Sliding Window Inference#

Sliding window inference on inputs with predictor. The outputs of predictor could be a tensor, a tuple, or a dictionary of tensors. Each output in the tuple or dict value is allowed to have different resolutions with respect to the input. e.g., the input patch spatial size is [128,128,128], the output (a tuple of two patches) patch sizes could be ([128,64,256], [64,32,128]). In this case, the parameter overlap and roi_size need to be carefully chosen to ensure the output ROI is still an integer. If the predictor's input and output spatial sizes are not equal, we recommend choosing the parameters so that overlap*roi_size*output_size/input_size is an integer (for each spatial dimension). When roi_size is larger than the inputs' spatial size, the input image are padded during inference. To maintain the same spatial sizes, the output image will be cropped to the original input size. inputs (Tensor) – input image to be processed (assuming NCHW[D]) roi_size (Union[Sequence[int], int]) – the spatial window size for inferences. When its components have None or non-positives, the corresponding inputs dimension will be used. if the components of the roi_size are non-positive values, the transform will use the corresponding components of img size. For example, roi_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. sw_batch_size (int) – the batch size to run window slices. predictor (Callable[…, Union[Tensor, Sequence[Tensor], Dict[Any, Tensor]]]) – given input tensor patch_data in shape NCHW[D], The outputs of the function call predictor(patch_data) should be a tensor, a tuple, or a dictionary with Tensor values. Each output in the tuple or dict value should have the same batch_size, i.e. NM'H'W'[D']; where H'W'[D'] represents the output patch's spatial size, M is the number of output channels, N is sw_batch_size, e.g., the input shape is (7, 1, 128,128,128), the output could be a tuple of two tensors, with shapes: ((7, 5, 128, 64, 256), (7, 4, 64, 32, 128)). In this case, the parameter overlap and roi_size need to be carefully chosen to ensure the scaled output ROI sizes are still integers. If the predictor's input and output spatial sizes are different, we recommend choosing the parameters so that overlap*roi_size*zoom_scale is an integer for each dimension. overlap (float) – Amount of overlap between scans. mode (Union[BlendMode, str]) – {"constant", "gaussian"} How to blend output of overlapping windows. Defaults to "constant". "constant": gives equal weight to all predictions. "gaussian": gives less weight to predictions on edges of windows. {"constant", "gaussian"} How to blend output of overlapping windows. Defaults to "constant". "constant": gives equal weight to all predictions. "gaussian": gives less weight to predictions on edges of windows. sigma_scale (Union[Sequence[float], float]) – the standard deviation coefficient of the Gaussian window when mode is "gaussian". Default: 0.125. Actual window sigma is sigma_scale * dim_size. When sigma_scale is a sequence of floats, the values denote sigma_scale at the corresponding spatial dimensions. padding_mode (Union[PytorchPadMode, str]) – {"constant", "reflect", "replicate", "circular"} Padding mode for inputs, when roi_size is larger than inputs. Defaults to "constant" See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html cval (float) – fill value for 'constant' padding mode. Default: 0 sw_device (Union[str, device, None]) – device for the window data. By default the device (and accordingly the memory) of the inputs is used. Normally sw_device should be consistent with the device where predictor is defined. device (Union[str, device, None]) – device for the stitched output prediction. By default the device (and accordingly the memory) of the inputs is used. If for example set to device=torch.device('cpu') the gpu memory consumption is less and independent of the inputs and roi_size. Output is on the device. progress (bool) – whether to print a tqdm progress bar. roi_weight_map (Optional[Tensor]) – pre-computed (non-negative) weight map for each ROI. If not given, and mode is not constant, this map will be computed on the fly. process_fn (Optional[Callable]) –

process inference output and adjust the importance map per window args (Any) – optional args to be passed to predictor. kwargs (Any) – optional keyword args to be passed to predictor. Note input must be channel-first and have a batch dim, supports N-D sliding window. Union[Tensor, Tuple[Tensor, …], Dict[Any, Tensor]]

Inferers#

A base class for model inference. Extend this class to support operations during inference, e.g. a sliding window method. Example code: Run inference on inputs with the network model. inputs (Tensor) – input of the model inference. network (Callable[…, Tensor]) – model for inference. args (Any) – optional args to be passed to network. kwargs (Any) – optional keyword args to be passed to network. NotImplementedError – When the subclass does not override this method.

SimpleInferer#

SimpleInferer is the normal inference method that run model forward() directly. Usage example can be found in the monai.inferers.Inferer base class. Unified callable function API of Inferers. inputs (Tensor) – model input data for inference. network (Callable[…, Tensor]) – target model to execute inference. supports callables such as lambda x: my_torch_model(x, additional_config) args (Any) – optional args to be passed to network. kwargs (Any) – optional keyword args to be passed to network.

