# CS7643: Deep Learning
# Assignment 3

Instructor: Zsolt Kira

Deadline: Mar 6 8:00 AM ET

- This assignment is due on the date and time posted on Canvas. We will have a 48-hour grace period for this assignment. However, no questions regarding the assignment are answered during the grace period in any form.
- Discussion is encouraged, but each student must write his/her own answers and explicitly mention any collaborators.
- Each student is expected to respect and follow the GT Honor Code. **We will apply anti-cheating software to check for plagiarism**. Anyone who is flagged by the software will automatically receive 0 for the homework and be reported to OSI.
- Please **do not change the filenames and function definitions** in the skeleton code provided, as this will cause the test scripts to fail and you will receive no points on those failed tests. You may also **NOT** change the import modules in each file or import additional modules that are not native Python packages.
- It is your responsibility to make sure that all code and other deliverables are in the correct format and that your submission compiles and runs. We will not manually check your code (this is not feasible given the class size). Thus, **non-runnable code in our test environment will directly lead to a score of 0**. Also, be sure to clean up print statements, etc. before submitting – the autograder will likely reject your entire submission and we would not be able to grant you any points.

...there is still little insight into the internal operation and behavior of these complex models, or how they achieve such good performance. From a scientific standpoint, this is deeply unsatisfactory. With-out clear understanding of how and why they work, the development of better models is reduced to trial-and-error.

*Zeiler and Fergus. 2014*

Interpretability matters. In order to build trust in intelligent systems and move towards their meaningful integration into our everyday lives, it is clear that we must build 'transparent' models that have the ability to explain why they predict what they predict.

*Selvaraju et al. 2016*

# Overview

This assignment has two main parts. The first part explores the use of different types of saliency methods, which provide insight into the decision-making processes of image convolutional networks (CNNs). The second part involves using gradient manipulation techniques to extract the content and style from different images and combine them to produce creative works. If you are unfamiliar with these concepts, review relevant course material and the required readings for this assignment. The concepts covered will provide you with valuable tools for model analysis, including explainability, bias determinations, and debugging, which can be extended to other domains, such as text and audio.

For this assignment, we will use `conda`[1] to create a Python environment with the required packages installed.[2][3]

```
conda env create -f environment.yaml
conda activate cs7643-a3
```

## Deliverables

You must submit your write-up and solution code to **Gradescope** by the assigned due date and time. The Gradescope Autograder will test your submissions, but keep in mind that the feedback provided is limited. To ensure your code is functioning properly, you should write your own tests and assertions. We also provide samples of the deliverables for you to check against your work.

*Note:* The code you submit should be vectorized wherever possible. Code that uses non-vectorized constructs like for-loops may not pass the Autograder, particularly when executing the forward pass for multiple images.

To submit your code to Gradescope, you will need to upload a zipped file containing all

[1]For more information, refer to: https://conda.io/miniconda.html.

[2]Note we have successfully tested these specifications on several platforms (GPU not necessary). To create and activate the environment, run the following commands in your shell: *conda create –name <environment-name> –file requirements.txt* and *conda activate <environment-name>*. Be sure to verify the correct environment is active before executing any commands.

[3]On certain setups, it may be necessary to update Conda first with the command: *conda update conda*. If you receive openssl issues on Windows, make sure your 'base' environment is active before executing the create environment command.

your code with the folder structure intact. You can run *collect_submission.py* to automate this process. Once complete, upload *assignment_3_submission.zip* to Gradescope *Assignment 3 Code*.

*Note:* Although passing Gradescope tests is important, it does not guarantee that your code is free of bugs. An image that looks similar to a sample image may still be the result of a flawed implementation. To avoid this, diligently follow the recipes outlined in the skeleton code and referenced papers. Keep in mind that images output by your work will show minor random run-to-run variations, which is expected and should not be altered.

For your write-up, a report template called `report-template-a3.pptx` has been provided in the root directory (./). Please follow the instructions for each question carefully and be sure to include your visualizations and stylized images. To help manage your time effectively, we recommend allocating a maximum of 350 words per answer, unless otherwise specified. Points may be deducted for incorrect tagging or if your report exceeds the maximum word count.

