DenseTorch

Jul 21, 2020

Contents

CHAPTER 1

DenseTorch: PyTorch Wrapper for Smooth Workflow with Dense Per-Pixel Tasks

Build Status Docs Status

This library aims to ease typical workflows involving dense per-pixel tasks in PyTorch. The progress in such tasks as semantic image segmentation and depth estimation have been significant over the last years, and in this library we provide an easy-to-setup environment for experimenting with given (or your own) models that reliably solve these tasks.

1.1 Installation

Python >= 3.6.7 is supported.

```
git clone https://github.com/drsleep/densetorch.git
cd densetorch
pip install -e .
```

1.2 Examples

Currently, we provide several models for single-task and multi-task setups:

- `resnet` ResNet-18/34/50/101/152.
- `mobilenet-v2` MobileNet-v2.
- `xception-65` Xception-65.
- `deeplab-v3+` DeepLab-v3+.
- `lwrf` Light-Weight RefineNet.
- `mtlwrf` Multi-Task Light-Weight RefineNet.

Examples are given in the `examples/` directory. Note that the provided examples do not necessarily reproduce the results achieved in corresponding papers, rather their goal is to demonstrate what can be done using this library.

1.3 Motivation behind the library

As my everyday research is concerned with dense per-pixel tasks, I found myself oftentimes re-writing and updating (occassionally improving upon) my own code for each project. With the number of projects being on the rise recently, such an approach was no longer easy to manage. Hence, I decided to create a simple to use and simple to extend upon backbone (pun is not intended) structure, which I would be able to share with the community and, hopefully, ease the experience for others in the field.

1.4 Future Work

This library is still work-in-progress. More examples and more models will be added. Contributions are welcome.

1.5 Documentation

Is available here.

1.6 Citation

If you found this library useful in your research, please consider citing

```
@misc{Nekrasov19,
  author = {Nekrasov, Vladimir},
  title = {DenseTorch},
  year = {2019},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/drsleep/densetorch}}
}
```

CHAPTER 2

Provided Models

Each model in DenseTorch is an encoder-decoder network, hence we provide several different encoders and decoders. The library supports various use cases where certain layers in the encoder can be marked as output layers and, consequently, as input layers to the decoder. Such a support is provided via a `return_layers` keyword argument when creating the encoder. Additionally, each encoder has the `info` property which is a dictionary with the information on the number of output channels and, perhaps, some additional entries specific to a concrete model.

2.1 Encoders

The following encoders are currently available:

1. ***ResNet-family*** (ResNet-18, 34, 50, 101, 152). Each encoder from this family has 4 various output layers with resolutions equal to 1/4, 1/8, 1/16 and 1/32, respectively. Hence, all the values passed in the `return_layers` argument must be strictly less than 4.
2. ***MobileNet-v2***. This encoder has 7 various output layers with resolutions equal to 1/2, 1/4, 1/8, 1/16, 1/16, 1/32, 1/32. All the values passed in the `return_layers` argument must be strictly less than 7.
3. ***Xception-65***. The encoder has 21 output layers with the resolutions of 1/4, 1/8 for the rest until 1/16 and 1/32. All the values passed in the `return_layers` argument must be strictly less than 21.

2.2 Decoders

Each decoder takes one or more layers with non-decreasing spatial resolutions and progressively merges them in a single set of feature maps with the highest resolution among the inputs. The following decoders are provided:

1. ***Multi-Task Light-Weight RefineNet***. This decoder only applies 1x1 convolutions followed by chained residual pooling blocks. Supports merging various combinations of input layers into a single layer – the only constraint is that the layers that are to be merged must have the same spatial dimensions; the relevant keyword argument is named `combine_layers`. When designing a specific encoder-decoder network, it is important to understand how the `combine_layers` and `return_layers` arguments interact with each other. For example, if a

given network produces 3 outputs and `return_layers` is set to `[1, 2]`, the outputs are zero-indexed and become `[0, 1]`, hence no index in the `combine_layers` can exceed 1.

2. ***Multi-Task DeepLab-v3+***. This decoder applies atrous spatial pyramid pooling layer together with several separable convolutions. Supports multiple skip-connections, does not support `combine_layers`.

