nnio

Ruslan Baynazarov

Jun 04, 2021

TABLE OF CONTENTS

nnio is a light-weight python package for easily running neural networks.

It supports running models on CPU as well as some of the edge devices:

- Google USB Accelerator
- Intel Compute Stick
- Intel integrated GPUs

For each device there exists an own library and a model format. We wrap all those in a single well-defined python package.

Look at this simple example:

```
import nnio

# Create model and put it on a Google Coral Edge TPU device
model = nnio.EdgeTPUModel(
    model_path='path/to/model_quant_edgetpu.tflite',
    device='TPU',
)
# Create preprocessor
preproc = nnio.Preprocessing(
    resize=(224, 224),
    batch_dimension=True,
)

# Preprocess your numpy image
image = preproc(image_rgb)

# Make prediction
class_scores = model(image)
```

nnio was developed for the Fast Sense X microcomputer. It has **six neural accelerators**, which are all supported by nnio:

- 3 x Google Coral Edge TPU
- 2 x Intel Myriad VPU
- an Intel integrated GPU

CHAPTER

ONE

INSTALLATION

nnio is simply installed with pip, but it requires some additional libraries. See *Installation*.

CHAPTER

TWO

USAGE

There are 3 ways one can use nnio:

1. Loading your saved models for inference - *Basic Usage*
2. Using already prepared models from our model zoo: *Model Zoo*
3. Using our API to wrap around your own custom models. *Extending nnio*

2.1 Installation

Basic installation is simple:

```
pip install nnio
```

To use one of three backends, additional installs are needed:

2.1.1 ONNX

To work with onnx backend, install onnxruntime package:

```
pip install onnxruntime
```

2.1.2 EdgeTPU

To work with EdgeTPU models, `tflite_runtime` is required. See the installation guide: https://www.tensorflow.org/lite/guide/python.

If you intend to only use CPU inference, tensorflow installation will be enough.

2.1.3 OpenVINO

To work with OpenVINO models user needs to install openvino package. The easiest way to do it is to use `openvino/ubuntu18_runtime` docker. The following command allows to pass all Myriad and GPU devices into docker container:

```
docker run -itu root:root --rm \
-v /var/tmp:/var/tmp \
--device /dev/dri:/dev/dri --device-cgroup-rule='c 189:* rmw' \
-v /dev/bus/usb:/dev/bus/usb \
```

(continues on next page)

(continued from previous page)

```
-v /etc/timezone:/etc/timezone:ro \
-v /etc/localtime:/etc/localtime:ro \
-v "$(pwd):/input" openvino/ubuntu18_runtime
```

2.2 Basic Usage

2.2.1 Using your saved models

nnio provides three classes for loading models in different formats:

- *nnio.ONNXModel*
- *nnio.EdgeTPUModel*
- *nnio.OpenVINOModel*

Loaded models can be simply called as functions on numpy arrays. Look at the example:

```
import nnio

# Create model and put it on TPU device
model = nnio.EdgeTPUModel(
    model_path='path/to/model_quant_edgetpu.tflite',
    device='TPU:0',
)
# Create preprocessor
preproc = nnio.Preprocessing(
    resize=(224, 224),
    dtype='uint8',
    padding=True,
    batch_dimension=True,
)

# Preprocess your numpy image
image = preproc(image_rgb)

# Make prediction
class_scores = model(image)
```

See also *nnio.Preprocessing* documentation.

2.2.2 Description of the basic model classes

class nnio.**ONNXModel**(*model_path: str*)

This class is used with saved onnx models.

Usage example:

```
# Create model
model = nnio.ONNXModel('path/to/model.onnx')
# Create preprocessor
```

(continues on next page)

(continued from previous page)

```
preproc = nnio.Preprocessing(
    resize=(300, 300),
    dtype='uint8',
    batch_dimension=True,
    channels_first=True,
)

# Preprocess your numpy image
image = preproc(image_rgb)

# Make prediction
class_scores = model(image)
```

Using this class requires onnxruntime to be installed. See *Installation*.

