Homework 3: Part 2

18739 Spring 2020

Due: Thursday, March 26, 2020 10:20am PST / 1:20pm EST

Academic Integrity Policy

This is an individual homework. Discussions are encouraged but you should write down your own code and answers. No collaboration on code development is allowed. We refer to CMU's Policy on Cheating and Plagiarism (https://www.cmu.edu/policies/). Any code included in your submission will be examined under Similarity Check automatically.

Late Day Policy

Your solutions should be uploaded to Gradescope (https://www.gradescope.com/) by the deadline. You have 8 late days in total to spend during the whole course but no more than 3 days can be applied to any one homework.

Written Exercises

If a homework contains written exercises, you will need to submit a `.pdf` file. Please indicate your Andrew ID and name on the first page. We will not accept hard-copies and do not hand-write your answers. Latex (Overleaf: https://www.overleaf.com) and Markdown are two recommended languages to generate the clean layout but you are free to use other software. You will need to submit the written part to HomeworkX-PDF on Gradescope, separately from you code submission.

Coding Exercises

Submit the coding portions of homeworks to Gradescope according to the following procedure:

1. Compress your `.py` and other requested files into a zip file called `submission.zip` (No other name is allowed otherwise Gradescope will fail to grade your submission). Do NOT put all files into a folder first and compress the folder. **Create the zip file directly on the top of all required files**. Each homework will specify the `.py` and other files expected to be included in the zip.
2. Submit `submission.zip` to Gradescope. Your code implementation will be auto-graded by Gradescope and you have an infinite number of submissions before the deadline but only the last submission will count towards the grading. Allow the server to take some time (around 2 min) for execution, which means that do not submit until the last moment when everyone is competing for resources.

Contents

Before you get started

Various exercises in this homework will refer to an ImageNet sample. You should get started by downloading it with the following command:

```
wget https://hw2dataset.s3.amazonaws.com/imagenet_14c_vgg16.npz
```

The method `load_ImageNet` of `hw3_utils` will help you load this dataset. The method returns images pre-scaled to vgg16, their ground truth class indices, their vgg16 predicted class indices, and their indices within a larger ImageNet sample. The dataset contains 100 images of 14 classes. You can use the methods described in Part 1 of this homework to map class indices to strings. The included classes are:

- 1: `goldfish, Carassius auratus`
- 163: `bloodhound, sleuthhound`
- 250: `Siberian husky`
- 283: `Persian cat`
- 284: `Siamese cat, Siamese`
- 340: `zebra`
- 354: `Arabian camel, dromedary, Camelus dromedarius`
- 356: `weasel`
- 386: `African elephant, Loxodonta africana`
- 397: `puffer, pufferfish, blowfish, globefish`
- 479: `car wheel`
- 511: `convertible`
- 734: `police van, police wagon, paddy wagon, patrol wagon, wagon, black Maria`
- 817: `sports car, sport car`

2 Influence-Directed Explanation

Influence-Directed Explanation [1] introduces the concept of *distributional influence* in explaining the internal behavior of neural networks.

Definition 3 (Distribution of Influence)**.** *Consider a feed-forward model $y = f(x)$ that takes an input $x \in \mathrm{R}^d$ and outputs y, a distribution of scores over a set of classes. We denote the $y_c = f_c(x)$ as the scores for the class c. A layer l decomposes the model into a function from model input to layer l's output, denoted h^l, and a function from layer l's output to the whole model output, denoted g^l, together forming the whole model $f(x) = g^l(h^l(x))$. Further, let a quantity of interest,q, be a function of the network's output scores (pre-softmax). The* distribution of influence $\mathcal{I}_q^l$ *at layer l is defined as*

$$\mathcal{I}_q^l \stackrel{\text{def}}{=} \int_{x \in \mathcal{X}} \frac{\partial q(f(x))}{\partial h^l(x)} dx \tag{1}$$

where $\mathcal{X}$ denotes the distribution of inputs. This is numerically approximated as

$$\mathcal{I}_q^l \approx \frac{1}{N} \sum_{i=1}^{N} \frac{\partial q(f(x))}{\partial h^l(x_i)} \tag{2}$$

where N is the size of the dataset, an empirical sample of $\mathcal{X}$.

