Tensorboard visuals#

Creates an animated gif out of an image tensor in ‘CHWD’ format and writes it with SummaryWriter. writer – Tensorboard SummaryWriter to write to tag (str) – Data identifier image_tensor (Union[ndarray, Tensor]) – tensor for the image to add, expected to be in CHWD format max_out (int) – maximum number of image channels to animate through frame_dim (int) – the dimension used as frames for GIF image, expect input data shape as CHWD, default to -3 (the first spatial dim) scale_factor (float) – amount to multiply values by. If the image data is between 0 and 1, using 255 for this value will scale it to displayable range global_step (Optional[int]) – Global step value to record None Creates an animated gif out of an image tensor in ‘CHWD’ format and returns Summary. tag (str) – Data identifier image (Union[ndarray, Tensor]) – The image, expected to be in CHWD format writer – the tensorboard writer to plot image max_out (int) – maximum number of image channels to animate through frame_dim (int) – the dimension used as frames for GIF image, expect input data shape as CHWD, default to -3 (the first spatial dim) scale_factor (float) – amount to multiply values by. if the image data is between 0 and 1, using 255 for this value will scale it to displayable range Summary Plot 2D or 3D image on the TensorBoard, 3D image will be converted to GIF image. Note Plot 3D or 2D image(with more than 3 channels) as separate images. And if writer is from TensorBoardX, data has 3 channels and max_channels=3, will plot as RGB video. data (Union[~NdarrayTensor, List[~NdarrayTensor]]) – target data to be plotted as image on the TensorBoard. The data is expected to have ‘NCHW[D]’ dimensions or a list of data with CHW[D] dimensions, and only plot the first in the batch. step (int) – current step to plot in a chart. writer – specify TensorBoard or TensorBoardX SummaryWriter to plot the image. index (int) – plot which element in the input data batch, default is the first element. max_channels (int) – number of channels to plot. frame_dim (int) – if plotting 3D image as GIF, specify the dimension used as frames, expect input data shape as NCHWD, default to -3 (the first spatial dim) max_frames (int) – if plot 3D RGB image as video in TensorBoardX, set the FPS to max_frames. tag (str) – tag of the plotted image on TensorBoard. None

Class activation map#

Compute class activation map from the last fully-connected layers before the spatial pooling. This implementation is based on: Zhou et al., Learning Deep Features for Discriminative Localization. CVPR ‘16, https://arxiv.org/abs/1512.04150 Examples N.B.: To help select the target layer, it may be useful to list all layers: See also monai.visualize.class_activation_maps.GradCAM nn_module (Module) – the model to be visualized target_layers (str) – name of the model layer to generate the feature map. fc_layers (Union[str, Callable]) – a string or a callable used to get fully-connected weights to compute activation map from the target_layers (without pooling). and evaluate it at every spatial location. upsampler (Callable) – An upsampling method to upsample the output image. Default is N dimensional linear (bilinear, trilinear, etc.) depending on num spatial dimensions of input. postprocessing (Callable) – a callable that applies on the upsampled output image. Default is normalizing between min=1 and max=0 (i.e., largest input will become 0 and smallest input will become 1). Compute the actual feature map with input tensor x. x – input to nn_module. class_idx – index of the class to be visualized. Default to None (computing class_idx from argmax) layer_idx – index of the target layer if there are multiple target layers. Defaults to -1. activation maps (raw outputs without upsampling/post-processing.) Computes Gradient-weighted Class Activation Mapping

(Grad-CAM). This implementation is based on: Selvaraju et al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, https://arxiv.org/abs/1610.02391 Examples N.B.: To help select the target layer, it may be useful to list all layers: See also monai.visualize.class_activation_maps.CAM Compute the actual feature map with input tensor x. x – input to nn_module. class_idx – index of the class to be visualized. Default to None (computing class_idx from argmax) layer_idx – index of the target layer if there are multiple target layers. Defaults to -1. activation maps (raw outputs without upsampling/post-processing.) Computes Gradient-weighted Class Activation Mapping (Grad-CAM++). This implementation is based on: Chattopadhyay et al., Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks, https://arxiv.org/abs/1710.11063 See also monai.visualize.class_activation_maps.GradCAM Compute the actual feature map with input tensor x. x – input to nn_module. class_idx – index of the class to be visualized. Default to None (computing class_idx from argmax) layer_idx – index of the target layer if there are multiple target layers. Defaults to -1. activation maps (raw outputs without upsampling/post-processing.) A model wrapper to run model forward/backward steps and storing some intermediate feature/gradient information. nn_module – the model to be wrapped. target_layer_names (Union[str, Sequence[str]]) – the names of the layer to cache. register_forward (bool) – whether to cache the forward pass output corresponding to target_layer_names. register_backward (bool) – whether to cache the backward pass output corresponding to target_layer_names. layer_id (Union[str, Callable]) – a layer name string or a callable. If it is a callable such as lambda m: m.fc, this method will return the module self.model.fc. a submodule from self.model. A linear intensity scaling by mapping the (min, max) to (1, 0). If the input data is PyTorch Tensor, the output data will be Tensor on the same device, otherwise, output data will be numpy array. Note: This will flip magnitudes (i.e., smallest will become biggest and vice versa). ~NdarrayTensor

