Homework Assignment 4

Response by: **Soham Roy(7028704)**

Due: **2:00pm Tuesday, 8 February 2024 on CISPA CMS**

Collaboration Policy: You should do this assignment by yourself and submit your own answers. You may discuss the problems with anyone you want and it is also fine to get help from anyone on problems with LaTeX or Jupyter/Python. You should note in the *Collaborators* box below the people you collaborated with.

Collaborators: Somrita Ghosh, Subrat Kishore Dutta

Implementation Problems. Below are two implementation problems that you need to provide your code implementations and results in Jupyter notebook. For the tasks which require explanations or descriptions (e.g., Task 3 of Problem 2), you can provide your answers in the submitted PDF.

Problem 1 (15 pts) In this problem, we will discuss and implement the *SmoothGrad* method for producing saliency map visualizations. The method is first introduced in the paper "SmoothGrad: removing noise by adding noise". Specifically, consider an input $\boldsymbol{x} \in \mathbb{R}^d$ and a classifier $f_\theta : \mathbb{R}^d \to \mathcal{Y}$ that we aim to probe, where $\mathcal{Y}$ denotes the set of class labels. Let $F_\theta : \mathbb{R}^d \to \mathbb{R}$ be function mapping from the input layer to the logit with respect to the predicted class $\hat{y} = f_\theta(\boldsymbol{x})$. *SoomthGrad* computes the saliency maps of F_θ at $\boldsymbol{x}$ based on the following rule:

$$\text{SmoothGrad}(F_\theta, \boldsymbol{x}; m, \sigma) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{x}+\boldsymbol{\delta}_i} F_\theta(\boldsymbol{x} + \boldsymbol{\delta}_i), \text{ where } \boldsymbol{\delta}_i \sim \mathcal{N}(0, \sigma^2).$$

Here, m represents the number of Gaussian samples and $\sigma > 0$ denotes the standard deviation parameter.

Task 1 (5pt): Answer the following question: How will the two parameters m and σ affect the computed saliency maps? Please also provide a brief explanation about your answer.

ANSWER - The sigma is the noise variable , it signifies the neighborhood of the pixel from which it will sample data points, which is analogous to the perturbation ball. The more the sigma , the bigger the neighborhood, the more noise will be incorporated. Ideally there should be a balance between the structure/sensitivity map and the structure of the object. so a too-high or too-low sigma could be detrimental to the experiment and could yield unassuming and erratic saliency maps.

The m is the no of data points that will be sampled and averaged, the more m the better and smoother the saliency maps, but also it comes at a cost of computation.

Task 2 (10pt): Now we implement the *SoomthGrad* method on a neural network and CIFAR-10 images. We will use the *pretrained ResNet18 model* as the classifier f_θ, and you need to randomly sample 5 testing CIFAR-10 images. You can refer to Problem 2 of Homework Assignment 3 on how to prepare the CIFAR-10 dataset. In particular, the number of Gaussian samples should be set as $m = 50$, and the the standard deviation parameter $\sigma \in \{0, 0.05, 0.1, 0.2, 0.3, 0.5\} * (x_{\max} - x_{\min})$, where $x_{\max}$ and $x_{\min}$ represent the largest and the smallest pixel value of the correpsonding sampled CIFAR-10 image $\boldsymbol{x}$.

Visualize the saliency maps as heatmaps for each combination of the 5 sampled CIFAR-10 images and the 6 σ parameters, and also visualize the sampled CIFAR-10 images. The produced figure should be in the similar format to Figure 3 of the "SmoothGrad" paper. In addition, briefly discuss the results (i.e., whether they are aligned with your answers for Task 1).

ANSWER- I see my result aligns with the claim from part 1, the experiment works best when the sigma is in 0.1-0.3, while the maps get fizzed and erratic when it is too low or high.

Problem 2 (25 pts) In this problem, we will consider a simple setting of out-of-distribution robustness on the MNIST dataset. First of all, a ood_mnist.ipynb file is provided in the jupyter_notebook folder. It provides the sample code for generating customized MNIST digits: *"0"* and *"8"*. The code is adapted from the Github repository: ec23-tutorial created by AIRI, where you can find the source code of all the functions. First, you should understand the content provided in the ood_mnist.ipynb file.

Instead of considering two MNIST digits and random forest, we will extend the results to the multi-class classification task of classifying ten MNIST digits using neural network. Specifically, you need to set the neural network using the CNN architecture with 4 convolution layers and 2 MLP layers, the same

architecture considered in Problem 3 of Homework Assignment 2. You can set the optimizer of SGD with learning rate 0.1, and train the model for 30 epochs.

