CSE 190: Programming Assignment 3

Fall 2018

Instructions

Due Sunday, November 11^{th}:

1. 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.
2. You will be using PyTorch (v 0.4.0), a Deep Learning library, for the tasks in this assignment. See the post on Piazza by Jenny as to how to access the UCSD GPU server, the data, and PyTorch. Additionally, any updates or modifications to the assignment will be pushed to the PA3-CNN repository on Github, though we will try making a point to announce these clearly on Piazza.
3. Please work in teams of 2-3 individuals (no more than 3). Under *very special circumstances*, we will allow you to do it solo. Please discuss your circumstances with us first to get approval.
4. Please submit in your assignment via Gradescope. The report should be in NIPS format or another "top" conference format (e.g. IEEE CVPR, ICML, ICLR) - we expect you to write at a high academic-level (to the best of your ability). Make sure your code is readable and well-documented - you may want to reuse it in the future.

Learning Objectives

1. Understand the basics of convolutional neural networks, including convolutional layer mechanics, max-pooling, and dimensions of layers.
2. Learn how to implement a CNN architecture in PyTorch for image/object classification using best practices.
3. Build intuition on the effects of modulating the design of a CNN by experimenting with your own (or "classic") architectures.
4. Learn how to address the imbalanced/rare class prediction problem in multiclass classification.
5. Visualize and interpret learned feature 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, but you may submit a single copy of this exercise in your report based on the consensus of your group.

For questions 1-3, consider the $(5 \times 5 \times 2)$ input with values on $[-1, 1]$ and the corresponding $(3 \times 3 \times 1)$ filter shown below.
(As in PyTorch, we refer to this *single* filter as being $(3 \times 3 \times 1)$ despite having two sub-filters, because it would produce an output with 1 channel. In a case where the input had 4 channels, the filter would have 4 sub-filters but still be $(3 \times 3 \times 1)$.)

Input Features

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

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

Filter 0

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

1	0	-1
1	0	0
1	1	1

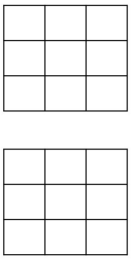
1. In the provided area below, fill in the values resulting from applying a convolutional layer to the input with no zero-padding and a stride of 1. Calculate these numbers before any activation function is applied. If there are any unused cells after completing the process in the provided tables, place an $\times$ in them.

Filtered Inputs

Output Feature Map

2. 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.

Maximally Activating Patch



3. **Spatial pooling:** Using your **output feature map** in question 1, applying *max-pooling* using a $[2\times2]$ kernel with a stride of 2. 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.

Max Pooled Output

4. **Number of learnable parameters**
Suppose we had the following architecture:

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

where all convs have a stride of 1 and no zero-padding. Conv1 has a kernel size of $[8 \times 8]$ with 12 output channels, conv2 has a $[8\times8]$ kernel with 10 output channels, conv3 has a $[6\times6]$ kernel with 8 output channels, and maxpool has a $[3 \times 3]$ kernel.
If ReLU is the activation function between each layer and the inputs are $[512 \times 512]$ greyscale images, what are:

(i) The number of input channels to **conv1**: _____

(ii) The number of input channels to **conv2**: _____

(iii) The number of input channels to **conv3**: _____

(iv) In order to add a fully-connected layer following the maxpool layer, we need to reshape the convolutional outputs to feed them into this hidden layer. Based on this architecture, what will the incoming dimensions to **fc1** be? Show your work.

Part II

Deep Convolutional Network for Thorax Disease Detection

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 a classification task which is increasingly relevant and important in the medical community - disease detection. We will build a deep CNN architecture and train it from scratch to predict thoracic disease(s) present in chest X-rays.