2.3 Typical Models

CHAPTER 3

Code Documentation

3.1 densetorch.nn

The *nn* module implements a range of well-established encoders and decoders.

class densetorch.nn.decoders.**DLv3plus**(*input_sizes*, *num_classes*, *skip_size=48*, *agg_size=256*, *rates=(6, 12, 18)*, ***kwargs*)

DeepLab-v3+ for Semantic Image Segmentation.

ASPP with decoder. Allows to have multiple skip-connections. More information about the model: https://arxiv.org/abs/1802.02611

Parameters

- **input_sizes** (*int, or list*) – number of channels for each input. Last value represents the input to ASPP, other values are for skip-connections.
- **num_classes** (*int*) – number of output channels.
- **skip_size** (*int*) – common filter size for skip-connections.
- **agg_size** (*int*) – common filter size.
- **rates** (*list of ints*) – dilation rates in the ASPP module.

forward(*xs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class densetorch.nn.decoders.**LWRefineNet**(*input_sizes*, *combine_layers*, *num_classes*, *agg_size=256*, *n_crp=4*)

Light-Weight RefineNet for Semantic Image Segmentation.

More information about the model: https://arxiv.org/abs/1810.03272

Parameters

- **input_sizes** (*int, or list*) – number of channels for each input.
- **combine_layers** (*list*) – which input layers should be united together (via element-wise summation) before CRP.
- **num_classes** (*int*) – number of output channels.
- **agg_size** (*int*) – common filter size.
- **n_crp** (*int*) – number of CRP layers in a single CRP block.

forward(*xs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class densetorch.nn.decoders.**MTDLv3plus**(*input_sizes*, *num_classes*, *skip_size=48*, *agg_size=256*, *rates=(6, 12, 18)*, ***kwargs*)

Multi-Task DeepLab-v3+ for Semantic Image Segmentation.

ASPP with decoder. Allows to have multiple skip-connections. More information about the model: https://arxiv.org/abs/1802.02611

Parameters

- **input_sizes** (*int, or list*) – number of channels for each input. Last value represents the input to ASPP, other values are for skip-connections.
- **num_classes** (*int*) – number of output channels.
- **skip_size** (*int*) – common filter size for skip-connections.
- **agg_size** (*int*) – common filter size.
- **rates** (*list of ints*) – dilation rates in the ASPP module.

forward(*xs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class densetorch.nn.decoders.**MTLWRefineNet**(*input_sizes*, *combine_layers*, *num_classes*, *agg_size=256*, *n_crp=4*, ***kwargs*)

Multi-Task Light-Weight RefineNet for Dense per-pixel tasks.

More information about the model: https://arxiv.org/abs/1809.04766

Parameters

- **input_sizes** (*int, or list*) – number of channels for each input.

- **combine_layers** (*list*) – which input layers should be united together (via element-wise summation) before CRP.
- **num_classes** (*int or list*) – number of output channels per each head.
- **agg_size** (*int*) – common filter size.
- **n_crp** (*int*) – number of CRP layers in a single CRP block.

forward(*xs*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

`densetorch.nn.mobilenetv2.`**mobilenetv2**(*pretrained=True*, ***kwargs*)
Constructs the mobilenet-v2 network.

Parameters **pretrained** (*bool*) – whether to load pre-trained weights.

Returns *nn.Module* instance.

`densetorch.nn.resnet.`**resnet18**(*pretrained=False*, ***kwargs*)
Constructs the ResNet-18 model.

Parameters **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet.

Returns *nn.Module* instance.

`densetorch.nn.resnet.`**resnet34**(*pretrained=False*, ***kwargs*)
Constructs the ResNet-34 model.

Parameters **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet.

Returns *nn.Module* instance.

`densetorch.nn.resnet.`**resnet50**(*pretrained=False*, ***kwargs*)
Constructs the ResNet-50 model.

Parameters **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet.

Returns *nn.Module* instance.

`densetorch.nn.resnet.`**resnet101**(*pretrained=False*, ***kwargs*)
Constructs the ResNet-101 model.

Parameters **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet.

Returns *nn.Module* instance.

`densetorch.nn.resnet.`**resnet152**(*pretrained=False*, ***kwargs*)
Constructs the ResNet-152 model.

Parameters **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet.

Returns *nn.Module* instance.

`densetorch.nn.xception.`**xception65**(*pretrained=False*, ***kwargs*)
Constructs the Xception-65 network.

