NVIDIA TRANSFER LEARNING TOOLKIT FOR INTELLIGENT VIDEO ANALYTICS

DU-09243-003 _v3.0 | March 2019

Getting Started Guide

TABLE OF CONTENTS

Chapter 1. OVERVIEW

NVIDIA Transfer Learning Toolkit is a Python package to enable NVIDIA customers the ability to fine-tune pre-trained models with customer's own data and export them for TensorRT based inference through an edge device.

Use the Transfer Learning Toolkit to perform these tasks:

- Download the model - Download pre-trained models.
- Evaluate the model - Evaluate models for target prediction.
- Train the model - Train or re-train data to create and refine models.
- Prune the model - Prune models to reduce size.
- Export the model - Export models to a compatible format.

Chapter 2. TRANSFER LEARNING TOOLKIT REQUIREMENTS

Using the Transfer Learning Toolkit requires the following:

Hardware Requirements

Minimum

- 4 GB system RAM
- 4 GB of GPU RAM
- Single core CPU
- 1 GPU - tested with Titan X (Pascal)
- 50 GB of HDD space

Recommended

- 32 GB system RAM
- 32 GB of GPU RAM
- 8 core CPU
- 4 GPUs - tested with Titan X (Pascal)
- 100 GB of SSD space

Software Requirements

- Ubuntu 16.04 LTS
- Cuda 9.0 - Included in the Transfer Learning Toolkit (TLT) docker.

> DeepStream 3.0 - NVIDIA SDK for IVA inference https://developer.nvidia.com/deepstream-sdk is recommended.

Model Requirements

Classification

- Input size: 3 * H * W (W, H >= 16)
- If using pretrained weights, the input size should be 3 * 224 * 224
- Input format: JPG, JPEG, PNG

Detection

- Input size: 3 * H * W (where H >= 272, W >= 480, and W,H are mutliples of 16)
- If using pretrained weights, the input size should be 3 * 384 * 1224
- Image format: JPG, JPEG, PNG
- Label format: KITTI detection

> The **`tlt-train`** tool does not support training on images of multiple resolutions, or resizing images during training. All of the images must be resized offline to the final training size and the corresponding bounding boxes must be scaled accordingly.

Installation Prerequisites

- Install the docker. See: https://www.docker.com/.
- NVIDIA GPU driver v384.xx. Download from https://www.nvidia.com/Download/index.aspx?lang=en-us.
- Install the Nvidia Docker from: https://github.com/NVIDIA/nvidia-docker.

Get an NGC API Key

- NVIDIA GPU Cloud account and API key - https://ngc.nvidia.com/

  1. Go to NGC and click the **Transfer Learning Toolkit** container in the **Catalog** tab. This message is displayed, **Sign in to access the PULL feature of this repository**.
  2. Enter your email address and click **Next** or click **Create an Account**.
  3. Choose your **organization** when prompted for Organization/Team.

     Your organization should be the first choice listed in the Set Your Organization dialog.
  4. Click **Sign In**.
  5. Select the **Containers** tab on the left navigation pane and click the **Transfer Learning Toolkit** tile.

> Save the API key in a secure location. You will need it to use the AI assisted annotation SDK.

Download the docker container

- Execute **`docker login nvcr.io`** from the command line and enter your username and password.
  - Username: $oauthtoken
  - Password: API_KEY
- Execute **`docker pull nvcr.io/nvidia/tlt-streamanalytics:v0.3_py2`**

Chapter 3. INSTALLATION

The Transfer Learning Toolkit (TLT) is available to download from the NGC. You must have an NGC account and an API key associated with your account.

3.1. Running the Transfer Learning Toolkit

Use this procedure to run the Transfer Learning Toolkit.

- **Run the toolkit**: Run the toolkit using this command. The docker starts in the **`/workplace`** folder by default.

```
docker run --runtime=nvidia -it nvcr.io/nvidia/tlt-
streamanalytics:v0.3_py2 /bin/bash
```

- **Access local directories**: To access local directories from inside the docker you need to mount them in the docker. Use this option, -v <source_dir>:<mount_dir>, to mount local directories in the docker. For example the command to run the toolkit mounting the **`/home/<username>/tlt-experiments directory`** in your disk to the **`/workspace/tlt-experiments`** in docker would be:

```
docker run --runtime=nvidia -it -v /home/<username>/tlt-experiments:/
workspace/tlt-experiments nvcr.io/nvidia/tlt-streamanalytics:v0.3_py2 /bin/
bash
```

- **Use the examples**: To run the examples that are available, enable the jupyter notebook included in the docker to run in your browser:

```
docker run --runtime=nvidia -it -v /home/<username>/tlt-experiments:/
workspace/tlt-experiments -p 8888:8888 tlt-streamanalytics:v0.3_py2
```

  Go to the examples folder: **`cd examples/`**

  Execute this command from inside the docker to run the jupyter notebook:

```
jupyter notebook --ip 0.0.0.0 --allow-root
```

  Copy and paste the link produced from this command into your browser to access the notebook. The **`/workspace/examples`** folder will contain a demo notebook.

3.2. Downloading the models

Getting a list of models

Use this command to get a list of models that are available.

```
tlt-pull --list_models -k $API_KEY -o $ORG -t $TEAM
```

Here is sample output from using this command.

`tlt-pull -k $API_KEY -lm -o nvtltea -t iva`

```
+--------------------+------------+----------------+----------------+----------+-----------+--------------+
| Name               | Org/Team   | Latest Version | Application    | Framework | Precision | Last Modified |
+--------------------+------------+----------------+----------------+----------+-----------+--------------+
| tlt_iva_classifica | nvtltea/iva | 1             | Classification | TLT      | FP32      | 2019-03-01   |
| tion_alexnet       |            |                |                |          |           | 20:41:41 UTC |
| tlt_iva_classifica | nvtltea/iva | 1             | Classification | TLT      | FP32      | 2019-03-02   |
| tion_googlenet     |            |                |                |          |           | 00:08:48 UTC |
| tlt_iva_classifica | nvtltea/iva | 1             | Classification | TLT      | FP32      | 2019-03-01   |
| tion_resnet18      |            |                |                |          |           | 23:56:49 UTC |
| tlt_iva_classifica | nvtltea/iva | 1             | Classification | TLT      | FP32      | 2019-03-01   |
| tion_resnet50      |            |                |                |          |           | 23:52:58 UTC |
| tlt_iva_classifica | nvtltea/iva | 1             | Classification | TLT      | FP32      | 2019-03-02   |
| tion_vgg16         |            |                |                |          |           | 00:03:52 UTC |
| tlt_iva_classifica | nvtltea/iva | 1             | Classification | TLT      | FP32      | 2019-03-02   |
| tion_vgg19         |            |                |                |          |           | 00:06:36 UTC |
| tlt_iva_object_det | nvtltea/iva | 1             | Detection      | TLT      | FP32      | 2019-03-02   |
| ection_googlenet   |            |                |                |          |           | 00:48:15 UTC |
| tlt_iva_object_det | nvtltea/iva | 1             | Detection      | TLT      | FP32      | 2019-03-02   |
| ection_resnet10    |            |                |                |          |           | 00:45:16 UTC |
| tlt_iva_object_det | nvtltea/iva | 1             | Detection      | TLT      | FP32      | 2019-03-02   |
| ection_resnet18    |            |                |                |          |           | 00:37:46 UTC |
| tlt_iva_object_det | nvtltea/iva | 1             | Detection      | TLT      | FP32      | 2019-03-02   |
| ection_vgg16       |            |                |                |          |           | 00:41:00 UTC |
+--------------------+------------+----------------+----------------+----------+-----------+--------------+
```

> The `-o`, `--org` and `-t`, `--team` are mandatory arguments. Please use **`nvtltea`** and **`iva`** for each argument respectively.

Getting a list of model versions

Use this command, using resnet10 as an example, to get a list of model versions that are available.

`tlt-pull -k $API_KEY -o nvtltea -t iva -lv -m tlt_iva_classification_resnet18`

```
+--------------------+------------+----------------+----------------+----------+-----------+--------------+
| Name               | Org/Team   | Latest Version | Application    | Framework | Precision | Last Modified |
+--------------------+------------+----------------+----------------+----------+-----------+--------------+
| tlt_iva_classifica | nvtltea/iva | 1             | Classification | TLT      | FP32      | 2019-03-01   |
| tion_resnet18      |            |                |                |          |           | 23:56:49 UTC |
+--------------------+------------+----------------+----------------+----------+-----------+--------------+
```

Downloading a model

Use the **`tlt-pull`** command to download the model from the NGC model registry:

```
tlt-pull --model_name $MODEL_NAME --version $VERSION -k $API_KEY -o $ORG -t
 $TEAM --dir ./path/to/save/model
```

The **`--version`** is a mandatory argument. Please use **`--list_versions`** to find the all the versions that are available.

Example output from using this command.