The **ChestX-ray14**[1] dataset contains 112,120 images (frontal-view X-rays) from 30,805 unique patients, where each image may be labeled with a single disease or multiple diseases. As such, we must design our CNNs for multiclass classification - clearly, applying softmax at the output layer will not achieve this and is therefore not an appropriate choice in this task. Additionally, the disease occurrences are not uniformly distributed - some may be more or less common than others - so using basic classification accuracy will not be a sufficient metric in evaluating your model's performance.
Please refer to the dataset README from NIH for more details about the dataset and problem setting.

Implementation Instructions

We have provided you with a data loader, **xray_dataloader.py**, **ChestXrayDataset**: a custom PyTorch Dataset class, specifically designed for the *ChestX-ray14* dataset, and a function **create_split_loaders()**, which partitions the dataset into train, validation, and test splits while creating PyTorch DataLoader object for each. The latter is an iterator over the elements in each split of the dataset which you will use to retrieve mini-batches of samples and their labels. Please familiarize yourself with the code and read the comments.

1. **Evaluating your model:**
   Before we discuss the neural network details, we must clarify how you will accurately and transparently evaluate your model's performance. When dealing with class imbalance, there are cases - particularly in disease and anomaly detection - where the naive prediction will result in very high overall accuracy (e.g. predicting "no disease"). As such, you will be reporting the following on a **per-class basis**:

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

   (ii) Precision: $\frac{|TP|}{|FP|+|TP|}$, where $|TP|, |FP|$ denotes the number of *true positives* and *false positives*, respectively

   (iii) Recall: $\frac{|TP|}{|TP|+|FN|}$, where $|FN|$ denotes the number of *false negatives*

   (iv) Balanced classification rate (BCR): $\frac{Precision+Recall}{2}$

   In addition, you will report:

   (v) The aggregated scores for precision, recall, and BCR, by computing an equally weighted average across all individual scores, respectively.

   (vi) A confusion matrix showing the classifications for all classes vs. all classes (demonstrating what your model "confused" as a different class).

2. **Create a baseline model:**
   You will create a simple convolutional neural net for the purposes of getting (i) acquainted with PyTorch, (ii) results to compare with more complex architectures and approaches to solving this multiclass classification problem. The baseline architecture is the following:

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

   Using the starter code provided in **baseline_cnn.py**, complete the implementation for the architecture as described in the comments, replacing the instances of __ with its corresponding missing value. This includes

finishing the **__init__()** and **forward()** functions.

To run the **baseline_cnn.py** model, we have additionally provided a basic Jupyter notebook, **train_model.ipynb**, containing nearly all the code needed to train the model. Based on your reasoning for determining the activation function for the output layer of the network, determine which loss criterion you should be optimizing - this may be something you are already familiar with (note that PyTorch loss criteria can be found in the **torch.nn** package). Additionally, use the Adam gradient descent optimizer, which can be found in the **torch.optim** package. Train the baseline model until the loss stops decreasing on the validation set.

Please note that this is a very bare-bones implementation which may perform decently on MNIST but pale in the light of a much more challenging problem as this. As such, this architecture may have "good" overall classification accuracy, but fail to address the class-imbalance problem.

In addition, make note that the data loader applies any specified color-scale conversion and transformations to your images *on-line*. In the next section, you are welcome to optimize this as you see fit (including preprocessing the complete dataset and saving a copy in your **home directory in the Kubernetes environment**.

3. **Experimentation and your solution:**
   To get a better sense of how your design choices affect model performance, we encourage you to experiment with various approaches to solving this multiclass classification problem using a deep CNN. You are welcome to make a copy of the **baseline_cnn.py** file and **train_model** notebook for this purpose (as well as convert the training code into a .py file). At a minimum, you should do the following:

   (i) Apply transformations to the input images: this can include using normalization, etc. Note that the original images are $[1024 \times 1024]$ dimensional, and in the **baseline_cnn.py** implementation, we resize them down to $[512 \times 512]$ to improve the speed of computation. You can try downsizing them further, but bear in mind that you may lose meaningful information at the cost of improved efficiency.

