ECE695DL: Homework 4

Qilei Zhang

Due Date: Monday, Feb 21, 2022

1 Introduction

This homework let us create three different convolutional image classification network, including single convolutional layer, multi convolutional layer and layers with padding. The Common Objects in COntext (COCO) dataset is used for the purpose of training and validation. A COCO dataset downloader is also created to download and downsample (64×64) a subset of COCO images using COCO API. The results is expected to have the loss function comparison of three nets. Additionally, confusion matrix will be provided for the validation to show how the test samples corresponding to that class were correctly and incorrectly classified.

2 Methodology

2.1 How to Run

2.1.1 To download dataset

Enter the required scripts in terminal. Below are the examples in the local environment:

```
python hw04_coco_downloader.py --root_path /Users/admin/Downloads/coco/Train/ --coco_json_path /Users/admin/Downloads/annotations2017/instances_train2017.json --class_list "airplane" "boat" "cat" "dog" "elephant" "giraffe" "horse" "refrigerator" "train" "truck" --images_per_class 2000
python hw04_coco_downloader.py --root_path /Users/admin/Downloads/coco/Train/ --coco_json_path /Users/admin/Downloads/annotations2014/instances_train2014.json --class_list "airplane" "boat" "cat" "dog" "elephant" "giraffe" "horse" "refrigerator" "train" "truck"  --images_per_class 1500
python hw04_coco_downloader.py --root_path /Users/admin/Downloads/coco/Val/ --coco_json_path /Users/admin/Downloads/annotations2014/instances_val2014.json --class_list "airplane" "boat" "cat" "dog" "elephant" "giraffe" "horse" "refrigerator" "train" "truck" --images_per_class 500
```

2.1.2 To train data

Run the `ok.py` in the terminal by entering required arguments including train data root folder and the category to classify. Following is an example:

```
python ok.py --root_path ./hw04_coco_data/Train/ --class_list "airplane"
↪   "boat" "cat" "dog" "elephant" "giraffe" "horse" "refrigerator" "train"
↪   "truck"
```

2.1.3 To validate model

Run the `hw04_validation.py` in the terminal by entering required arguments including validation data root folder, network file folder and the category to classify. Following is an example:

```
python hw04_validation.py --root_path ./hw04_coco_data/Val/ --net_path
↪   /Users/benwhite/Downloads/ --class_list "airplane"  "boat" "cat" "dog"
↪   "elephant" "giraffe" "horse" "refrigerator" "train" "truck"
```

2.2 Prepare Task

(1). The *ExperimentsWithCIFAR* example in DLStudio is run to illustrate how the neutral network is construct through *pytorch*.

(2). Package *pycocotools* is need to install with `sudo` because some root authority may needed to install successfully. Cocoapi is also downloaded from Link provided. Additionally, annotation files including a dozen JSON files are downloaded that are available to be parsed.

2.3 Main Task

(1). A complete construction of neural network consist of creating network structure, dataloader, and a "ignition" code to start the process. Validation part is quite similar to the training part. It will create a empty network structure similar to the training one. Then it read the trained network to process the validation dataset.

(2). A COCO downloader is build that can handle arbitrary number of classes and any number of per-class images. The relevant image URLs are queried by `cocoAPI` and downloaded using `requests` python package.

(3). The `dataloader` function embedded in the `hw04_training.py` can handle arbitrary number of classes and any number of per-class images. It returns the processed images and labels that can be enumerated in the training process.

