Neuralet Edge Vision

Release 0.0.1

Neuralet

Nov 08, 2021

CONTENTS:

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 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 the 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 to train 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] [--label_map LABEL_MAP] [--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 a directory in which the script will save the output video file.

`MODEL_PATH` is the path to the model file or directory. For `x86` devices, it should be a directory that 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.

`label_map` is a `pbtxt` file which contains a series of mappings that connects a set of class IDs with the corresponding class names. For example if your detector predict 3 as the object label, with the help of label_map.pbtxt you can map this label to corresponding class. If you pass the model_path you should pass this argument too. A sample to this file can be find in utils/mscoco_label_map.pbtxt . If you use our Adaptive Learning service the model label map is exist in output.zip file

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

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

`OUT_WIDTH` and `OUT_HEIGHT` are the resolutions of output video.

CHAPTER

FOUR

ADAPTIVE LEARNING

Neuralet Adaptive Learning Service is designed to customize the object detection models to the provided datasets and specified environments by customers. For more information, please visit our blog post.

4.1 Client

There are two methods for users to call different endpoints of Adaptive Learning API. The first method uses the `curl`'s command-line tool in which the user should type raw commands, and the other more straightforward way is to use our simple Python Client interface.

For using our client interface, you only need to have python on your machine, clone our repository to your system, and go to the provided directory using the commands below.

```
git clone https://github.com/neuralet/edge-object-detection.git
cd edge-object-detection/services/adaptive-learning/client
```

Either you prefer to use curl or our interface, we are going to address a step-by-step guide to run Adaptive Learning using each of the methods in the following.

4.1.1 Step 0: Authentication

At the beginning, you need to go to our API home page and sign up using your email address and password. By clicking on the sign-up button, a verification link will be sent to your email address. After verifying your account, you can sign in to Neuralet's Adaptive Edge Vision API. Now you are able to get your token using the provided section and keep it for the next steps.

4.1.2 Step 1: Preparing Video File

After logging in and getting your token, you should upload the video file you want to feed Adaptive Learning. Since our API only accepts **zip files** as the input format, you need to create a compressed zip file from your video file, preferably using the name of `input.zip`.

```
zip -j input.zip PATH_TO_VIDEO_FILE
```

4.1.3 Step 2: Uploading Video

After creating the `input.zip` file, you are ready to upload it to Neuralet's Servers. After the uploading process, you will receive a unique id (`UUID`), which will be required later in the config file step.

You can upload your file using one of the following methods:

Client Code:

If you are using the client code, you should first save your token (provided to you in step0) as a text file and type the path to this file instead of `TOKEN_PATH` in the python command below. The `FILE_PATH` is the address to the `input.zip` file that you have created in step 1. After running the python command successfully, the unique id (`UUID`) will show up.

```
python3 client.py --token TOKEN_PATH upload_file --file_path FILE_PATH --label crowded_
↪street --video_names "elm_street.mp4,helm_street.mp4"
```

Curl Command:

If you prefer to work with the curl, you need to follow these two stages to upload your video and get your unique id.

- Get Upload URL:

  In this part, you should only copy your token (provided to you in step 0) and paste it instead of `TOKEN` in the command.

  ```
  curl -X GET "https://api.neuralet.io/api/v1/file/upload/" -H  "accept: application/
  ↪json" -H "Authorization: Bearer TOKEN"
  ```

  Note: This endpoint has 2 more optional parameters:

  1. label: Label indicates that in the future you can distinct the uploads from each other (in current situation only distinction between uploads is their ids which can be difficult to read).
  2. video_names (comma-separated names): Video names included in zip file in comma-separated format for future use in website (like `video1.mp4,video2.mp4`).

  If you intend to pass these 2 parameters your url will be like this:

  ```
  curl -X GET https://api.neuralet.io/api/v1/file/upload/?label=crowded-street&video_
  ↪names=elm_street.mp4,helm_street.mp4 -H  "accept: application/json" -H
  ↪"Authorization: Bearer TOKEN"
  ```

After running this command, it will return a json containing two items. The first key is `name`, which has your unique id (`UUID`), and you should keep it for the config step. The second key is `upload_object` which contains 2 keys: `url` and `fields`.