Once you have completed your write-up, upload it to Gradescope *Assignment 3 Written*, ensuring that you assign the correct pages to the corresponding questions.[4]

# 1 Part I: Network Visual Inspections

In the first section of Part I, we will apply gradient and signal methods on top of SqueezeNet, a compact CNN model that achieves high performance on the ImageNet dataset while being significantly smaller than other models, such as VGG and ResNet. SqueezeNet has a file size of less than 5 megabytes, making it well-suited for deployment on memory-limited devices. According to the original paper, SqueezeNet achieves a top-1 accuracy of 60% and a top-5 accuracy of 80% on the ImageNet dataset.

In this section, we will implement the following techniques:

- **Class Model Visualizations**: We will synthesize an image to maximize the classification score of a particular class to provide insights into what the network focuses on when classifying images of that class.
- **Class-Specific Saliency Maps for Images**: We will generate image-specific saliency maps to quickly determine which parts of an image influenced the network's classification decision for a particular class.
- **GradCAM**: We will use Gradient Class Activation Mapping (GradCAM) to highlight the areas in an image that are most relevant to a given label.

## 1.1 Pre-requisite Reading

Before we dive into coding Part I, it is important to familiarize ourselves more with the key papers and ideas surrounding the approaches that will be implemented. Below are related papers that will provide you with a deeper understanding of the concepts and techniques that will be used. Once you have read the papers, answer the questions that

[4]Points may be deducted for incorrect tagging.

follow to solidify your understanding. Links to the papers can be found in the assignment materials or by conducting a quick search online.

- Matthew D Zeiler, Rob Fergus, "Visualizing and Understanding Convolutional Networks", Visualizing and Understanding Convolutional Networks, 2013.
- Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
- Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization", 2016

### 1.1.1 Questions

1. How does Zeiler's *deconvnets* work and how are they related to Simonyan's class visualization methods?
2. Explain how Simonyan et al. are able to produce a class model visualization starting with a zero image? How is this different than their image-specific class saliency method?
3. Why do Selvaraju et. al place such high value on the last convolutional layer? Why is GradCAM considered to be *class discriminative*?

## 1.2 Class Model Visualizations

First up are class model visualizations. This idea was first presented by Simonyan et al. and later extended by Yosinski et al.[5] to include regularization techniques that improve the quality of the generated image.

Concretely, let $I$ be an image and let $y$ be a target class. Let $s_y(I)$ be the score that a convolutional network assigns to the image $I$ for class $y$; note that these are raw unnormalized scores, not class probabilities. We wish to generate an image $I^*$ that achieves a high score for the class $y$ by solving the problem

$$I^* = \arg\max_I s_y(I) - R(I)$$

where $R$ is a (possibly implicit) regularizer (note the sign of $R(I)$ in the argmax: we want to minimize this regularization term). We can solve this optimization problem using gradient ascent, computing gradients with respect to the generated image. We will use L2 regularization (squared L2 norm) of the form:

$$R(I) = \lambda \|I\|_2^2 \equiv \lambda \sum_{i=1}^{n} x_i^2$$

[5] Yosinki et al., "Understanding Neural Networks Through Deep Visualization", ICML 2015 Deep Learning Workshop

**and** implicit regularization as suggested by [3] by periodically blurring the generated image. We can solve this problem using gradient ascent on the generated image.

Your tasks are as follows:

1. Follow the instructions in *visualizers/class_visualization.py* to implement functions that manually compute class visualizations.
2. Run `./class_visualization.py` and verify your plots were saved.

   ✓ `./visualization/class_visualization.png`

### 1.3 Saliency map

A saliency map tells us the degree to which each pixel in a given image affects the classification score for that image. To compute it, we compute the gradient of the unnormalized score corresponding to the correct class (which is a scalar) with respect to the pixels of the image. If the image has shape (3, H, W), then this gradient will also have shape (3, H, W); for each pixel in the image, this gradient tells us the amount by which the classification score will change if the pixel changes by a small amount at that pixel. To compute the saliency map, we take the absolute value of this gradient, then take the maximum value over the 3 input channels element-wise; the final saliency map thus has shape (H, W), and all entries are non-negative.