3 Implementation and Results

3.1 Main Program Code

3.1.1 hw04_coco_downloader.py

```python
import argparse
import requests
import os
import torch
import random
import numpy as np

from PIL import Image
from pycocotools.coco import COCO
# from requests.exceptions import ConnectionError, ReadTimeout, TooManyRedirects, MissingSchema, InvalidURL

# If want to use preset input
# class Fake_args():
#     def __init__(self, data_type='val2017'):
#         self.root_path = '/Users/admin/Downloads/coco/'
#         self.coco_json_path = '{}/annotations2017/instances_{}.json'.format(self.root_path,data_type)
#         #self.class_list = {0: "airplane", 1: "boat"}
#         self.class_list = ["airplane", "boat"]
#         self.images_per_class = 5


class Downloader:
    def __init__(self, info):
        self.root = info.root_path
        self.json = info.coco_json_path
        self.class_list = info.class_list
        self.images_per_class = info.images_per_class
        self.cat_folder = dict.fromkeys(self.class_list)
        self.coco = COCO(self.json)

    def make_folder(self):
        for cat in self.class_list:
            folder_root = self.root + cat
            self.cat_folder[cat] = folder_root
            if not os.path.exists(folder_root):
                os.makedirs(folder_root)

    def get_image(self):
```

```
        for cat in self.class_list:
            cat_id = self.coco.getCatIds(cat)
            img_id = self.coco.getImgIds(catIds=cat_id)
            imgs = self.coco.loadImgs(img_id)
            save_number = 0
            img_index = 0
            while save_number < self.images_per_class:
                # for number in range(self.images_per_class):
                img_path = self.root + cat + "/" + imgs[img_index]['file_name']
                save_check = self.save_image(img_path, imgs[img_index]['coco_url'])
                if save_check:
                    self.resize_image(img_path)
                    save_number += 1
                img_index += 1

        print("Download Finished")

    @staticmethod
    def resize_image(img_save_path):
        image = Image.open(img_save_path)
        if image.mode != "RGB":
            image = image.convert(mode="RGB")
        image_resized = image.resize((64, 64), Image.BOX)
        image_resized.save(img_save_path)

    @staticmethod
    def save_image(img_save_path, img_url):
        try:
            img_response = requests.get(img_url, timeout=1)
        except requests.exceptions as e:
            return False
        with open(img_save_path, 'wb') as img_f:
            img_f.write(img_response.content)
        return True


if __name__ == "__main__":
    seed = 0
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    np.random.seed(seed)
    # torch.backends.cudnn.deterministic = True
    # torch.backends.cudnn.benchmarks = False
```

```python
    os.environ['PYTHONHASHSEED'] = str(seed)

    parser = argparse.ArgumentParser(description='HW04 CocoDownloader')
    parser.add_argument('--root_path', required=True, type=str)
    parser.add_argument('--coco_json_path', required=True, type=str)
    parser.add_argument('--class_list', required=True, nargs='*', type=str)
    parser.add_argument('--images_per_class', required=True, type=int)

    args, args_other = parser.parse_known_args()

    # args = Fake_args()

    myDownloader = Downloader(args)
    myDownloader.make_folder()
    myDownloader.get_image()
```

3.1.2 ok.py

```python
import random
import numpy
import os
import argparse
import torchvision.transforms as tvt
import torch.utils.data
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

from model import QZhangNet
from hw04_training import run_code_for_training
from hw04_training import qzDatasetClass

if __name__ == '__main__':
    if torch.cuda.is_available():
        device = 'cuda:0'
    else:
        device = 'cpu'

    parser = argparse.ArgumentParser(description='HW04 Training')
    parser.add_argument('--root_path', required=True, type=str)
    parser.add_argument('--class_list', required=True, nargs='*', type=str)
    args, args_other = parser.parse_known_args()

    seed = 0
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
```

```
    numpy.random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)

    # Load and normalize data
    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    batch_size = 10

    # Local Fake
    # catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe", "horse", "refrigerator", "train", "truck"]
    # catList = ["airplane", "boat"]
    # dataFolder = "./hw04_coco_data/Train/"
    catList = args.class_list
    dataFolder = args.root_path
    catNum = len(catList)


    trainSet = qzDatasetClass(dataFolder, catList, transform)
    trainLoader = DataLoader(dataset=trainSet, batch_size=batch_size, shuffle=True, num_workers=4)
    print('first')

    # First Net
    net1 = QZhangNet(net_type=1, class_num=catNum)
    print('done1')
    net1Loss, iter1 = run_code_for_training(net1, trainLoader, learning_rate=1e-3,
                                            momentum_set=0.9, epochs=10, device=device, net_type=1)
    print('done first')

    # Second Net
    net2 = QZhangNet(net_type=2, class_num=catNum)
    print('done2')
    net2Loss, iter2 = run_code_for_training(net2, trainLoader, learning_rate=1e-3,
                                            momentum_set=0.9, epochs=10, device=device, net_type=2)
    print('done second')

    # Third Net
    net3 = QZhangNet(net_type=3, class_num=catNum)
    print('done3')
    net3Loss, iter3 = run_code_for_training(net3, trainLoader, learning_rate=1e-3,
```

```
                                                                momentum_set=0.9, epochs=10,
                                                                ↪  device=device, net_type=3)
    print('done third')

    plt.plot(net1Loss, label='Net1')
    plt.plot(net2Loss, label='Net2')
    plt.plot(net3Loss, label='Net3')
    plt.legend()
    plt.show()

    print('done4')
```