By finding the neuron at layer l with highest influence score, we locate the expert neuron "most" responsible for quantity of interest q. Given a vector of pre-softmax class scores s, some relevant quantities of interest include:

- s_c where c is the predicted class.
- s_y where y is the ground truth class.
- $s_c - s_y$ where c is the true class and y is the predicted class.
- $s_{c_1} - s_{c_2}$ where c_1, c_2 are two different classes.

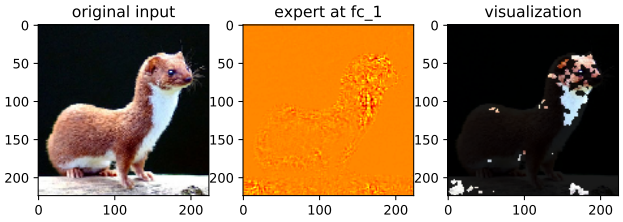
After we find the expert neuron most responsible for a particular quantity of interest, we can visualize the expert neuron by finding the input features with high influence on the expert neuron's activation. We do this using the gradient of the output of expert neuron w.r.t. the input. Formally,

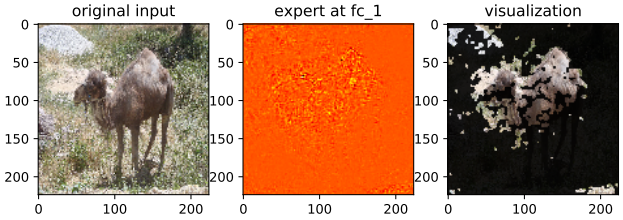
Definition 4 (Expert Neuron Attribution)**.** *Consider a feed-forward model $y = f(x)$ that takes an input $x \in \mathrm{R}^d$ and outputs y. Given layer l with the distribution of influence $\mathcal{I}_c^l$ towards class c. Let $\mathcal{I}_c^l[i]$ and $h_i^l(x)$ be the influence score and the output for the i-th neuron at layer l, respectively, and let i^* be the most expert neuron at layer l, or $i^* \stackrel{\text{def}}{=} \arg\max_i \mathcal{I}_c^l[i]$. The* expert neuron attribution *score $A(x)$ is defined as*

$$A(x) \stackrel{\text{def}}{=} x \odot \nabla_x h_{i*}^l(x) \tag{3}$$

Coding Exercise 10. [8 points] *Implement the Influence-Directed Explanation in the class* `InflDirected` *in of* `hw3_infl.py`.

Requirement: When computing the gradient of a class score, use the pre-softmax score. For simplicity, your implementation will receive full scores **even if you only consider the internal influence on dense layers**. See the following example of the expected results for quantity of interest, $q = s_c$ where $c = \arg\max(f(x))$, the predicted class:





Written Exercise 11. [3 points] *Report the indices of the expert neuron at* `fc_1` *layer w.r.t. the* `camel` *and* `weasel` *classes, (A) when the distribution of interest is the entire dataset and (B) when the distribution of interest is just the subset of* `camel` *or* `weasel` *images, respectively. Also provide visualizations of these experts on a representative input image from the respective distribution of interest.*

Discuss what you observe: do the experts correspond to some human-understandable concept? Are they the same for both camels and weasels? Did the differences in distribution of interest have any impact? It might be useful to visualize the expert on more than a single image to derive some conclusions.

Written Exercise 12. [3 points] *Pick two related classes from the sample we provided different than* `camel` *or* `weasel`*. Which classes have you chosen? Report the same information as the previous exercise. For (B), use the appropriate subsets of only the classes you have chosen.*

Discuss what you observe: do the experts correspond to some human-understandable concept? Are they the same for both of your chosen classes? Did the differences in distribution of interest have any impact?

Written Exercise 13. [4 points] *For the two classes you picked in the previous exercise, report and visualize the expert neuron (in the same layer) that distinguishes the first class from the second. You will need to use a quantity of interest that compares scores of the two classes. For the quantity of interest, use images from the (A) entire dataset and (B) only images belonging to one of your two classes. Also report your quantity of interest. Discuss what you observe: do the experts correspond to some human-understandable concept?*

Did the differences in distribution of interest have any impact?

[2 bonus points] Bonus points possible for especially thoughtful discussions.

Include the code you have used to answer the above questions as part of your `hw3_infl.py` submission.

3 Attribution Method Evaluation *[26 points]*

In this section you are going to implement metrics to evaluate the performance of attribution methods and apply them to the three methods you have implemented in this homework: Saliency Map, Integrated Gradient and Influence-Directed Explanations.