SlidingWindowInferer#

Sliding window method for model inference, with sw_batch_size windows for every model.forward(). Usage example can be found in the monai.inferers.Inferer base class. roi_size (Union[Sequence[int], int]) – the window size to execute SlidingWindow evaluation. If it has non-positive components, the corresponding inputs size will be used. if the components of the roi_size are non-positive values, the transform will use the corresponding components of img size. For example, roi_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. sw_batch_size (int) – the batch size to run window slices. overlap (float) – Amount of overlap between scans. mode (Union[BlendMode, str]) – {"constant", "gaussian"} How to blend output of overlapping windows. Defaults to "constant". "constant”: gives equal weight to all predictions. "gaussian”: gives less weight to predictions on edges of windows. {"constant", "gaussian"} How to blend output of overlapping windows. Defaults to "constant". "constant”: gives equal weight to all predictions. "gaussian”: gives less weight to predictions on edges of windows. sigma_scale (Union[Sequence[float], float]) – the standard deviation coefficient of the Gaussian window when mode is "gaussian". Default: 0.125. Actual window sigma is sigma_scale * dim_size. When sigma_scale is a sequence of floats, the values denote sigma_scale at the corresponding spatial dimensions. padding_mode (Union[PytorchPadMode, str]) – {"constant", "reflect", "replicate", "circular"} Padding mode when roi_size is larger than inputs. Defaults to "constant" See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html cval (float) – fill value for ‘constant’ padding mode. Default: 0 sw_device (Union[str, device, None]) – device for the window data. By default the device (and accordingly the memory) of the inputs is used. Normally sw_device should be consistent with the device where predictor is defined. device (Union[str, device, None]) – device for the stitched output prediction. By default the device (and accordingly the memory) of the inputs is used. If for example set to device=torch.device('cpu') the gpu memory consumption is less and independent of the inputs and roi_size. Output is on the device. progress (bool) – whether to print a tqdm progress bar. cache_roi_weight_map (bool) – whether to precompute the ROI weight map. cpu_thresh (Optional[int]) – when provided, dynamically switch to stitching on cpu (to save gpu memory) when input image volume is larger than this threshold (in pixels/voxels). Otherwise use "device". Thus, the output may end-up on either cpu or gpu. Note sw_batch_size denotes the max number of windows per network inference iteration, not the batch size of inputs. inputs (Tensor) – model input data for inference. network (Callable[…, Union[Tensor, Sequence[Tensor], Dict[Any, Tensor]]]) – target model to execute inference. supports callables such as lambda x: my_torch_model(x, additional_config) args (Any) – optional args to be passed to network. kwargs (Any) – optional keyword args to be passed to network. Union[Tensor, Tuple[Tensor, …], Dict[Any, Tensor]]

SaliencyInferer#

SaliencyInferer is inference with activation maps. cam_name (str) – expected CAM method name, should be: "CAM", "GradCAM" or "GradCAMpp". target_layers (str) – name of the model layer to generate the feature map. class_idx (Optional[int]) – index of the class to be visualized. if None, default to argmax(logits). args – other optional args to be passed to the __init__ of cam. kwargs – other optional keyword args to be passed to __init__ of cam. Unified callable function API of Inferers. inputs (Tensor) – model input data for inference. network (Module) – target model to execute inference. supports callables such as lambda x: my_torch_model(x, additional_config) args (Any) – other optional args to be passed to the __call__ of cam. kwargs (Any) – other optional keyword args to be passed to __call__ of cam.

SliceInferer#

SliceInferer extends SlidingWindowInferer to provide slice-by-slice (2D) inference when provided a 3D volume. A typical use case could be a 2D model (like 2D segmentation UNet) operates on the slices from a 3D volume, and the output is a 3D volume with 2D slices aggregated. Example: spatial_dim (int) – Spatial dimension over which the slice-by-slice inference runs on the 3D volume. For example 0 could slide over axial slices. 1 over coronal slices and 2 over sagittal slices. args – other optional args to be passed to the __init__ of base class SlidingWindowInferer. kwargs – other optional keyword args to be passed to __init__ of base class SlidingWindowInferer. Note roi_size in SliceInferer is expected to be a 2D tuple when a 3D volume is provided. This allows sliding across slices along the 3D volume using a selected spatial_dim. inputs (Tensor) – 3D input for inference network (Callable[…, Union[Tensor, Sequence[Tensor], Dict[Any, Tensor]]]) – 2D model to execute inference on slices in the 3D input args (Any) – optional args to be passed to network. kwargs (Any) – optional keyword args to be passed to network. Union[Tensor, Tuple[Tensor, …], Dict[Any, Tensor]] Wrapper handles inference for 2D models over 3D volume inputs. Union[Tensor, Tuple[Tensor, …], Dict[Any, Tensor]] previous Engines next Event handlers