SSD OBJECT DETECTION

CONTENTS: -

Overview

In this documentation, we are going to discuss brief knowledge about Object Detection and learn the whole procedure to run any objection detection model into a Linux based system.

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class in digital images and videos. Well-researched domains of object detection include face detection and pedestrian detection.

As declared this documentation is for the Linux based System, we will discuss each and every point clearly about the installation needed to train and evaluate the model.

Our model is based on the technique called SSD for Object Detection. So, we are going to know the details related to this topic in next topic.

What is SSD?

SSD or Single Shot Detector, SSD is designed for object detection in real-time. As different CNNs has many bottleneck like:-

1. Training the data is unwieldy too long.

2. Training happens in multiple phases (e.g. training reason proposal vs classifier).

3. Network is too slow at inference time (i.e when dealing with non-training data).

Therefore, to address these bottlenecks many architectures were created in these few years , enabling real-time object detection. The most famous ones are YOLO and SSD. We are going to explore SSD in this after understanding this it will be easy to understand YOLO too.

To better understand SSD, let's start explaining each term of its name:

Single Shot- This means that the tasks of object localization and classification are done in a *single forward pass* of the network.

Multibox- This is the approach for bounding box regression developed by szegedy.

Detector- The network is an object detector that also classifies those detected objects.

Architecture- The VGG-16 architecture is used in SSD because of its best performance in high quality image classification tasks and its popularity for problems where transfer learning helps in improving results. Instead of the original VGG fully connected layers, a set of *auxiliary* convolutional layers were added, thus enabling to extract features at multiple scales and progressively decrease the size of the input to each subsequent layer.

SSD does not use a delegated region proposal network. Instead, it resolves to a very simple method. It computes both the location and class scores using small convolution filters. After extracting the feature maps, SSD applies 3 × 3 convolution filters for each cell to make predictions. (These filters compute the results just like the regular CNN filters.) Each filter outputs 25 channels: 21 scores for each class plus one boundary box.

Just like Deep Learning, we can start with random predictions and use gradient descent to optimize the model. However, during the initial training, the model may fight with each other to determine what shapes (pedestrians or cars) to be optimized for which predictions.

For more resources

- https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab
- https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11
- https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06
- https://arxiv.org/abs/1611.10012

Installation

Installation of TensorFlow for gpu , cpu and NVIDIA Drivers and Toolkit is defined in our last version of documentation which is related to image classification on the CIFAR10 model. Link of that documentation is below: -

Package Installation

After creating new virtual environment, we had to install some packages before installing different models. So given below are the few packages to install only we have to do is change the package name after **python3 -m pip install <package name> <=version>.**

- python3 -m pip install pillow
- python3 -m pip install lxml
- python3 -m pip install contextlib2
- python3 -m pip install opencv – python
- python3 -m pip install coco
- python3 -m pip install protobuf - compiler

NOTE: - According to different versions of ubuntu, there should be some more packages to install which will be asked to install during these packages install so

same process to install them also as written above. Ex: Like in coco installation it is asked to install crontab.

Test of Installation

As there is important to crosscheck what we did till now is correct or not. As above installation codes are correct otherwise, we can't proceed ahead but to check we provided few codes in this we are just trying to run some basic code of TensorFlow to check it is working or not.

- python3
- import tensorflow as tf

Downloading of TensorFlow Models

_Now it's time to clone our TensorFlow models for the all the tasks to be done in this object detection models.

- git clone https://github.com/tensorflow/models.git

Protobuf Installation

The TensorFlow Object Detection API uses protobufs to configure models and training parameters. Therefore, we have to install protobuf for it so first make a path to the research directory via models.

- Move to the research directory (cd/models/research) and compile the Protobuf libraries as follows: -

  First make a directory of **protoc_3.0**

- mkdir protoc_3.0

  Then move inside the directory.

- cd protoc_3.0

  Inside the directory we have to install the latest protobuf according to our use.

- wget https://github.com/google/protobuf/releases/download/v3.0.0/protoc-3.0.0-linux-x86_64.zip
- unzip protobuf.zip

  After unzipping the zipped file we have to run the following code given below

- Add libraries to PYTHONPATH—export PYTHONPATH= $PYHTONPATH:'pwd':'pwd'/slim

  As TensorFlow models are core package for object detection, it's convenient to add specific folder to our environment variables.

- ./protoc_3.0/bin/protoc object_detection/protos/*.proto –python out=.

Setup completion

- python setup.py build
- python setup.py install

Testing the Installation

- python object_detection/builders/model_builder_test.py

COCO API Installation(optional)

Coco is a large image dataset designed for object detection, segmentation, person keypoints detection and caption generation. This package provides Matlab, Python and Lua APIs that assists in loading, parsing, and visualizing the annotations in **COCO.**

For COCO API installation I am providing you a video link(https://www.youtube.com/watch?v=COlbP62-B-U HYPERLINK HYPERLINK) by which we can easily understand the installation process.

In our model, we have created our own dataset for training and testing purpose. So, after this whole process of data conversion from .xml to .csv and .csv to .record is written.

Creating Tfrecord files

- For the purpose of training and testing, Divide images into two new different folders Train and Test. Divide the total number of images in your required ratio. Most of the times the ratio should be 80:20 and 90:10 if the number of images is 100.