`tlt-pull -k $API_KEY -o nvtltea -t iva -v 1 -m tlt_iva_classification_resnet18 -d ./models`

```
Downloaded 82.41 MB in 4s, Download speed: 20.55 MB/s
----------------------------------------------------
Transfer id: tlt_iva_classification_resnet18_v1 Download status: Completed.
Downloaded local path: /tmp/tmpxZVFfV/tlt_iva_classification_resnet18_v1
Total files downloaded: 2
Total downloaded size: 82.41 MB
Started at: 2019-03-12 18:51:08.367526
Completed at: 2019-03-12 18:51:12.379689
Duration taken: 4s seconds
----------------------------------------------------
Finished downloading tlt_iva_classification_resnet18
```

Chapter 4. PREPARING INPUT DATA STRUCTURE

The chapter provides instructions on preparing your data for use by the Transfer Learning Toolkit.

4.1. Data input for classification

Classification expects a directory of images with the following structure, where each class has its own directory with the class name. The naming convention for **train/val/test** can be different, because the path of each set is individually specified in the spec file. See Specification file for classification for more information.

```
|--dataset_root:
    |--train
        |--audi:
            |--1.jpg
            |--2.jpg
        |--bmw:
            |--01.jpg
            |--02.jpg
    |--val
        |--audi:
            |--3.jpg
            |--4.jpg
        |--bmw:
            |--03.jpg
            |--04.jpg
    |--test
        |--audi:
            |--5.jpg
            |--6.jpg
        |--bmw:
            |--05.jpg
            |--06.jpg
```

4.2. Data input for detection

The object detection tool expects label input as tfrecords for training. In order to convert the data into tfrecords, the Transfer Learning Toolkit provides a dataset converter tool to export a KITTI format object detection dataset to **`tlt`** ingestible tfrecords.

This is an example data structure to organize your data to be used as input for the dataio conversion tool. The KITTI file format requires this structure.

```
|--dataset root
  |-- images
      |-- 000000.jpg
      |-- 000001.jpg
            .
            .
      |-- xxxxxx.jpg
  |-- labels
      |-- 000000.txt
      |-- 000001.txt
            .
            .
      |-- xxxxxx.txt
  |-- kitti_seq_to_map.json
```

- The **`images`** directory contain the images to train the model.
- The **`labels`** directory contain the labels to the corresponding images.
- The **`kitti_seq_to_map.json`** file contains and sequence to frame id mapping for the frames in the **`images`** directory. This is an optional file, and is useful if the data needs to be split into an N folds sequence. If the data is split into a random 80:20 split, this file can be ignored.

All the images and labels in the training dataset should be of the same resolution.

4.2.1. Label files

A KITTI format label file is a text file containing one line per object. Each line has multiple fields. Here's an example:

Number Elements	Parameter Name	Description	Type	Range	Example
1	Class names	The class the object belongs	String	NA	Person, Car, Road_Sign, Don't care
1	Truncation	How much of the object has left image boundaries	Float	0.0, 1.0	0.0

Number Elements	Parameter Name	Description	Type	Range	Example
1	Occlusion	Occlusion state [0 = fully visible, 1 = partly occluded, 2 = largely occluded, 3 = unknown]	Integer	[0, 3]	2
1	Alpha	Observation Angle of object	Float	[-pi, pi]	0.146
4	Bounding box coordinates	Location of the object in the image	Float 0 (based index)	[0 to image_width], [0 to image_height], [top_left, image_width], [bottom_right, image_height]	100 120 180 160
3	3-D dimensions	Height, width, length of the object (in meters)	Float	NA	1.65 1.67 3.64
3	Location	3-D object location x, y, z in camera corrdinates (in meters)	Float	NA	-0.65 1.71 46.7
1	Rotation_y	Rotation ry around the Y-axis in camera coordinates	Float	[-pi, pi]	-1.59

This sums the total number of elements per object to 15. Here is a sample text file:

```
car 0.00 0 -1.58 587.01 173.33 614.12 200.12 1.65 1.67 3.64 -0.65 1.71 46.70
 -1.59
cyclist 0.00 0 -2.46 665.45 160.00 717.93 217.99 1.72 0.47 1.65 2.45 1.35 22.10
 -2.35
pedestrian 0.00 2 0.21 423.17 173.67 433.17 224.03 1.60 0.38 0.30 -5.87 1.63
 23.11 -0.03
```

In the image there are 3 objects with parameters indicated above. For this tool, consider the class names and bbox coordinates fields, because you're training for the class and object coordinates. You can use a sample file like this:

```
car 0.00 0 0.00 587.01 173.33 614.12 200.12 0.00 0.00 0.00 0.00 0.00 0.00 0.00
cyclist 0.00 0 0.00 665.45 160.00 717.93 217.99 0.00 0.00 0.00 0.00 0.00 0.00
 0.00
pedestrian 0.00 0 0.00 423.17 173.67 433.17 224.03 0.00 0.00 0.00 0.00 0.00 0.00
 0.00
```

4.2.2. Sequence mapping file

This is an optional json file that includes the mapping between the frames in the images directory and the names of video sequences from which the frames were extracted. This information is needed while doing an N-fold split of the dataset. Using this method takes frames from sequences that don't repeat in other folds.

The json dictionary file would be as follows:

```
{
  "video_sequence_name": [list of strings(frame idx)]
}
```

Consider a sample dataset with six sequences, the **kitti_seq_to_frames.json** file would look like this.

```
{
  "2011_09_28_drive_0165_sync": ["003193", "003185", "002857", "001864",
 "003838",
  "007320", "003476", "007308", "000337", "004165", "006573"],
  "2011_09_28_drive_0191_sync": ["005724", "002529", "004136", "005746"],
  "2011_09_28_drive_0179_sync": ["005107", "002485", "006089", "000695"],
  "2011_09_26_drive_0079_sync": ["005421", "000673", "002064", "000783",
 "003068"],
  "2011_09_28_drive_0035_sync": ["005540", "002424", "004949", "004996",
 "003969"],
  "2011_09_28_drive_0117_sync": ["007150", "003797", "002554", "001509"]
}
```

4.2.3. TFRecords conversion spec file

The dataio conversion tool uses a spec file to define the parameters required to convert KITTI format data to the tfrecords that the dashnet tool accepts as input. This is a prototxt format file with the following parameters.

```
{
  kitti_config {
    root_directory_path = Path to the dataset root
    image_dir_name = Relative path to the directory containing images
    label_dir_name = Relative path to the directory containing labels
    point_clouds_dir = Relative path to the directory containing point cloud
 data
    calibrations_dir = Relative path to the directory containing calibration
 data
    kitti_sequence_to_frames_file = KITTI sequence to frame json map.
    image_extension = Image extension (all images are required to be of the same
 format.)

    num_partitions = Number of partitions to split the data (N folds)
    num_shards = Number of shards per fold (this makes is faster to iterate)
    partition_mode = Type of partition supported [choices are 'random' and
 'sequence']
    val_split = Percentage split of the data for validation
```

```
  }
  image_directory_path: Path to the data root
}
```

4.2.4. Using the dataio conversion tool

KITTI is the accepted dataset format for image detection. The KITTI dataset must be converted to the TFRecord file format before passing to detection training. Use this command to do the conversion:

```
tlt-dataset-convert [-h] -d DATASET_EXPORT_SPEC -o OUTPUT_FILENAME
                         [-f VALIDATION_FOLD] [-v]
```

You can use these optional arguments:

- **`-h, --help`**: Show this help message and exit
- **`-d , --dataset-export-spec`**: Path to the detection dataset spec containing config for exporting .tfrecords.
- **`-o output_filename`**: Output file name.
- **`-f , -validation-fold`**: Indicate the validation fold in 0-based indexing. This is required when modifying the training set but otherwise optional.
- **`-v, --verbose`**: Flag to get detailed logs during the conversion process.

Here's an example of using the command with the dataset:

```
tlt-dataset-convert -d <path_to_tfrecords_conversion_spec> -o
 <path_to_output_tfrecords>
```

Output log from executing **`tlt-dataset-convert`**:

```
Using TensorFlow backend.
2018-11-06 00:41:27,318 - iva.dashnet.dataio.build_converter - INFO -
 Instantiating a kitti converter
2018-11-06 00:41:27,371 - dlav.drivenet.dataio.dataset_converter_lib - INFO -
 Writing partition 0, shard 0
/usr/local/lib/python2.7/dist-packages/iva/dashnet/dataio/
kitti_converter_lib.py:255: VisibleDeprecationWarning: Reading unicode strings
 without specifying the encoding argument is deprecated. Set the encoding, use
 None for the system default.
..
..
2018-11-06 00:42:14,494 - dlav.drivenet.dataio.dataset_converter_lib - INFO -
 Writing partition 0, shard 9
2018-11-06 00:42:19,442 - dlav.drivenet.dataio.dataset_converter_lib - INFO -
Wrote the following numbers of objects:
bicycle: 5
automobile: 121487
heavy_truck: 7748
person: 40
road_sign: 11580
motorcycle: 7
rider: 670
..
..
2018-11-06 00:47:34,515 - dlav.drivenet.dataio.dataset_converter_lib - INFO -
 Writing partition 4, shard 9
2018-11-06 00:47:42,614 - dlav.drivenet.dataio.dataset_converter_lib - INFO -
Wrote the following numbers of objects:
bicycle: 2
automobile: 41932
```

```
heavy_truck: 7004
person: 51
road_sign: 8933
rider: 1281

2018-11-06 00:47:42,614 - dlav.drivenet.dataio.dataset_converter_lib - INFO -
Wrote the following numbers of objects:
bicycle: 45
automobile: 295471
heavy_truck: 36321
person: 541
road_sign: 53045
motorcycle: 10
rider: 3308
```

Chapter 5. TRAINING THE MODEL

This chapter discusses using the **tlt-train** command to train models with single and multiple GPUs.