Occlusion sensitivity#

This class computes the occlusion sensitivity for a model’s prediction of a given image. By occlusion sensitivity, we mean how the probability of a given prediction changes as the occluded section of an image changes. This can be useful to understand why a network is making certain decisions. As important parts of the image are occluded, the probability of classifying the image correctly will decrease. Hence, more negative values imply the corresponding occluded volume was more important in the decision process. Two torch.Tensor will be returned by the __call__ method: an occlusion map and an image of the most probable class. Both images will be cropped if a bounding box used, but voxel sizes will always match the input. The occlusion map shows the inference probabilities when the corresponding part of the image is occluded. Hence, more -ve values imply that region was important in the decision process. The map will have shape BCHW(D)N, where N is the number of classes to be inferred by the network. Hence, the occlusion for class i can be seen with map[...,i]. The most probable class is an image of the probable class when the corresponding part of the image is occluded (equivalent to occ_map.argmax(dim=-1)). See: R. R. Selvaraju et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. https://doi.org/10.1109/ICCV.2017.74. Examples: See also monai.visualize.occlusion_sensitivity.OcclusionSensitivity. Occlusion sensitivity constructor. nn_module (Module) – Classification model to use for inference mask_size (Union[int, Sequence]) – Size of box to be occluded, centred on the central voxel. If a single number is given, this is used for all dimensions. If a sequence is given, this is used for each dimension individually. n_batch (int) – Number of images in a batch for inference. verbose (bool) – Use progress bar (if tqdm available). mode (Union[str, float, Callable]) – what should the occluded region be replaced with? If a float is given, that value will be used throughout the occlusion. Else, gaussian, mean_img and mean_patch can be supplied: gaussian: occluded region is multiplied by 1 - gaussian kernel. In this fashion, the occlusion will be 0 at the center and will be unchanged towards the edges, varying smoothly between. When gaussian is used, a weighted average will be used to combine overlapping regions. This will be done using the gaussian (not 1-gaussian) as occluded regions count more. mean_patch: occluded region will be replaced with the mean of occluded region. mean_img: occluded region will be replaced with the mean of the whole image. what should the occluded region be replaced with? If a float is given, that value will be used throughout the occlusion. Else, gaussian, mean_img and mean_patch can be supplied: gaussian: occluded region is multiplied by 1 - gaussian kernel. In this fashion, the occlusion will be 0 at the center and will be unchanged towards the edges, varying smoothly between. When gaussian is used, a weighted average will be used to combine overlapping regions. This will be done using the gaussian (not 1-gaussian) as occluded regions count more. mean_patch: occluded region will be replaced with the mean of occluded region. mean_img: occluded region will be replaced with the mean of the whole image. overlap (float) – overlap between inferred regions. Should be in range 0<=x<1. activate (Union[bool, Callable]) – if True, do softmax activation if num_channels > 1 else do sigmoid. If False, don't do any activation. If callable, use callable on inferred outputs. Occlude with a constant occlusion. Multiplicative is zero, additive is constant value. Tuple[float, Tensor] Crop the meshgrid so we only perform occlusion sensitivity on a subsection of the image. Tuple[MetaTensor, SpatialCrop, Sequence] For Gaussian occlusion, Multiplicative is 1-Gaussian, additive is zero. Default sigma of 0.25 empirically shown to give reasonable kernel, see here: Project-MONAI/MONAI#5230. Tuple[Tensor, float] Predictor function to be passed to the sliding window inferer. Takes a cropped meshgrid, referring to the coordinates in the input image. We use the index of the top-left corner in combination mask_size to figure out which region of the image is to be occluded. The occlusion is performed on the original image, x, using cropped_region * mul + add. mul and add are sometimes pre-computed (e.g., a constant Gaussian blur), or they are sometimes calculated on the fly (e.g., the mean of the occluded patch). For this reason occ_mode is given. Lastly, activate is used to activate after each call of the model. cropped_grid (Tensor) – subsection of the meshgrid, where each voxel refers to the coordinate of the input image. The meshgrid is created by the OcclusionSensitivity class, and the generation of the subset is determined by sliding_window_inference. nn_module (Module) – module to call on data. x (Tensor) – the image that was originally passed into OcclusionSensitivity.__call__. mul (Union[Tensor, float]) – occluded region will be multiplied by this. Can be torch.Tensor or float. add (Union[Tensor, float]) – after multiplication, this is added to the occluded region. Can be torch.Tensor or float. mask_size (Sequence) – Size of box to be occluded, centred on the central voxel. Should be a sequence, one value for each spatial dimension. occ_mode (str) – might be used to calculate mul and add on the fly. activate (Union[bool, Callable]) – if True, do softmax activation if num_channels > 1 else do sigmoid. If False, don't do any activation. If callable, use callable on inferred outputs. module_kwargs – kwargs to be passed onto module when inferring Tensor

