Training tensorflow Model

First you need to download these files:

- Cuda
- CuDNN
- Tensorflow GPU
- Libraries Ex: (Jupyter notebook, matplotlib, pandas)
- Tensorflow model from github
- Protobuf

Step 1:

For installing Cuda, CuDNN & tensorflow-GPU you need to follow this link!
https://github.com/satishtilwani123/Deep-Learning/blob/master/Installation%20Cuda%2C%20Cudnn%2C%20tf-gpu/Cuda%2C%20Cndnn%20%26%20Tensorflow%20installation.pdf

Step 2:

Install Jupyter notebook!
`$ sudo pip install jupyter`

Install matplotlib!
`$ sudo pip install matplotlib`

Install pandas!
`$ sudo pip install pandas`

Step 3:

Download Tensorflow model from this link:
https://github.com/tensorflow/models

Step 4:

Download protobuf compiler from this link:
https://github.com/protocolbuffers/protobuf/releases

If you are using linux/ubuntu then you need to download **protoc-3.4.0-linux-x86_64.zip** or any updated version.

After Downloading we will extract using this command:
`$ tar -xvf protoc-3.4.0-linux-x86_64.tar.gz` **#Replace this with actual download file**

Open bash file:
`$ sudo nano ~/.bashrc`

Add these path's into bash file:
`export PATH="$PATH:/usr/local/protoc-3.4.0-linux-x86_64/bin/"`
#Replace above path with the actual path of your protoc file.

Start protobuf compilation:

- Open terminal
- `$ cd Tensorflow/models/research`
- `$ protoc object_detection/protos/*.proto --python_out=.`

#If it will show an error then it means protobuf is not installed correctly.

Step 5: Training custom object detector

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.

You need to follow these Steps:

- **How to organize your workspace/training files**
  First create a folder of named tensorflow having structure:

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

  Models is the directory which we downloaded in **step 3.**

  Now structure of training_demo will be like this:

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

  Annotations contain .pbtxt file & records file.
  Exported-models contain models exported after training.
  Images contain test and train images and *.xml files.
  Models contain all checkpoints & config files.
  Pre-trained-models contain SSD mobilenet fpn lite.

  You can also add more directories as per your requirements.

- **How to prepare/annotate image dataset**
  Download LabelImg:
  - Inside your TensorFlow folder, create a new directory, name it addons and then **cd** into it.
  - To download the package you can either use this link:

https://github.com/tzutalin/labelImg

- After downloading clone it in **Tensorflow/addons**

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

Install dependencies & compiling packages:

cd into **TensorFlow/addons/labelImg** and run the following commands:

`conda install pyqt=5`

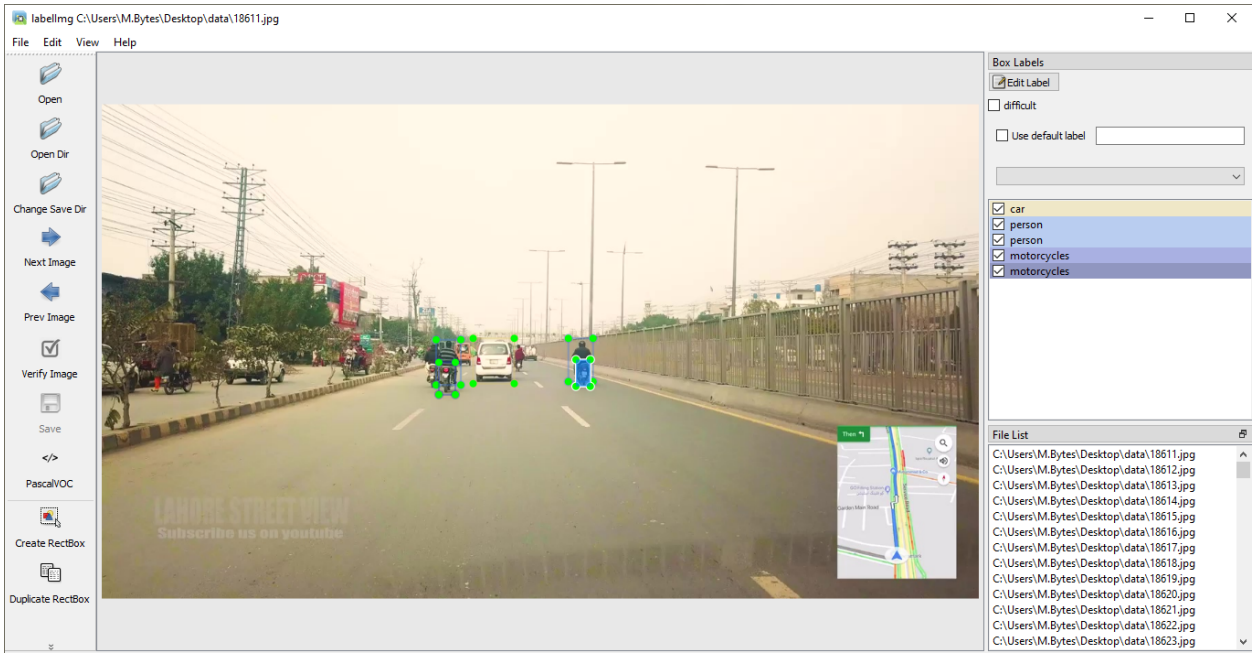
`pyrcc5 -o libs/resources.py resources.qrc`

