A Illustrative examples of insertion/deletion evaluations

This section describes the detailed process of insertion/deletion evaluation. Figure 1 shows the process of the insertion for each attribution method. The insertion game starts from the base image (e.g., blurred image or zero image) and inserting the original pixel value in decreasing order of corresponding attribution values. The pixels with high attribution values are inserted first, and the image ends up with the original image. If the attribution method correctly captures the order of relevance of the pixels, the softmax output should increase in the early stage. Thus, the attribution method that obtains the curve with high AUC would be regarded as more reliable attribution. We provide the AUC score in the legend of the first row. We identify that SPI outperforms other methods with high AUC score. The insertion sequence also qualitatively identifies that SPI inserts the bird pixels first, while other methods insert them in the late stage.

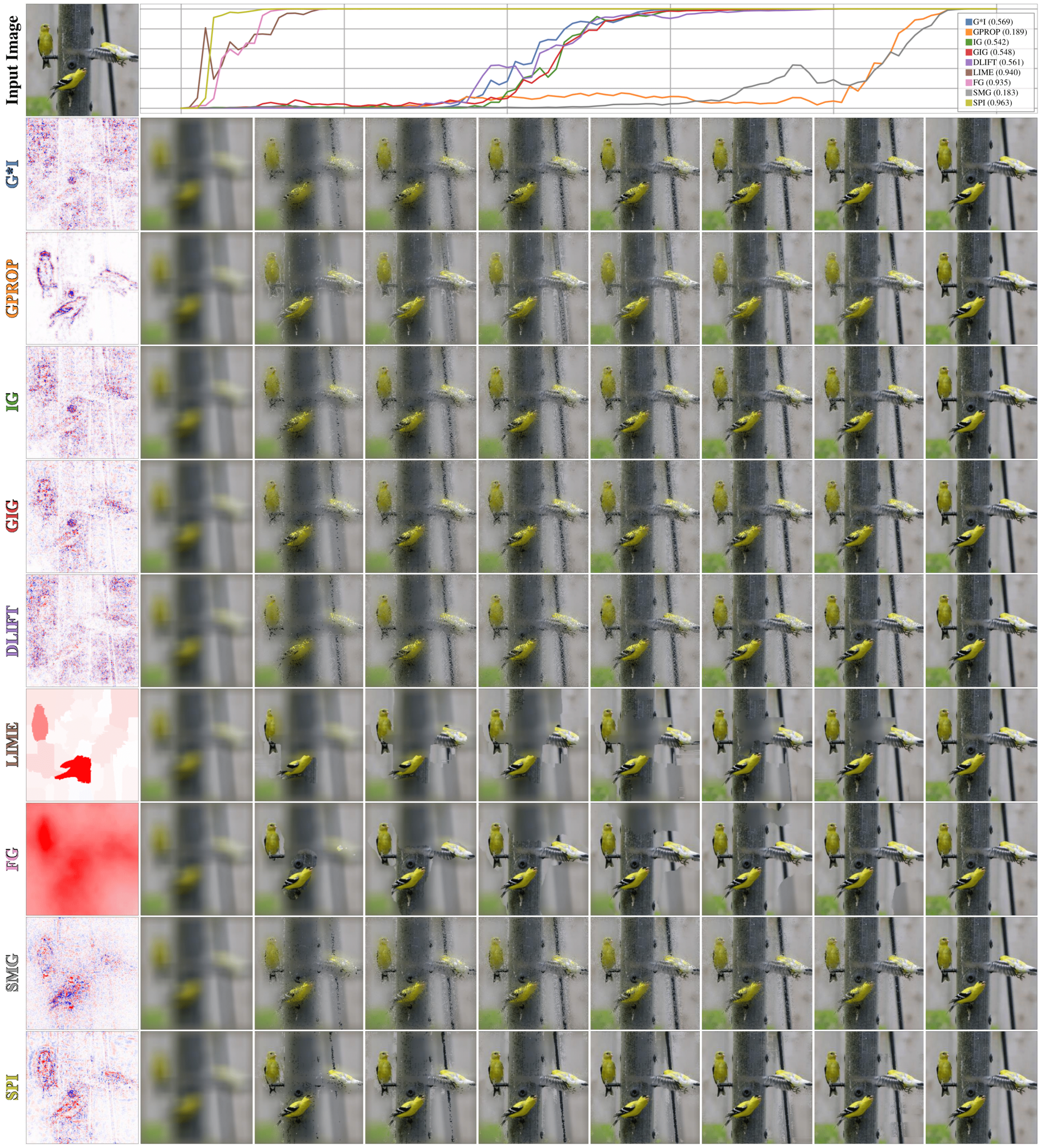


Figure 1: An illustrative example of the detailed process in the insertion evaluation. The curves in the first row depicts the softmax trends along the insertion order of the pixel computed by each attribution result. Each row depicts the attribution heatmap obtained by each method and corresponding insertion sequence. The sequence is sampled with equidistant intervals for the visualization.

Figure 2 shows the process of the deletion for each attribution method. In similar way as the insertion game, the deletion game starts from the original image and change the original pixel values in increase order of corresponding attribution values until the original image becomes the base image (e.g., blurred image or zero image). If the attribution method correctly captures the order of relevance of the pixels, the softmax output should decrease in the early stage. Thus, the attribution method that obtains the curve with low AUC would be regarded as more reliable attribution. From Figure 2, we confirm that SPI outperforms other method with low AUC score.

