

BME646 and ECE60146: Homework 4

Spring 2023

Due Date: 11:59pm, Feb 20, 2023

TA: Fangda Li (li1208@purdue.edu)

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. **Late submissions will be accepted with penalty: -10 points per-late-day, up to 5 days.**

1 Introduction

There are two main goals of this homework:

1. To introduce you to the MS-COCO dataset, which stands for Microsoft Common Objects in COtext. You will be tasked to create your own classification dataset based on images and annotations taken from the COCO dataset.
2. To have you create your own Convolutional Neural Network (CNN) in PyTorch for image classification.

2 Background

2.1 About the COCO Dataset

Owing to its rich annotations, the COCO dataset has been used extensively by researchers for training and evaluating their frameworks for image classification, object detection, self-supervised learning, pose estimation, etc. To understand the motivations behind its creation and what makes it challenging, you should go ahead and download the paper that introduced the COCO dataset by clicking on:

<https://arxiv.org/pdf/1405.0312.pdf>

You should at least read the Introduction section of the paper.

In this homework, you will be asked to create your own image dataset for classification. For any aspiring machine learning practitioner, the ability to curate data and labels that fit his or her own needs is one of the most important skills to have. Therefore, we have designed this homework to provide an introductory exercise for this very purpose. More specifically, you will

be downloading a part of the full COCO dataset. Then, you will familiarize yourself with the COCO API, which provides a convenient interface to the otherwise complicated annotation files. And lastly, you will create your own image dataset for classification using the downloaded COCO files and the COCO API.

2.2 About the Image Classification Network You Will Write Code For

A good starting point for you would be to review the networks you see in the inner class `ExperimentsWithCIFAR` of the `DLStudio` module. The network you will be creating is likely to be very similar to the two examples — `Net` and `Net2` — shown in that section of `DLStudio`. After installing `DLStudio`, play with the two networks by changing the parameters of the convolutional and the fully connected layers and see what that does to the classification accuracy. For that, you will need to execute the following script in the `Examples` directory of the `DLStudio` module¹:

`playing_with_cifar10.py`

As with the `DLStudio` example code mentioned above, the classification network you will create will use a certain number of convolutional layers and, at its top, will contain one or more fully connected (FC) layers (also known as Linear layers). The number of output nodes at the final layer will be 10, which is equal to the number of image classes you will be working with.

In your experiments with the classification network, pay attention to the changing resolution in the image tensor as it is pushed up the resolution hierarchy of a CNN. This is particularly important when you are trying to estimate the number of nodes you need in the first fully connected layer at the top of the network. Depending on the sizes of the convolutional kernels you will use, you may also need to pay attention to the role played by padding in the convolutional layers.

¹The CIFAR-10 dataset will be downloaded automatically by running this script. The CIFAR image dataset, made available by the University of Toronto, is considered to be the fruit-fly of DL. The dataset consists of 32×32 images, 50,000 for training and 10,000 for testing that can easily be processed in your laptop. Just Google “download CIFAR-10 dataset” for the website from where you can download the dataset.

3 Programming Tasks

3.1 Creating Your Own Image Classification Dataset

In this exercise, you will create your own dataset based on the following steps:

1. The first step is to install the COCO API in your `conda` environment. The Python version of the COCO API — `pycocotools` provides the necessary functionalities for loading the annotation JSON files and accessing images using class names. The `pycocoDemo.ipynb` demo available on the COCO API GitHub repository [1] is a useful resource to familiarize yourself with the COCO API. You can install the `pycocotools` package with the following command:

```
conda install -c conda-forge pycocotools
```

2. Now, you need to download the image files and their annotations. The COCO dataset comes in 2014 and 2017 versions. For this homework, you will be using the **2014 Train images**. You can download them directly from this page:

<https://cocodataset.org/#download>

On the same page, you will also need to download the accompanying annotation files: **2014 Train/Val annotations**. Unzip the two archives you just downloaded.

3. Your main task is to use those files to create your own image classification dataset. Note that you can access the class labels of the images stored in the `instances_train2014.json` file using the COCO API. You have total freedom on how you organize your dataset as long as it meets the following requirements:

- It should contain 1500 training and 500 validation images for each of the following five classes:

```
['airplane', 'bus', 'cat', 'dog', 'pizza']
```

This will amount to 7.5k training images and 2.5k validation images in total and there should be no duplicates. All images should be taken from the **2014 Train images** set you just downloaded.

- When saving your images to disk, resize them to 64×64 . You can use the PIL module for that.
4. In your report, make a figure of a selection of images from your created dataset. You should plot at least 3 images from each of the five classes.

3.2 Image Classification using CNNs – Training and Validation

Once you have prepared the dataset, you now need to implement and test the following CNN tasks:

CNN Task 1: In the following network, you will notice that we are constructing instances of `torch.nn.Conv2d` in the mode in which it only uses the valid pixels for the convolutions. But, as you now know based on the Week 5 lecture (and slides), this is going to cause the image to shrink as it goes up the convolutional stack.

