

# Homework 3

## CS 5787 Deep Learning

### Spring 2023

Instructor: Prof. Alejandro (Alex) Jaimes

Discussion Section Leader: Jack Morris

TAs: Andrew Bennett, Yair Schiff

Graders: Eylul Ertay, Anastasiia Sorokina, Zhengchun Shang, Jamie Cao

**Due: 3/31/23, midnight**

Your homework submission must cite any references used (including articles, books, code, websites, services (e.g., ChatGPT, GitHub copilot, etc), and personal communications). All solutions must be written in your own words, and you must program the algorithms yourself. **If you do work with others, you must list the people you worked with.** Submit your solutions as a PDF to Canvas.

Your homework solution must be typed, you may use  $\text{\LaTeX}$  if you wish. Homework must be output to PDF format. I suggest using <http://overleaf.com> if you use  $\text{\LaTeX}$  to create your document. Overleaf is free and can be accessed online.

Note that this assignment contains two problems for 40 total bonus points, while the required questions add up to 110 points.

This HW accounts for 15% of the total grade.

**If told to implement an algorithm, don't use a toolbox, or you will receive no credit. Do not post your code to a public web repository (e.g., GitHub).**

## Problem 1 - BatchNorm (5 points)

Discuss the pros and cons of using BatchNorm. When does BatchNorm fail? Discuss alternatives and their pros and cons. What properties would an ideal normalization method have?

**Solution:**

## Problem 2 - Using a Pre-Trained CNN (10 points total)

For this problem you must use PyTorch. For this problem you will use a CNN that has been trained on ImageNet-1k. You may use any pre-trained architecture developed from 2014 onwards (e.g., not AlexNet, but a VGG or ResNet model would be fine). The CNN needs to produce  $d$ -dimensional embeddings that are fed into the 1000-way output layer, where  $d$  is typically 512, 2048, or 4096. You will be re-using the CNN you select in the next problem for transfer learning with embeddings.

### Part 1 - Using Pre-Trained Deep CNN (5 points)

Run it on `peppers.jpg`. Output the top-3 predicted categories and the probabilities. State the pre-trained CNN model that you used.

Make sure to pre-process the input image appropriately. Look at the toolbox documentation for the pre-trained model you use to determine how to do this.

**Solution:**

### Part 2 - Visualizing Feature Maps (5 points)

Write code to visualize the feature maps in the network as images. You will likely need to normalize the values between 0 and 1 to do this.

Choose five interesting feature maps from early in the network, five from the the middle of the network, and and five close to the end of the network. Display them to us and discuss the structure of the feature maps. Try to find some that are interpretable, and discuss the challenges in doing so.

**Solution:**

## Problem 3 - Transfer Learning with a Pre-Trained CNN (20 points)

For this problem you may use a toolbox (and you must use PyTorch). We will do image classification using the Oxford Pet Dataset. The dataset consists of 37 categories with about 200 images in each of them. You can find the dataset here: <http://www.robots.ox.ac.uk/~vgg/data/pets/>

Rather than using the final ‘softmax’ layer of the CNN as output to make predictions as we did in problem 2, instead we will use the CNN as a feature extractor to classify the Pets dataset. For each image, grab features from the last hidden layer of the neural network, which will be the layer **before** the 1000-dimensional output layer (around 500–6000 dimensions). You will need to resize the images to a size compatible with your network (usually  $224 \times 224 \times 3$ , but look at the documentation for the pre-trained system you selected). You should grab the output just after the last hidden layer or after global pooling (if it is 1000-dimensional, you will know you did it wrong).

After you extract these features for all of the images in the dataset, normalize them to unit length by dividing by the  $L_2$  norm. Train a linear classifier of your choice<sup>1</sup> with the training CNN features, and then classify the test CNN features. Report mean-per-class accuracy and discuss the classifier you used.

