Assignment 3: Classification and Interpretability

Due date: TBA

Assignment by Anastasia Razdaibiedina

Submission: please submit a completed Google Colab notebook through Classroom. We provide you with a starter notebook that has code sections that you should fill in, and written questions that you should answer.

Google Colab notebook: https://drive.google.com/file/d/1T-ylxRFeeF1EbwcEOWPWvHDZpQxf-Zkc/view?usp=sharing

The Google Colab notebook contains all the detailed instructions, starter code and questions. **You should create a copy of the starter Google Colab notebook and finish all the tasks there**.

The assignments are individual work. You should attempt to answer all questions for this assignment. Most of the tasks can be completed at least partially even if you could not finish earlier questions. If you were unable to run the experiments, please discuss what outcomes you might hypothetically expect from the experiments. If you have any questions during completion of this assignment, please don't hesitate to contact TAs for help.

Introduction

Interpreting deep neural networks remains a challenging task in machine learning. Although neural networks are obtaining state-of-the art results for tasks involving complex datasets, we cannot always explain their output. In case of image classification, it is not always clear which pixels of the image are driving the prediction and what patterns the model has learned. In this assignment you will implement several *interpretability methods* and try to examine *what the neural network has learned*.

While working on this assignment, you will turn into a computational biologist whose task is to **classify and understand microscopy data**. Your dataset comes from an actual research lab from the University of Toronto [2].

In your Google Colaboratory notebook you will:

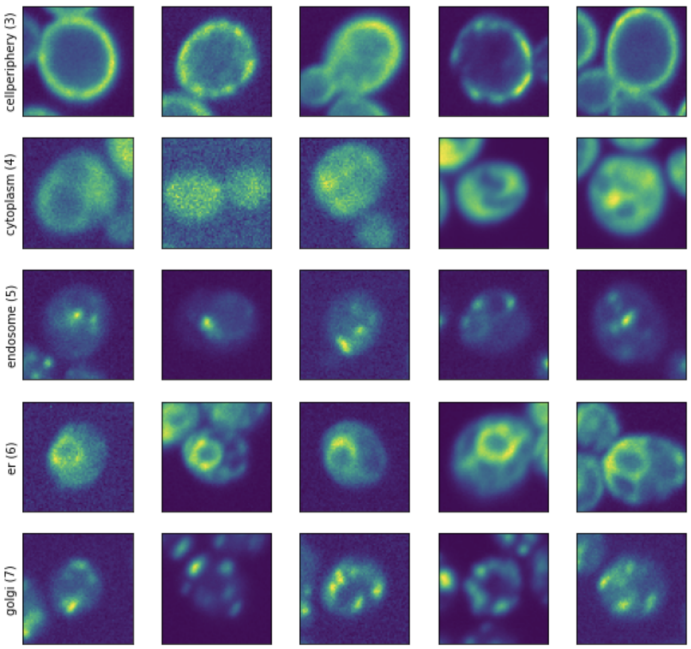
(1) implement and train from scratch a convolutional neural network to classify cellular components.
(2) investigate convolutional layers of the model through visualizing weight matrices and layer activations.

(3) understand what the network has learned using saliency maps.

Dataset

We will be classifying different parts of a cell using yeast single-cell data (Fig. 1) [2]. All the data that you need for this homework is shared with you, and you can access it through Google Drive (all the details are in Google Colab).

FIGURE 1. Example of images from classes 2-7, each row corresponding to different sub-cellular compartment. There are 19 compartment classes in total in the dataset.



The dataset has 19 classes in total, each class stands for a different cellular compartment. Various biological functions happen in different compartments, for example:

- cell periphery (class 3) is a part of the cell that interacts with outside environment, you can see how it's highlighted in green as a circular boundary of a cell.
- cytoplasm (class 4) corresponds to all contents of a cell, it suspends different organelles and supports the whole cell.

If we can correctly classify cellular compartments, we can better understand how cells work, respond to drugs and change over time.