   (ii) Try two distinct, additional architectures - this includes making significant changes in the number of layers, activation functions, dimensions of the convolutional filters, etc. For each, choose some k s.t. $1 < k \leq 4$ and perform k-fold cross validation to evaluate. You may choose the best version of each implementation to evaluate on. Simply adding an additional layer is not sufficient!

   (iii) Address the rare class or imbalanced class problem. This is often done by **enforcing** that the network learn to categorize the infrequently seen classes. You can use: (1) weighted loss, (2) using batches of evenly distributed class representation (resampling). At a minimum, you must implement your own weighted/balanced loss criterion.

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*:

0. **Abstract**
   The **Abstract** should serve as a $\approx$ one paragraph synopsis of the work in your report, including the task (e.g. multinomial regression on dataset X), how you approached it, and a quick overview of your results.

1. **Introduction**
   The **Introduction** should describe the problem statement or task, why it's important, and any necessary background your audience may need to know. Keep in mind that since we're using Xavier weight initialization and batch normalization (see starter code), you should understand the basic mathematics behind these concepts at a minimum, how they affect your network parameters, and why they're useful. As such, these should be discussed in either the **Introduction** or **Methods** sections.

2. **Related Work**
   The **Related Work** section should review any work you used to inspire your approach - this includes previous research in the specific problem you address, as well as any core ideas you build upon. You should include citations in standard reference format with this itemized in a **References** section at the end of the report. This is where using LaTeX and BibTex can come in very handy.

3. **Methods**
   In the **Methods** section, you should describe the implementation and architectural details of your system - in particular, this addresses how you approached the problem and the design of your solution. For those who believe in reproducible science, this should be fine-grained enough such that somebody could implement your model/algorithm and reproduce the results you claim to achieve.

   (i) **Baseline:** You should describe the baseline architecture, stating the appropriate activation function on the last layer of the network, and the loss criterion you optimized.

   (ii) **Experimentation:** Encapsulate your two experimental CNN architectures, each in a 2-column table, which the first column indicate the particular layer of your network, the second column state the layer's dimensions (e.g. in-channels, out-channels, kernel-size, padding/stride) and activation function/non-linearity.
   Describe the type of k-fold cross validation you implemented, any regularization techniques you used, parameter initialization methods, gradient descent optimization, and how you addressed the class-imbalance problem (the latter in detail).

4. **Results:**
   In the **Results** section, you should demonstrate your models' performance compared with the baseline implementation. You should include all the performance metrics described in Implementation Instructions part 1 for **each implementation based on your test set results**. Please organize these results into a series of concise tables. The formatting is your choice, so long as it is easily interpretable.

   Additionally, you should include the following for each architecture:

   (i) A single plot showing both training and validation *loss* curves

   (ii) A single plot showing training and validation *accuracy* curves

   (iii) Visualizations of the learned filter maps of your trained network. Select 2 filters from an early layer, 2 filters from a middle layer, and 2 filters from a later layer.

5. **Discussion:**
   The **Discussion** section should highlight the meaning of your results and the approaches followed in this work. At a minimum, please discuss the following important points, but feel free to go beyond this:

   (i) How did the performance of your implementations differ; hypothesize why this was the case.

   (ii) What common confusions between classes did your model exhibit; why might this have occurred? Did this differ significantly between your implementations?

   (iii) What do the different performance metrics tell us about the 3 models' ability to identify diseases? What does Precision / Recall seem to tell us as opposed to accuracy or our plotted loss?

   (iv) Discuss the visualizations of the feature maps and how they vary with depth.

6. **Authors' Contributions and References**
   Each group member must write a small section describing their contributions to the project. Do not forget to cite your references.

References

[1] X. Wang, Y. Peng , L. Lu Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. Department of Radiology and Imaging Sciences, September 2017. https://arxiv.org/pdf/1705.02315.pdf