TorchIO will be featured at the PyTorch Developer Day 2021 (December 1st & 2nd)

TorchIO

Search

Inference

Here is an example that uses a grid sampler and aggregator to perform dense inference across a 3D image using patches:

```
>>> import torch
>>> import torch.nn as nn
>>> import torchio as tio
>>> patch_overlap = 4, 4, 4  # or just 4
>>> patch_size = 88, 88, 60
>>> subject = tio.datasets.Colin27()
>>> subject
Colin27(Keys: ('t1', 'head', 'brain'); images: 3)
>>> grid_sampler = tio.inference.GridSampler(
...     subject,
...     patch_size,
...     patch_overlap,
... )
>>> patch_loader = torch.utils.data.DataLoader(grid_sampler, batch_size=4)
>>> aggregator = tio.inference.GridAggregator(grid_sampler)
>>> model = nn.Identity().eval()
>>> with torch.no_grad():
...     for patches_batch in patch_loader:
...         input_tensor = patches_batch['t1'][tio.DATA]
...         locations = patches_batch[tio.LOCATION]
...         logits = model(input_tensor)
...         labels = logits.argmax(dim=tio.CHANNELS_DIMENSION, keepdim=True)
...         outputs = labels
...         aggregator.add_batch(outputs, locations)
>>> output_tensor = aggregator.get_output_tensor()
```

Grid sampler

`GridSampler`

```
class torchio.data.GridSampler(subject: Optional[torchio.data.subject.Subject] =
    None, patch_size: Optional[Union[int, Tuple[int, int, int]]] = None,
    patch_overlap: Union[int, Tuple[int, int, int]] = (0, 0, 0), padding_mode:
    Optional[Union[str, float]] = None)                                    [source]
```

Bases: `torchio.data.sampler.sampler.PatchSampler`

Extract patches across a whole volume.

Grid samplers are useful to perform inference using all patches from a volume. It is often used with a `GridAggregator`.

PARAMETERS

- **subject** – Instance of `Subject` from which patches will be extracted. This argument should only be used before instantiating a `GridAggregator`, or to precompute the number of patches that would be generated from a subject.
- **patch_size** – Tuple of integers (w, h, d) to generate patches of size $w \times h \times d$. If a single number n is provided, $w = h = d = n$. This argument is mandatory (it is a keyword argument for backward compatibility).
- **patch_overlap** – Tuple of even integers (w_o, h_o, d_o) specifying the overlap between patches for dense inference. If a single number n is provided, $w_o = h_o = d_o = n$.
- **padding_mode** – Same as `padding_mode` in `Pad`. If `None`, the volume will not be padded before sampling and patches at the border will not be cropped by the aggregator. Otherwise, the volume will be padded with $\left(\frac{w_o}{2}, \frac{h_o}{2}, \frac{d_o}{2}\right)$ on each side before sampling. If the sampler is passed to a `GridAggregator`, it will crop the output to its original size.

Example:

```
>>> import torchio as tio
>>> sampler = tio.GridSampler(patch_size=88)
>>> colin = tio.datasets.Colin27()
>>> for i, patch in enumerate(sampler(colin)):
...     patch.t1.save(f'patch_{i}.nii.gz')
...
>>> # To figure out the number of patches beforehand:
>>> sampler = tio.GridSampler(subject=colin, patch_size=88)
>>> len(sampler)
8
```

Note

Adapted from NiftyNet. See this NiftyNet tutorial for more information about patch based sampling. Note that `patch_overlap` is twice `border` in NiftyNet tutorial.

Grid aggregator

`GridAggregator`

```
class torchio.data.GridAggregator(sampler: torchio.data.sampler.grid.GridSampler,
    overlap_mode: str = 'crop')                                             [source]
```

Aggregate patches for dense inference.

This class is typically used to build a volume made of patches after inference of batches extracted by a `GridSampler`.

PARAMETERS

- **sampler** – Instance of `GridSampler` used to extract the patches.
- **overlap_mode** – If `'crop'`, the overlapping predictions will be cropped. If `'average'`, the predictions in the overlapping areas will be averaged with equal weights. See the grid aggregator tests for a raw visualization of both modes.

Note

Adapted from NiftyNet. See this NiftyNet tutorial for more information about patch-based sampling.

`add_batch(batch_tensor: torch.Tensor, locations: torch.Tensor) → None` [source]

Add batch processed by a CNN to the output prediction volume.

PARAMETERS

- **batch_tensor** – 5D tensor, typically the output of a convolutional neural network, e.g. `batch['image'][torchio.DATA]`.
- **locations** – 2D tensor with shape $(B, 6)$ representing the patch indices in the original image. They are typically extracted using `batch[torchio.LOCATION]`.

`get_output_tensor() → torch.Tensor` [source]

Get the aggregated volume after dense inference.

v: latest