Your tasks are as follows:

1. Follow the instructions in `./saliency_visualization.py` and implement functions to manually create saliency maps.
2. Run `./saliency_visualization.py` and verify your plots were saved:

   ✓ `./visualization/saliency_visualization.png`

### 1.4 GradCAM

Gradient Class Activation Mapping (GradCAM)is a technique that highlights the regions of an image that the network uses to make a particular prediction using a heatmap overlaid on top of the source image. In this task, we will implement GradCAM to visualize the activation maps of various layers of a pre-trained convolutional neural network and examine the results to gain insights into the network's behavior.

Your tasks are as follows:

1. Follow the instructions in `./gradcam_visualization.py` and implement GradCAM.
2. Run *./gradcam_visualization.py* and verify your plots were saved:

   ✓ `./visualization/gradcam_visualization.png`

## 2 Part II: Style Transfer

Another task closely related to image gradients is style transfer. Style transfer is a technique that allows us to apply the style of one image to the content of another,

resulting in a new image that combines the two. This technique has become increasingly popular in computer vision and deep learning, as it allows us to generate blended images that combine the content of one image with the style of another. We will study and implement the style transfer technique from:

- Gatys et al., "Image Style Transfer Using Convolutional Neural Networks", CVPR 2015

The general idea is to take two images (a content image and a style image), and produce a new image that reflects the content of one but the artistic "style" of the other. We will do this by first formulating a loss function that matches the content and style of each respective image in the feature space of a deep network, and then performing gradient descent on the pixels of the image itself.

In this assignment, we will also use SqueezeNet as our feature extractor which can easily work on a CPU machine. Similarly, if computational resources are not any problem for you, a larger network which may enhance the visual outputs.

## 2.1 Content Loss

We can generate an image that reflects the content of one image and the style of another by incorporating both in our loss function. We want to penalize deviations from the content of the content image and deviations from the style of the style image. We can then use this hybrid loss function to perform gradient descent **not on the parameters** of the model, but instead **on the pixel values** of our original image.

Let's first write the content loss function. Content loss measures how much the feature map of the generated image differs from the feature map of the source image. We only care about the content representation of one layer of the network (say, layer $\ell$), that has feature maps $A^\ell \in \mathbb{R}^{1\times C_\ell \times H_\ell \times W_\ell}$. $C_\ell$ is the number of channels in layer $\ell$, $H_\ell$ and $W_\ell$ are the height and width. We will work with reshaped versions of these feature maps that combine all spatial positions into one dimension. Let $F^\ell \in \mathbb{R}^{N_\ell \times M_\ell}$ be the feature map for the current image and $P^\ell \in \mathbb{R}^{N_\ell \times M_\ell}$ be the feature map for the content source image where $M_\ell = H_\ell \times W_\ell$ is the number of elements in each feature map. Each row of $F^\ell$ or $P^\ell$ represents the vectorized activations of a particular filter, convolved over all positions of the image. Finally, let $w_c$ be the weight of the content loss term in the loss function.

Then the content loss is given by:

$L_c = w_c \times \sum_{i,j}(F_{ij}^\ell - P_{ij}^\ell)^2$

1. Implement Content Loss in *style_modules/content_loss.py*

You can check your implementation by running the 'Test content loss' function. The expected error should be 0.0

## 2.2 Style Loss

Now we can tackle the style loss. For a given layer $\ell$, the style loss is defined as follows:

First, compute the Gram matrix G which represents the correlations between the responses of each filter, where F is as above. The Gram matrix is an approximation to the covariance matrix – we want the activation statistics of our generated image to match the activation statistics of our style image, and matching the (approximate) covariance is one way to do that. There are a variety of ways you could do this, but the Gram matrix is nice because it's easy to compute and in practice shows good results.