Test your Installation:

Run this following command:

`python labelImg.py`

Start annotating/labeling images:



This youtube video helps you to understand the labeling mechanism:
https://www.youtube.com/watch?t=9m13s&v=K_mFnvzyLvc&feature=youtu.be

Partition the dataset into train & test:

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 choose whatever ratio suits your needs. Insert (test, train) folders into images

Create Label map:

TensorFlow requires a label map, which mainly maps each of the used labels to 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 contains 2 labels, dogs and cats:

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

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

Insert *.pbtxt file into annotations folder.

- ### How to generate tf records of such datasets

  Download tf-record script from this link:
  https://github.com/satishtilwani123/Deep-Learning/tree/master/TF-Record

  Put this code in **scripts/preprocessing** directory inside tensorflow

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

  Run this command inside **scripts/preprocessing** directory:
  Generate train.record

`python3 generate_tfrecord.py *path/train -l *path/label_map.pbtxt -o *path/train.record`

Generate test.record
`python3 generate_tfrecord.py *path/test -l *path/label_map.pbtxt -o *path/test.record`

- **How to configure simple training pipeline**
  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.
  there exist a number of models you can use, all of which are listed below: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

  After downloading the model, decompress it 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
└─ ...
```

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.

 # specify number of classes

1. num_classes: 4

 # specify batch size, increase batch size will affect on memory

2. Batch_size: 2

 # Path to checkpoint of pre-trained model

3. Fine_tune_checkpoint: "pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt-0"

 # if you to classify "classification" else for localization "detection"

4. Fine_tune_checkpoint_type: "detection"

 # path to the label map file in train_input_reader

5. Label_map_path: "annotations/label_map.pbtxt"

 # path to the train.record file in train_input_reader

6. Input_path: "annotations/train.record"

 # path to the label map file in eval_input_reader

7. Label_map_path: "annotations/label_map.pbtxt"

 # path to the test.record file in eval_input_reader

8. Input_path: "annotations/test.record"

 # specify total steps inside cosine_decay_learning_rate

9. total_steps: 25000

 # specify number of steps below fine_tune_checkpoint

10. Num_steps: 25000

 # Optional

 # if model can't detect properly so minimize learning rate

 # warmup learning rate must be minimum than base learning rate

11. Learning_rate_base: 0.003999999910593033

12. Warmup_learning_rate: 0.0013333000242710114

- **How to train a model & monitor its progress**
  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:
  `python3 model_main_tf2.py --model_dir=models/my_ssd_resnet50_v1_fpn --pipeline_config_path=models/my_ssd_resnet50_v1_fpn/pipeline.config`

  Once training start it's look like this:

```
...
WARNING:tensorflow:Unresolved object in checkpoint: (root).model._box_predictor._base_tower_layers_for_
W0716 05:24:19.105542  1364 util.py:143] Unresolved object in checkpoint: (root).model._box_predictor._
WARNING:tensorflow:Unresolved object in checkpoint: (root).model._box_predictor._base_tower_layers_for_
W0716 05:24:19.106541  1364 util.py:143] Unresolved object in checkpoint: (root).model._box_predictor._
WARNING:tensorflow:Unresolved object in checkpoint: (root).model._box_predictor._base_tower_layers_for_
W0716 05:24:19.107540  1364 util.py:143] Unresolved object in checkpoint: (root).model._box_predictor._
WARNING:tensorflow:Unresolved object in checkpoint: (root).model._box_predictor._base_tower_layers_for_
W0716 05:24:19.108539  1364 util.py:143] Unresolved object in checkpoint: (root).model._box_predictor._
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore or tf.keras.Model.load_w
W0716 05:24:19.108539  1364 util.py:151] A checkpoint was restored (e.g. tf.train.Checkpoint.restore or
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
...
```

  If you want to stop training then write : **CTRL-C** in terminal.
  If you want to again start training from the remaining number of steps then:
  1. Move to **training_demo/models/my_ssd_resnet50_v1_fpn/**
  2. Open **pipeline.config** file
  3. Change path of **fine_tune_checkpoint:**
  4. Specify path of newly created checkpoints in **models/my_ssd_resnet50_v1_fpn/**
  5. Specify as:
     Fine_tune_checkpoint: models/my_ssd_resnet50_v1_fpn/ckpt-6
     # Replace ckpt-6 with newly created checkpoint

  ### Monitor the progress using tensorboard:

  To start a new TensorBoard server, we follow the following step:
  - **cd** into the **training_demo** folder
  - `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):

Hope you will enjoy this article and successfully train the model
Created by: Safe Drive