# Homework 3 Part 1

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 start

In this homework, we provide you with a pretrained VGG16[1] network for classifying a subset[2] of ImageNet[3]. You will build explanations for this model in the homework. We provide the following helper functions in `hw3_utils.py` in the Part One.

1. `get_model: tf.Session -> vgg16` returns a class containing the architecture of VGG16.
2. `get_tensor_names: () -> List[str]` returns names of all tensors in the current graph.
3. `ImageNet_name2idx: str -> int` returns the index of a class if given a name.
4. `ImageNet_idx2name: int -> str` returns the name of a class if given an index.
5. `binary_mask: ndarray, ndarray -> ndarray` overlays the attribution map on the input image to differentiate input features according to the attribution scores. You could use the default argument or adjust them after you carefully read the docstrings.
6. We also import from tensorflow a few useful methods for processing images: `img_to_array`, `load_img`, `resize_images`. You can see their use for running the VGG16 network at the bottom of `vgg16.py`. You might need to install the `pillow` (which provides `PIL`) pip package to make use of these methods.

Also, make sure you have installed `PIL`, `tqdm` and `scipy` in addition to the environment we use in the homework one. You can run the following commands to install these packages in conda:

```
conda install scipy tqdm

conda install -c anaconda pillow
```

# 1 Gradient-Based Attribution Methods

In this part, we are going to implement two gradient-based attribution methods that interpret the feature importance by attribution scores.

[1] https://arxiv.org/pdf/1409.1556.pdf
[2] http://image-net.org/challenges/LSVRC/2014/browse-synsets
[3] http://image-net.org/index

## 1.1 Saliency Map

Sailency Map [1] compute the local gradient of a class of interest w.r.t an input to find the attribution scores. Moreover, current work has found that the product of input $\times$ grad produces clearer visualization result. In the homework, we use the following definition for Saliency Map:

**Definition 1** (Saliency Map)**.** *Consider a model $y = f(x)$ that takes an input $x \in \mathrm{R}^d$ and outputs $y$, a distribution of scores for each class. We denote the $y_c = f_c(x)$ as the scores for the class $c$. The Saliency Map $S_c(x)$ for class $c$ is defined as*

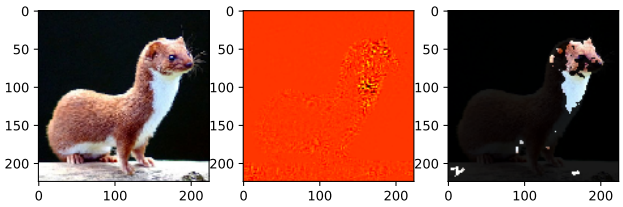
$$S_c(x) = x \odot \nabla_x f_c(x) \tag{1}$$

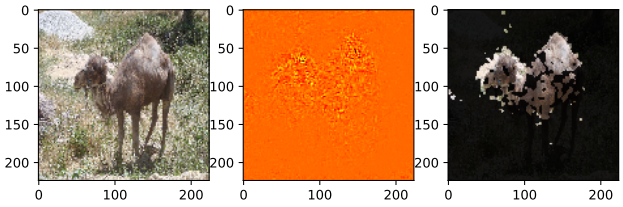
*where $\odot$ denotes the element-wise multiplication.*

**Coding Exercise 1.** [4 points] *Implement Def 1 within the* `SaliencyMap` *class of* `hw3_attribution.py`*. The main method is* `__call__` *but you will also need to implement* `_define_ops`*, which is called only a single time, to add any tensorflow operations into the computational graph you find necessary and store them in the class* `self`*. The method* `__call__` *should not create any operations.*

*Hints:*

- The expected Saliency Map output on a weasel (`laska.png` of class `"weasel"`) and a camel (`camel.png` of class `"Arabian␣camel,␣dromedary,␣Camelus␣dromedarius"`) towards their respective classes' scores (pre-softmax) are given below. The left image is the input, the middle is the saliency, the right is the saliency-masked input.




- For testing, use the pre-softmax scores instead of the post-softmax probability as the target of the explanation. You can use the names of tensors to get access to the input/output of a TensorFlow graph.