5.1. Training a classification model

Use the **tlt-train** command to tune a pre-trained model:

```
tlt-train [-h] {classification} --gpus <num GPUs>
                [train.py python arguments]
```

The **tlt-train** command with the train.py arguments:

```
tlt-train [-h] classification --gpus <num GPUs>
                -k <encoding key>
                -r <result directory>
                [-e SPEC_FILE]
                [-v]
```

Required arguments:

- **classification** - User specific encoding key to save or load a .trim model.
- **-k, --key** - User specific encoding key to save or load a .tlt model.
- **-r, --results_dir** is the directory to store predictions and output.

Optional arguments:

- **--gpus NUM_GPUS** : number of GPUs to use for training
- **-e, --experiment_spec** the path to the experiment specification for classification.

See the Specification file for classification section for more details.

Here's an example of using the tlt-train command:

```
tlt-train classification -e /workspace/tlt_drive/spec/spec.cfg -r /workspace/
output -k $YOUR_KEY
```

Output Log

Here's the output log from the successful use of this command:

```
Using TensorFlow backend.
..
______________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to

==============================================================================
input_1 (InputLayer)            (None, 3, 224, 224)  0

..
..
..
_________________________________________________________________________________
predictions (Dense)             (None, 20)           10260       flatten_1[0][0]

=================================================================================
Total params: 11,558,548
Trainable params: 11,546,900
Non-trainable params: 11,648
_________________________________________________________________________________

Epoch 1/80
124/311 [==========>...................] - ETA: 49s - loss: 4.1188 - acc:
 0.06592018-10-11 22:09:13.292358: W tensorflow/core/framework/allocator.cc:101]
 Allocation of 38535168 exceeds 10% of system memory.
```

5.2. Training a detection model

Detection training

Use this command to perform image detection training on the model.

tlt-train usage with the train.py arguments:

```
tlt-train [-h] detection --gpus <num GPUs>
                -k <encoding key>
                -r <result directory>
                [-e SPEC_FILE]
                [-v]
```

Required arguments

- **`{detection}`** : Training **`type`**, each option triggers detection training respectively.
- **`-r, --results_dir`** : Path to a folder where experiment outputs should be written.
- **`-k, --key`** : User specific encoding key to save or load a **`.tlt`** model.

Optional arguments

- **`--gpus NUM_GPUS`**: Number of GPUs to use and processes to launch for training. The default value is: 1.
- **`-e, --experiment_spec_file`**: Path to the spec file. Absolute path or relative path to the working directory. The default value: **`spec`** from **`spec_loader.py`**.

See the Specification file for detection section for more details.

Usage example

Here's an example of a 2 GPU training session, the command line would look like this.

```
tlt-train detection -e <path_to_spec_file> \
                    -r <path_to_experiment_output> \
                    -k <API_KEY_to_load_the_model> \
                    -n <name_string_for_the_model> \
                    --gpus 2
```

Output Log

Here's the output log from the successful use of this command:

```
Using TensorFlow backend.
2018-11-06 01:03:16.402006: I tensorflow/core/common_runtime/gpu/
gpu_device.cc:1356] Found device 0 with properties:
name: TITAN X (Pascal) major: 6 minor: 1 memoryClockRate(GHz): 1.531
..
..
_____________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to

=============================================================================
input_1 (InputLayer)            (None, 3, 544, 960)  0

..
==============================================================================
Total params: 11,555,983
Trainable params: 11,544,335
Non-trainable params: 11,648
..
..
2018-11-06 01:04:06,173 [INFO] tensorflow: Running local_init_op.
..
INFO:tensorflow:loss = 0.07203477, epoch = 0.0, step = 0
2018-11-06 01:05:14,270 [INFO] tensorflow: loss = 0.07203477, epoch = 0.0, step
 = 0
INFO:tensorflow:Saving checkpoints for step-1.
..
2018-11-06 01:05:44,920 [INFO] tensorflow: loss = 0.05362146, epoch =
 0.0663716814159292, step = 15 (5.978 sec)
INFO:tensorflow:global_step/sec: 0.555544
..
===========  ======  ======  ======
 class mAP    easy    hard    mdrt
===========  ======  ======  ======
    car      91.06   84.50   84.50
  cyclist     0.00    7.70    7.70
pedestrian    0.00    0.00    0.00
===========  ======  ======  ======
Time taken to run /usr/local/lib/python2.7/dist-packages/iva/dashnet/scripts/
train.pyc:main: 0:45:45.071311.
```

Chapter 6. EVALUATING THE MODEL

When training is complete, the model is stored in the output directory of your choice in $OUTPUT_DIR. Evaluate a model using the **`tlt-evaluate`** command:

```
tlt-evaluate {classification,detection} [-h] [<arguments for classification or
 detection>]
```

Positional arguments: `[classification, detection]` : Choose to evaluate a classification or detection model.

Optional arguments: Vary depending on whether or not the model is a classification or detection model

See Evaluating a model for classification for more information on using this command to evaluate a classification model.

See Evaluating a model for detection for more information on using this command to evaluate a detection model.

6.1. Evaluating a model for classification

Execute **`tlt-evaluate`** on a classification model.

```
tlt-evaluate classification [-h] [-d TARGET_DIR] -pm PRETRAINED_MODEL -k KEY [-w
 WORKERS] [-b BATCH_SIZE] [-v]
```

Required arguments

- **`[classification, detection]`** : Choose [classification] to run evaluation for classification models.
- **`-d, --target_dir`** : Directory containing target dataset.
- **`-pm, --pretrained_model`** : Path to the model file to use for evaluation.
- **`-k , -key`** : Provide the encryption key to decrypt the model (API KEY).

Optional arguments

- **`-h, --help`** : show this help message and exit.
- **`-b, --batch_size`** : Batch size to use for evaluation.

If you followed the example in Training a classification model, you can run the evaluation:

```
tlt-evaluate classification -d /workspace/tlt_drive/examples/classification/
dataset/make_1k/val \
                            -pm /workspace/output/weights/resnet_003.tlt \
                            -b 32 -k $YOUR_KEY
```

The resulting log file will be similar to this:

```
Using TensorFlow backend.
..
..
______________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to

==============================================================================
input_1 (InputLayer)            (None, 3, 224, 224)  0

==============================================================================
Total params: 11,558,548
Trainable params: 11,546,900
Non-trainable params: 11,648
______________________________________________________________________________
Found 1923 images belonging to 20 classes.
```

6.2. Evaluating a model for detection

Execute **`tlt-evaluate`** to evaluate a detection model.

```
tlt-evaluate detection [-h] [-e EXPERIMENT_SPEC_FILE] -m MODEL_FILE -k KEY [--
use_training_set] [-v]
```

Required arguments:

```
[classification, detection] : Choose [detection] to run evaluation for detection
 models.
```

- **`-e, --experiment_spec_file`** : Experiment spec file to set up the evaluation experiment. This should be the same as training spec file.
- **`-m, --model`** : Path to the model file to use for evaluation.
- **`-k , --key`** : Provide the encryption key to decrypt the model (API KEY).

Optional arguments:

- **`-h, --help`** : show this help message and exit.
- **`--use_training_set`**: Set this flag to run evaluation on training + validation dataset.
- **`-v, --verbose`**: Flag to generate detailed log.

