CSE 151B: Programming Assignment 3

Fall 2020

Instructions

Due Sunday, November 15^{th}:

1. Please submit your assignment on Gradescope. There are two components to this assignment: written homework (Problems I.1-6), and a programming part. As in previous assignments, your report for the programming part will be written using LaTeXor Word in **NeurIPS format** (NeurIPS is the top machine learning conference, and it is now dominated by deep nets - it will be good practice for you to write in that format!). The templates, both in **Word** and LaTeX are available from the 2015 NeurIPS format site.

2. For the programming part, please work in pairs or triples. I would very much prefer that you not request different team sizes!! If you think you have a good argument for such, please discuss your circumstances with your TA, who will then present your case to me. Again, **don't forget to include a paragraph for each team member in your report that describes what each team member contributed to the project.**

3. You need to submit all of the source codes files and a *readme.txt* file that includes detailed instructions on how to run your code.

   You should write clean code with consistent format, as well as explanatory comments. Do not submit any of your output plot files or .pyc files, just the .py files and a readme.txt that explains how to run your code.

4. Using any off-the-shelf code is strictly prohibited.

5. Any form of copying, plagiarizing, grabbing code from the web, having someone else write your code for you, etc., is cheating. We expect you all to do your own work, and when you are on a team, to pull your weight. Team members who do not contribute will not receive the same scores as those who do. Discussions of course materials and homework solutions are encouraged, but you should write the final solutions to the written part alone. Books, notes, and Internet resources can be consulted, but not copied from. Working together on homework must follow the spirit of the **Gilligan's Island Rule** (Dymond, 1986): No notes can be made (or recording of any kind) during a discussion, and you must watch one hour of Gilligan's Island or something equally insipid before writing anything down. Suspected cheating has been and will be reported to the UCSD Academic Integrity office.

6. **Start early!** If you have any questions or uncertainties about the assignment instructions in Part 1 or Part 2, please ask about them as soon as possible (preferably on Piazza, so everybody can benefit). We want to minimize any possible confusion about this assignment and have tried very hard to make it understandable and easy for you to follow.

7. You will be using PyTorch (v 1.2.0 or above), a Deep Learning library, for the tasks in this assignment. Also you are allowed to only use Numpy, Matplotlib and PIL Imaging library apart from the libraries that come along with the Python installation. Steps to access the UCSD GPU cluster are explained in the following sections.

Learning Objectives

1. Understand the basics of convolutional neural networks, including convolutional layer mechanics, max-pooling, and dimensions of layers.
2. Build a custom CNN architecture to classify North American birds.
3. Build intuition on the effect of different building blocks used in a CNN model
4. Learn best practices of transfer learning model by fine tuning a model to classify birds.
5. Visualize and interpret learned activation and weight maps of a CNN.

Part I

Understanding Convolutional Network Basics

This portion of the assignment is to build your intuition and understanding the basics of convolutional networks - namely how convolutional layers learn feature maps and how max pooling works - by manually simulating these. We expect each team member to complete this task individually

For questions 1-4, consider the $(5 \times 5 \times 2)$ input with values in range $[-1, 1]$ and the corresponding $(3 \times 3 \times 2)$ filter shown below. You can assume the bias for each is 0. Each filter consists of kernels. Here, we have two kernels. Each kernel dimension is $(3 \times 3 \times 1)$. Also, in general, a filter bank is a collection of many filters. In this case our filter bank has a single filter. (**Note**: Don't confuse with kernels in Pytorch. In Pytorch, they use use the word "kernel" to denote "filter bank")

1	0	1	-1	1
1	1	0	0	0
-1	0	1	0	-1
0	1	1	0	-1
1	1	0	0	-1

1	0	-1
1	0	-1
1	0	-1

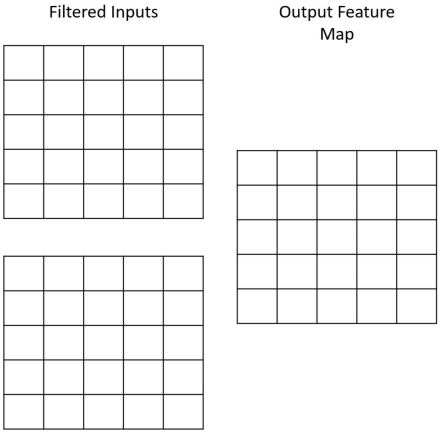
1	0	-1	0	1
1	0	-1	0	-1
-1	1	1	1	0
0	0	1	0	0
1	-1	0	-1	1

