Neuralet Edge Vision

Release 0.0.1

Neuralet

Jan 28, 2021

CONTENTS:

1 Getting Started **3**
- 1.1 X86 . . . 3
- 1.2 X86 nodes with GPU . . . 3
- 1.3 Nvidia Jetson Devices . . . 3
- 1.4 Coral Dev Board . . . 4
- 1.5 AMD64 node with a connected Coral USB Accelerator . . . 4

2 Export Models to Edge Devices **5**
- 2.1 Compile tflite Models to Edge TPU Models . . . 5
- 2.2 Export TensorFlow protobuf models to TRT engines . . . 5

3 Run Inference **7**

4 Adaptive Learning **9**
- 4.1 Client . . . 9
- 4.2 Adaptive Learning Config File . . . 10

5 Indices and tables **13**

This is the neuralet edge vision module. With this module you can run object detection models on various edge devices and create an adaptive learning session to customize object detection models to your specific environment. for more information please visit our website.or reach out to hello@neuralet.com.

CHAPTER

ONE

GETTING STARTED

You can run Object Detection module on various platforms

1.1 X86

You should have website Docker on your system.

```
# 1) Build Docker image
docker build -f x86.Dockerfile -t "neuralet/object-detection:latest-x86_64_cpu" .

# 2) Run Docker container:
docker run -it -v "$PWD":/repo neuralet/object-detection:latest-x86_64_cpu
```

1.2 X86 nodes with GPU

You should have Docker and Nvidia Docker Toolkit on your system.

```
# 1) Build Docker image
docker build -f x86-gpu.Dockerfile -t "neuralet/object-detection:latest-x86_64_gpu" .

# 2) Run Docker container:
Notice: you must have Docker >= 19.03 to run the container with `--gpus` flag.
docker run -it --gpus all -v "$PWD":/repo neuralet/object-detection:latest-x86_64_gpu
```

1.3 Nvidia Jetson Devices

You need to have JetPack 4.3 installed on your Jetson device.

```
# 1) Build Docker image
docker build -f jetson-nano.Dockerfile -t "neuralet/object-detection:latest-jetson_
↪nano" .

# 2) Run Docker container:
Notice: you must have Docker >= 19.03 to run the container with `--gpus` flag.
docker run -it --runtime nvidia --privileged -v "$PWD":/repo neuralet/object-
↪detection:latest-jetson_nano
```

1.4 Coral Dev Board

```
# 1) Build Docker image
docker build -f coral-dev-board.Dockerfile -t "neuralet/object-detection:latest-coral-
↪dev-board" .

# 2) Run Docker container:
docker run -it --privileged -v "$PWD":/repo neuralet/object-detection:latest-coral-
↪dev-board
```

1.5 AMD64 node with a connected Coral USB Accelerator

```
# 1) Build Docker image
docker build -f amd64-usbtpu.Dockerfile -t "neuralet/object-detection:latest-amd64" .

# 2) Run Docker container:
docker run -it --privileged -v "$PWD":/repo neuralet/object-detection:latest-amd64
```

CHAPTER

TWO

EXPORT MODELS TO EDGE DEVICES

With Neuralet Edge Object Detection module you can easily export your trained model to Nvidia's Jetson Devices and Google Edge TPUs.

2.1 Compile tflite Models to Edge TPU Models

In amd64 with connected USB Accelerator docker container run:

```
python3 exporters/edgetpu_exporter.py --tflite_file TFLITE_FILE --out_dir OUT_DIR
```

Where `TFLITE_FILE` should be a quantized model. You can use our adaptive learning API for training a quantized object detection model.

For more information about quantization techniques of deep neural networks you can read our blog.

2.2 Export TensorFlow protobuf models to TRT engines

In Jetson docker container run:

```
python3 exporters/trt_exporter.py --pb_file PB_FILE --out_dir OUT_DIR [ --num_classes␣
↪NUM_CLASSES]
```

Where `PB_FILE` is a protobuf frozen graph tensorflow model.