Parameters **pretrained** (*bool*) – whether to load pre-trained weights.

Returns *nn.Module* instance.

3.2 densetorch.engine

The *engine* module contains metrics and losses typically used for the tasks of semantic segmentation and depth estimation. Also contains training and validation functions.

class `densetorch.engine.losses.`**`CosineDistanceLoss`**(*ignore_index=0*, *dim=1*)

Cosine Distance Loss for 3D surface normals estimation.

Parameters **`ignore_index`** (*`float`*) – value to ignore in the target when computing the loss.

`forward`(*x*, *target*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class `densetorch.engine.losses.`**`InvHuberLoss`**(*ignore_index=0*)

Inverse Huber Loss for depth estimation.

The setup is taken from https://arxiv.org/abs/1606.00373

Parameters **`ignore_index`** (*`float`*) – value to ignore in the target when computing the loss.

`forward`(*x*, *target*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class `densetorch.engine.metrics.`**`AngularError`**(*ignore_index=0*)

AngularError computational block for 3D surface normals estimation.

Parameters **`ignore_index`** (*`float`*) – value to ignore in the target when computing the metric.

`name`

descriptor of the estimator.

Type str

class `densetorch.engine.metrics.`**`MeanIoU`**(*num_classes*)

Mean-IoU computational block for semantic segmentation.

Parameters **`num_classes`** (*`int`*) – number of classes to evaluate.

`name`

descriptor of the estimator.

Type str

class densetorch.engine.metrics.**RMSE**(*ignore_index=0*)
Root Mean Squared Error computational block for depth estimation.

Parameters **ignore_index** (*float*) – value to ignore in the target when computing the metric.

name
descriptor of the estimator.

Type str

densetorch.engine.trainval.**maybe_cast_target_to_long**(*target*)
Torch losses usually work on Long types

densetorch.engine.trainval.**train**(*model*, *opts*, *crits*, *dataloader*, *loss_coeffs=(1.0,)*, *freeze_bn=False*, *grad_norm=0.0*)
Full Training Pipeline.

Supports multiple optimisers, multiple criteria, multiple losses, multiple outputs. Assumes that the model.eval() property has been set up properly before the function call, that the dataloader outputs have the correct type, that the model outputs do not require any post-processing bar the upsampling to the target size. Criteria, loss_coeff, and model's outputs all must have the same length, and correspond to the same keys as in the ordered dict of dataloader's sample.

Parameters

- **model** – PyTorch model object.
- **opts** – list of optimisers.
- **crits** – list of criterions.
- **dataloader** – iterable over samples. Each sample must contain *image* key and >= 1 optional keys.
- **loss_coeffs** – list of coefficients for each loss term.
- **freeze_bn** – whether to freeze batch norm parameters in the module.
- **grad_norm** – if > 0, clip gradients' norm to this value.

densetorch.engine.trainval.**trainbal**(*model*, *dataloader*)
Full Training Pipeline with balanced model.

Assumes that the model.eval() property has been set up properly before the function call, that the dataloader outputs have the correct type, that the model outputs do not require any post-processing bar the upsampling to the target size.

Parameters

- **model** – PyTorch model object.
- **dataloader** – iterable over samples. Each sample must contain *image* key and >= 1 optional keys.

densetorch.engine.trainval.**validate**(*model*, *metrics*, *dataloader*)
Full Validation Pipeline.

Support multiple metrics (but 1 per modality), multiple outputs. Assumes that the dataloader outputs have the correct type, that the model outputs do not require any post-processing bar the upsampling to the target size. Metrics and model's outputs must have the same length, and correspond to the same keys as in the ordered dict of dataloader's sample.

Parameters

- **model** – PyTorch model object.

- **metrics** – list of metric classes. Each metric class must have update and val functions, and must have 'name' attribute.
- **dataloader** – iterable over samples. Each sample must contain *image* key and >= 1 optional keys.

3.3 densetorch.data

The *data* module implements datasets and relevant utilities used for data pre-processing. It supports multi-modal data.

class densetorch.data.datasets.**MMDataset**(*data_file*, *data_dir*, *data_list_sep*, *data_list_columns*, *masks_names*, *ignore_indices*, *depth_scale=1.0*, *transform=None*)

Multi-Modality dataset.

Works with any datasets that contain image and any number of 2D-annotations.