You can see the result in section below (save it for next part):

```
"upload_object":{
    "url":"https://neuralet-adaptive.s3.amazonaws.com/",
    "fields":{
        "key":"fdef1df4-afdb-11eb-b2c8-9a66e4b8f595/input.zip",
        "AWSAccessKeyId":"ASIA...",
        "x-amz-security-token":"IQoJb3...",
        "policy":"eyJleHBpcmF0aW9uI...",
        "signature":"M3bcjKw..."
```

(continues on next page)

(continued from previous page)

```
    }
}
```

- Upload File: In this stage, you must put the `upload_object.url` you have copied in the last part in place of UploadURL. Use data in `upload_object.fields` from last part to pass as form-data. For the `FILE_PATH`, you need to input the path to the input.zip file you have created in the first step.

```
curl "UploadURL" -F "key=UUID/input.zip" -F "AWSAccessKeyId=ASIA..." -F "x-amz-
↪security-token=IQoJb3..." -F "policy=eyJleHBpcmF0aW9uI..." -F "signature=M3bcjKw..
↪." -F "file=@FILE_PATH"
```

4.1.4 Step 3: Configure Your Training

In order to start the adaptive learning process on your uploaded file, you should tune and modify the `sample_config.json` file presented in our repository.

There are two mandatory fields in the sample_config file, which you are required to set. First, you must copy-paste your unique id (`UUID` provided to you in the previous step) in front of the `UploadUUID` field. The second field is `VideoFile`, in which you should put your video file's name against it. (Please pay attention that this is the name of your original video, e.g., `softbio.mp4`)

```
UploadUUID = Your_Unique_ID
VideoFile = Your_video_File_Name.mp4
```

In addition, there are a few more fields presented in the sample config file that you can modify based on your requirements. For example, in the config file's `Classes` field, you can choose between 90 different Object Categories of COCO's dataset by writing your desired classes' name with a comma-separated format to train your model. Notice that the default value (`coco`) will train all of the 90 object categories. You can find the 90 classes of COCO's dataset in their original research paper. Furthermore, it is possible to change `QuantizedModel` value for the Student network.

To do this, you need to adjust the sample config file on the `configs/` directory. Thus, we have prepared a brief explanation for each of the config files' parameters and options in the following table. You can also use the sample config file in `configs/sample_config.json`.

Table 1: Config File Fields

Parameter	Options	Comments
Teacher/UploadUUID	UUID	Unique id of uploaded input.zip file.
Teacher/VideoFile	string	Name of the video you zipped and uploaded.
Teacher/Classes	comma-seperated string	A list of class names that you want to train your model on. These classes should be a subset of COCO classes. You can find the COCO's category names in their original paper. To train on all of the 90 COCO classes, just put `'coco'`.
Student/QuantizedModel	true or false	whether to train the student model with quantization aware strategy or not. This is especially useful when you want to deploy the final model on an edge device that only supports `Int8` precision like Edge TPU. By applying quantization aware training the App will export a `tflite` too.

4.1.5 Step 4: Start a Training Job

Up until now, you have uploaded your video file and tuned the config file's parameters for training. Now you are ready to request to train your adaptive learning model. At the end of this step, by running the command using either the Client code or `curl`, you will get a **Job id** that you should keep for monitoring your training status in the next steps.

Client Code:

As same as the second step, you need to input the path to your token text file instead of `TOKEN_PATH` and the address of your config file in the `CONFIG_FILE` field.

```
python3 client.py --token TOKEN_PATH train --config_path CONFIG_PATH
```

Curl Command:

Again, similar to the second step, you should copy-paste the token we have provided to you at the beginning instead of `TOKEN`. Additionally, you must give the path to your config file in the `JSON_CONFIGFILE_PATH` field.

```
curl -X POST "https://api.neuralet.io/api/v1/model/train/" -H "accept: application/json"␣
↪-H "Content-Type: application/json" -H "Authorization: Bearer TOKEN" -d @JSON_
↪CONFIGFILE_PATH
```

4.1.6 Step 5: Get Job Status

At this moment, your model is training on the Neuralet's servers that may take from a few hours to a couple of days to finish based on the video length. Meanwhile, if you want to know your model's status at each moment, you are going to use this command. In this stage, you can request a job status using the **Job id** generated in the last step to observe the operation progress.