Evaluate the model

If you followed the example in Training a detection model, you can run the evaluation:

```
tlt-evaluate detection -e <path to training spec file>\
                       -m <path to the model> \
                       -k <key to load the model>
```

This command runs evaluation on the same validation set, that was set during training. To evaluate on a test set with ground truth labeled, follow these steps:

1. Create tfrecords for this training set using the steps in Preparing input data structure.
2. Update the dataloader configuration part of the training spec file to include the newly generated tfrecords. For more information on the dataset config, see the Dataloader.

```
dataset_config {
  data_sources: {
    tfrecords_path: "<path to training tfrecords root>/<tfrecords_name*>"
    image_directory_path: "<path to training data root>"
  }
  image_extension: "jpg"
  target_class_mapping {
      key: "car"
      value: "car"
  }
  target_class_mapping {
      key: "automobile"
      value: "car"
  }
  ..
  ..
  ..
  target_class_mapping {
      key: "person"
      value: "pedestrian"
  }
  target_class_mapping {
      key: "rider"
      value: "cyclist"
  }
  validation_data_source: {
    tfrecords_path: "<path to testing tfrecords root>/<tfrecords_name*>"
    image_directory_path: "<path to testing data root>"
  }
}
```

The resulting log file will be similar to this:

```
Using TensorFlow backend.
..
..
packages/iva/dashnet/evaluation/build_evaluator.pyc: Found 1802 samples in
 validation set
_______________________________________________________________________________

Layer (type)                    Output Shape         Param #     Connected to

===============================================================================
input_1 (InputLayer)            (None, 3, 544, 960)  0


=================================================================================
Total params: 11,555,983
Trainable params: 11,544,335
Non-trainable params: 11,648
_________________________________________________________________________________
INFO:tensorflow:Graph was finalized.
2018-10-22 19:55:24,136 [INFO] tensorflow: Graph was finalized.
..
..
```

```
Validation cost: 0.002105
AP:
===========  ==========  ======  ======  ======
 class mAP    combined    easy    hard    mdrt
===========  ==========  ======  ======  ======
    car         3.39     51.59   40.72   40.72
  cyclist       0.03      0.00    0.08    0.08
pedestrian      0.00      0.00    0.00    0.00
===========  ==========  ======  ======  ======
Time taken to run /usr/local/lib/python2.7/dist-packages/iva/dashnet/scripts/
evaluate.pyc:main: 0:02:06.424072.
```

Chapter 7. USING INFERENCE ON A MODEL

The **`tlt-infer`** tool produces a classification of images to produce a prediction on a single image or a directory of images.

7.1. Inference on a classification model

Execute `tlt-infer` on a classification model trained on the Transfer Learning Toolkit

```
tlt-infer classification [-h] [-m MODEL] [-i IMAGE] [-d IMAGE_DIR]
                    [-b BATCH_SIZE] [-k KEY] [-cm CLASSMAP][-v]
```

Here are the parameters of the **`tlt-infer`** tool:

```
-h, --help: Show this help message and exit
-m MODEL, --model MODEL: Path to the pretrained model (TLT model).
-i IMAGE, --image IMAGE: A single image file for inference.
-d IMAGE_DIR, --image_dir IMAGE_DIR: The directory of input images for inference.
-b BATCH_SIZE, --batch_size BATCH_SIZE: Inference batch size, default=1
-k KEY, --key KEY: Key to load model.
-cm CLASSMAP, --class_map CLASSMAP: The json file that specifies the class index
 and label mapping.
-v, --verbose: Flag used to generate more detailed logs.
```

Sample output using single image mode

```
Single Image Mode
____________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to

============================================================================
input_1 (InputLayer)            (None, 3, 224, 224)  0


____________________________________________________________________________
conv1 (Conv2D)                  (None, 16, 112, 112) 2368        input_1[0][0]


____________________________________________________________________________
...
...
____________________________________________________________________________
2018-11-05 18:46:16,248 [INFO] root: Current predictions: [[2.0956191e-04
 4.7424308e-08 6.0529976e-07 1.5379728e-05 4.9668059e-05
  2.3047665e-05 8.3990363e-07 2.1063986e-06 3.9042366e-06 9.8465785e-07
  7.9830796e-05 8.4068454e-08 1.3434786e-06 1.6271177e-05 1.1729119e-06
```

```
  9.9955863e-01 2.9604094e-05 2.6558594e-06 3.4933796e-06 7.3329272e-07]]
2018-11-05 18:46:16,248 [INFO] root: Class label = 15
2018-11-05 18:46:16,248 [INFO] root: Class name = mercedes
```

Execution using -d or directory mode

A result.csv file is created and stored in the directory you specifies using **`-d`**. The result.csv has the following format, where the second column shows the file path and third shows the predicted class name.

```
0,/home/tmp/1.jpg,A
0,/home/tmp/2.jpg,B
0,/home/tmp/3.jpg,C
```

7.2. Inference on a detection model

Execute the `tlt-infer` tool on an object detection model trained on the Transfer Learning Toolkit

This section describes using the **`tlt-infer`** tool for object detection networks used to visualize bboxes, or generate frame by frame KITTI format labels on a single image or a directory of images. Execute the **`tlt-infer`**:

```
tlt-infer detection [-h] [-m MODEL] [-i INFERENCE_INPUT] [-o INFERENCE_OUTPUT]
                    [-bs BATCH_SIZE] [-k] [-bo] [-cp CLUSTER_PARAMS_FILE]
                    [-lw LINE_WIDTH] [-g GPU_SET] [--output_cov OUTPUT_COV]
                    [--output_bbox OUTPUT_BBOX] -ek ENC_KEY [-v]
```

Here are the parameters of the **`tlt-infer`** tool:

```
-h, --help: Show this help message and exit
-m MODEL, --model MODEL: TLT model file path
-i INFERENCE_INPUT, --inference_input INFERENCE_INPUT: The directory of input
 images or a single image for inference.
-o INFERENCE_OUTPUT, --inference_output INFERENCE_OUTPUT: The directory to the
 output images and labels. The annotated images are in inference_output/
images_annotated and labels are in image_dir/labels
-bs BATCH_SIZE, --batch_size BATCH_SIZE: Inference batch size, default=1
-k, --kitti_dump: Flag to enable KITTI dump
-bo, --box_overlay: Flag to enable image overlay
-cp CLUSTER_PARAMS_FILE, --cluster_params_file CLUSTER_PARAMS_FILE: Bbox post
 processing json file.
-lw LINE_WIDTH, --line_width LINE_WIDTH: Overlay linewidth
-g GPU_SET, --gpu_set GPU_SET: GPU index to choose
--output_cov OUTPUT_COV: Name of the coverage layer. Note: If there is a reshape
 layer,then this is the layer just before reshape.
--output_bbox OUTPUT_BBOX: Name of output bbox layer. Note: If there is a reshape
 layer,then this is the layer just before reshape.
-ek ENC_KEY, --enc_key ENC_KEY: Key to load model.
-v, --verbosity: Flag to set for more detailed logs.
```

> The inference tool requires the **`cluster_params.json`** file to configure the post processing block.

A sample json file is found in the Specification file for inference section. Use this clusterfile with the pretrained models in the NGC. Use the **`-k`** and **`-bo`** parameters to generate the output file. Use these parameters to save the output labels and overlay images respectively to the output path. The output labels are saved in **`output_path/labels`** and the overlain images will be saved at **`output_path/images_annotated`**.

Output log from executing tlt-infer:

```
Using TensorFlow backend.
2018-11-05 16:56:08.557935: I tensorflow/core/common_runtime/gpu/
gpu_device.cc:1356] Found device 0 with properties:
name: TITAN Xp major: 6 minor: 1 memoryClockRate(GHz): 1.582
pciBusID: 0000:02:00.0
..
..
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 3, 384, 1240)      0
..
..
0it [00:00, ?it/s]
  0%|                                                    | 0/32 [00:00<?, ?it/s]
  3%|█#                                           | 1/32 [00:00<00:04,  7.50it/s]
  ..
100%|███████████████████████████████████████████| 23/23 [00:03<00:00,  7.18it/s]
1it [00:10, 10.85s/it]
  0%|                                                    | 0/32 [00:00<?, ?it/s]
  3%|█#                                           | 1/32 [00:00<00:03,  7.92it/s]
  ..
100%|███████████████████████████████████████████| 32/32 [00:04<00:00,  6.87it/s]
2it [00:19,  9.67s/it]
..
..
5it [00:40,  8.07s/it]
2018-11-05 16:56:52,571 [INFO] iva.dashnet.scripts.inference: Inference complete
```

Chapter 8.
PRUNING THE MODEL

Pruning removes parameters from the model to reduce the model size without compromising the integrity of the model itself using the **`tlt-prune`** command.