3.1 Visual Comparisons

To visualize the highlighted area based on attribution scores, we provide you with some visualization utilities in `Visualization.py` that you may find useful for visual comparisons. For the next exercise you need to select 10 images from a representative samples shown below. Select 5 from the first collection and 5 from the second. The first is sample of images from each class that was correctly classified while the second set includes examples of incorrectly classified images. You can retireve your selection from the dataset we provided by their imagenet index shown above each sampe.

Figure 1: Correct samples.

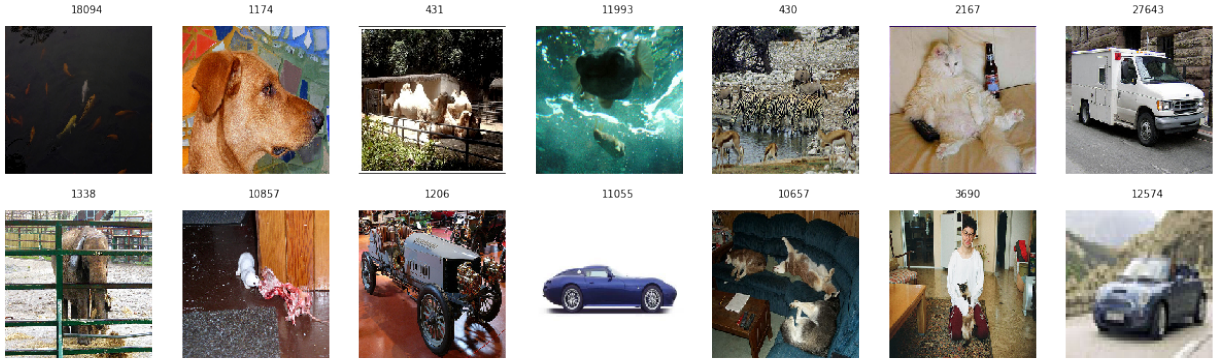


Figure 2: Incorrect samples.

Written Exercise 14. [5 points] *Use* `point_cloud` *and* `binary_mask` *methods to construct visualizations for Saliency Map, Integrated Gradient, Influence Directed Explanations attributions for the* ***10*** *different input images you selected, 5 from the correctly predicted sample and 5 from the incorrectly predicted sample. An example of Saliency Map is given below.* ***DO label the visualizations with the method used and the ImageNet index of the images, and whether it is a correctly classified image nor not.***

For selecting the expert neuron for influence directed explanations, use the pre-softmax score for the predicted class as the quantity of interest, the whole dataset as the distribution of interest, and `fc_1` *as the layer.*

Include your code for generating the attribution maps and visualizations as `hw3_vc.py`.

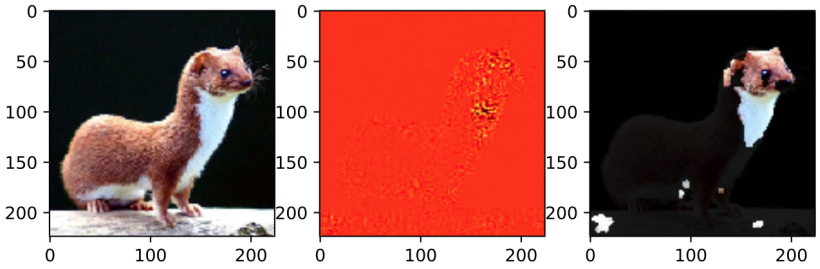


Figure 3: **Left**: the original input **Middle**: the point cloud generated by `point_cloud` **Right**: the masked input with saliency map generated by `binary_mask`

Written Exercise 15. [4 points] *Compare and contrast the attributions produced by the three methods. Example points of comparison may include, but should not be limited to:*

- *Is one of the attribution methods consistently more convincing to you as a human who presumably can recognize the objects pictured?*
- *How do the methods fare in relation to pointing out elements of an image that should definitely have no relationship with the pictured object?*
- *Are some methods easier to interpret than others? What characteristics make an attribution easier to interpret?*
- *Is visual comparison a good approach to distinguish attribution methods? why or why not?*
- [2 bonus points] *Bonus points available for this exercise for especially thoughtful comparison.*

3.2 Average Drop %

Unfortunately visual evaluations are subjective and objective methods are hard to come by. In this homework we will explore two objective methods. The first is Average % Drop. This method and the one that follows operates on attributions that do not distinguish between the color channels. **To create a pixel attribution from the full attribution you have worked with so far by setting setting the attribution of a pixel as the average attribution of its three channels.**