Client Code:

Enter the address to your token text file and your Job id, respectively, in the provided `TOKEN_PATH` and `JOBID` fields of the command and run it.

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

Curl Command:

You only need to repeat the previous step and copy-paste your token in the `TOKEN` field, and input your job id in the given field for `JOB_ID`.

```
curl -X POST "https://api.neuralet.io/api/v1/model/status/" -H  "accept: application/json
↪" -H  "Content-Type: application/json" -H "Authorization: Bearer TOKEN" -d "{\"job_id\
↪":\"JOB_ID\"}"
```

By running the command and sending your request to our API, you may get one of the following messages for either the Teacher or Student models each time you request for the status (Overall Status):

Table 2: Status Massages

Message	Description
Allocating Resource	We are Allocating Resources (e.g., a computing machine) to your job.
Building	We have allocated the resources, and the program is Building an environment (installing the required packages) to start your job.
Training	The Training process has started. An Adaptive Learning Job is Running.
Wrapping Up	Your training is about to finish and is Saving data and completing the job.
Finished	The job has been finished successfully.
Failed	If the process faces an infrastructural or hardware problem such as Neuralet's server failure, you will see this message.
Not Reached Yet	It usually appears as the student model's status, which means the job's workflow has not reached the student model's training phase yet. I.e., while the teacher model is running, the student model's status will be Not Reached Yet.
Unexpected Error	An internal error has occurred

Also you get more specific status such as individual status for Teacher and Student plus their progress on the job.

4.1.7 Step 6: Download your model

Finally, you have reached the final step, and the job has finished successfully. Now you can download your Adaptive Learning's trained student model. After running one of the below commands based on your preference, you will receive a file named `output.zip` that we will explain the contents in the next section.

Client Code:

As you would probably know, you should insert the address to your token file in the `TOKEN_PATH` field and replace your job id with `JOBID`, just like what you did in step five.

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

Curl Command:

If you are using the curl, there are two stages here to finally get your output file:

- Get your upload link:

  You only need to act like step five once more for replacing the `TOKEN` and `JOB_ID` fields using the token and job id you have saved before. Running this command will return an `upload_link` which you need in the next part.

  ```
  curl -X POST "https://api.neuralet.io/api/v1/file/download/" -H "accept:␣
  ↪application/json" -H "Authorization: Bearer TOKEN" -H "Content-Type: application/
  ↪json" -d "{\"job_id\":\"JOB_ID\"}"
  ```

- Download your file:

  Now by putting the `upload_link` that you have received in the previous step against the provided field and running the command, your output file's download process will start.

```
wget "upload_link" -O output.zip
```

What does the output.zip file contain?

After extracting the output.zip file in your computer, you will see the main directory of this zip file named `train_outputs`, which contains all of the Adaptive Learning files and directories. Here we will walk through the files and directories inside the `train_ouputs` and present a brief explanation of their contents.

First, we are going to introduce the most important files inside the `train_ouputs`:

`validation_vid.mp4` :

This is a video with a maximum length of 40 seconds, which compares the results of running an SSD-MobileNet-V2 model trained on COCO (Baseline model) and the Adaptive Learning trained (Student) model on a validation set video (Not used in the training process).

`label_map.pbtxt` :

This `pbtxt` file contains a series of mappings that connects a set of class IDs with the corresponding class names. To run the inference code of this module, you should pass this file to the script to classify each object with the right name.

`events.out.tfevents` :

If you want to monitor and analyze your training process, you can open this file using **TensorBoard** and observe each step of the Adaptive Learning model training process.

So far, we have introduced the most important files in the `train_outputs` directory. Now we are going to explain the contents of the `train_outputs/frozen_graph directory`.

`train_outputs/frozen_graph` :

Actually, this is the main directory of our trained model, which contains all of the required files for inferencing and exporting to the edge devices.

`train_outputs/frozen_graph/frozen_inference_graph.pb` :

For running your model on Jetson, you should pass this file to the export module that we have built for edge object detection. So it will export and create a TensorRT engine for you.

