) C:\Users\cheta\Desktop\abhinav_project\TFODCourse>

```

Figure 15: Training Steps

```
In [20]: command = "python {}_model_dir/ --pipeline_config_path={} --checkpoint_dir={}".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'], paths['MODEL_PATH'])

In [21]: print(command)

python Tensorflow/models/research/object_detection/model_main_tf.py --model_dir=Tensorflow/workspace/models/my_ssd_mobnet --pipeline_config_path=Tensorflow/workspace/models/my_ssd_mobnet/pipeline.config --checkpoint_dir=Tensorflow/workspace/models/my_ssd_mobnet
```

Figure 16: Evaluation

[illegible]

Figure 17: Evaluation on Terminal

- For Google Colab to run Tensorboard from train folder kindly follow the path:  

```
reload_ext tensorboard
load_ext tensorboard
!cd
/Tensorflow/workspace/models/ssd_640_640/train/
tensorboard --logdir .
```
- For Google Colab to run Tensorboard from eval folder: `reload_ext tensorboard`  
`load_ext tensorboard`  
  

```
!cd /Tensorflow/workspace/models/ssd_640_640/eval/
tensorboard --logdir .
```
- Kindly refer the following figure 18 and figure 19 to see the TensorBoard outputs.
- We can even detect the object by inputting an image. Kindly refer the figure 20
- In the final portion of the code we can run our our live feed cam to detect an image using our webcam. Kindly refer the figure 21, figure 22 and figure 23
- we can finally save the model and the checkpoint so that one dose not have to run the complete program again. Kindly refer the figure 24



Figure 18: Tensorboard Loss and Learning graph

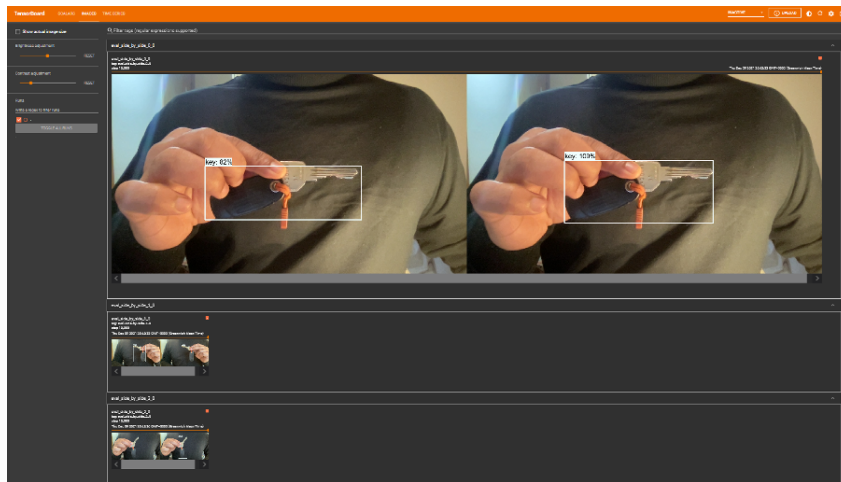


Figure 19: Testing using an image from test folder

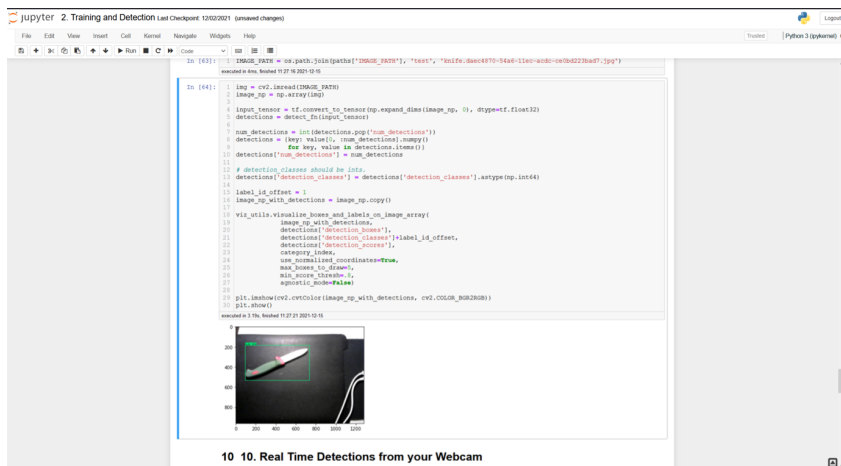


Figure 20: Detecting the object from an image

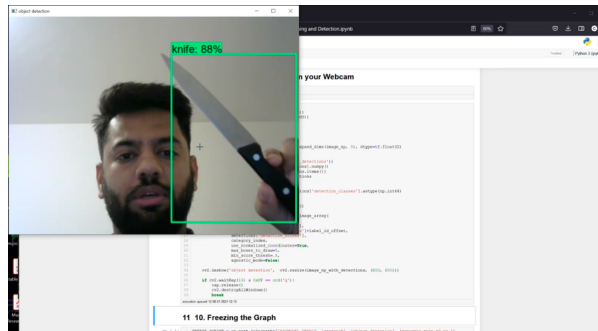


Figure 21: Detecting and object using live feed 1

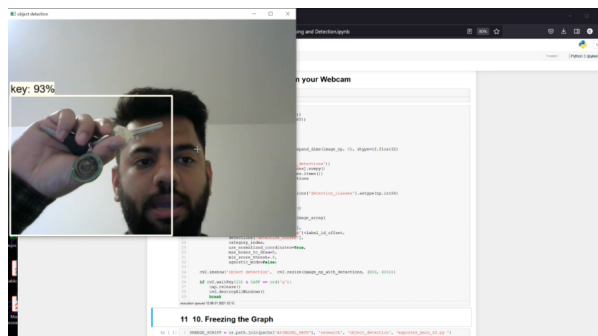


Figure 22: Detecting and object using live feed 2

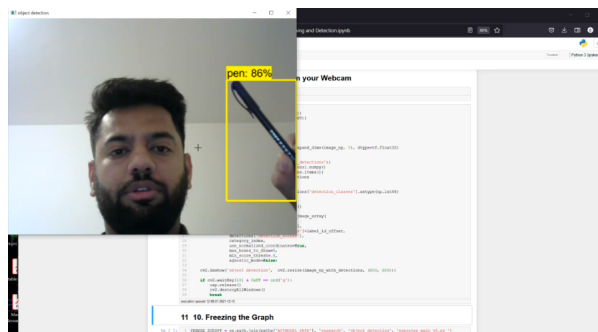


Figure 23: Detecting and object using live feed 3

## 8. Load Train Model From Checkpoint

```
In [7]: import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util

In [11]: # Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-5')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

Figure 24: Load the checkpoints and save model