Given a feature map $F^\ell$ of shape $(1, C_\ell, M_\ell)$, the Gram matrix has shape $(1, C_\ell, C_\ell)$ and its elements are given by:

$$G^\ell_{ij} = \sum_k F^\ell_{ik} F^\ell_{jk}$$

Assuming $G^\ell$ is the Gram matrix from the feature map of the current image, $A^\ell$ is the Gram Matrix from the feature map of the source style image, and $w_\ell$ a scalar weight term, then the style loss for the layer $\ell$ is simply the weighted Euclidean distance between the two Gram matrices:

$$L^\ell_s = w_\ell \sum_{i,j} \left(G^\ell_{ij} - A^\ell_{ij}\right)^2$$

In practice we usually compute the style loss at a set of layers $\mathcal{L}$ rather than just a single layer $\ell$; then the total style loss is the sum of style losses at each layer:

$$L_s = \sum_{\ell \in \mathcal{L}} L^\ell_s$$

1. Implement Style Loss in *style_modules/style_loss.py*

You can check your implementation by running the 'Test style loss' function. The expected error should be 0.0

## 2.3 Total Variation Loss

It turns out that it's helpful to also encourage smoothness in the image. We can do this by adding another term to our loss that penalizes wiggles or **total variation** in the pixel values. This concept is widely used in many computer vision task as a regularization term.

You can compute the total variation as the sum of the squares of differences in the pixel values for all pairs of pixels that are next to each other (horizontally or vertically). Here we sum the total-variation regularization for each of the 3 input channels (RGB), and weight the total summed loss by the total variation weight, $w_t$:

$L_{tv} = w_t \times \sum_{c=1}^{3} \left( \sum_{i=1}^{H} \sum_{j=1}^{W-1} (x_{i,j+1,c} - x_{i,j,c})^2 + \sum_{i=1}^{H-1} \sum_{j=1}^{W} (x_{i+1,j,c} - x_{i,j,c})^2 \right)$

You may not see this loss function referenced, but you should be able to implement it based on this equation.

**You should try to provide an efficient vectorized implementation.**

1. Implement Style Loss in *style_modules/tv_loss.py*

You can check your implementation by running 'Test total variation loss' function. The expected error should be 0.0

### 2.4 Style Transfer

You have implemented all the loss functions in the paper. Now we're ready to string it all together. Please read the entire function: figure out what are all the parameters, inputs, solvers, etc. **The update rule in function style_transfer of style_utils.py is held out for you to finish.**

As the final step, run the script *style_transfer.py* to generate stylized images.

### 2.5 Style Transfer - Unleash Your Creativity

You now have the structure built to transfer style from one image to another. For this section, select two images of your choosing - a content image and a style image - from any non-copyrighted source. Now affect a style transfer to generate a stylized image. Note that some image preprocessing may be required before your selected images are input into the code you developed above. You can examine images provided by us for the previous section to get an idea what may be required.

Include your two selected images (before) and the stylized image (after) in the report.

## 3 Wrap-up

Finally, choose one of the papers below to read and analyze with respect to this assignment. Then provide a short summary regarding the papers main contributions, followed by your observations and personal takeaways.

1. Mukund Sundararajan, Ankur Taly, Qiqi Yan, "Axiomatic Attribution for Deep Networks", ICML 2017
2. Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, Been Kim, "Sanity Checks for Saliency Maps", arXiv 2018
3. Quan Zheng, Ziwei Wang, Jie Zhou, Jiwen Lu. "Shap-CAM: Visual Explanations for Convolutional Neural Networks based on Shapley Value" arXiv 2022

## 4 Sample Outputs

We provide some sample outputs for your reference to verify the correctness of your code. Use these images to help you verify your approach is correct. The images will not match 100%, but should be very close in the effects produced and the textures and colors created.