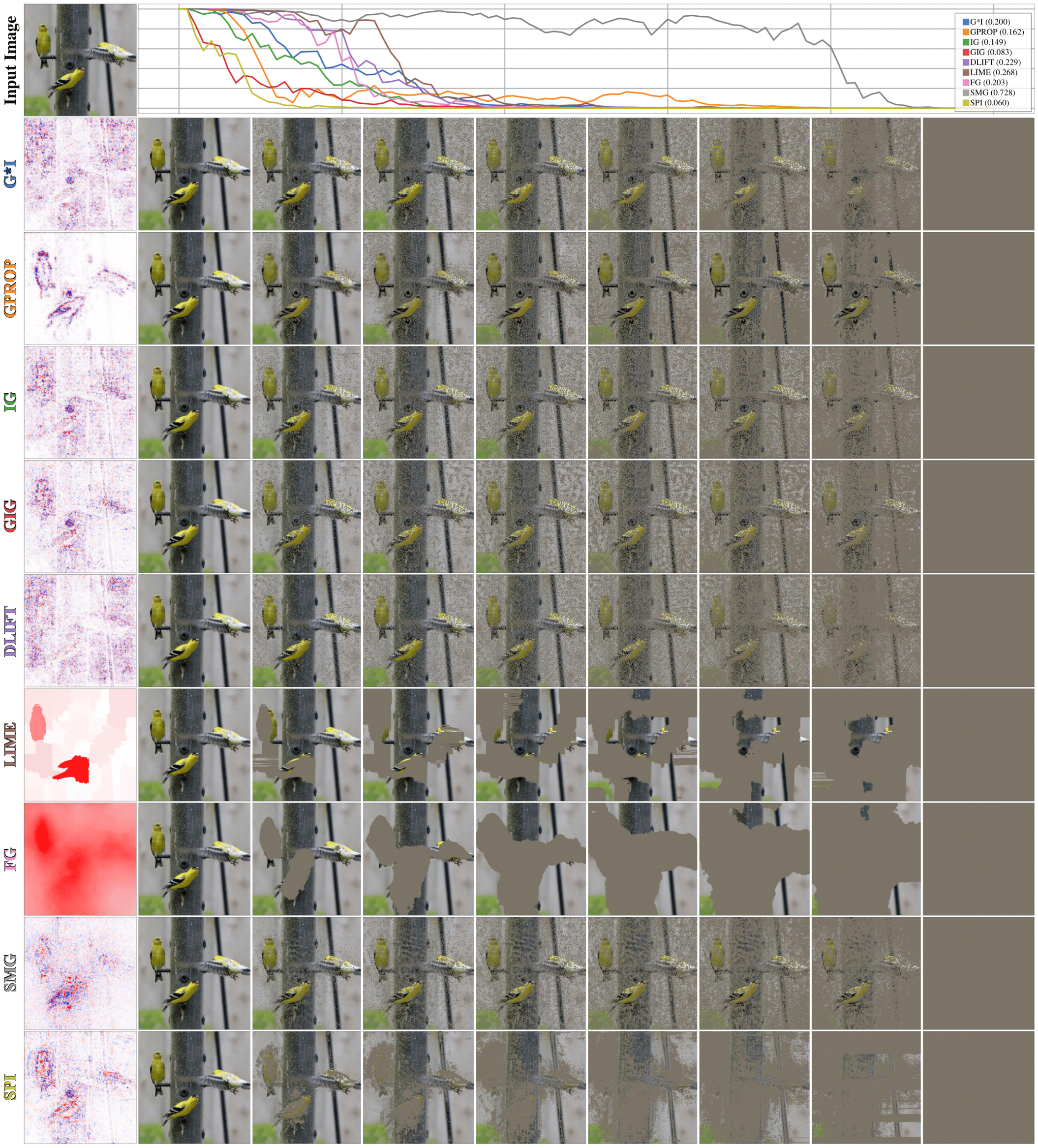