__init__(*model_path: str*)

Parameters **model_path** – URL or path to the .onnx model

forward(**inputs*, *return_info=False*)
This method is called when the model is called.

Parameters

- ***inputs** – numpy arrays, Inputs to the model
- **return_info** – bool, If True, will return inference time

Returns numpy array or list of numpy arrays.

get_input_details()

Returns human-readable model input details.

get_output_details()

Returns human-readable model output details.

class nnio.**EdgeTPUModel**(*model_path: str*, *device='CPU'*)
This class works with tflite models on CPU and with quantized tflite models on Google Coral Edge TPU.

Using this class requires some libraries to be installed. See *Installation*.

__init__(*model_path: str*, *device='CPU'*)

Parameters

- **model_path** – URL or path to the tflite model
- **device** – str. `CPU` by default. Set `TPU` or `TPU:0` to use the first EdgeTPU device. Set `TPU:1` to use the second EdgeTPU device etc.

forward(**inputs*, *return_info=False*)
This method is called when the model is called.

Parameters

- ***inputs** – numpy arrays, Inputs to the model

- **return_info** – bool, If True, will return inference time

Returns numpy array or list of numpy arrays.

`get_input_details()`

Returns human-readable model input details.

`get_output_details()`

Returns human-readable model output details.

`property n_inputs`
number of input tensors

`property n_outputs`
number of output tensors

`class nnio.OpenVINOModel(model_bin: str, model_xml: str, device='CPU')`
This class works with OpenVINO models on CPU, Intel GPU and Intel Movidius Myriad.

Using this class requires some libraries to be installed. See *Installation*.

`__init__(model_bin: str, model_xml: str, device='CPU')`

Parameters

- **model_bin** – URL or path to the openvino binary model file
- **model_xml** – URL or path to the openvino xml model file
- **device** – str. Choose Intel device: `CPU`, `GPU`, `MYRIAD` If there are multiple devices in your system, you can use indeces: `MYRIAD:0` but it is not recommended since Intel automatically chooses a free device.

`forward(inputs, return_info=False)`

Parameters

- **inputs** – numpy array, input to the model
- **return_info** – bool, If True, will return inference time

Returns numpy array or list of numpy arrays.

2.3 Model Zoo

Table of Contents

2.3.1 Using pretrained models

Some popular models are already built in nnio. Example of using SSD MobileNet object detection model on CPU:

```
# Load model
model = nnio.zoo.onnx.detection.SSDMobileNetV1()

# Get preprocessing function
preproc = model.get_preprocessing()

# Preprocess your numpy image
image = preproc(image_rgb)

# Make prediction
boxes = model(image)
```

Here `boxes` is a list of *nnio.DetectionBox* instances.

2.3.2 ONNX

Classification

class nnio.zoo.onnx.classification.**MobileNetV2**

MobileNetV2 classifier trained on ImageNet

Model is taken from the ONNX Model Zoo.

__init__()

forward(*image*, *return_scores=False*, *return_info=False*)

Parameters

- **image** – np array. Input image
- **return_scores** – bool. If `True`, return class scores.
- **return_info** – bool. If `True`, return inference time.

Returns `str`: class label.

`get_preprocessing`()

Returns *nnio.Preprocessing* object.

`property labels`

Returns list of ImageNet classification labels

Detection

`class` nnio.zoo.onnx.detection.**SSDMobileNetV1**

SSDMobileNetV1 object detection model trained on COCO dataset.

Model is taken from the ONNX Model Zoo.

Here is the webcam demo of this model working.

`__init__`()

`forward`(*image*, *return_info=False*)

Parameters

- **image** – np array. Input image
- **return_info** – bool. If `True`, return inference time.

Returns list of *nnio.DetectionBox*

`get_preprocessing`()

Returns *nnio.Preprocessing* object.

`property labels`

Returns list of COCO labels

Re-Identification

`class` nnio.zoo.onnx.reid.**OSNet**

Omni-Scale Feature Network for Person Re-ID taken from here and converted to onnx.