The **`tlt-prune`** command includes these parameters:

```
tlt-prune [-h] -pm PRETRAINED_MODEL -o OUTPUT_DIR -k KEY
            [-n NORMALIZER]
            [-eq EQUALIZATION_CRITERION]
            [-pg PRUNING_GRANULARITY]
            [-pth PRUNING_THRESHOLD]
            [-nf MIN_NUM_FILTERS]
            [-el [EXCLUDED_LAYERS [EXCLUDED_LAYERS ...]]]
            [-v]
```

Required arguments:

- **`pm, --pretrained_model`** : Path to pretrained model.
- **`-o, --output_dir`** : Path to output checkpoints.
- **`-k, --key`** : Key to load a .tlt model

Optional arguments

- **`-h, --help`**: Show this help message and exit.
- **`-n, –normalizer`** : `max` to normalize by dividing each norm by the maximum norm within a layer; `L2` to normalize by dividing by the L2 norm of the vector comprising all kernel norms. (default: `max`)
- **`-eq, --equalization_criterion`** : Criteria to equalize the stats of inputs to an element wise op layer. Options are [arithmetic_mean, geometric_mean, union, intersection]. (default: `union`)
- **`-pg, -pruning_granularity`**: Pruning granularity: number of filters to remove at a time. (default:8).
- -pth, : Threshold to compare normalized norm against. (default:0.1)
- **`-nf, --min_num_filters`** : Minimum number of filters to keep per layer. (default:16)
- **`-el, --excluded_layers`**: List of excluded_layers. Examples: -i item1 item2 (default: [])
- **`-v, --verbose`**: Flag to get detailed logs during the conversion process.

After pruning, the model needs to be retrained.

Re-training the pruned model

Pruning a model may result in a decrease in accuracy. To re-gain accuracy, re-train the model in your dataset. See Training the model.

Using the Prune command

Here's an example of using the **`tlt-prune`** command:

```
tlt-prune -pm /workspace/output/weights/resnet_003.tlt \
              -o /workspace/output/weights/resnet_003_pruned \
              -eq union \
              -pth 0.7 -k $API_KEY
```

Using this command produces a log similar to this:

```
Using TensorFlow backend.
2018-10-12 00:12:38.772343: I tensorflow/core/common_runtime/gpu/
gpu_device.cc:1356] Found device 0 with properties:
name: TITAN Xp major: 6 minor: 1 memoryClockRate(GHz): 1.582
pciBusID: 0000:01:00.0
totalMemory: 11.91GiB freeMemory: 10.58GiB
..
..
..
2018-10-12 00:12:45,132 [INFO] modulus.pruning.pruning: Pruning model and
 appending pruned nodes to new graph
2018-10-12 00:13:10,642 [INFO] /usr/local/lib/python2.7/dist-packages/iva/
common/tlt_prune.pyc: Pruning ratio: 0.0194629982936
```

Chapter 9.
EXPORTING THE MODEL

The Transfer Learning Toolkit includes the tlt-export command to export and prepare TLT models for Deploying to DeepStream. The **`tlt-export`** command optionally generates the calibration cache for TensorRT INT8 engine calibration.

Exporting the model decouples the training process from inference and allows conversion to TensorRT engines outside the TLT environment. TensorRT engines are specific to each hardware configuration and should be generated for each unique inference environment, but the same exported TLT model may be used universally.

Int8 mode overview

TensorRT engines can be generated in INT8 mode to improve performance, but require a calibration cache at engine creation-time. The calibration cache is generated using a specified directory of calibration data if **`tlt-export`** is run with the **`--data_type`** flag set to **`int8`**. The **`--cal_data_file`** argument should be a path to a directory of images. Pre-generating calibration information and caching it removes the need for the calibration data on the inference machine and the cache is usually smaller than a single data image. Using the calibration cache also speeds up engine creation as building the cache can take several minutes to generate depending on the size and number of batches ran and would need to be repeated during each engine creation step.

Using the int8 data type to generate a calibration cache

```
tlt-export [-h] [--enc_key ENC_KEY] [-o OUTPUT_FILE]
                [--data_type {fp32,fp16,int8}]
                [--outputs OUTPUTS]
                [--input_dims INPUT_DIMS]
                [--cal_data_file CAL_DATA_FILE]
                [--cal_cache_file CAL_CACHE_FILE]
                [--batches BATCHES]
                [--cal_batch_size CAL_BATCH_SIZE]
                [--max_batch_size MAX_BATCH_SIZE]
                [--max_workspace_size MAX_WORKSPACE_SIZE]
                [-v]
                input_file
```

Required arguments:

- **`input_file`**: Path to the model exported using **`tlt-export`**.
- **`-k ENCODE_KEY`**: API key used to download the model with tlt-pull.

- **`-O OUTPUTS`**: Comma-separated list of output blob names.
  - For classification use: **predictions/Softmax**
  - For detection use: **output_bbox/BiasAdd,output_cov/Sigmoid**

Optional arguments:

- **`-o, --output_file OUTPUT_FILE`** : Path to save the exported model to. The default is **`./<input_file>.etlt`**.
- **`-data_type {fp32,fp16,int8}`**: Desired engine data type, generates calibration cache if in int8 mode. The default value is fp32.

INT8 specific required arguments:

- **`--cal_data_file CAL_DATA_FILE`** : Directory of data used to create the calibration cache and will be used to calibrate the engine.
- **`--input_dims INPUT_DIMS`** : Comma separated list of input dimensions in CHW order.

INT8 specific optional arguments:

- **`--cal_cache_file CAL_CACHE_FILE`**: Path to save the calibration cache file. The default value is **`./cal.bin`**.
- **`--batches BATCHES`**: Number of batches to use for calibration and inference testing.The default value is 10.
- **`--cal_batch_size CAL_BATCH_SIZE`**: Batch size to use for calibration. The default value is 8.
- **`--max_batch_size MAX_BATCH_SIZE`**: Maximum batch size of TensorRT engine. The default value is 16.
- **`--max_workspace_size MAX_WORKSPACE_SIZE`** : Maximum workspace size of TensorRT engine. The default value is: 1073741824 = 1<<30)

Exporting a model

Using the **`resnet10_detection_kitti`** model as an example, model available in TLT using tlt-pull.

```
tlt-export --enc_key $API_KEY -o $EXPORT_PATH --input_dims $INPUT_DIMS --outputs
 $OUTPUT_NODES $MODEL_PATH
Using TensorFlow backend.
2018-11-02 18:59:43,347 [INFO] tlt.encoding.tlt_export: Loading model from
 resnet10_kitti_multiclass_v1.tlt
..
2018-11-02 18:59:47,572 [INFO] tensorflow: Restoring parameters from /tmp/
tmp8crUBp.ckpt
INFO:tensorflow:Froze 82 variables.
2018-11-02 18:59:47,701 [INFO] tensorflow: Froze 82 variables.
Converted 82 variables to const ops.
2018-11-02 18:59:48,123 [INFO] tlt.encoding.tlt_export: Converted model was
 saved into resnet10_kitti_multiclass_v1.etlt
2018-11-02 18:59:48,123 [INFO] tlt.encoding.tlt_export: Input node: input_1
2018-11-02 18:59:48,124 [INFO] tlt.encoding.tlt_export: Output node(s):
 ['output_bbox/BiasAdd', 'output_cov/Sigmoid']
```

Chapter 10. DEPLOYING TO DEEPSTREAM

The Transfer Learning Toolkit (tlt) is designed to facilitate use of DeepStream video analytics. For DeepStream deployment, create a TensorRT engine in the DeepStream environment on an inference device using the **`tlt-converter`**. Machine specific optimizations are done as part of the engine creation process, so a distinct engine should be generated for each environment and hardware configuration. If the inference environment's TensorRT or CUDA libraries are updated – including minor version updates – new engines should be generated. Running an engine that was generated with a different version of TensorRT and CUDA is not supported and will cause unknown behavior that affects inference speed, accuracy, and stability.

The **`tlt-converter`** takes a model that was exported in the TLT docker using **`tlt-export`** and converts it to a TensorRT engine. For more information regarding exporting models and INT8 inference mode please see Exporting the Models. Un-exported TLT models and non-TLT models are not supported and will not work with the **`tlt-converter`**.

Setup and Execution

The tlt-converter executable is distributed as part of the tlt-docker and must be copied from **`/workspace/tools/tlt-converter`** to the inference machine. The converter requires TensorRT 5.0 to be installed and available in the **`LD_LIBRARY_PATH`** to run successfully.

1. Extract the **`tlt-converter`** from the **`tlt-docker`** image by launching the image with a mounted local directory and copying **`/workspace/tools/tlt-converter`** to the mounted directory.
2. Copy the the **`tlt-converter`** executable and your exported trained model to your inference machine.
3. Locate the **`tlt-converter`** inside your inference environment and add its parent directory to the system path.
4. Run the converter following the instructions below and use the resulting TensorRT engine in DeepStream or for standalone inference.

Using the tlt-converter

```
tlt-converter [-h] [-k ENCODE_KEY] [-d INPUT_DIMENSIONS]
```

```
                [-o OUTPUTS]
                [-e ENGINE_FILE_PATH]
                [-c CACHE_FILE]
                [-b BATCH_SIZE]
                [-m MAX_BATCH_SIZE]
                [-t {fp32,fp16,int8}]
                [-w MAX_WORKSPACE_SIZE]
                [-i {nchw, nhwc, nc}]
                [-v]
                input_file
```

Required arguments:

- **`input_file`**: Path to the model exported using **`tlt-export`**.
- **`-k ENCODE_KEY`**: API key used to download the model with tlt-pull.
- **`-d INPUT_DIMENSIONS`**: Comma-separated list of input dimensions that should match the dimensions used for tlt-export. Unlike tlt-export this cannot be inferred from calibration data.
- **`-O OUTPUTS`**: Comma-separated list of output blob names that should match the output configuration used for tlt-export.

