BME646 and ECE60146: Homework 4

Spring 2023
Due Date: 11:59pm, Feb 5, 2024
TA: Akshita Kamsali (akamsali@purdue.edu)

Turn in typed solutions via BrightSpace. Read all submission instructions carefully. Late submissions will be accepted with penalty: **-10 points per-late-day, up to 5 days**.

1 Introduction

The two main goals of this homework:

1. To introduce you to the competition-grade and very famous COCO dataset of images. A more correct name for the COCO dataset is MS-COCO. The acronym stands for Microsoft Common Objects in COntext. It is frequently used by researchers to showcase the power of their neural networks for solving problems in image segmentation, classification, object detection, etc. You will be tasked to create your own image classification dataset as a relatively small subset of COCO. The dataset you create will consist of the images and their annotations as described later in this homework.

2. Your second goal will be to write PyTorch code for a CNN (Convolutional Neural Network) for image classification. You will train the CNN using the dataset you will extract from COCO as mentioned above. Since this will be your first exercise in image classification, this part of the homework will also suggest that you develop your classification insights quickly by experimenting with DLStudio's inner class `ExperimentsWithCIFAR` using the far simpler CIFAR dataset that consists of just 32×32 images.

2 Background

2.1 About the COCO Dataset

Owing to its rich annotations, the COCO dataset, first published in 2014 [3], continues to be a most important resource in deep learning. A recent very famous paper from Meta presented a powerful neural network for image segmentation called Segment Anything (SAM) [2]. It was trained using the COCO dataset. With its versatile annotations, the dataset can be used to train networks for all kinds of tasks including image classification, object detection, self-supervised learning, pose estimation, and more. To understand the motivations behind its inception and to appreciate the challenges faced in constructing the dataset, see the original paper [3] on COCO. You should at least read the Introduction section of the paper.

For this homework, you will download a part of the full COCO dataset and familiarize yourself with the COCO API, which provides a convenient interface to the otherwise complicated annotation files. Finally, 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

DLStudio's inner class `ExperimentsWithCIFAR` will serve as a sandbox for quickly coming up to speed on this part of the Homework. The network you have to create is likely to be very similar to the two examples — `Net` and `Net2` — shown in that part of DLStudio. After installing DLStudio, play with these two networks by changing the parameters of the convolutional and the fully connected layers and see what that does to the classification accuracy. Regarding DLStudio, download its zip archive from its main doc page, install the module, and go to its Examples directory to get hold of the script named below.[1]:

```
python 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 is equal to the number of image classes you will be working with, here 10.

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

[1]The CIFAR-10 dataset will be downloaded automatically when you run the script `playing_with_cifar10.py`. 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 "CIFAR-10 dataset" for more information regarding the dataset.

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.

3 Programming Tasks

3.1 Using COCO to Create Your Own Image Classification Dataset

Note: Don't be concerned about the initial large size of the complete dataset you download. You will utilize only a subset of the entire dataset, which you will extract following the provided instructions. Make sure to execute these steps accurately, save the subset data, and keep it for future assignments.

Through this exercise, you will create a custom dataset which is a subset of the COCO dataset:

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 [2]:

```
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 **2017 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: **2017 Train/Val annotations**. Unzip the two archives you just downloaded.

[2]The following command may change based on your version of conda, please check for the appropriate conda/pip command to install pycocotools

3. You 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_train2017.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 1600 training and 400 validation images for each of the following five classes:

     `['boat', 'cake', 'couch', 'dog', 'motorcycle']`

     This will amount to 8000 training images and 2000 validation images in total and there should be no duplicates. All images should be taken from the **2017 Train images** set you just downloaded.

   - When saving your images to disk, resize them to 64×64.

     You may use the `opencv` and `PIL` module to perform above operations.

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.

Do NOT submit any dataset, original or custom, to Brightspace.

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*. You may also modify the code by using `nn.Sequential` to implement this.

```python
class HW4Net(nn.Module):
    def __init__(self):
        super(HW4Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.fc1 = nn.Linear(XXXX, 64)
        self.fc2 = nn.Linear(64, XX)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        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:

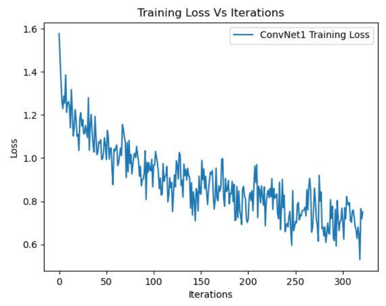
```python
net = net.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(
    net.parameters(), lr=1e-3, betas=(0.9, 0.99))
epochs = 7
for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(train_data_loader):
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if (i+1) % 100 == 0:
            print("[epoch: %d, batch: %5d] loss: %.3f" \
                % (epoch + 1, i + 1, running_loss / 100))
            running_loss = 0.0
```

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

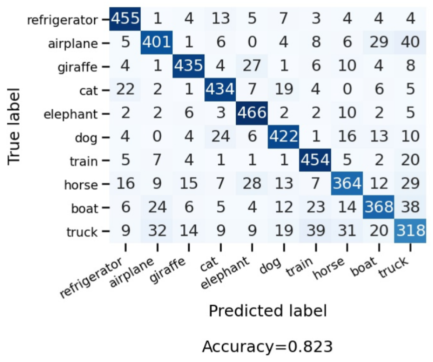
CNN Task 2: In the `HW4Net` class as shown, we used the class `torch.nn.Conv2d` class without padding. In this task, construct instances of this class with padding. Specifically, add a padding of one to the all the convolutional layers. Now calculate the loss again and compare with the loss for the case 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*.

Before you proceed further, identify the number of parameters in each of your network.



(a) Training loss for the three CNNs.



True label \ Predicted label	refrigerator	airplane	giraffe	cat	elephant	dog	train	horse	boat	truck
refrigerator	455	1	4	13	5	7	3	4	4	4
airplane	5	401	1	6	0	4	8	6	29	40
giraffe	4	1	435	4	27	1	6	10	4	8
cat	22	2	1	434	7	19	4	0	6	5
elephant	2	2	6	3	466	2	2	10	2	5
dog	4	0	4	24	6	422	1	16	13	10
train	5	7	4	1	1	1	454	5	2	20
horse	16	9	15	7	28	13	7	364	12	29
boat	6	24	6	5	4	12	23	14	368	38
truck	9	32	14	9	9	19	39	31	20	318

Accuracy=0.823

(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 hyperparamters. The confusion matrix shown is for a different dataset and is for illustration only.

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 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**. Further, include the confusion matrix for each of the three networks on the validation set. Add a table with net name, corresponding number of parameters and classification accuracy. 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 pdf file a typed self-contained pdf report with source code and results. Rename your .pdf file as hw4_<First Name><Last Name>.pdf
3. Turn in a zipped file, it should include all source code files (only .py files are accepted). Rename your .zip file as hw4_<First Name><Last Name>.zip .
4. **Do NOT submit your network weights.**
5. **Do NOT submit your dataset.**
6. 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.
7. 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.
8. Regrade requests regarding failing to follow instructions are not accepted.
9. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

10. To help better provide feedback to you, make sure to **number your figures**.

References

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

[2] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

[3] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.