`train_outputs/frozen_graph/detect.tflite` :

If you have had set your `QuantizedModel` as `true` in the config file, this file would be available to you inside the frozen_graph directory. The importance of this file is for exporting your model to the EdgeTPU. In this case, our EdgeTPU exporter accepts this `detect.tflite` file as an input to create an edgetpu compiled tflite file.

`train_outputs/frozen_graph/saved_model` :

This is the last important directory we are introducing here. The `frozen_graph/saved_model` contains a TensorFlow `saved-model` for inferencing on X86s.

4.2 Client Management

4.2.1 Kill Job

When your model is training, you can cancel your job. In this stage, you can request a kill job using the **Job id** generated in the `Step 4: Start a Training Job`.

Client Code:

Enter the address to your token text file and your Job id, respectively, in the provided `TOKEN_PATH` and `JOBID` fields of the command and run it.

```
python3 client.py --token TOKEN_PATH kill_job --job_id JOBID
```

Curl Command:

You only need to repeat the previous step and copy-paste your token in the `TOKEN` field, and input your job id in the given field for `JOB_ID`.

```
curl -X POST "https://api.neuralet.io/api/v1/model/kill/" -H  "accept: application/json"␣
↪-H  "Content-Type: application/json" -H "Authorization: Bearer TOKEN" -d "{\"job_id\":\
↪"JOB_ID\"}"
```

4.2.2 User Jobs

Get User jobs list.

Client Code:

Enter the address to your token text file. respectively, in the provided `TOKEN_PATH` field of the command and run it.

```
python3 client.py --token TOKEN_PATH user_jobs --page 1
```

Curl Command:

You only need to repeat the previous step and copy-paste your token in the `TOKEN` field. Also you can change the page number to see other pages.

```
curl "https://api.neuralet.io/api/v1/users/me/jobs?page=1" -H "Authorization: Bearer␣
↪TOKEN"
```

Response:


```
{
    "jobs": [
        {
        "job_id": "WcLbF1VOB904wk/aMNsfU1==",
        "created_at": "2021-04-05T21:23:31.815000"
        },
        {
        "job_id": "/3I5rFqL+E4sQyskPTLNWg==",
        "created_at": "2021-03-07T16:49:41.249000"
        }
    ],
    "number_of_pages": 1,
    "current_page": 1
}
```

4.2.3 User Uploads

Get User uploads list.

Client Code:

Enter the address to your token text file. respectively, in the provided TOKEN_PATH field of the command and run it.

```
python3 client.py --token TOKEN_PATH user_uploads --page 1
```

Curl Command:

You only need to repeat the previous step and copy-paste your token in the TOKEN field. Also you can change the page number to see other pages.

```
curl "https://api.neuralet.io/api/v1/users/me/uploads?page=1" -H "Authorization: Bearer␣
↪TOKEN"
```

Response:


```
{
    "uploads": [
        {
        "name": "fdef2df4-afdb-11eb-b2c8-9a66efb8f595",
        "label": "crowded-street-number-1",
        "created_at": "2021-05-08T09:01:35.795000",
        "video_names": [
            "video1.mp4",
            "video2.mp4"
        ]
        },
        {
        "name": "5ed05020-afaa-11eb-b7cx-6ec41806e103",
        "label": "",
        "created_at": "2021-05-07T17:33:43.757000",
        "video_names": [
            "video11.mp4",
            "video12.mp4"
        ]
        }
    ],
    "number_of_pages": 1,
    "current_page": 1
}
```


4.2.4 User Info

Get User detail info.

Client Code:

Enter the address to your token text file. respectively, in the provided TOKEN_PATH field of the command and run it.

```
python3 client.py --token TOKEN_PATH user_detail
```

Curl Command:

You only need to repeat the previous step and copy-paste your token in the `TOKEN` field.

```
curl "https://api.neuralet.io/api/v1/users/me/detail" -H "Authorization: Bearer TOKEN"
```

Response:


```
{
    "email": "test@test.com",
    "is_active": true,
    "is_superuser": false,
    "is_verified": true
}
```

CHAPTER

FIVE

DEPLOY MODEL USING NVIDIA DEEPSTREAM

You can perform inference using NVIDIA Deepstream to boost your model run-time speed. You need a trained model which can be a retrained custom model downloaded from the Adaptive Learning module or a pretrained general one from the Tensorflow Model Zoo.