1	1	1
0	0	0
-1	-1	-1

Input Feature maps

Filter

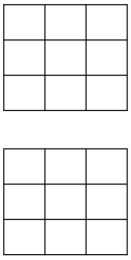
1. In the provided area below, fill in the values resulting from applying a convolutional layer to the input with no zero-padding and a stride of 1. Calculate these numbers before **any activation function** is applied. If there are any unused cells after completing the process in the provided tables, place an $\times$ in them. (**Note: filtered inputs** are the output you get right after performing convolution operation between input feature maps and the filter. Output feature map is the output after **combining** both the filtered inputs) (3 pts)



2. What kind of features do the kernels in Part 1 detect? (1 pt)

3. Think about what ideal 3×3 patch from each of the input channels would maximally activate the corresponding 3×3 filter. Fill in these maximally activating patches in the area below. If there are any unused cells after completing the process in the provided tables, place an $\times$ in them. (2pts)

Maximally
Activating Patch



4. **Spatial pooling:** Using your **output feature map** in question 1, apply *max-pooling* using a $[2 \times 2]$ kernel with a stride of 1. Recall from lecture that spatial pooling gives a CNN invariance to small transformations in the input, and max-pooling is more widely used over sum or average pooling due to empirically better performance. (2 pts)

Max Pooled
Output

5. **Dilated Convolutions:** Maxpooling helps the network learn the global context of input feature maps but loses spatial resolution. Dilated convolutions allow for larger filters without adding additional parameters. You can read more on Dilated convolutions here. Consider the same input feature map used for question 1. Also consider the same filters provided for question 1. Fill in the values below after dilated convolution (with dilation factor 2) between the input feature map and the filters. (2 pts)

Filtered Inputs Output feature Map

6. **Number of learnable parameters and feature map dimensions**
Suppose we had the following architecture:

```
inputs -> conv1 -> conv2 -> maxpool -> conv3 -> fc1 -> fc2 (outputs)
```

All convolutions have stride 1 and no zero padding unless mentioned. conv1 has a 5x5 kernel with 5 output channels. conv2 has a 7x7 kernel with 10 output channels. maxpool has 3x3 kernel with stride 2. conv3 has single output channel with a 13x13 kernel and stride=10. fc2 has 512 input units. You are supposed to predict 10 class outputs.
If ReLU is the activation function between each layer and the inputs are $[256 \times 256]$ RGB images, what are:

(a) **conv1**, **conv2**, **conv3** filter shape

(b) **conv1** output feature map shape

(c) **conv2** output feature map shape

(d) **maxpool** output feature map shape

(e) **conv3** output feature map shape

(f) Number of input and output units of **fc1** and **fc2** layer

Show your work with calculations (6 pts)

Part II
Deep Convolutional Network for Bird Classification

Problem statement: Deep convolutional neural networks have been applied to a broad range of problems and tasks within Computer Vision. In this part of the assignment, we will explore and build an image classification Deep CNN model to classify birds. We will try this with our own CNN model developed from scratch as well as use two of the pre-trained models and fine-tune them to our use case.

We will be using **Caltech-UCSD birds dataset** for training our model. This dataset has images of 200 categories of North American birds. The dataset has around 12,000 images out of which 6000 are training images and the rest are test images. We will be training a CNN model to classify just 20 categories instead of all 200 mainly due to resource constraints on the UCSD GPU cluster.

Implementation Instructions

Please familiarize yourself with Pytorch before implementing. One of the good places to start with Pytorch is this. Clone the PA repository from here We have provided you with a barebone datasets class (**datasets.py**), CNN model class (**model.py**) and a train script (**train.ipynb**). This is to help your file structure organization. You can add other files if necessary
The following are the implementation steps to follow.

1. **Evaluation metrics**
   (a) **Evaluating your model:**
   Before we discuss the neural network details, we must clarify how you will accurately and transparently evaluate your model's performance. Since there is no class imbalance problem, you will be just using the following metric:

   Accuracy: percent correct predictions $= \frac{\text{total correct predictions}}{\text{total number of samples}}$

2. **Dataset loading**
   (a) The first step for training any model is preparing your dataset. The dataset is on the github repo link posted above. The dataset location is **birds_dataset**/ - relative to the main repository. The content of this location is as below
      i. **images** - Folder. All the train and test images are under it
      ii. **classes.txt** - Names of all the classes
      iii. **test_list.txt** - Contains lines of tuples (*path,label*) for test images. where path is the relative path of the image from **image** folder. label is the class number in the range [0-19].
      iv. **train_list.txt** - Contains lines of tuples (*path,label*) for train images. where path is the relative path of the image from **image** folder. label is the class number in the range [0-19].
   (b) You have to write the dataset class under **datasets.py**. In particular you need to implement **__getitem__** and **__len__** methods under it. You can do all your data pre-processing (normalization, cropping, resizing etc) here.(**Note:** The Train and test datasets contain RGB images.)
   (c) Once the dataset object is created, you need to create a dataloader in **train.py**. This dataloader is used to iterate over your dataset while training.