Here is the webcam demo of this model working.

`__init__`()

`forward`(*image*, *return_info=False*)

Parameters

- **image** – np array. Input image of a person.
- **return_info** – bool. If `True`, return inference time.

Returns np.array of shape `[512]` - person appearence vector. You can compare them by cosine or Euclidian distance.

`get_preprocessing`()

Returns *nnio.Preprocessing* object.

2.3.3 OpenVINO

Detection

class nnio.zoo.openvino.detection.**SSDMobileNetV2**(*device='CPU'*, *lite=True*, *threshold=0.5*)
SSDMobileNetV2 object detection model trained on COCO dataset.

Model is taken from openvino and converted to openvino.

Here is the webcam demo of an analogous model (*nnio.zoo.onnx.detection.SSDMobileNetV1*) working.

__init__(*device='CPU'*, *lite=True*, *threshold=0.5*)

Parameters

- **device** – str. Choose Intel device: `CPU`, `GPU`, `MYRIAD`. If there are multiple devices in your system, you can use indeces: `MYRIAD:0` but it is not recommended since Intel automatically chooses a free device.
- **threshold** – float. Detection threshold. It affects sensitivity of the detector.
- **lite** – bool. If True, use SSDLite version (idk exactly how it is lighter).

forward(*image*, *return_info=False*)

Parameters

- **image** – np array. Input image of a person.
- **return_info** – bool. If `True`, return inference time.

Returns list of *nnio.DetectionBox*

get_preprocessing()

Returns *nnio.Preprocessing* object.

property labels

Returns list of COCO labels

Re-Identification

class nnio.zoo.openvino.reid.**OSNet**(*device='CPU'*)
Omni-Scale Feature Network for Person Re-ID taken from here and converted to openvino.

Here is the webcam demo of this model (onnx version) working.

__init__(*device='CPU'*)

Parameters **device** – str. Choose Intel device: `CPU`, `GPU`, `MYRIAD`. If there are multiple devices in your system, you can use indeces: `MYRIAD:0` but it is not recommended since Intel automatically chooses a free device.

forward(*image*, *return_info=False*)

Parameters

- **image** – np array. Input image of a person.

- **return_info** – bool. If `True`, return inference time.

Returns np.array of shape `[512]` - person appearence vector. You can compare them by cosine or Euclidian distance.

get_preprocessing()

Returns *nnio.Preprocessing* object.

2.3.4 EdgeTPU

Classification

class nnio.zoo.edgetpu.classification.**MobileNet**(*device='CPU'*, *version='v2'*)

MobileNet V2 (or V1) classifier trained on ImageNet

Model is taken from the google-coral repo

__init__(*device='CPU'*, *version='v2'*)

Parameters

- **device** – str. `CPU` by default. Set `TPU` or `TPU:0` to use the first EdgeTPU device. Set `TPU:1` to use the second EdgeTPU device etc.
- **version** – str. Either `v1` or `v2`.

forward(*image*, *return_scores=False*)

Parameters

- **image** – np array. Input image
- **return_scores** – bool. If `True`, return class scores.

Returns `str`: class label.

get_preprocessing()

Returns *nnio.Preprocessing* object.

property labels

Returns list of ImageNet classification labels

Detection

class nnio.zoo.edgetpu.detection.**SSDMobileNet**(*device='CPU'*, *version='v2'*, *threshold=0.5*)

MobileNet V2 (or V1) SSD object detector trained on COCO dataset.

Model is taken from the google-coral repo.

Here is the webcam demo of an analogous model (*nnio.zoo.onnx.detection.SSDMobileNetV1*) working.

__init__(*device='CPU'*, *version='v2'*, *threshold=0.5*)

Parameters

- **device** – str. `CPU` by default. Set `TPU` or `TPU:0` to use the first EdgeTPU device. Set `TPU:1` to use the second EdgeTPU device etc.
- **version** – str. Either "v1" or "v2"
- **threshold** – float. Detection threshold. Affects the detector's sensitivity.

forward(*image*, *return_info=False*)

Parameters

- **image** – np array. Input image
- **return_info** – bool. If `True`, return inference time.