Optional arguments:

- **`-e, ENGINE_FILE_PATH`**: Path to save the engine to. (default: ./saved.engine)
- **`-t {fp32,fp16,int8}`**: Desired engine data type, generates calibration cache if in int8 mode. The default value is fp32.
- **`-w MAX_WORKSPACE_SIZE`**: Maximum workspace size for the TensorRT engine. The default value is **`1<<30`**.
- **`-i {nchw, nhwc, nc}`**: Input dimension ordering, all other tlt command use NCHW. The default value is nchw.

INT8 Mode Arguments:

- **`-c CACHE_FILE`**: Path to calibration cache file, only used in int8 mode. The default value is **`./cal.bin`**.
- **`-b BATCH_SIZE`** : Batch size to use for engine calibration. The default value is 8.
- **`-m MAX_BATCH_SIZE`**: Maximum batch size of TensorRT engine. The default value is 16.

Generating an Engine

Using the **`resnet10_detection_kitti`** model as an example, model available in Transfer Learning Toolkit using **`tlt-pull`** and using the exported model from the example in Exporting the model.

Chapter 11. CONFIGURING THE EXPERIMENT

11.1. Specification file for classification

Here's an example of classification training.

Inside the docker, assume you have mounted the shared tlt drive at **`/workspace/tlt_drive`** and pull a model from the NGC registry, in this example, **`/workspace/resnet18.tlt`**.

Example specification file for classification

```
model_config {
  arch: "resnet",
  n_layers: 18
  input_image_size: "3,224,224"
}
train_config {
  train_dataset_path: "/workspace/tlt_drive/classification/make_1k/train"
  val_dataset_path: "/workspace/tlt_drive/classification/make_1k/val"
  pretrained_model_path: "/workspace/resnet18.tlt"
  optimizer: "sgd"
  batch_size_per_gpu: 64
  n_epochs: 80
  n_workers: 16

  # regularizer
  reg_config {
    type: "L2"
    scope: "Conv2D,Dense"
    weight_decay: 0.00005
  }

  # learning_rate
  lr_config {
    scheduler: "step"
    learning_rate: 0.006
    step_size: 10
    gamma: 0.1
  }
}
```

Here's an example of using tlt-train:

```
tlt-train classification -e /workspace/tlt_drive/spec/make_spec.cfg -r /
workspace/output -k $YOUR_KEY
```

This command produces a log similar to this:

```
Log
Using TensorFlow backend.
..
______________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to

==============================================================================
input_1 (InputLayer)            (None, 3, 224, 224)  0

...
_________________________________________________________________________________
predictions (Dense)             (None, 20)           10260       flatten_1[0][0]

=================================================================================
Total params: 11,558,548
Trainable params: 11,546,900
Non-trainable params: 11,648
_________________________________________________________________________________
Epoch 1/80
124/311 [==========>...................] - ETA: 49s - loss: 4.1188 - acc:
 0.06592018-10-11 22:09:13.292358: W tensorflow/core/framework/allocator.cc:101]
 Allocation of 38535168 exceeds 10% of system memory.
```

11.2. Specification file for detection

Here's an example of a spec file used for detection. Detection training spec file components include:

- Model configuration in **`model_config`**
- Rasterization configration in **`bbox_rasterizer_config`**
- Post processing configuration in **`post_processing_config`**
- Training configuration in **`training_config`**
- Cost function configuration in **`cost_function_config`**
- Augmentation configuration in **`augmentation_config`**
- Evaluator configuration in **`evalutaion_config`**
- Dataloader configuration in **`dataset_config`**

This is a template for generating a specification file for a detection model.

```
random_seed: 42
model_config {
  template: "resnet"
  num_layers: 18
  use_pooling: False
  use_batch_norm: True
  dropout_rate: 0.0
  training_precision: {
    backend_floatx: FLOAT32
  }
  objective_set: {
    cov {}
    bbox {
```

```
      scale: 35.0
      offset: 0.5
    }
  }
}

bbox_rasterizer_config {
  target_class_config {
    key: "car"
    value: {
      cov_center_x: 0.5
      cov_center_y: 0.5
      cov_radius_x: 0.4
      cov_radius_y: 0.4
      bbox_min_radius: 1.0
    }
  }
  target_class_config {
    key: "cyclist"
    value: {
      cov_center_x: 0.5
      cov_center_y: 0.5
      cov_radius_x: 1.0
      cov_radius_y: 1.0
      bbox_min_radius: 1.0
    }
  }
  target_class_config {
    key: "pedestrian"
    value: {
      cov_center_x: 0.5
      cov_center_y: 0.5
      cov_radius_x: 1.0
      cov_radius_y: 1.0
      bbox_min_radius: 1.0
    }
  }
  deadzone_radius: 0.67
}

cost_function_config {
  target_classes {
    name: "car"
    class_weight: 1.0
    coverage_foreground_weight: 0.05
    objectives {
      name: "cov"
      initial_weight: 1.0
      weight_target: 1.0
    }
    objectives {
      name: "bbox"
      initial_weight: 10.0
      weight_target: 10.0
    }
  }
  target_classes {
    name: "cyclist"
    class_weight: 1.0
    coverage_foreground_weight: 0.05
    objectives {
      name: "cov"
      initial_weight: 1.0
      weight_target: 1.0
    }
    objectives {
      name: "bbox"
```

```
      initial_weight: 10.0
      weight_target: 1.0
    }
  }
  target_classes {
    name: "pedestrian"
    class_weight: 1.0
    coverage_foreground_weight: 0.05
    objectives {
      name: "cov"
      initial_weight: 1.0
      weight_target: 1.0
    }
    objectives {
      name: "bbox"
      initial_weight: 10.0
      weight_target: 10.0
    }
  }
  enable_autoweighting: True
  max_objective_weight: 0.9999
  min_objective_weight: 0.0001
}

training_config {
  batch_size_per_gpu: 16
  num_epochs: 80
  learning_rate {
    soft_start_annealing_schedule {
      min_learning_rate: 5e-6
      max_learning_rate: 5e-4
      soft_start: 0.1
      annealing: 0.7
    }
  }
  regularizer {
    type: L1
    weight: 3e-9
  }
  optimizer {
    adam {
      epsilon: 1e-08
      beta1: 0.9
      beta2: 0.999
    }
  }
  cost_scaling {
    enabled: False
    initial_exponent: 20.0
    increment: 0.005
    decrement: 1.0
  }
}
augmentation_config {
  preprocessing {
    output_image_width: 960
    output_image_height: 544
    min_bbox_width: 1.0
    min_bbox_height: 1.0
  }
  spatial_augmentation {
    hflip_probability: 0.5
    vflip_probability: 0.0
    zoom_min: 1.0
    zoom_max: 1.0
    translate_max_x: 8.0
    translate_max_y: 8.0
```

```
  }
  color_augmentation {
    color_shift_stddev: 0.0
    hue_rotation_max: 25.0
    saturation_shift_max: 0.2
    contrast_scale_max: 0.1
    contrast_center: 0.5
  }
}
postprocessing_config {
  target_class_config {
    key: "car"
    value: {
      clustering_config {
        coverage_threshold: 0.005
        dbscan_eps: 0.13
        dbscan_min_samples: 0.05
        minimum_bounding_box_height: 20
      }
    }
  }
  target_class_config {
    key: "cyclist"
    value: {
      clustering_config {
        coverage_threshold: 0.005
        dbscan_eps: 0.15
        dbscan_min_samples: 0.05
        minimum_bounding_box_height: 20
      }
    }
  }
  target_class_config {
    key: "pedestrian"
    value: {
      clustering_config {
        coverage_threshold: 0.005
        dbscan_eps: 0.15
        dbscan_min_samples: 0.05
        minimum_bounding_box_height: 20
      }
    }
  }
}
dataset_config {
  data_sources: {
    tfrecords_path: "/workspace/tlt-experiments/tfrecords/iva_its_sequencewise/
iva_its_sequencewise*"
    image_directory_path: "/media/tlt_home/examples/detection/dataset"
  }
  image_extension: "jpg"
  target_class_mapping {
      key: "car"
      value: "car"
  }
  target_class_mapping {
      key: "automobile"
      value: "car"
  }
  target_class_mapping {
      key: "heavy_truck"
      value: "car"
  }
  target_class_mapping {
      key: "person"
      value: "pedestrian"
  }
```

```
  target_class_mapping {
      key: "rider"
      value: "cyclist"
  }
  validation_fold: 0
}
evaluation_config {
  validation_period_during_training: 10
  first_validation_epoch: 40
  minimum_detection_ground_truth_overlap {
    key: "car"
    value: 0.7
  }
  minimum_detection_ground_truth_overlap {
    key: "cyclist"
    value: 0.5
  }
  minimum_detection_ground_truth_overlap {
    key: "pedestrian"
    value: 0.5
  }
  evaluation_bucket_config {
    key: "easy"
    value {
      minimum_height: 40
      maximum_height: 9999
      minimum_occlusion: 0
      maximum_occlusion: 0
      truncation {
          minimum: 0.0
          maximum: 0.15
      }
    }
  }
  evaluation_bucket_config {
    key: "mdrt"
    value {
      minimum_height: 25
      maximum_height: 9999
      minimum_occlusion: 0
      maximum_occlusion: 1
      truncation {
          minimum: 0.0
          maximum: 0.3
      }
    }
  }
  evaluation_bucket_config {
    key: "hard"
    value: {
      minimum_height: 25
      maximum_height: 9999
      minimum_occlusion: 0
      maximum_occlusion: 2
      truncation {
          minimum: 0.0
          maximum: 0.5
      }
    }
  }
  importance_weights {
    weight_camera {
      key: 'front'
      value: 0.5
    }
    weight_cvip {
      key: true
```

```
      value: 10.0
    }
    weight_cvip {
      key: false
      value: 0.0
    }
    weight_detection_in_path {
      key: true
      value: 1.0
    }
    weight_detection_in_path {
      key: false
      value: 0.0
    }
    weight_height {
      minimum_height: 0
      maximum_height: 25
      weight: 0.0
    }
    weight_height {
      minimum_height: 25
      maximum_height: 100
      weight: 1.0
    }
    weight_height {
      minimum_height: 100
      maximum_height: 9999
      weight: 5.0
    }
    weight_occlusion {
      key: -1
      value: 0.0
    }
    weight_occlusion {
      key: 0
      value: 3.0
    }
    weight_occlusion {
      key: 1
      value: 0.0
    }
    weight_occlusion {
      key: 2
      value: 0.0
    }
    weight_occlusion {
      key: 3
      value: 0.0
    }
  }
}
```