Task 1: implementing and training the model

You can see that each compartment can be easily distinguished by eye. However, in large-scale studies it is not efficient to manually label the images, because dataset sizes can reach millions of data points. So we will train a convolutional neural network (CNN) to classify the images for us.

The starter code and exploratory data analysis are provided in Google Colab notebook. Your tasks are also outlined in the notebook:

- Task 1a (coding): fill in code for convolutional layer at the model initialization section.
- Task 1b (coding): fill in code for loss function definition.
- Task 1c (theory): answer the given questions.
  (1) Why do we have 19 nodes in our last layer?
  (2) Does max pooling layer have learnable parameters? Explain why.
  (3) Do you observe overfitting of your model? Justify your answer with a train / test accuracy plot that you produced.
  (4) Why do we need dropout layer?

Task 2: visualizing features

In task 1 you have trained the model and it should be reaching 70% test accuracy for 19 classes. We are happy about this result, but *we still don't know what the neural network is actually seeing.*

In tasks 2 and 3 you will try to understand what is happening inside the black box. We will start our model's interpretation through **visualizing features**. You can read more about feature visualization in this lecture: Interpretability lecture (U of T, Deep Learning Winter 2020).

In this task you will try two ways of model interpretation:

(1) **weight matrices**
(2) **layers activations**

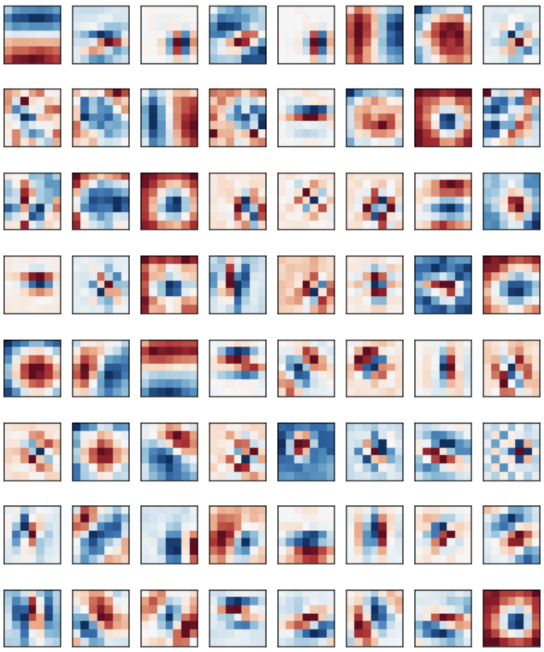
Weight matrices help to understand first-layer features if we visualize them. Since the first convolutional layer operates directly on the input image pixels, plotting its weights can show patterns that the network is trying to detect with its first layer [1].

Further layers are more challenging and cannot be interpreted just by looking at the deeper layers' weights because they operate in feature space (rather than pixel space).

As an example, you will plot weight matrices from the first convolutional layer of the pretrained model. In your Colab notebook you will load DenseNet121 model pretrained on ImageNet dataset, and plot the first layer's weights (Fig. 2). First convolutional layer of DenseNet121 network has 64 channels and kernel size is 7x7, hence in Fig.2 you can see 64 different filters, each of them detecting its own pattern (circles, lines, curves and so on).

Layer activations (also called feature maps or feature activations) help to investigate deeper convolutional layers [3, 1]. Layer activations are obtained by applying several rounds of convolutional filters to the image. Basically we are feeding an image to the network, but stop half-way to see intermediate output from prior conv layers.

FIGURE 2. Weight matrices from the 1st convolutional layer of the pretrained DenseNet-121 network.



Your tasks are outlined in the notebook:

- Task 2a (coding): define variable w in the starter code (it corresponds to 1st convolutional layer weights of the cell classifier model that you trained).
- Task 2b (coding): plot feature activations of 2nd convolutional layer of your CNN by finishing the provided starter code.
- Task 2c (theory): summarize in 3-4 sentences differences between model interpretation through weight matrices and feature activations.