There have been provided two versions of Deepstream modules which are compatible with different OS module versions as follows:

dGPU model Platform and OS Compatibility:

Deepstream	OS	Cuda	Cudnn	TensorRT release	Display Driver
5.1	Ubuntu 18.04	11.1	CuDNN 8.0+	TRT 7.2.X	R460.32
5.0	Ubuntu 18.04	10.2	cuDNN 7.6.5+	TRT 7.0.0	R450.51

Jetson model Platform and OS Compatibility:

Deep-stream	Jetson Platforms	OS	Cuda	Cudnn	Jet-pack	TensorRT release
5.1	Nano, AGX Xavier, TX2, TX1, Jetson NX	L4T Ubuntu 18.04 Release 32.5.1	10.2	cuDNN 8.0.0.x	4.5.1 GA	TRT 7.1.3
5.0	Nano, AGX Xavier, TX2, TX1, Jetson NX	L4T Ubuntu 18.04 Release 32.4.3	10.2	cuDNN 8.0.0.x	4.4 GA	TRT 7.1.3

You need to change directory to the appropriate deepstream version based on your platform and OS components.

```
cd deepstream/5.0/
```

or

```
cd deepstream/5.1/
```

5.1 Run on x86:

First, you need to copy your frozen_inference_graph which can be located in the `train_outputs/frozen_graph/frozen_inference_graph.pb` path to deepstream-data directory. You can skip this step if you want to perform inference on the pretrained SSD-Mobilenet-v2-coco from the model zoo.

```
cp train_outputs/frozen_graph/frozen_inference_graph.pb deepstream-data/
```

Then, you need to prepare a label file at the next step. If you have retrained a model using the adaptive learning module, there is a *label_map.pbtxt* file in the train_outputs directory. If you have trained your model with all of the coco labels, you can skip this step as the label file will be downloaded automatically.

5.1:

```
docker build -f deepstream-51-x86.Dockerfile -t "neuralet/object-detection:deepstream5.1-
↪x86" .

docker run -it  --runtime nvidia --gpus all -e videoPath=<video_file_path> --env␣
↪labelPath= <label_map.pbtxt_file_path> -v "$PWD/../../":/repo neuralet/object-
↪detection:deepstream5.1-x86
```

5.0:

```
docker build -f deepstream-x86.Dockerfile -t "neuralet/object-detection:deepstream5.0-x86
↪" .

docker run -it --gpus all --runtime nvidia -e videoPath=<video_file_path> --env␣
↪labelPath=<label_map.pbtxt_file_path> -v "$PWD/../..":/repo  neuralet/object-
↪detection:deepstream5.0-x86
```

5.2 Run on Jetson Devices:

First you need to generate TensorRT using `exporter/trt_exporter.py` script inside `jetson-nano` docker. It will generate a `frozen_inference_graph.bin` engine file and you need to copy it inside the deepstream data directory.

Then, you need to prepare a label file at the next step. If you have retrained a model using the adaptive learning module, there is a `label_map.pbtxt` file in the train_outputs directory. If you have trained your model with all of the coco labels, you can skip this step as the label file will be downloaded automatically.

5.1:

```
docker build -f deepstream-51-jetson.Dockerfile -t "neuralet/object-detection:deepstream-
↪jetson-4-5" .
docker run --runtime nvidia -e videoPath=<path-to-video-file> --env labelPath=<label-
↪pbtxt-path> --privileged -it -v "$PWD/../../":/repo neuralet/object-
↪detection:deepstream-jetson-4-5
```

5.0

```
docker build -f deepstream-jetson.Dockerfile -t "neuralet/object-detection:deepstream-
↪jetson-4-4" .
docker run --runtime nvidia -e videoPath=<path-to-video-file> --env labelPath=<label-
↪pbtxt-path> --privileged -it -v $PWD/../..:/repo neuralet/object-detection:deepstream-
↪jetson-4-4
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search