3. **Baseline Model**
   (a) Once your data is ready to be used for training, you need to create a CNN model architecture. We have provided a **baseline** CNN model to start with in the repo. The basic building blocks of the architecture looks like below:

```
conv1 -> conv2 -> conv3 -> maxpool1 -> conv4 -> avgpool -> fc1 -> fc2
```

4. **Training**

   (a) Train a **baseline model** with the following design choices: Use the **cross entropy** loss function and **Xavier Initialization** to train your model for 25 epochs (complete passes through the training set). Use the **Adam** optimizer with learning rate= 10^{-4}. Use two holdout/validation sets (say, 60 images with 3 images for each category for each holdout set). Save the best weights over the 25 epochs.

   (b) Provide plots of loss for training and holdout sets, and a separate plot for the performance (percent correct) of each. Report your final test set performance.

   (c) For **25** epochs, you should get around 20-25% test set accuracy (which is way better than chance, but nothing to write home about). **Note:** Training time can vary a great deal on the UCSD GPU cluster, depending on the load. On average on an unloaded machine, you can expect around 1 min per epoch on the cluster.

5. **Custom Model**

   (a) Now, try to improve over the baseline model. You can add two extra convolutional layers (required) along with other layers like maxpool, batchnorm, dropout, etc., layers (if they help) and create a **custom model**. Be sure to check out Andrej Karpathy's blog to get some pointers on how to train a Neural Network.

   (b) Train this custom model. You can tune the hyperparameters: #epochs, learning rate, learning rate decay schedule, activation functions, batch size, weight initialization method and optimization techniques. Try to get the best possible results without overfitting. Remember, don't test on the test set until you have gotten your best holdout set performance. **Note:** Use 2-way cross validation with two holdout sets with 60 images each for hyperparameter tuning. Make sure that each holdout set has an even distribution with 3 images per class.

   (c) Provide plots of loss for training and holdout sets, and a separate plot for the performance of each. Report your final test set performance.

   (d) The accuracy of the model will be pretty modest. Anything around 47% accuracy on test dataset is probably good enough.

6. **Transfer Learning**

   (a) Now we will be using pre-trained models and fine tune them for our classification objective. Use both vgg-16 (with batch normalization) and resnet 18 for this purpose.

   (b) Remove the last fully connected layer and replace it with an appropriate-sized one for your problem.

   (c) First, freeze all of the weights except to your new fully connected layer and just train those.

   (d) Second, try fine-tuning (back-propagating into the pre-trained model) to see if you can improve on the results. You can try selectively freezing layers if you wish.

   (e) Provide the usual plots for parts c and d, as well as final accuracy.

   (f) You can do hyper parameter tuning and use other techniques including selective weight freezing to improve accuracy. Again, use 2 holdout sets with 60 images each for hyperparameter tuning.

7. **Weight maps and Feature maps**

   (a) To get an intuitive understanding of what is happening let us analyze the weight maps first. Plot all the first layer weight maps of your best custom model, vgg16 and resnet model. Explain what you observe visually.

   (b) Now, we need to analyze the feature maps. Take the first image in the test dataset (image corresponding to 1st line of **test_list.txt**) and pass it through the CNN model. Get three feature activation maps: One from the initial layer, one from a middle layer, and one from the final convolutional layer. Show them in the report. Explain what you observe. Do this for all the three models. [Hint: Make use of **hooks** in Pytorch.]