Returns list of *nnio.DetectionBox*

get_preprocessing()

Returns *nnio.Preprocessing* object.

property labels

Returns list of COCO labels

class nnio.zoo.edgetpu.detection.**SSDMobileNetFace**(*device='CPU'*, *threshold=0.5*)

MobileNet V2 SSD face detector.

Model is taken from the google-coral repo.

__init__(*device='CPU'*, *threshold=0.5*)

Parameters

- **device** – str. `CPU` by default. Set `TPU` or `TPU:0` to use the first EdgeTPU device. Set `TPU:1` to use the second EdgeTPU device etc.
- **threshold** – float. Detection threshold. Affects the detector's sensitivity.

forward(*image*, *return_info=False*)

Parameters

- **image** – np array. Input image
- **return_info** – bool. If `True`, return inference time.

Returns list of *nnio.DetectionBox*

get_preprocessing()

Returns *nnio.Preprocessing* object.

Re-Identification

class nnio.zoo.edgetpu.reid.**OSNet**(*device='CPU'*)

Omni-Scale Feature Network for Person Re-ID taken from torchreid and converted to tflite.

This is the quantized version. It is not as accurate as its onnx and openvino versions.

Here is the webcam demo of this model (onnx version) working.

__init__(*device='CPU'*)

Parameters **device** – str. `CPU` by default. Set `TPU` or `TPU:0` to use the first EdgeTPU device. Set `TPU:1` to use the second EdgeTPU device etc.

forward(*image*, *return_info=False*)

Parameters

- **image** – np array. Input image of a person.
- **return_info** – bool. If `True`, return inference time.

Returns np.array of shape `[512]` - person appearence vector. You can compare them by cosine or Euclidian distance.

get_preprocessing()

Returns *nnio.Preprocessing* object.

Segmentation

class nnio.zoo.edgetpu.segmentation.**DeepLabV3**(*device='CPU'*)

DeepLabV3 instance segmentation model trained in Pascal VOC dataset.

Model is taken from the google-coral repo.

__init__(*device='CPU'*)

Parameters **device** – str. `CPU` by default. Set `TPU` or `TPU:0` to use the first EdgeTPU device. Set `TPU:1` to use the second EdgeTPU device etc.

forward(*image*)

Parameters **image** – np array. Input image

Returns numpy array. Segmentation map of the same size as the input image: `shape=[batch, 513, 513]`. For each pixel gives an integer denoting class. Class labels are available through `.labels` attribute of this object.

get_preprocessing()

Returns *nnio.Preprocessing* object.

property labels

Returns list of Pascal VOC labels

2.4 Utils

2.4.1 nnio.Preprocessing

class nnio.**Preprocessing**(*resize=None*, *dtype='uint8'*, *divide_by_255=False*, *means=None*, *stds=None*, *scales=None*, *padding=False*, *channels_first=False*, *batch_dimension=False*, *bgr=False*)

This class provides functionality of the image preprocessing.

Example:

```
preproc = nnio.Preprocessing(
    resize=(224, 224),
    dtype='float32',
    divide_by_255=True,
    means=[0.485, 0.456, 0.406],
    stds=[0.229, 0.224, 0.225],
    batch_dimension=True,
    channels_first=True,
)

# Use with numpy image
image_preprocessed = preproc(image_rgb)

# Or use to read image from disk
image_preprocessed = preproc('path/to/image.png')

# Or use to read image from the web
image_preprocessed = preproc('http://www.example.com/image.png')
```

Object of this type is returned every time you call `get_preprocessing()` method of any model from *Model Zoo*.

__eq__(*other*)

Compare two `Preprocessing` objects. Returns `True` only if all preprocessing parameters are the same.