Model for object detection

Core object detection can be configured using the **`model_config`** option in the spec file. Here are the parameters:

- **`template`**: This defines the architecture of the back bone feature extractor to be used to train. The supported architectures include, Resnets, Vgg's and Googlenet
- **`num_layers`**: This parameter defines the depth of the feature extractor, for scalable templates. In our case, the templates that support this are resnets and vgg. The depths that we support for these templates are resnets:

- **resnets**: 10, 18, 50
- **vgg**: 16, 19

> Googlenet is a fixed architecture network, so this parameter is ignored.

- **pretrained_model_file**: This parameter defines the path to a pretained tlt model file. Note that in case the tlt model file is invoked, the template and num_layers parameters are ignored. They are still required to be present in the spec file for setting up the experiment. With detection, if the pretrained model has a different number of classes than the your training dataset the pretrained model file can't be used.
- **use_pooling**: Choose between using strided convolutions or MaxPooling while down sampling. When true, we use MaxPooling to down sample, however for the object detection network, we recommend setting this to False and use strided convolutions.
- **use_batch_norm**: Boolean variable to use batch normalization layers or not.
- **dropout_rate**: Probability for drop out.
- **training_precision**: This parameters contains a nested parameter that sets the precision of the back-end training framework. Currently we support only float32.
- **objective_set**: This defines what objectives is this network being trained for. For object detection networks, we set to learn cov and bbox. These parameters should not be altered for the current training pipeline.

Code sample

```
model_config {
  template: "resnet"
  num_layers: 18
  use_pooling: False
  use_batch_norm: True
  dropout_rate: 0.0
  training_precision: {
    backend_floatx: FLOAT32
  }
  objective_set: {
    cov {}
    bbox {
      scale: 35.0
      offset: 0.5
    }
  }
}
```

Bounding box ground truth generator

The **bbox_rasterizer_config** field in the spec file configures the bounding box ground truth generator. In this example the replicate entries is based on the number of classes, with the key as the class name. A 3 class object detector training for cars, pedestrians and cyclists the rasterizer spec would be as follows:

```
bbox_rasterizer_config {
  target_class_config {
    key: "car"
    value: {
      cov_center_x: 0.5
      cov_center_y: 0.5
      cov_radius_x: 0.4
      cov_radius_y: 0.4
      bbox_min_radius: 1.0
    }
  }
  target_class_config {
    key: "cyclist"
    value: {
      cov_center_x: 0.5
      cov_center_y: 0.5
      cov_radius_x: 0.4
      cov_radius_y: 0.4
      bbox_min_radius: 1.0
    }
  }
  target_class_config {
    key: "pedestrian"
    value: {
      cov_center_x: 0.5
      cov_center_y: 0.5
      cov_radius_x: 0.4
      cov_radius_y: 0.4
      bbox_min_radius: 1.0
    }
  }
  deadzone_radius: 0.67
}
```

Post processor

Define the parameters that configure the post processor. In each class you train for, the **postprocessing_config** has a **target_class_config** element, which defines the clustering parameters for the class. The parameters for each target class are:

- **key**: class name
- **value**: containing a clustering_config parameters defining parameters for the DBSCAN clustering algorithm. The DBSCAN algorithm helps cluster the valid predictions to a box per object.

The **clustering_config** element configures the clustering block for this class:

- **coverage_threshold**: minimum confidence generated by the network to filter out valid bboxes
- **dbscan_eps**: epsilon value for the dbscan algorithm
- **dbscan_min_samples**: minimum samples parameter for the dbscan algorithm
- **minimum_bounding_box_height**: minimum height in pixels to consider as a valid detection.

In this example, the postprocessor is defined for a 3 class network learning **`car`**, **`cyclist`**, and **`pedestrian`**.

```
postprocessing_config {
  target_class_config {
    key: "car"
    value: {
      clustering_config {
        coverage_threshold: 0.005
        dbscan_eps: 0.15
        dbscan_min_samples: 0.05
        minimum_bounding_box_height: 20
      }
    }
  }
  target_class_config {
    key: "cyclist"
    value: {
      clustering_config {
        coverage_threshold: 0.005
        dbscan_eps: 0.15
        dbscan_min_samples: 0.05
        minimum_bounding_box_height: 20
      }
    }
  }
  target_class_config {
    key: "pedestrian"
    value: {
      clustering_config {
        coverage_threshold: 0.005
        dbscan_eps: 0.15
        dbscan_min_samples: 0.05
        minimum_bounding_box_height: 20
      }
    }
  }
}
```

Cost function

Configure the cost function to include the classes for which you are training. For each class you want to train, add a new entry of the target classes to the spec file. Nvidia recommends that you do not change the parameters within the spec file for best performance with these classes. The other parameters remain the same.

```
cost_function_config {
  target_classes {
    name: "car"
    class_weight: 1.0
    coverage_foreground_weight: 0.05
    objectives {
      name: "cov"
      initial_weight: 1.0
      weight_target: 1.0
    }
    objectives {
      name: "bbox"
      initial_weight: 10.0
      weight_target: 10.0
    }
  }
  target_classes {
```

```
    name: "cyclist"
    class_weight: 1.0
    coverage_foreground_weight: 0.05
    objectives {
      name: "cov"
      initial_weight: 1.0
      weight_target: 1.0
    }
    objectives {
      name: "bbox"
      initial_weight: 10.0
      weight_target: 1.0
    }
  }
  target_classes {
    name: "pedestrian"
    class_weight: 1.0
    coverage_foreground_weight: 0.05
    objectives {
      name: "cov"
      initial_weight: 1.0
      weight_target: 1.0
    }
    objectives {
      name: "bbox"
      initial_weight: 10.0
      weight_target: 10.0
    }
  }
  enable_autoweighting: True
  max_objective_weight: 0.9999
  min_objective_weight: 0.0001
}
```

Trainer

Here are the parameters used to configure the trainer:

- **`batch_size_per_gpu`**: Number of image per GPU
- **`num_epochs`**: Number of epochs to train on
- **`learning rate`**: Defines what kind of learning rate annealing scheduling use. Currently, only soft_start_annealing_schedule, which has the following nested parameters, is supported:
  - **`min_learning_rate`**: Minimum learning late to be seen during the entire experiment
  - **`max_learning_rate`**: Maximum learning rate to be seen during the entire experiment
  - **`soft start`**: Time to be lapsed before warm up (expressed in percentage of progress between 0 and 1)
  - **`annealing`**: Time to start annealing the learning rate
- **`reglarizer`**: Configures the regularizer to be used while training and contains the following nested params
  - **`type`**: Type or regularizer to use. We support NO_REG, L1 or L2

- **`weight`**: The floating point value for regularizer weight
- > Nvidia suggests using L1 regularizer when training a network before pruning as L1 regularization helps making the networks more prunable.
- **`optimizer`**: Configures the optimizer to be used. For object detection, only ADAM is currently supported. Adam contains nested parameters for the optimizer:
  - **`epsilon`**
  - **`beta1`**
  - **`beta2`**
- cost_scaling: This parameter is to be left fixed for your training pipe.

```
training_config {
  batch_size_per_gpu: 16
  num_epochs: 80
  learning_rate {
    soft_start_annealing_schedule {
      min_learning_rate: 5e-6
      max_learning_rate: 5e-4
      soft_start: 0.1
      annealing: 0.7
    }
  }
  regularizer {
    type: L1
    weight: 3e-9
  }
  optimizer {
    adam {
      epsilon: 1e-08
      beta1: 0.9
      beta2: 0.999
    }
  }
  cost_scaling {
    enabled: False
    initial_exponent: 20.0
    increment: 0.005
    decrement: 1.0
  }
}
```

Augmentation module

The augmentation module provides some basic pre-processing and augmentation when training. The augmentation_config contains three elements :

- **`preprocessing`**: This parameter defines the input dimensions to the network and minimum dimensions to filter out bboxes during training.
- **`spatial_augmentation`**: This module supports basic augmentation such as flip, zoom and translate which may be configured.
- **`color_augmentation`**: This module configures the color space transformations, such as color shift, hue_rotation, saturation shift, and contrast adjustment.