   Feature map and weight map examples from some layer of a model are shown in **Figure 1**. (**Note:** This is just an example to give you an idea of how the weight maps and feature maps will look)

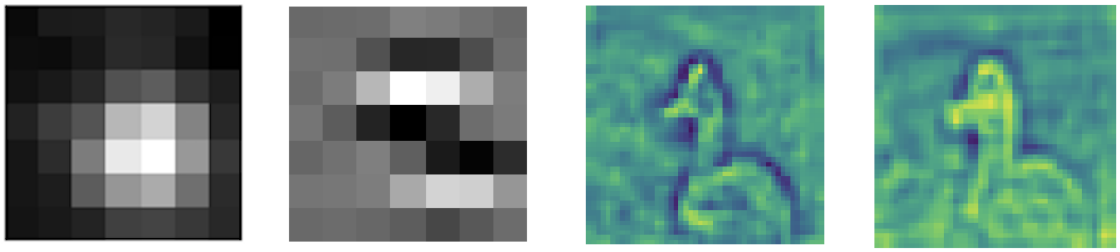


Figure 1: (a), (b) Example Weight maps corresponding to two filters of a layer. (c),(d) Example Feature maps corresponding to two filters of a layer

What to include in your report

In addition to learning very useful and applicable skills in deep learning and computer vision, this portion of the assignment will help you learn to write a scientific report. Please include an abstract and the following *7 sections*:

1. **Title:** (1 pt) The title has to be informative and not generic like '151B PA3' Report. Also you have to include author's list.
2. **Abstract:** (1 pt) Short description of what you did. Please mention any key findings, interesting insights, and final results (i.e., percent correct, not loss numbers)
3. **Introduction:** (1 pt) Similar to abstract but with more details - here, you introduce the problem, perhaps reference a couple of previous papers on this dataset, and why this is interesting.
4. **Related Work:** (1 pt) Cite any paper/slides for the methods you have used for this project. Full credit will be given if you reference (and briefly describe) two previous papers using this dataset. Also you can cite any Pytorch library pages that helped you implement the PA.
5. **Models** (5 pts)
   (a) Start this section by describing the baseline model in brief.
   (b) Then, you will explain the architectural details of your custom CNN - in particular, this addresses how you approached the problem and the design of your solution.
   (c) You should describe the custom model architecture, including the appropriate activation function on the last layer of the network, and the loss criterion you optimized. You may give a bit of history of what you tried, leading up to your final model.
   (d) Encapsulate your custom CNN architecture in a 2-column table, with the first column indicating the particular layer of your network, the second column stating the layer's dimensions (e.g. in-channels, out-channels, kernel-size, padding/stride) and activation function/non-linearity.
   (e) Write a few lines on vgg16 and resnet18 architecture by including any novel ideas used in them.
6. **Experiments** (10 pts)
   (a) Provide the plots described above - the baseline model accuracy and error over epochs for the training and holdout sets, and final accuracy on the test set. Since you have 2 holdout sets, you can plot the averages over the training and holdout sets. Do the same for the next part.
   (b) Show the same for your best custom model. For the major things you tried, give the holdout accuracy in a table with a brief name of each model. The models should be described well in the text, with a short name that you use in the table. Make a table for the same containing experiment name, train and test accuracy (for the best model of the experiment). If you have any intuition for why a model didn't work well, give it here.
   (c) Repeat the same for both vgg16 and resnet18 models
7. **Feature map and weights analysis** (5 pts)

(a) Show the plots of the first convolutional layer weight maps of best custom model, vgg16 and resnet18 models. Explain what you observe visually.

(b) Show the feature maps as explained in the previous section. Explain what you observe visually.

8. **Discussion** (5 pts)
   This section should summarize all your findings, what worked and what didn't, and possible reasons for each. This should at bare minimum include

   (a) Compare the baseline, custom, vgg16 and resnet18 models. Also, mention future steps to improving the custom model

   (b) What type of hyper parameter tuning helped in improving the model and reasons for it

   (c) Other design choices/building blocks that helped in improving the model

   Feel free to include any other points that you feel needs to be discussed.

9. **Authors' Contributions** (1pt)
   Each group member must write a short paragraph describing their contributions to the project.

10. **References** Don't forget to cite your references in the body of the report!

UCSD GPU cluster

UCSD provides access to its GPU cluster through UCSD Datahub. You are free to use any other platforms but we won't be responsible for any glitches, bugs or delays in them.
Steps to access UCSD datahub are as follows.

1. **Method 1**

   (a) Log into UCSD Datahub using Single Sign On method.

   (b) There will be two notebooks visible for the class. Choose the one that allocates you GPU.

   (c) Once the notebook is spawned, A JupyterHub Dashboard will appear.

2. **Method 2**

   (a) ssh into dsmlp-login.ucsd.edu using *ssh username@dsmlp-login.ucsd.edu*. Before that, you will need to connect through VPN if you are not connected to on campus network.

   (b) Use *launch-scipy-ml-gpu.sh* to launch a Jupyter Notebook with GPU access.

More details on using Datahub is provided on this link

Start early on the PA. Datahub GPUs might get over utilized towards the end of PA and you may not be able to get access to GPU when you need it. Happy Coding !!