CHAPTER

THREE

RUN INFERENCE

On any of the docker containers you can run sample inference to get an output video:

```
python3 inference.py --device DEVICE --input_video INPUT_VIDEO --out_dir OUT_DIR \
                [--model_path MODEL_PATH] [--threshold THRESHOLD]  [--input_width␣
↪INPUT_WIDTH]\
                [--input_height INPUT_HEIGHT] [--out_width OUT_WIDTH] [--out_height␣
↪OUT_HEIGHT]
```

Where:

`DEVICE` should be one of the `x86`, `edgetpu` or `jetson`.

`INPUT_VIDEO` is the path to the input video file.

`OUT_DIR` is directory in which the output video file will be saved.

`MODEL_PATH` is the path to the model file or directory. for `x86` devices it should be a directory which contains the `saved_model` directory. For `edgetpu` it should be a compiled `tflite` file, and for `jetson` devices it should be a `TRT Engine` file.

`threshold` is the detector's threshold to detect objects.

`INPUT_WIDTH` and `INPUT_HEIGHT` are width and height of the input of the model.

`OUT_WIDTH` and `OUT_HEIGHT` are resolution of output video.

CHAPTER

FOUR

ADAPTIVE LEARNING

Adaptive learning is the process of customization of object detection models with user provided data and environments. For more information please visit our blog post.

4.1 Client

Neuralet adaptive learning service includes client/server side. By the client side you can start an adaptive learning task on cloud and get the model after training.

Run the docker container based on your device and the below commends inside the container:

```
cd services/adaptive-learning/client
```

#Step 1:

Create an `input.zip` file from the video file you want to feed to Adaptive Learning.

```
zip input.zip PATH_TO_VIDEO_FILE
```

#Step 2:

Upload the zip file and get a unique id:

```
python3 client.py upload_file --file_path FILE_PATH
```

#Step 3:

Add the previous step's unique id to the `UploadUUID` field and the video file name to the `VideoFile` field of config file. You can find a more comprehensive explanation of config file and its fields in the next section. Note: You can use the sample config file in `configs/sample_config.ini`

#Step 4:

Initiate a new job and get your job's ID:

```
python3 client.py train --config_path CONFIGPATH
```

#Step 5:

Get a job status (enter the job id at JASKID)

```
python3 client.py get_status --job_id JOBID
```

#Step 6:

Download the trained model whenever the job has been finished.

```
python3 client.py download_file --job_id JOBID
```

4.2 Adaptive Learning Config File

To customize the Adaptive Learning framework based on your needs, you must configure the sample config file on `configs/` directory. there is a brief explanation on each parameter of config files in the following table:

Table 1: a title

Parameter	Options	Comments
Teacher/MinScore	a float number between 0 and 1	the teacher model threshold for detecting an object
Teacher/UploadUUID	a UUID	unique id of uploaded input.zip file
Teacher/VideoFile	string	name of the video you zipped and uploaded
Teacher/MinDetectionPerFrame	an integer number	The teacher only stores the frame only if it detects at least this many objects, otherwise it discards the frame.
Teacher/SaveFrequency	An integer bigger than 0	The teacher model will store video frames and the corresponding predictions with this frequency
Teacher/PostProcessing	One of `'background_filter'` or `''`	Background filter will apply a background subtraction algorithm on video frames and discards the bounding boxes in which their background pixels rate is higher than a defined threshold.
Teacher/ImageFeature	One of the `'foreground_mask'`, `'optical_flow_magnitude'`, `'foreground_mask && optical_flow_magnitude'` or `''`	This parameter specifies the type of input feature engineering that will perform for training. `'foreground_mask'` replaces one of the RGB channels with the foreground mask. `'optical_flow_magnitude'` replaces one of the RGB channels with the magnitude of optical flow vectors and, `'foreground_mask && optical_flow_magnitude'` performs two feature engineering technique at the same time as well as changing the remaining RGB channel with the grayscale transformation of the frame. For more information about feature engineering and their impact on the model's accuracy, visit our blog
Student/BatchSize	An integer bigger than 0	The student's training batch size
Student/Epochs	An integer bigger than 0	The student's training number of epochs in each round
Student/ValidationSplit	An float between 0 and 1	the portions of data which will be used for validation in each training round.
Student/QuantizationAware	true or false	whether to train the student model with quantization aware strategy or not. This is specially useful when you want to deploy the final model on a edge device that only supports *Int8* precision like Edge TPU. By applying quantization aware training the student model will be exported a *tflite* too.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search