Parameters

- **data_file** (*string*) – Path to the data file with annotations.
- **data_dir** (*string*) – Directory with all the images.
- **data_list_sep** (*string*) – Separator between the columns in data_file.
- **data_list_columns** (*list of strings*) – Column names in data_file.
- **masks_names** (*list of strings*) – Keys for each annotation mask (e.g., 'segm', 'depth').
- **ignore_indices** (*list of floats or ints*) – Ignore values for each annotation mask in the same order as *masks_names*.
- **depth_scale** (*float*) – Scaling factor for depth.
- **transform** (*callable, optional*) – Optional transform to be applied on a sample.

Raises ValueError – if the number of columns in data_file is not equal to *len(data_list_columns)*.

static read_image(*x*)

Simple image reader

Parameters x (*str*) – path to image.

Returns image as *np.array*.

class densetorch.data.utils.**Normalise**(*scale*, *mean*, *std*, *depth_scale=1.0*)

Normalise a tensor image with mean and standard deviation. Given mean: (R, G, B) and std: (R, G, B), will normalise each channel of the torch.*Tensor, i.e. channel = (scale * channel - mean) / std

Parameters

- **scale** (*float*) – Scaling constant.
- **mean** (*sequence*) – Sequence of means for R,G,B channels respecitvely.
- **std** (*sequence*) – Sequence of standard deviations for R,G,B channels respecitvely.
- **depth_scale** (*float*) – Depth divisor for depth annotations.

class densetorch.data.utils.**Pad**(*size*, *img_val*, *msk_vals*)

Pad image and mask to the desired size.

Parameters

- **`size`** (*`int`*) – minimum length/width.
- **`img_val`** (*`array`*) – image padding value.
- **`msk_vals`** (*`list of ints`*) – masks padding value.

`class` `densetorch.data.utils.`**`RandomCrop`**(*crop_size*)
Crop randomly the image in a sample.

Parameters **`crop_size`** (*`int`*) – Desired output size.

`class` `densetorch.data.utils.`**`RandomMirror`**
Randomly flip the image and the mask

`class` `densetorch.data.utils.`**`ResizeAndScale`**(*side*, *low_scale*, *high_scale*, *shorter=True*)
Resize shorter/longer side to a given value and randomly scale.

Parameters

- **`side`** (*`int`*) – shorter / longer side value.
- **`low_scale`** (*`float`*) – lower scaling bound.
- **`high_scale`** (*`float`*) – upper scaling bound.
- **`shorter`** (*`bool`*) – whether to resize shorter / longer side.

`class` `densetorch.data.utils.`**`ToTensor`**
Convert ndarrays in sample to Tensors.

`densetorch.data.utils.`**`albumentations2densetorch`**(*augmentation*)
Wrapper to use Albumentations within DenseTorch dataset.

Parameters **`augmentation`** – either a list of augmentations or a single augmentation

Returns A composition of augmentations

`densetorch.data.utils.`**`denormalise`**(*tensor_bchw*, *scale*, *mean_c*, *std_c*)
Reversed normalisation

Parameters

- **`tensor_bchw`** (*`torch.tensor`*) – 4D tensor of shape BxCxHxW
- **`scale`** (*`float`*) – scale value
- **`mean_c`** (*`np.ndarray`*) – mean array of shape (C,)
- **`std_c`** (*`np.ndarray`*) – standard deviation array of shape (C,)

Returns Un-normalised torch tensor.

`densetorch.data.utils.`**`densetorch2torchvision`**(*augmentation*)
Wrapper to use DenseTorch augmentations within torchvision dataset.

Parameters **`augmentation`** – either a list of augmentations or a single augmentation

Returns A composition of augmentations.

`densetorch.data.utils.`**`get_loaders`**(*train_batch_size*, *val_batch_size*, *train_set*, *val_set*, *num_stages=1*, *num_workers=8*, *train_shuffle=True*, *val_shuffle=False*, *train_pin_memory=False*, *val_pin_memory=False*, *train_drop_last=False*, *val_drop_last=False*)
Create train and val loaders

3.4 densetorch.misc

The *misc* module has various useful utilities.

class `densetorch.misc.utils.`**`AverageMeter`**(*momentum=0.99*)

Simple running average estimator.

Parameters **`momentum`** (*`float`*) – running average decay.

`update`(*val*)

Update running average given a new value.