3.1.3 model.py

```
# import torch
import torch.nn as nn
import torch.nn.functional as functional


class QZhangNet(nn.Module):
    def __init__(self, net_type=1, class_num=10):
        super(QZhangNet, self).__init__()
        self.net_type = net_type
        self.conv1 = nn.Conv2d(3, 128, 3)  # Default Value
        self.conv1_3 = nn.Conv2d(3, 128, 3, padding=1)
        self.conv2 = nn.Conv2d(128, 128, 3)  # (B)
        self.pool = nn.MaxPool2d(2, 2)

        self.fc1_1 = nn.Linear(128 * 31 * 31, 1000)
        self.fc1_2 = nn.Linear(128 * 14 * 14, 1000)
        self.fc1_3 = nn.Linear(128 * 15 * 15, 1000)
        self.fc2 = nn.Linear(1000, class_num)

    def forward(self, x):
        """
        Uncomment the next statement and see what happens to the
        performance of your classifier with and without padding.
        Note that you will have to change the first arg in the
        call to Linear in line (C) above and in the line (E)
        shown below.  After you have done this experiment, see
        if the statement shown below can be invoked twice with
        and without padding.  How about three times?
        """
        if self.net_type == 1:
            x = self.pool(functional.relu(self.conv1(x)))
            x = x.view(-1, 128 * 31 * 31)  # (E)
```

```
                x = functional.relu(self.fc1_1(x))
            elif self.net_type == 2:
                x = self.pool(functional.relu(self.conv1(x)))
                x = self.pool(functional.relu(self.conv2(x)))
                x = x.view(-1, 128 * 14 * 14)
                x = functional.relu(self.fc1_2(x))
            elif self.net_type == 3:
                x = self.pool(functional.relu(self.conv1_3(x)))
                x = self.pool(functional.relu(self.conv2(x)))
                x = x.view(-1, 128 * 15 * 15)
                x = functional.relu(self.fc1_3(x))
            x = self.fc2(x)
            return x
```