**Solution:**

## Problem 4 - Training a Small CNN (55 points total)

### Part 1 (25 points)

For this problem you must use a toolbox. Train a CNN with three hidden convolutional layers that use the Mish activation function. Use  $32 \ 7 \times 7$  filters for the first layer, followed by  $2 \times 2$  max pooling (stride of 2). The next two convolutional layers will use  $64 \ 3 \times 3$  filters followed by the Mish activation function. Prior to the softmax layer, you should have an average pooling layer that pools across the preceding feature map. Do not use a pre-trained CNN. AdamW is the recommended optimizer, but you may try Ranger, RAdam, or something else more recent if you would like, but you will probably have to add in an external toolbox to do this. Tune your hyperparameters on a small version of the dataset to make sure your training loss goes down and things are set up correctly.

Train your model using all of the CIFAR-10 training data, and evaluate your trained system on the CIFAR-10 test data.

---

<sup>1</sup>You could use the softmax classifier you implemented for homework 1 or any toolbox you prefer.

Visualize all of the  $7 \times 7 \times 3$  filters learned by the first convolutional layer as an RGB image array (I suggest making a large RGB image that is made up of each of the smaller images). This visualization of the filters should be similar to the ones we saw in class. Note that you will need to normalize each filter to display them. Let  $\mathbf{H}_t$  be the  $t$ 'th filter learned by the network. The normalized version  $\mathbf{H}'_t$  for *visualization* is then given by

$$\mathbf{H}'_t = \frac{\mathbf{H}_t}{2(\|\mathbf{H}_t\|_\infty + \epsilon)} + \frac{1}{2},$$

where  $\epsilon > 0$  (e.g.,  $\epsilon = 0.0001$ ). This will make all values in the filter be values from 0 to 1 so that they can be displayed as an RGB image.

Display the training loss as a function of epochs. What is the accuracy on the test data? How did you initialize the weights? What optimizer did you use? Discuss your architecture and hyper-parameters.

**Solution:**

IMAGE SHOWING THE FILTERS  
 TRAINING LOSS AS FUNCTION OF EPOCHS  
 TEST DATA ACCURACY  
 WEIGHT INITIALIZATION INFORMATION  
 DESCRIBE HYPER-PARAMETERS

## Part 2 (20 points)

Using the same architecture as in part 1, add in batch normalization between each of the hidden layers. Compare the training loss with and without batch normalization as a function of epochs. What is the final test accuracy? Visualize the filters.

**Solution:**

## Part 3 (10 points)

Can you do better with a deeper and better network architecture? Optimize your CNN's architecture to improve performance. You may get significantly better results by using smaller filters for the first convolutional layer. Describe your model's architecture and your design choices. What is your final accuracy?

Note: Your model should perform better than the one in Part 1 and Part 2.

**Solution:**

## Problem 5 - Vision Transformers (20 points)

In this problem, you will redo problem 3, but instead of transfer learning with a CNN, you will do transfer learning with a Vision Transformer. Identify a strong vision transformer architecture for transfer learning that was pre-trained on ImageNet-1K (note that some pre-trained models are pre-trained on larger datasets). We suggest using Swin Transformers or later incarnations of them. You will have to download the pre-trained model and the appropriate code.

Briefly discuss the Vision Transformer you selected. Compare performance of the Vision Transformer to the CNN you used earlier in terms of overall performance. Are there some images that the CNN gets correct but the Swin Transformer misclassifies, and vice versa?

**Solution:**

## Code Appendix

## BONUS PROBLEMS

### Problem 6 - BatchNorm - Investigations (10 bonus points)

In this **bonus** problem, you will study BatchNorm's properties for a single 'dot product' neuron, but the results are the same if you have multiple neurons in a layer or use convolutional units.

Let

$$x_i = \mathbf{w}^T \mathbf{h}_i + b$$

be the output of the neuron, where  $\mathbf{h}_i \in \mathcal{R}^d$  is a vector of inputs to the neuron,  $\mathbf{w} \in \mathcal{R}^d$  are the weights of the neuron, and  $b$  is the bias. BatchNorm is applied in the subsequent 'layer' before the non-linearity. After computing the activation of the neuron, the BatchNorm transformation of the output activations is given by

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}},$$

where the mean of the activation in the mini-batch is given by

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i,$$

the variance is given by

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2,$$

and where  $m$  is the mini-batch size.