The new running average estimate is given as a weighted combination of the previous estimate and the current value.

Parameters **`val`** (*`float`*) – new value

class `densetorch.misc.utils.`**`Balancer`**(*model*, *opts*, *crits*, *loss_coeffs*)

Wrapper for balanced multi-GPU training.

When forward and backward passes are fused into a single nn.Module object, the multi-GPU consumption is distributed more equally across the GPUs.

Parameters

- **`model`** (*`nn.Module`*) – PyTorch module.
- **`opts`** (*`list or single instance of torch.optim`*) – optimisers.
- **`crits`** (*`list or single instance of torch.nn or nn.Module`*) – criterions.
- **`loss_coeffs`** (*`list of single instance of float`*) – loss coefficients.

`forward`(*inp*, *targets=None*)

Forward and (optionally) backward pass.

When targets are provided, the backward pass is performed. Otherwise only the forward pass is done.

Parameters

- **`inp`** (*`torch.tensor`*) – input batch.
- **`targets`** (*`None or torch.tensor`*) – targets batch.

Returns Forward output if *targets=None*, else returns the loss value.

class `densetorch.misc.utils.`**`Saver`**(*args*, *ckpt_dir*, *best_val=0*, *condition=<function Saver.<lambda>>*, *save_interval=100*, *save_several_mode=<built-in function any>*)

Saver class for checkpointing the training progress.

`maybe_load`(*ckpt_path*, *keys_to_load*)

Loads existing checkpoint if exists.

Parameters

- **`ckpt_path`** (*`str`*) – path to the checkpoint.
- **`keys_to_load`** (*`list of str`*) – keys to load from the checkpoint.

Returns the epoch at which the checkpoint was saved.

`maybe_save`(*new_val*, *dict_to_save*)

Maybe save new checkpoint

`densetorch.misc.utils.`**`broadcast`**(*x*, *num_times*)

Given an element, broadcast it number of times and return a list. If it is already a list, only the first element will be copied.

Parameters

- **`x`** – input.
- **`num_times`** (*`int`*) – how many times to copy the element.

Returns list of length num_times.

`densetorch.misc.utils.`**`compute_params`**(*model*)

Compute the total number of parameters.

Parameters **`model`** (*`nn.Module`*) – PyTorch model.

Returns Total number of parameters - both trainable and non-trainable (int).

`densetorch.misc.utils.`**`create_optim`**(*optim_type*, *parameters*, ***kwargs*)

Initialise optimisers.

Parameters

- **`optim_type`** (*`string`*) – type of optimiser - either 'SGD' or 'Adam'.
- **`parameters`** (*`iterable`*) – parameters to be optimised.

Returns An instance of torch.optim.

Raises ValueError if optim_type is not either of 'SGD' or 'Adam'.

`densetorch.misc.utils.`**`create_scheduler`**(*scheduler_type*, *optim*, ***kwargs*)

Initialise schedulers.

Parameters

- **`scheduler_type`** (*`string`*) – type of scheduler – either 'poly' or 'multistep'.
- **`optim`** (*`torch.optim`*) – optimiser to which the scheduler will be applied to.

Returns An instance of torch.optim.lr_scheduler

Raises ValueError if scheduler_type is not either of 'poly' or 'multistep'.

`densetorch.misc.utils.`**`ctime`**()

Returns current timestamp in the format of hours-minutes-seconds.

`densetorch.misc.utils.`**`get_args`**(*func*)

Get function's arguments.

Parameters **`func`** (*`callable`*) – input function.

Returns List of positional and keyword arguments.

`densetorch.misc.utils.`**`make_list`**(*x*)

Returns the given input as a list.

`densetorch.misc.utils.`**`polyschedule`**(*max_epochs*, *gamma=0.9*)

Poly-learning rate policy popularised by DeepLab-v2: https://arxiv.org/abs/1606.00915

Parameters

- **`max_epochs`** (*`int`*) – maximum number of epochs, at which the multiplier becomes zero.
- **`gamma`** (*`float`*) – decay factor.

Returns Callable that takes the current epoch as an argument and returns the learning rate multiplier.

`densetorch.misc.utils.`**`set_seed`**`(`*seed*`)`

Setting the random seed across *torch*, *numpy* and *random* libraries.

Parameters **`seed`** (*`int`*) – random seed value.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index