Here is a sample augmentation config element:

```
augmentation_config {
  preprocessing {
    output_image_width: 960
    output_image_height: 544
    min_bbox_width: 1.0
    min_bbox_height: 1.0
  }
  spatial_augmentation {
    hflip_probability: 0.5
    vflip_probability: 0.0
    zoom_min: 1.0
    zoom_max: 1.0
    translate_max_x: 8.0
    translate_max_y: 8.0
  }
  color_augmentation {
    color_shift_stddev: 0.0
    hue_rotation_max: 25.0
    saturation_shift_max: 0.2
    contrast_scale_max: 0.1
    contrast_center: 0.5
  }
}
```

Evaluation

The evaluator in the detection training pipe can be configured using the evaluation_config params. The evaluation values, easy, mdrt (moderate), hard, and weighted are defined may be configured as mentioned below. These evaluation values filter detections based on object height, truncation, and occlusion and evaluate mAP on the filtered set of objects.

The mAP calculation method is based on the PASCAL VOC and KITTI evaluation methods.

```
evaluation_config {
  validation_period_during_training: 10
  first_validation_epoch: 40
  minimum_detection_ground_truth_overlap {
    key: "car"
    value: 0.7
  }
  minimum_detection_ground_truth_overlap {
    key: "cyclist"
    value: 0.5
  }
  minimum_detection_ground_truth_overlap {
    key: "pedestrian"
    value: 0.5
  }
  evaluation_bucket_config {
    key: "easy"
    value {
      minimum_height: 40
      maximum_height: 9999
      minimum_occlusion: 0
      maximum_occlusion: 0
```

```
      truncation {
          minimum: 0.0
          maximum: 0.15
      }
    }
  }
  evaluation_bucket_config {
    key: "mdrt"
    value {
      minimum_height: 25
      maximum_height: 9999
      minimum_occlusion: 0
      maximum_occlusion: 1
      truncation {
          minimum: 0.0
          maximum: 0.3
      }
    }
  }
  evaluation_bucket_config {
    key: "hard"
    value: {
      minimum_height: 25
      maximum_height: 9999
      minimum_occlusion: 0
      maximum_occlusion: 2
      truncation {
          minimum: 0.0
          maximum: 0.5
      }
    }
  }
  importance_weights {
    weight_camera {
      key: 'front'
      value: 0.5
    }
    weight_cvip {
      key: true
      value: 10.0
    }
    weight_cvip {
      key: false
      value: 0.0
    }
    weight_detection_in_path {
      key: true
      value: 1.0
    }
    weight_detection_in_path {
      key: false
      value: 0.0
    }
    weight_height {
      minimum_height: 0
      maximum_height: 25
      weight: 0.0
    }
    weight_height {
      minimum_height: 25
      maximum_height: 100
      weight: 1.0
    }
    weight_height {
      minimum_height: 100
      maximum_height: 9999
      weight: 5.0
```

```
    }
    weight_occlusion {
      key: -1
      value: 0.0
    }
    weight_occlusion {
      key: 0
      value: 3.0
    }
    weight_occlusion {
      key: 1
      value: 0.0
    }
    weight_occlusion {
      key: 2
      value: 0.0
    }
    weight_occlusion {
      key: 3
      value: 0.0
    }
  }
}
```

Dataloader

This section defines the parameters to configure the dataloader. Here, we define the path to the data we want to train on and class mapping for classes in the dataset to the classes that the network would be trained for. The parameters in the dataset config are

- **`data sources`**: Contains the path to the tfrecords
- **`image extension`**: Extension of the images
- **`target class mapping`**: Maps the class names in the tfrecords to the target class to be trained in the network.
- **`validation fold`**: In case of an n fold tfrecords, we define the index of the fold to validate on. For sequence wise validation we may choose the validation fold in the range [0, N-1]. However, for a random split tfrecords, we force the validation fold index to 0 as the tfrecord is just 2-fold.

```
dataset_config {
  data_sources: {
    tfrecords_path: "<path to the training tfrecords roots/tfrecords train
 pattern*"
    image_directory_path: "path to the training data source"
  }
  image_extension: "jpg"
  target_class_mapping {
      key: "car"
      value: "car"
  }
  target_class_mapping {
      key: "automobile"
      value: "car"
  }
  target_class_mapping {
      key: "heavy_truck"
      value: "car"
  }
  target_class_mapping {
      key: "person"
```

```
      value: "pedestrian"
  }
  target_class_mapping {
      key: "rider"
      value: "cyclist"
  }
  validation_fold: 0
}
```

In this example the tfrecords is assumed to be multi fold, and the fold number to validate on is defined. If you want to validate on a different tfrecords than those defined in the training set then, use the **`validation_data_source`** field to define this. In this case, remove the **`validation_fold`** field from the spec.

`validation_data_source: {tfrecords_path: " <path to tfrecords to validate on>/tfrecords validation pattern"image_directory_path: " <path to validation data source>"}`

Specification file for inference

This spec file for inference is used to set up the post processing block. These are the parameters:

- **`dbscan_criterion`**: The criterion to cluster the bboxes. For this release, we only support "IOU" Intersection over Union.
- **`dbscan_eps`**: The minimum distance between to bboxes to be considered in the same cluster.
- **`dbscan_min_samples`**: The minimum number of samples that required to consider a valid cluster.
- **`min_cov_to_cluster`**: This is the equivalent to the converage threshold in the post processing config in the Post processor section. It acts as a first level filter to send valid bboxes to the clustering algorithm.
- **`min_obj_height`**: The minimum height in pixels to filter out noisy bboxes.
- **`target_classes`**: The list of classes the networks has been trained for. The order of the list must be the same as that during training.
- **`confidence_th`**: The confidence threshold to cluster out bboxes after clustering.
- **`output_map`**: The class mapping from the target classes in the network to the labels that maybe output to the kitti labels file.
- **`color`**: The color of the bboxes for each class. This is important when visualizing the boxes.
- **`postproc_classes`**: This parameter is used incase you would like to filter out and visualize only a subset of classes.
- **`image_height`**: The height of the image at inference.
- **`image_width`**: The width of the image at inference.

```
{
    "dbscan_criterion": "IOU",
    "dbscan_eps": {
        "cyclist": 0.4,
```

```
        "car": 0.25,
        "default": 0.15,
        "pedestrian": 0.4
    },
    "dbscan_min_samples": {
        "cyclist": 0.05,
        "car": 0.05,
        "default": 0.0,
        "pedestrian": 0.05
    },
    "min_cov_to_cluster": {
        "cyclist": 0.075,
        "car": 0.075,
        "default": 0.005,
        "pedestrian": 0.005
    },
    "min_obj_height": {
        "cyclist": 4,
        "car": 4,
        "pedestrian": 4,
        "default": 2
    },
    "target_classes": ["car", "cyclist", "pedestrian"],
    "confidence_th": {
        "car": 0.3,
        "cyclist": 0.3,
        "pedestrian": 0.2
    },
    "output_map": {
        "pedestrian" : "pedestrian",
        "car" : "car",
        "cyclist" : "cyclist"
    },
    "color": {
        "car": "green",
        "pedestrian": "magenta",
        "cyclist": "cyan"
    },
    "postproc_classes": ["car", "cyclist", "pedestrian"],
    "image_height":384,
    "image_width": 1240
}
```

Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, cuFFT, cuSPARSE, DIGITS, DGX, DGX-1, DGX Station, GRID, Jetson, Kepler, NVIDIA GPU Cloud, Maxwell, NCCL, NVLink, Pascal, Tegra, TensorRT, Tesla and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the Unites States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2019 NVIDIA Corporation. All rights reserved.

www.nvidia.com

NVIDIA.