**Written Exercise 2.** [1 points] *Discuss and justify the use of pre-softmax score.*

**Written Exercise 3.** [1 points] *What would be a possible drawback of multiplying the input with the gradient ?*

## 1.2 Integrated Gradient

Integrated Gradient [3] aims to solve the vanishing gradient problem in Saliency Map, while it satisfies several desirable axioms.

**Definition 2** (Integrated Gradient)**.** *Consider a model $y = f(x)$ that takes an input $x \in \mathrm{R}^d$ and outputs $y$, a distribution of scores for each class. We denote the $y_c = f_c(x)$ as the scores for the class $c$ and $x_b$ as the baseline input. The Integrated Gradient $IG_c(x, x_b)$ for class $c$ is defined as*

$$IG_c(x, x_b) = (x - x_b) \odot \int_0^1 \nabla_x f_c(x_b + t(x - x_b))dt \tag{2}$$

*where $\odot$ denotes the element-wise multiplication. In the implementation, we use the following equation to approximate Eq. 2.*

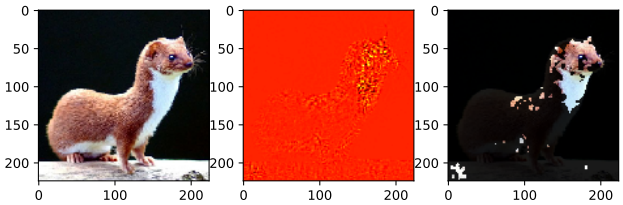
$$IG_c(x, x_b) \approx (x - x_b) \odot \frac{1}{N} \sum_{i=1}^{N} \nabla_x f_c \left( x_b + (x - x_b)\frac{i}{N} \right) \tag{3}$$

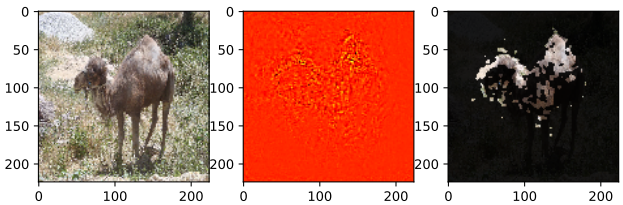
*where $N$ is the number of steps used for the approximation.*

**Coding Exercise 4.** [6 points] *Implement Def. 2 within the* `IntegratedGrad` *class of* `hw3_attribution.py` *(use the pre-softmax score as well). You should implement three kinds of baseline input:* `black` *for black image,* `white` *for white image and* `random` *for random noise sampled from an isotropic Gaussian with zero mean and standard deviation of* 0.1*. You should not create any computational graph nodes in* `__call__`*.*

*Hints:*

- The expected Integrated Gradients attribution of a weasel and a camel on their respective classes' scores (pre-softmax) are given below. The left image is the input, the middle is the saliency, the right is the saliency-masked input.



**Written Exercise 5.** [1 points] *Explain the* completeness *axiom which Integrated Gradient satisfies.*

**Written Exercise 6.** [2 points] *Given a model* $y = min(2x_1 + 3x_2, 1)$ *and a baseline* $x_1 = x_2 = 0$*, compute the integrated gradient for point* $x_1 = 2, x_2 = 1$*.*

### 1.3 Saliency in Linear Models

Given linear model for $d$-dimensional inputs defined by the $C$-class score function $f(x) \stackrel{\text{def}}{=} Wx+b$, analytically derive the attribution of score $f_c(x)$ in terms of vectors $x$, $b$ and matrix $W$ according to:

**Written Exercise 7.** [2 points] *Saliency Map method.*

**Written Exercise 8.** [3 points] *Integrated Gradients method with baseline* $x_b = \vec{0}$*.*

Also:

**Written Exercise 9.** [2 points] *Is it useful to ask for the same according to the Influence Directed Explanations method as described by Leino et al. [2] ?*

## Submission

Submission guidelines will be included in Part 2.

## References


[1] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Mueller. How to explain individual classification decisions, 2009.

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

[3] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.