Gradient-based saliency maps#

Based on Springenberg and Dosovitskiy et al. https://arxiv.org/abs/1412.6806, compute gradient-based saliency maps by backpropagating positive graidents and inputs (see _AutoGradReLU). See also Springenberg and Dosovitskiy et al. Striving for Simplicity: The All Convolutional Net (https://arxiv.org/abs/1412.6806) Compute gradient-based saliency maps based on both GuidedBackpropGrad and SmoothGrad. Compute averaged sensitivity map based on n_samples (Gaussian additive) of noisy versions of the input image x. See also Smilkov et al. SmoothGrad: removing noise by adding noise https://arxiv.org/abs/1706.03825 Given an input image x, calling this class will perform the forward pass, then set to zero all activations except one (defined by index) and propagate back to the image to achieve a gradient-based saliency map. If index is None, argmax of the output logits will be used. See also Simonyan et al. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps (https://arxiv.org/abs/1312.6034)

Utilities#

Blend an image and a label. Both should have the shape CHW[D]. The image may have C==1 or 3 channels (greyscale or RGB). The label is expected to have C==1. image (Union[ndarray, Tensor]) – the input image to blend with label data. label (Union[ndarray, Tensor]) – the input label to blend with image data. alpha (Union[float, ndarray, Tensor]) – this specifies the weighting given to the label, where 0 is completely transparent and 1 is completely opaque. This can be given as either a single value or an array/tensor that is the same size as the input image. cmap (str) – specify colormap in the matplotlib, default to hsv, for more details, please refer to: https://matplotlib.org/2.0.2/users/colormaps.html. rescale_arrays (bool) – whether to rescale the array to [0, 1] first, default to True. transparent_background (bool) – if true, any zeros in the label field will not be colored. Create a 3D volume figure as a grid of images. volume – 3D volume to display. data shape can be BCHWD, CHWD or HWD. Higher dimensional arrays will be reshaped into (-1, H, W, [C]), C depends on channel_dim arg. A list of channel-first (C, H[, W, D]) arrays can also be passed in, in which case they will be displayed as a padded and stacked volume. fig – matplotlib figure or Axes to use. If None, a new figure will be created. title (Optional[str]) – title of the figure. figsize – size of the figure. frames_per_row (Optional[int]) – number of frames to display in each row. If None, sqrt(firstdim) will be used. frame_dim (int) – for higher dimensional arrays, which dimension from (-1, -2, -3) is moved to the -3 dimension. dim and reshape to (-1, H, W) shape to construct frames, default to -3. channel_dim (Optional[int]) – if not None, explicitly specify the channel dimension to be transposed to the last dimensionas shape (-1, H, W, C). this can be used to plot RGB color image. if None, the channel dimension will be flattened with frame_dim and batch_dim as shape (-1, H, W). note that it can only support 3D input image. default is None. vmin – vmin for the matplotlib imshow. vmax – vmax for the matplotlib imshow. every_n (int) – factor to subsample the frames so that only every n-th frame is displayed. interpolation (str) – interpolation to use for the matplotlib matshow. show – if True, show the figure. fill_value – value to use for the empty part of the grid. margin (int) – margin to use for the grid. dtype – data type of the output stacked frames. kwargs – additional keyword arguments to matplotlib matshow and imshow. See also https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.matshow.html Example previous Event handlers next Utilities