__init__(*resize=None*, *dtype='uint8'*, *divide_by_255=False*, *means=None*, *stds=None*, *scales=None*, *padding=False*, *channels_first=False*, *batch_dimension=False*, *bgr=False*)

Parameters

- **resize** – `None` or `tuple`. (width, height) - the new size of image
- **dtype** – `str` or `np.dtype`. Data type
- **divide_by_255** – `bool`. Divide input image by 255. This is applied before `means`, `stds` and `scales`.
- **means** – `float` or iterable or `None`. Substract these values from each channel
- **stds** – *float* ` or iterable or `None`. Divide each channel by these values
- **scales** – `float` or iterable or `None`. Multipy each channel by these values
- **padding** – `bool`. If `True`, images will be resized with the same aspect ratio
- **channels_first** – `bool`. If `True`, image will be returned in `[B]CHW` format. If `False`, `[B]HWC`.

- **batch_dimension** – bool. If True, add first dimension of size 1.
- **bgr** – bool. If True, change channels to BRG order. If False, keep the RGB order.

__str__()

Returns full description of the Preprocessing object

forward(*image*, *return_original=False*)
Preprocess the image.

Parameters

- **image** – np.ndarray of type uint8 or str RGB image If str, it will be concerned as image path.
- **return_original** – bool. If True, will return tuple of (preprocessed_image, original_image)

2.4.2 nnio.DetectionBox

class nnio.**DetectionBox**(*x_min*, *y_min*, *x_max*, *y_max*, *label=None*, *score=1.0*)

__init__(*x_min*, *y_min*, *x_max*, *y_max*, *label=None*, *score=1.0*)

Parameters

- **x_min** – float in range [0, 1]. Relative x (width) coordinate of top-left corner.
- **y_min** – float in range [0, 1]. Relative y (height) coordinate of top-left corner.
- **x_max** – float in range [0, 1]. Relative x (width) coordinate of bottom-right corner.
- **y_max** – float in range [0, 1]. Relative y (height) coordinate of bottom-right corner.
- **label** – str or None. Class label of the detected object.
- **score** – float. Detection score

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

draw(*image*, *color=(255, 0, 0)*, *stroke_width=2*, *text_color=(255, 0, 0)*, *text_width=2*)
Draws the detection box on an image

Parameters

- **image** – numpy array.
- **color** – RGB color of the frame.
- **stroke_width** – boldness of the frame.
- **text_color** – RGB color of the text.
- **text_width** – boldness of the text.

Returns Image with the box drawn on it.

2.5 Extending nnio

2.5.1 Using our API to wrap around your own custom models

nnio.Model is an abstract class from which all models in nnio are derived. It is easy to use by redefining `forward` method:

```
class MyClassifier(nnio.Model):
    def __init__(self):
        super().__init__()
        self.model = SomeModel()

    def forward(self, image):
        # Do something with image
        result = self.model(image)
        # For example, classification
        if result == 0:
            return 'person'
        else:
            return 'cat'

    def get_preprocessing(self):
        return nnio.Preprocessing(
            resize=(224, 224),
            dtype='float',
            divide_by_255=True,
            means=[0.485, 0.456, 0.406],
            stds=[0.229, 0.224, 0.225],
            batch_dimension=True,
            channels_first=True,
        )
```

We also recommend to define `get_preprocessing` method like in *Model Zoo* models. See *nnio.Preprocessing*. We encourage users to wrap their loaded models in such classes. *nnio.Model* abstract base class is described below:

2.5.2 nnio.Model

class nnio.**Model**

abstract forward(**args*, ***kwargs*)

This method is called when the model is called.

Parameters

- ***inputs** – numpy arrays, Inputs to the model
- **return_info** – bool, If True, will return inference time

Returns numpy array or list of numpy arrays.

get_input_details()

Returns human-readable model input details.

`get_output_details()`

Returns human-readable model output details.

`get_preprocessing()`

Returns *nnio.Preprocessing* object.

INDEX

Symbols

D

E

F

G

L

M

N

O

P

S