Task 1 (5 pts): Construct three additional MNIST testing datasets: *colored_MNIST*, *grayscaled_MNIST*, and *rotated_MNIST*. To be more specific, *colored_MNIST* consist of colored testing MNIST image by randomly applying one of the three color transformations (i.e., red, green and blue) provided in the function gen_colored(), *greyscaled_MNIST* consist of grayscaled MNIST testing images by randomly applying one of the three transformations provided in function gen_weighted_grayscale(), while *rotated_MNIST* consists of rotated MNIST testing images by applying the transformation using the function rr(). All the functions are detailed in the ec23-tutorial repo. All the testing datasets should contain 10000 images.

Task 2 (5 pts): Train a neural network with the required architecture and learning scheduler on the original MNIST training dataset. Report the mean accuracy and standard deviation over 3 repeated trials on each of the three constructed MNIST testing datasets and the original MNIST testing dataset. Do you observe an obvious accuracy drop when tested on customized datasets? Also, please provide a brief discussion about the results.

ANSWER Yes i see a accuracy drop when i test the customized datasets on the standard trained model. the results i got are Train accuracy 99.01666666666667 (on training data)
Test accuracy 98.72999999999999 (on test data)
Colored_MNIST mean Test accuracy 98.60 standard deviation 0.04
Grayscale_MNIST Test accuracy 98.43 standard deviation 0.05
Rotated_MNIST Test accuracy 45.71 standard deviation 0.11

The experiment and result can also be found in the notebook

Task 3 (5 pts): Next, we want to improve the out-of-distribution robustness performance of our learning algorithm. One typical method for OOD robustness is to apply various types of data augmentations to the training images. Read the paper "AUGMIX: A Simple Data Processing Method to Improve Robustness and Uncertainty", then summarize the existing data augmentation techniques and their proposed AUGMIX method that is designed for enhancing the model robustness against distribution shift.

ANSWER - the existing vannilla data augmentation techniques which are generally applied for experiments involve rotation, shear, color, illumination changes etc. In a vanilla augmentation method ,it is a common practice to use this one or or maybe a combination of them on a dataset. This blind application has a disadvantage as it substantially deteriorates the real image under data shift, and more often so the augmented image barely resembles the original image which is the shift in data. The authors claim that this fails to generalize to unseen distortions, as networks have a tendency to memorize properties of the specific training distortion Such augmentations do not do any good. The author proposes "AugMix" where the working principle involves mixing the results from compositions of augmentation operations, where a set of k (typically 3) augmentation chains, each composed of 3 operations, are selected and applied independently to k copies of the original image. The resulting k images are then mixed together linearly with random weights, and this augmented image is mixed linearly with the original non-augmented image. The algorithm involves generating convex weights from a Beta distribution, randomly sampling k augmentation chains, and combining the resulting images from these chains by mixing them with convex coefficients randomly sampled from a Dirichlet distribution. This creates diverse augmented images through a combination of multiple augmentation operations and mixing them with random weights, leading to better generalization performance and improved accuracy on corrupted data.

Task 4 (10 pts): Adapt the data augmentation techniques discussed in the paper to improve the out-

of-distribution robustness on MNIST. Instead of considering our constructed synthetic datasets, we consider a better benchmark dataset created for evaluating OOD robustness for MNIST, named MNIST-C, which is introduced in the paper "MNIST-C: A Robustness Benchmark for Computer Vision". The dataset consists of a variety of corrupted MNIST images, which can be downloaded here: https://github.com/google-research/mnist-c. To simplify the task, you only need to consider three types of corruptions (out of the total 15 types), i.e., *Motion Blur*, *Fog*, and *Zigzag*. Describe how you augment the training data for improving OOD robustness. Report the testing accuracy of the vanilla model trained in Task 2 and the model learned with your proposed augmentation technique on each of the three corrupted datasets.

ANSWER - For the augmentation I have used the Augmix augmentation technique (from the authors repo) without the use of the addition of the JS Divergence term to the loss functions as done by the authors of the paper. I get significantly better results than the vanilla model. The results can be found in the notebook.

Problem 3 (bonus, 5 pts) Read the paper "Accuracy on the Line: On the Strong Correlation Between Out-of-Distribution and In-Distribution Generalization" and write a one-page short review about the paper. You can follow the general structure of the paper reviews you are supposed to submit on Feb 1st.

End of Homework Assignment 4 (PDF part)
Don't forget to submit your nice Jupyter notebook!