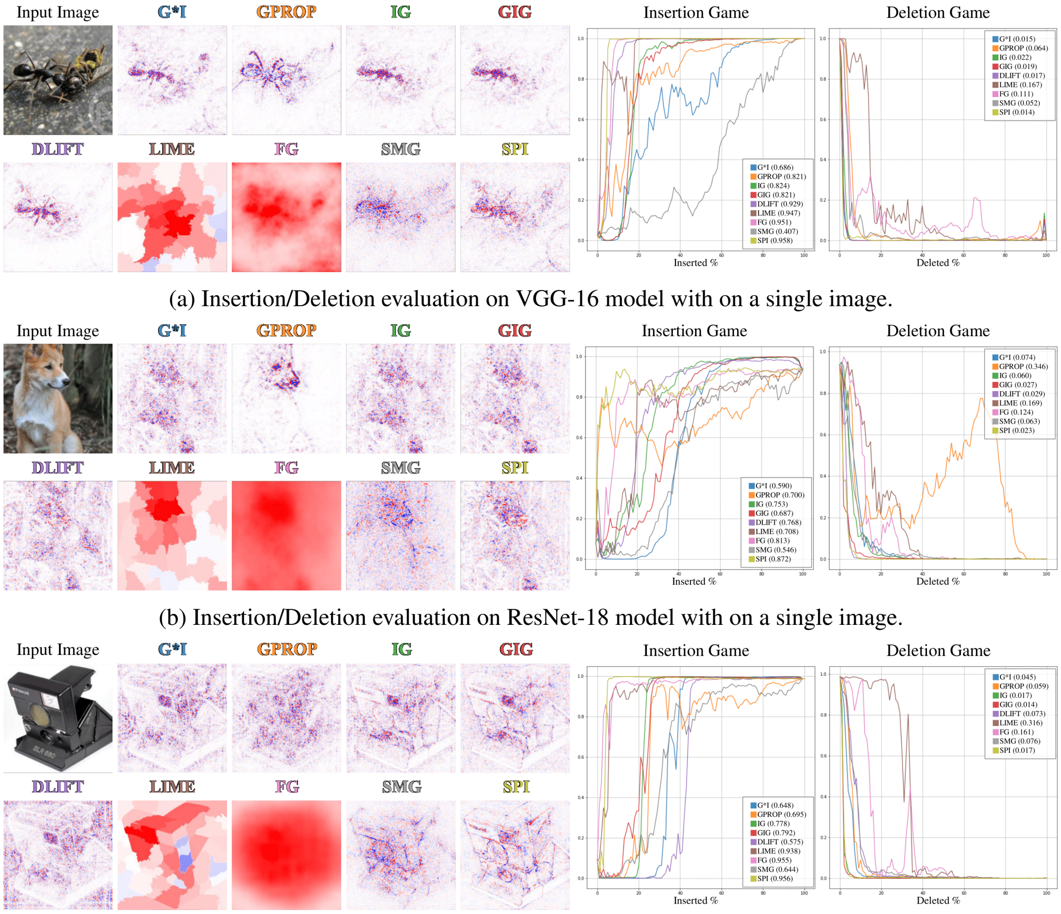