Definition 5 (Average Drop %)**.** *Given a (pixel) attribution map A for an input x and the model $f(x)$, let $M_A(x)$ denote a mask function that keeps only the pixels in x whose pixel attribution score in A is positive while setting all other pixels to 0. The Average Drop % score AD is then defined for a set of instances D as:*

$$AD(D) \stackrel{\text{def}}{=} 100\% \times \frac{1}{|D|} \sum_{x_i \in D} \frac{\max\left[f_{c_i}(x_i) - f_{c_i}(M_{A_i}(x_i)), 0\right]}{f_{c_i}(x)} \tag{4}$$

where $f_{c_i}(x_i)$ is the pre-softmax output score for class c for instance x_i and $c_i \stackrel{\text{def}}{=} \arg\max(f(x_i))$.

Coding Exercise 16. [3 points] *Implement this evaluation metric with* `AverageDrop` *in* `hw3_evaluation.py`.

Written Exercise 17. [2 points] *Use your implementation to compute the average drop score for the 100 images of a class of your choosing for the three attribution methods of Section 3.1. Report the class you chose and the scores in the written part. This should be a 3 by 1 table.*

Written Exercise 18. [2 points] *Based on the definition of Average Drop %, what does this metric evaluate about an attribution map? Does a higher Average Drop % indicate an attribution map is better than another and in what sense? Discuss whether Average Drop % is a reasonable objective measure of attribution goodness.* [2 bonus points] *Bonus points available for especially thoughtful discussion.*

3.3 Necessity Ordering

Necessity Ordering Score [2] is another objective measure of an attribution's fidelity. It applies to individual images/attributions as opposed to entire datasets as was the case for average drop:

Definition 6 (Necessity Ordering Score)**.** *Given a (pixel) attribution map A of an input x and a model $f(x)$, denote $\pi_A(x)$ as a list of pixels of x ordered by their positive attribution score in A, biggest first. Denote $R(i, \pi_A)$ as the masking of x that replaces all the pixels up to the i-th according to ordering π_A by 0. Everything else about x is left unchanged. The Necessity Ordering Score for the attribution map $N_o(x, A)$ is defined as*

$$N_o(x, A) \stackrel{\text{def}}{=} \frac{1}{N+1} \sum_{i}^{N} \max \left[f_c(R(i, \pi_A)) - f_c(\vec{0}), 0 \right] \tag{5}$$

where N is the number of features, $f_c(\cdot)$ denotes the pre-softmax output score for class c, and $\vec{0}$ is an all-zero "blank" input.

The above removes pixels one by one and can be computationally intensive; about half of 224×224 of the input pixels or ≈ 25088 are positive features. For your implementation you can remove M features at a time (you can use $M = 1000$).

Coding Exercise 19. [5 points] *Implement this evaluation metric with `N_Ord` in `hw3_evaluation.py`.*

Written Exercise 20. [3 points] *Use your implementationto evaluate the **10** images and their attribution maps of Sec 3.1. Report the scores in the written part. This should be a 10 by 3 table.*

Written Exercise 21. [2 points] *Based on the definition of Necessity Ordering, what does this metric evaluate about an attribution map? Does a higher Necessity Ordering indicate an attribution map is better than another and in what sense? Discuss whether Necessity Ordering is a reasonable objective measure of attribution goodness.* [2 bonus points] *Bonus points available for especially thoughtful discussion.*

4 Submission

Submit all your `.py` files to Gradescope. The required list of files is noted below. Not all of the coding exercises will be graded by Gradescope so you need to make sure you have commented your code in a clear way. For the written exercise, follow the instructions in the beginning of this document and make sure you label each written exercise answer with the corresponding exercise number.

- `hw3_attribution.py` (from Part 1)
- `hw3_infl.py`
- `hw3_vc.py`
- `hw3_evaluation.py`

References

[1] Klas Leino, Linyi Li, Shayak Sen, Anupam Datta, and Matt Fredrikson. Influence-directed explanations for deep convolutional networks. *arXiv preprint arXiv:1802.03788*, 2018.

[2] Zifan Wang, PiotrPiotr Mardziel, Anupam Datta, and Matt Fredrikson. Interpreting interpretations: Organizing attribution methods by criteria, 2020.