

MIT CSAIL
6.8300/1 Advances in Computer Vision
Spring 2023

Problem Set 4

Posted: Tuesday, Mar 14, 2023

Due: Tuesday 11:59 pm, Mar 21, 2023

The relevant material for Pset 4 was covered in Lectures 9 and 10.

6.8300 students are expected to finish **all** problems.

6.8301 students are expected to finish all problems, **except 2c, 2d, 4d**.

Please note that 6.8301 students will *not* receive credit for 2c, 2d, 4d.

We provide a Python notebook with the code to be completed. You can run it locally or on Google Colab. To use Colab, upload it to Google Drive and double-click the notebook (or right-click and select Open with Google Colaboratory), which will allow you to complete the problems without setting up your own environment.

Submission Instructions: We will be using Gradescope for the remainder of the term. To submit Pset 4, just navigate to the assignment submission window on Canvas like you did for the previous problem sets and you will find a link that takes you to Gradescope. Unlike Canvas, which requires PDF files, Gradescope allows us to view Python notebooks directly. For Pset 4, all you will need to submit is your Python notebook on Gradescope. Please make sure to run all the cells before submitting your notebook, so that we can inspect and grade your output in addition to your code. If a problem requires you to write text or you would like to write comments, the easiest way to do so is by adding a new text cell and writing into it.

Attention: Failure to follow the submission instructions will result in point deductions. For example, not running all the cells before submitting your notebook or submitting a zip file instead of just your notebook will be penalized.

Late Submission Policy: If your problem set is submitted within 7 days (rounding up) of the original deadline, you will receive partial credit. Such submissions will be penalized by a multiplicative coefficient that linearly decreases from 1 to 0.5.

In this problem set, you will be experimenting with neural networks using PyTorch. To reduce training time, we recommend using GPU acceleration. Colab comes with free GPU support. On Colab, select GPU as your runtime type as follows:

Runtime → **Change runtime type** → **Hardware accelerator** → **GPU** → **Save**.

If you exceed your GPU usage limit on Colab, don't fret. You can also complete your problem set using a regular CPU in a very reasonable amount of time.

Problem 1 *Filter Visualization* (2 points)

Convolutional kernels in different layers are used to extract information from the input image at different levels. In this problem, we will focus on visualizing the filters of the ResNet network pretrained on the Places365 dataset. You must download the model and the pretrained weights first.

- (a) (1 pt) Implement `normalize_tensor` to normalize the input kernel for better visualization.
- (b) (1 pt) Visualize filters for another convolutional layer in ResNet (pure black is not acceptable).

Problem 2 *Internal Activation Visualization* (6.8301: 2 points, 6.8300: 5 points)

In this problem, we will visualize the activations of the internal units in the model. We will chop the fully connected layers of ResNet and visualize the activations of the last convolutional layer into different locations of the input image.

- (a) (1 pt) Create a version of the ResNet model without the last two layers by completing `remove_last_layers` to expose the last convolutional layers in the `generate_featuremap_unit` function.
- (b) (1 pt) As you can see, unit 300 activates the mountain part of the image. Find other units that detect 1) the sky and 2) buildings in the image.
- (c) (1.5 pts) **[6.8300 only]** Each unit contributes differently to the final prediction. The contribution weight for each unit is determined by the weights of the fully connected layer (last layer in ResNet). If we deactivate top-5 units (force the kernel to be zeros) that have the maximum weights for the top 1 category of some input image, the prediction for that category will drop dramatically. The code to find the index of those units is given, try to deactivate those units and compare the difference between the original and new top 5 predictions. **Show the feature map of that unit.**
- (d) (1.5 pts) **[6.8300 only]** In (c), we deactivate top-5 units and observe dramatic changes in predictions. Try to deactivate fewer units and. **Report the lowest number of dropped units required to change the top prediction class.**

Problem 3 *Class Activation Map (CAM)* (3 points)

So far, we know the output of the last convolutional layer activate different parts of the input image. In this problem, we will explore how to visualize which parts of the image are responsible for the final decision. Use the chopped ResNet from problem 2 as the model.

- (a) (1 pt) Running the chopped ResNet will produce a tensor with the size of $(1 \times 2048 \times 8 \times 8)$. 1 is the batch size (b), 2048 is the number of channels (c) and 8 & 8 is the height (h) and width (w) of the kernel. Convert the output tensor to a new tensor with the size of $(hw \times c)$.

(b) (1 pt) Feed the new tensor from (a) to the fully connected layer of ResNet to compute the weighted average. You will get a output tensor with the size of $(hw \times 365)$, where 365 is the number of classes in the Places365 Dataset.

(c) (1 pt) Show the feature maps (combined with input image) of the top 5 predicted categories and explain the relationship between the feature activation and the corresponding category.

Problem 4 *Network Training* (6.8301: 3 points, 6.8300: 5 points)

The goal of this problem is to train a small convolutional neural network to classify images of clothing items from the FashionMNIST dataset. You'll first fill in critical components of a simple PyTorch training pipeline, evaluate the model on the test set, and explore the impact of specific design choices and hyperparameters on the model's performance.

(a) (1 pt) Read and execute the notebook cells until you reach the training loop section. Here you will complete the function `train` that trains a network for 1 epoch by filling in the missing code snippets.

(b) (1 pt) Implement the `get_prediction` function which returns the index of the predict class given an image and a network and is called during the evaluation routine. Also implement the accuracy computation in the `(evaluate)` function

(c) (1 pt) Here, we bring all of the pieces together to train the network. Instantiate an optimizer to update the weights of the network during training. Run training and validation for 10 epochs and report your final validation accuracy.

(d) (2 pts) **[6.8300 only]** We want you to get a feel for the impact of specific design choices on the performance of the network. Experiment with two or more of the following hyperparameters / techniques:

- Data augmentation
- Weight initialization
- Number of layers, or number of layer features
- Type of optimizer
- Learning rate and/or schedule
- Regularization

For the techniques you choose, plot the top-1 accuracy of your modified network against the top-1 accuracy of the original network for both the training and validation sets. Try several different hyperparameter values! For example, if you choose to modify the learning rate, you can plot a chart of learning rate vs. top-1 accuracy. Briefly describe the techniques you tried, and suggest an explanation for your results. **Full credit will only be given if at least 90% validation accuracy is achieved (i.e., the accuracy must be above 89.9999999%).**