Figure 2: An illustrative example of the detailed process in the deletion evaluation. The curves in the first row depicts the softmax trends along the deletion order of the pixel computed by each attribution method. Each row shows the attribution heatmap obtained by each method and the corresponding deletion sequence. The sequence is sampled with equidistant intervals for the visualization.

B Insertion/Deletion evaluation for single image

We also provide the change of softmax output in insertion/deletion game with the single image to clarify the evaluation metric for three models trained on ImageNet. Figure 3 shows the attribution heatmap and the corresponding curve for each method.



(a) Insertion/Deletion evaluation on VGG-16 model with on a single image.

(b) Insertion/Deletion evaluation on ResNet-18 model with on a single image.

(c) Insertion/Deletion evaluation on Inception-V3 model with on a single image.

Figure 3: The comparison of attribution methods on a single image with insertion/deletion evaluation for three models trained on ImageNet.

C Implementation Code

We provide the implementation of our algorithm in Python.

```python
import torch
def SPI(
    inputs,        # (tensor) Input tensor to be explained.
    model_func,    # (function) The function of model to be explained.
    target,        # (integer) Target class to be explained.
    alpha=10,      # (float) Concentraion parameter.
    n_paths=30,    # (integer) Number of paths to be sampled.
    n_steps=30,    # (integer) Number of steps for each path.
    ):

    # Initialize Beta distribution for Stick-Breaking Process
    beta_dist = torch.distributions.Beta(1, alpha)
    # Define a linear sequence which is used in Stick-Breaking Process
    linear = torch.linspace(0,1,n_steps)
    linear = linear.reshape([-1,]+[1,]*len(inputs.shape))
    rand_igs = []
    for i in range(n_paths):
        # Sample alpha path according to Stick-Breaking Process
        rand_alphas = beta_dist.sample([n_steps*10, 1, *inputs.shape[1:]])
        rand_alphas[1:] *= (1-rand_alphas[:-1]).cumprod(dim=0)
        u = torch.rand_like(rand_alphas) < linear
        rand_alphas = torch.mul(rand_alphas[None], u)
        # Append auxilary zeros and ones to ensure the path from 0 to 1.
        rand_alphas = torch.cat([torch.zeros_like(inputs[1:])[None],
                                rand_alphas,
                                torch.ones_like(inputs[1:])[None]], dim=0)

        # Multiply the input to build the Stick-Breaking Path
        rand_xpath = torch.mul(rand_alphas, inputs[None,]).requires_grad_(True)
        # Collect the gradients along the path
        rand_gpath = []
        for _x in rand_xpath:
            _y = model_func(_x)[range(len(inputs)), target]
            _g = torch.autograd.grad(_y.sum(), _x)[0]
            rand_gpath.append(_g)
        rand_gpath = torch.stack(rand_gpath)
        # Use Riemann sum to integrate the gradients
        rand_ig = torch.mul(rand_xpath[1:]-rand_xpath[:-1],
                            (rand_gpath[1:]+rand_gpath[:-1])/2).sum(dim=0)
        rand_igs.append(rand_ig)
    return torch.mean(rand_igs, dim=0)
```