Your first task is to run the network as shown. Let's call this single layer CNN as *Net1*.

```

1 class HW4Net(nn.Module):
2     def __init__(self):
3         super(HW4Net, self).__init__()
4         self.conv1 = nn.Conv2d(3, 16, 3)
5         self.pool = nn.MaxPool2d(2, 2)
6         self.conv2 = nn.Conv2d(16, 32, 3)
7         self.fc1 = nn.Linear(XXXX, 64)
8         self.fc2 = nn.Linear(64, XX)
9
10    def forward(self, x):
11        x = self.pool(F.relu(self.conv1(x)))
12        x = self.pool(F.relu(self.conv2(x)))
13        x = x.view(x.shape[0], -1)
14        x = F.relu(self.fc1(x))
15        x = self.fc2(x)
16        return x

```

Note that the value for XXXX will vary for each CNN architecture and finding this parameter for each CNN is your homework task. XX denotes the number of classes. In order to experiment with a network like the one shown above, your training routine can be as simple as:

```

1 net = net.to(device)
2 criterion = torch.nn.CrossEntropyLoss()
3 optimizer = torch.optim.Adam(

```

```

4     net.parameters(), lr=1e-3, betas=(0.9, 0.99))
5 epochs = 7
6 for epoch in range(epochs):
7     running_loss = 0.0
8     for i, data in enumerate(train_data_loader):
9         inputs, labels = data
10        inputs = inputs.to(device)
11        labels = labels.to(device)
12        optimizer.zero_grad()
13        outputs = net(inputs)
14        loss = criterion(outputs, labels)
15        loss.backward()
16        optimizer.step()
17        running_loss += loss.item()
18        if (i+1) % 100 == 0:
19            print("[epoch: %d, batch: %5d] loss: %.3f" \
20                  % (epoch + 1, i + 1, running_loss / 100))
21        running_loss = 0.0

```

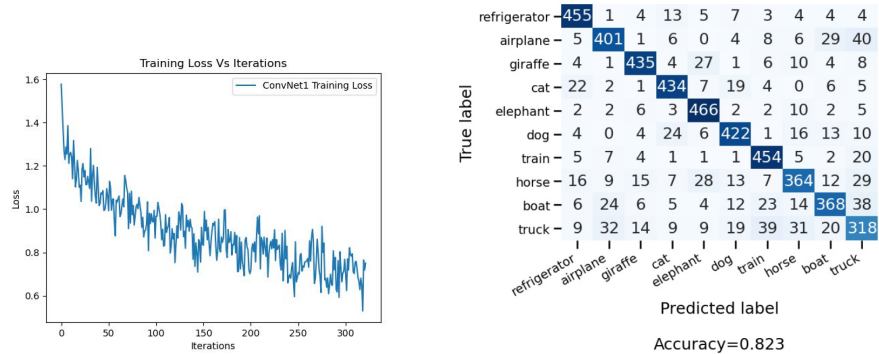
where the variable `net` is an instance of `HW4Net`.

CNN Task 2: In the `HW4Net` class as shown, we used the class `torch.nn.Conv2d` class without padding. In this task, construct instances of this class with padding. Specifically, add a padding of one to the all the convolutional layers. Now calculate the loss again and compare with the loss for the case when no padding was used. This is the second CNN architecture, *Net2* for this homework.

CNN Task 3: So far, both *Net1* and *Net2* can be only considered as very shallow networks. Now in this task, we would like you to experiment with a deeper network. Modify the `HW4Net` class to chain at least 10 extra convolutional layers between the second conv layer and the first linear layer. Each new convolutional layer should have 32 in-channels, 32 out-channels, a kernel size of 3 and padding of 1. In the `forward()` method, the output of each conv layer should be fed through an activation function before passed into the next layer. Note that you would also need to update the value of `XXXX` accordingly. The resulting network will be the third CNN architecture — *Net3*.

Note that in order to train and evaluate your CNNs, you will need to implement your own `torch.utils.data.Dataset` and `DataLoader` classes for loading the images and labels. This is similar to what you have implemented in HW2.

For evaluating the performance of your CNN classifier, you need to write your own code for calculating the confusion matrix. For the dataset that you



(a) Training loss for the three CNNs. (b) Sample confusion matrix.

Figure 1: Sample output, training loss and validation confusion matrix. The plotting options are flexible. Your results could vary based on your choice of hyperparameters. The confusion matrix shown is for a different dataset and is for illustration only.

created, your confusion matrix will be a 5×5 array of numbers, with both the rows and the columns standing for the 5 classes in the dataset. The numbers in each row should show how the test samples corresponding to that class were correctly and incorrectly classified. You might find `scikit-learn` and `seaborn` python packages useful for this task. Fig. 1b shows a sample plot of the training loss and a sample confusion matrix. It's important to note that your own plots could vary based on your choice of hyperparameters.

In your report, you should include a figure that plots the training losses of all three networks together. Additionally, include the confusion matrix for each of the three networks on the validation set. Finally, include your answers to the following questions:

1. Does adding padding to the convolutional layers make a difference in classification performance?
2. As you may have known, naively chaining a large number of layers can result in difficulties in training. This phenomenon is often referred to as *vanishing gradient*. Do you observe something like that in *Net3*?
3. Compare the classification results by all three networks, which CNN do you think is the best performer?
4. By observing your confusion matrices, which class or classes do you think are more difficult to correctly differentiate and why?

5. What is one thing that you propose to make the classification performance better?

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Your pdf must include a description of
 - The figures and descriptions as mentioned in Sec. 3.
 - Your source code. Make sure that your source code files are adequately commented and cleaned up.
2. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (only .py files are accepted). Rename your .zip file as hw3_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.
3. **Do NOT submit your network weights.**
4. For all homeworks, you are encouraged to use .ipynb for development and the report. If you use .ipynb, please convert it to .py and submit that as source code.
5. You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission. **If you are submitting late, do it only once on BrightSpace.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.
6. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**
7. To help better provide feedbacks to you, make sure to **number your figures.**

References

- [1] COCO API - <http://cocodataset.org/>. URL <https://github.com/cocodataset/cocoapi>.