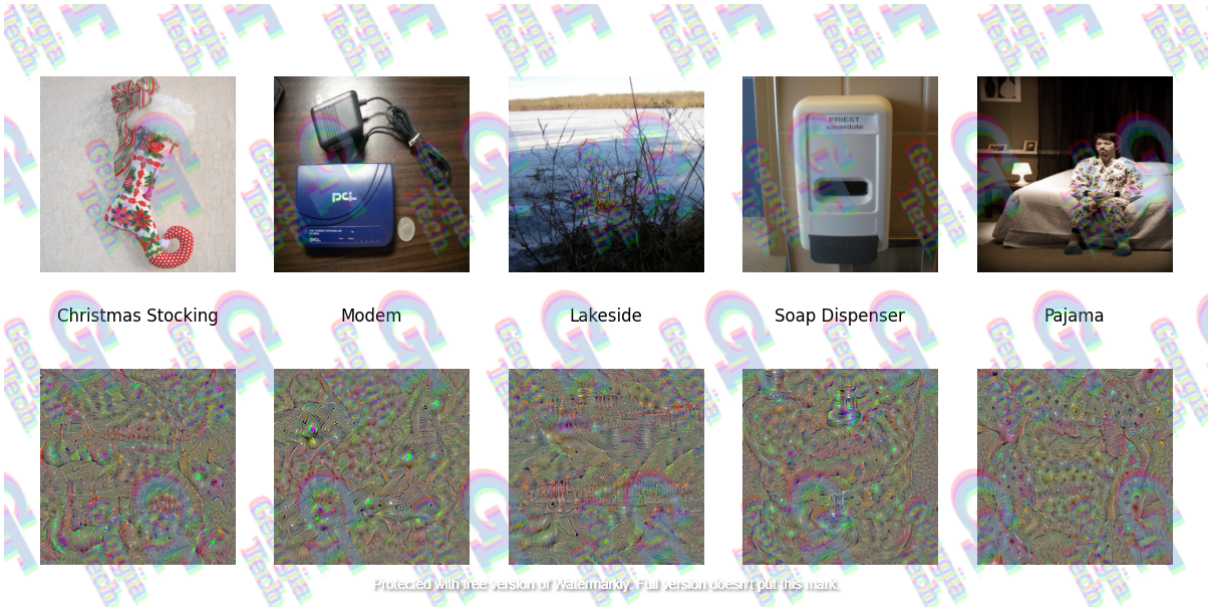


Figure 1: Example - Class Visualization


Figure 2: Example - Saliency Visualization

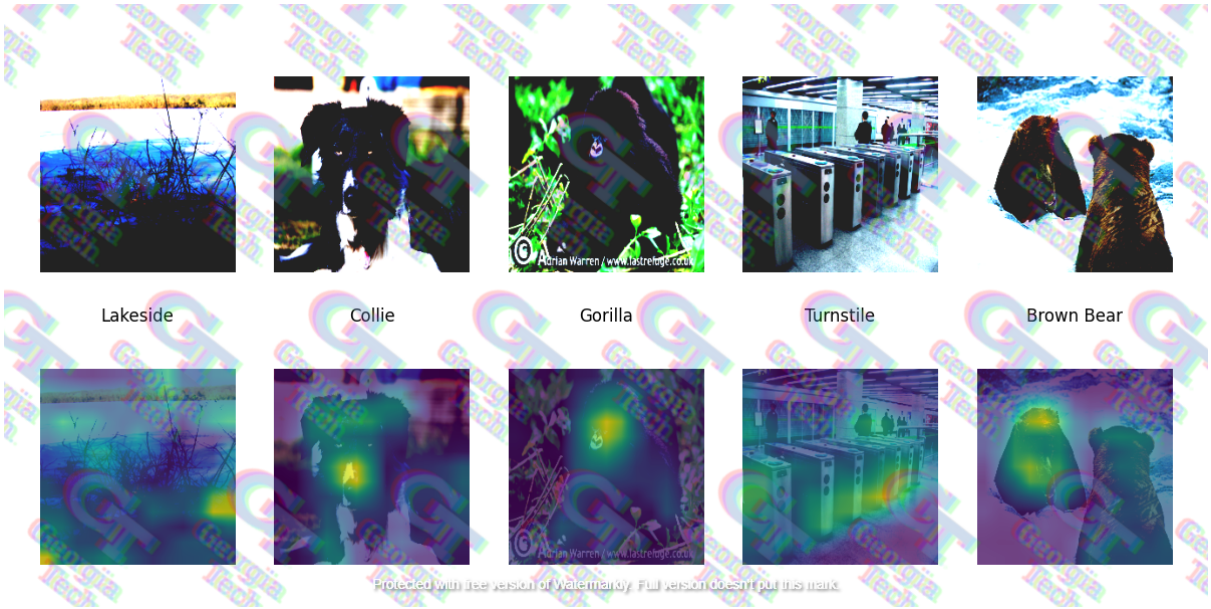


Figure 3: Example - GradCAM visualization


Figure 4: Example - Style Transfer


Figure 5: Example - Style Transfer


Figure 6: Example - Style Transfer