Substitute in the neuron's activation function into the BatchNorm equations and simplify. What does this tell you about how BatchNorm impacts the weights and the bias of the neuron? How should you adjust your neural network's architecture if using BatchNorm?

**Solution:**

## Problem 7 - Fooling Convolutional Neural Networks (30 bonus points)

In this problem you will fool the pre-trained convolutional neural network of your choice. One of the simplest ways to do this is to add a small amount of adversarial noise to the input image, which causes the correct predicted label  $y_{true}$  to switch to an incorrect adversarial label  $y_{fool}$ , despite the image looking the same to our human visual system.

### Part 1 (20 bonus points)

More formally, given an input image  $\mathbf{X}$ , an ImageNet pre-trained network will give us  $P(y|\mathbf{X})$ , which is a probability distribution over labels and the predicted label can be computed using the argmax function. We assume the network has been trained to correctly classify  $\mathbf{X}$ . To create an adversarial input, we want to find  $\hat{\mathbf{X}}$  such that  $\hat{\mathbf{X}}$  will be misclassified as  $y_{fool}$ . To ensure that  $\hat{\mathbf{X}}$  does not look radically different from  $\mathbf{X}$  we impose a constraint on the distance between the original and modified images, i.e.,  $\|\mathbf{X} - \hat{\mathbf{X}}\|_{\infty} \leq \epsilon$ , where  $\epsilon$  is a small positive number. This model can be trained using backpropagation to find the adversarial example, i.e.,

$$\hat{\mathbf{X}} = \arg \min_{\mathbf{X}'} \left( Loss(\mathbf{X}', y_{fool}) + \frac{\lambda}{2} \|\mathbf{X}' - \mathbf{X}\|_{\infty} \right),$$

where  $\lambda > 0$  is a hyperparameter and  $\|\cdot\|_{\infty}$  denotes the infinity norm for tensors.

To do this optimization, you can begin by initializing  $\mathbf{X}' \leftarrow \mathbf{X}$ . Then, repeat the following two steps until you are satisfied with the results (or convergence):

$$\begin{aligned}\mathbf{X}' &\leftarrow \mathbf{X}' + \lambda \frac{\partial}{\partial \mathbf{X}'} P(y_{\text{fool}} | \mathbf{X}') \\ \mathbf{X}' &\leftarrow \text{clip}(\mathbf{X}', \mathbf{X} - \epsilon, \mathbf{X} + \epsilon)\end{aligned}$$

where the `clip` function ‘clips’ the values so that each pixel is within  $\epsilon$  of the original image. You may use the neural network toolbox of your choice to do this optimization, but we will only provide help for PyTorch. You can read more about this approach here: <https://arxiv.org/pdf/1707.07397.pdf>. Note that the details are slightly different.

Demonstrate your method on four images. The first image should be ‘peppers,’ which was used in an earlier problem. Show that you can make the network classify it as a space shuttle (ImageNet class id 812). You can choose the other three photos, but ensure that they contain an ImageNet object and make the network classify it as a different class. Ensure that the pre-trained CNN that you use outputs the correct class as the most likely assignment and give its probability. Then, show the ‘noise image’ that will be added to the original image. Then, show the noise+original image along with the new most likely class and the new largest probability. The noise+original image should be perceptually indistinguishable from the original image (to your human visual system). You may use the ImageNet pre-trained CNN of your choice (e.g., VGG-16, ResNet-50, etc.), but mention the pre-trained model that you used. You can show your results as a  $4 \times 3$  array of figures, with each row containing original image (titled with most likely class and probability), the adversarial noise, and then the new image (titled with most likely class and probability).

**Solution:**

## Part 2 (10 bonus points)

The method we deployed to make adversarial examples is not robust to all kinds of transformations. To examine the robustness of this, take the four adversarial images you created in part 1 and show how the following image manipulations affect the predicted class probabilities: mirror reflections (flip the image), a crop that contains about 80% of the original object, a 30 degree rotation, and converting the image to grayscale and then replicating the three gray channels to make a faux RGB image. Show the modified adversarial images and title them with the new most likely class and probabilities. Discuss the results and why you think it did or did not work in each case.

**Solution:**