Task 3: gradient maps

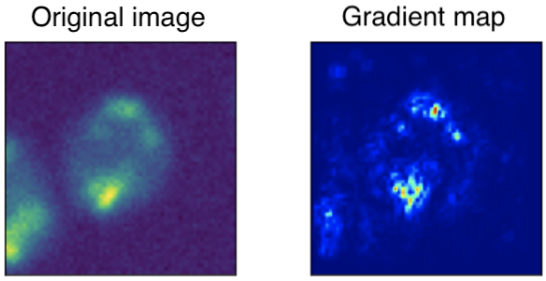
Finally, you will investigate model through **gradient maps**. Gradient map (also called **sensitivity map**) *shows how much individual pixels in an image contribute to the prediction.*

In this part of the assignment, you will learn about:

- **Vanilla gradient map** [4]
- **SmoothGrad** [5]

Vanilla gradient map shows how much every pixel of the image drives the prediction [4]. Gradient maps look like heatmaps with hot regions standing for regions that contain most predictive patterns (Fig. 3). Let's take a look at mathematics behind that.

FIGURE 3. Gradient map for one of the cell images from yeast single-cell dataset.



In task 1 you have trained a convolutional neural network that gives score $S_c(I)$ for an image I belonging to a class c ($S_c(I)$ is logit). Final classification result will be class with maximal score:

$$class(I) = argmax_{c \in C} S_c(I)$$

In case of CNNs, score $S_c(I)$ is a highly non-linear function of the input image I, but we can approximate it with first-order Taylor expansion:

$$S_c(I) \approx w^T I + b$$

Hence, **class sensitivity map** for image I is defined as derivative of the predicted class logit to the input image:

$$M_c = \frac{\partial S_c}{\partial I}$$

Sensitivity map M_c will have the same shape as image I and basically answers the question *How much does change in every pixel makes change in network's prediction of class c?* You can read more about gradient maps in this paper.

SmoothGrad is a more advanced approach for getting sensitivity maps that makes gradient map sharper and less noisy [5]. **SmoothGrad works by averaging several gradient maps** the following way:

(1) make N slightly different copies of the same image by adding Gaussian noise to the input image;
(2) compute N vanilla gradient maps for the "noisy" images;
(3) get an averaged gradient map from N vanilla gradient maps.

Mathematically SmoothGrad sensitivity map can be written as:

$$\hat{M}_c(I) = \frac{1}{n} \sum_{1}^{n} M_c(I + \mathcal{N}(0, \sigma))$$

Where I is our input image and M_c is vanilla gradient map (defined above). The idea behind SmoothGrad is to make the sensitivity map robust to noise: pixels which are important for the prediction will get high values in gradient maps from different noise additions. For better understanding of the material it would be useful to read the original SmoothGrad paper.

In your Google Colab notebook you have several exercises that will help you to get practical experience with gradient maps:

- Task 3a (coding): in your Colab notebook finish the function for calculating vanilla gradient map.
- Task 3b (coding): in your Colab notebook finish the function for calculating SmoothGrad gradient map.
- Task 3c (theory): answer the following question:
  - discuss advantages of sensitivity maps for model interpretation;
  - explain why SmoothGrad is better than vanilla gradient maps.

What to submit

For this assignment you should complete tasks 1a-c, 2a-c and 3a-c in the Google Colab notebook. Please copy the notebook to your Google Drive and finish the assignment there. Once completed, submit the final .ipynb file through Classroom. You should answer the theoretical questions also in the notebook according to the instructions.

References

[1] Jimmy Ba. University of Toronto course CSC413/2516 Lecture 6: Interpretability. https://csc413-2020.github.io/assets/slides/lec06.pdf, 2020. [Online; accessed Feb 2020].

[2] Oren Z Kraus, Ben T Grys, Jimmy Ba, Yolanda Chong, Brendan J Frey, Charles Boone, and Brenda J Andrews. Automated analysis of high-content microscopy data with deep learning. *Molecular systems biology*, 13(4):924, 2017.

[3] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[4] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[5] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.