Converting .xml to .csv

- make a file 'xml_to csv.py' by using 'gedit' in terminal and paste code from (https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#converting-xml-to-csv)

To Create train data:

- python3 xml_to_csv.py -i [PATH_TO_IMAGES_FOLDER]/train -o [PATH_TO_ANNOTATIONS_FOLDER]/train_labels.csv

To Create test data:

- python3 xml_to_csv.py -i [PATH_TO_IMAGES_FOLDER]/test -o [PATH_TO_ANNOTATIONS_FOLDER]/test_labels.csv

Converting from .csv to .record

- make a file 'generate_tfrecord.py' by using 'gedit' in terminal and paste code from (

https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#converting-from-csv-to-record)

- set image path in main function for train images and test images (path = os.path.join(os.getcwd(), FLAGS.img_path)) change ‘ FLAGS.img_path’ to image directories.

- for error (TypeError: None has type NoneType, but expected one of: int, long) goto (**def** class_text_to_int(row_label):) and return 0 for ‘else’ block.

- python3 generate_tfrecord.py --label= <LABEL> --csv_input=train_labels.csv --output_path=train.record

- python3 generate_tfrecord.py --label= <LABEL> --csv_input=test_labels.csv --output_path=test.record

Configuring a Training Pipeline

A **machine learning pipeline** is used to help automate **machine learning** workflows. They operate by enabling a sequence of data to be transformed and correlated together in a model that can be tested and evaluated to achieve an outcome, whether positive or negative.

In this model, we are going to reuse one of the pre-trained models to train our model. If you would try to run a new model then have a look at https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/configuring_jobs.md

In this we are going to use ssd_inception_v2_coco to train our model due to its speed and mAP. If you want to use a different models then have a look at this list

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#coco-trained-models-coco-models

First of all, we need to get ourselves a same pipeline configuration file we want to use so download it from the link given just above.

The file downloaded will be on tar.gz format then uncompress it to have access to files inside this and go through the code of pre-trained model then do the changes in the code you want to do like changing the value of num-classes, decay factor etc.

Model Training

- goto **cd models/research**
- **export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim**
- **./protoc_3.0/bin/protoc object_detection/protos/*.proto**

  –-python_out=.
- goto **cd object_detection**
- Download the SSD network (we use ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03.tar.gz)
- **wget http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03.tar.gz**
- Extract a file run a command **tar xvzf ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03.tar.gz**
- cd training
- make a same name network config file in **gedit ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03.config**
- *copy the network code from* ***https://github.com/tensorflow/models/blob/master/research/obje***

ct_detection/samples/configs/ssd_mobilenet_v1_300x300_coco14_sync.config

and paste to your own file

ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03.config

- *move the* ***ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03.config*** *to training directory*
- ***ope****n* ***ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03.config*** *file*
- *if you only want to train for '****Tip****' ten (num_classes=1) otherwise equal to number of row_label i.e. 3 in this case*
- *inside config file find* ***fine_tune_checkpoint: "ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03/model.ckpt"*** *below this line there is* ***num_steps: 20000*** *you can set as your need. (in my case 20,000 steps)*
- ***train_input_reader: {***

tf_record_input_reader {

input_path: "data/train.record" ***#*** *train.reecord location*

}

label_map_path: "data/objt.pbtxt" ***#*** *pbtxt file location*

}

- *same for* ***eval_input_reader*** *, input_path is* ***test.record*** *and pbtxt file location is same*
- *for training upto 20k steps run the code* ***python3 train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/ssd_mobilenet_v2_quantized_300x300_coco.config***
- *for evaluation* ***python3 eval.py --logtostderr --pipeline_config_path=training/ssd_mobilenet_v2_quantized_300x300_coco.config --checkpoint_dir=training/ --eval_dir=training/***

Making a pbtxt file

- ***gedit objt.pbtxt***
- *paste the below code in a file*
- ***item {***

id: 1

name: 'tip'}

- ***item {***

id: 2
name: 'top'}

- ***item {***

id: 3
name: 'all'}

Exporting a Trained Inference Graph

- *Goto* ***cd models/research/object_detection/training***
- ***simply sort all the files inside*** *training by descending time and pick the*model.ckpt-**file that comes first in the list.*
- ***cd ..***
- *run the command* ***python3 export_inference_graph.py --input_type image_tensor --pipeline_config_path training/ssd_mobilenet_v2_quantized_300x300_coco.config --trained_checkpoint_prefix training/model.ckpt-20000 --output_directory trained-inference-graphs/output_inference_graph_v1.pb***

Changing IOU value from 0.5 to 0.75 & 0.90

- *goto* ***cd models/research/object_detection/utills***
- *open* ***object_detection_evaluation.py***
- *change all text of* ***matching_iou_threshold=0.5 to matching_iou_threshold=0.75 or 0.90***
- *save the file and run* ***cd ..***
- *In object_detection directory run the command* ***python3 eval.py --logtostderr --pipeline_config_path=training/ssd_mobilenet_v2_quantized_300x300_coco.config --checkpoint_dir=training/ --eval_dir=training/***

Monitoring the job using Tensorboard

First activate the TensorFlow gpu using the code: -

```
activate tensorflow_gpu
```

Then go to the training folder and run this code: -

```
tensorboard --logdir=training\
```