3.1.4 hw04_training.py

```
import random
import numpy
import os
# import sys
import glob
import argparse
from PIL import Image
import torch
import torchvision.transforms as tvt
import torch.utils.data
from torch.utils.data import DataLoader, Dataset
import matplotlib.pyplot as plt
from model import QZhangNet


class qzDatasetClass(Dataset):
    def __init__(self, root, category_list, trans=None):
        """
        Make use of the arguments from the calling
        routine to initialise the variables
        e.g.image path lists for cat and dog classes
        you could also maintain label_array
        0 - - airplane
        1 - - boat
        2 - - cat.
        .
        .
        9 - - truck
        Initialise the required transform
        """
```

```
        self.transform = trans
        self.category_list = category_list
        self.root = root
        self.path_list = []
        self.img_info = []

        for cat in self.category_list:
            cat_path = self.root + cat + "/"
            self.path_list.append(cat_path)
            cat_label = self.category_list.index(cat)
            search = cat_path + "*"
            for img in glob.glob(search):
                img_info = [cat_label, img]
                self.img_info.append(img_info)

    def __len__(self):
        """
        return the total number of images
        refer pytorch documentation for more details
        """
        return len(self.img_info)

    def __getitem__(self, idx):
        img_path = self.img_info[idx][1]
        label = self.img_info[idx][0]
        image = Image.open(img_path)
        im_ts = self.transform(image)
        return im_ts, label


def run_code_for_training(net, trainLoader, learning_rate=1e-3, momentum_set=0.9, epochs=10,
                          device='cuda:0', net_type=1):
    net = net.to(device)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate, momentum=momentum_set)
    loss_running_record = []
    Iter_record = []
    for epoch in range(epochs):
        running_loss = 0.0
        for i, data in enumerate(trainLoader):
            print(i)
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
```

```python
            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if (i + 1) % 500 == 0:
                points = running_loss / float(500)
                loss_running_record.append(points)
                Iter_record.append(len(loss_running_record))
                print("\n[epoch:%d, batch:%5d] loss: %.3f" % (epoch + 1, i + 1, points))
                running_loss = 0.0
    # netpath = './net1.pth'
    netpath = './net' + str(net_type) + '.pth'
    torch.save(net.state_dict(), netpath)
    print('Finished Training')
    return loss_running_record, Iter_record


if __name__ == '__main__':
    #If want to run directly

    # Training on GPU or CPU
    if torch.cuda.is_available():
        device = 'cuda:0'
    else:
        device = 'cpu'

    seed = 0
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    numpy.random.seed(seed)
    # torch.backends.cudnn.deterministic = True
    # torch.backends.cudnn.benchmarks = False
    os.environ['PYTHONHASHSEED'] = str(seed)

    # Load and normalize data
    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    # parser = argparse.ArgumentParser(description='HW04 Training')
    # parser.add_argument('--root_path', required=True, type=str)
    # parser.add_argument('--class_list', required=True, nargs='*', type=str)
```

```
    # args, args_other = parser.parse_known_args()

    # catList = args.class_list
    # root = args.root_path

    batch_size = 10

    catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe", "horse", "refrigerator", "train", "truck"]
    dataFolder = "./hw04_coco_data/Train/"
    trainSet = qzDatasetClass(dataFolder, catList, transform)
    trainLoader = DataLoader(dataset=trainSet, batch_size=batch_size, shuffle=True , num_workers=4)
    print('Loader Created')

    # First Net
    net1 = QZhangNet()
    print('Net Created')
    net1Loss, iter1 = run_code_for_training(net1, trainLoader, learning_rate=1e-3,
                                            momentum_set=0.9, epochs=10, device=device, net_type=1)
    plt.plot(net1Loss, label='Net1')
    plt.legend()
    plt.show()

    print('Done')
```

3.1.5 hw04_validation.py

```
import copy
import random
import torch
import os
import argparse
import numpy as np
import torchvision.transforms as tvt
import seaborn as sns
import matplotlib.pyplot as plt
from model import QZhangNet
from hw04_training import qzDatasetClass
from torch.utils.data import DataLoader
# from torch.utils.data import DataLoader, Dataset


def validation(net, val_loader, mat_size, device='cpu'):
```

```python
    confusion_mat = np.zeros([mat_size, mat_size], dtype=int)
    net = copy.deepcopy(net)
    net = net.to(device)
    for i, data in enumerate(val_loader):
        print(i)
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = net(inputs)
        # print(outputs)
        label_pred = []
        for output in outputs:
            label_pred.append(torch.argmax(output))
        for record_number in range(len(labels)):
            confusion_mat[labels[record_number]][label_pred[record_number]] += 1
    return confusion_mat


def plot_confusion_matrix(conf_mat, label_list, net_type=1):
    size = len(conf_mat)
    cat_size = np.sum(conf_mat) / size
    labels = []
    for row in range(size):
        rows = []
        for col in range(size):
            count = conf_mat[row][col]
            percent = "%.2f%%" % (count / cat_size * 100)
            label = str(count) + '\n' + str(percent)
            rows.append(label)
        labels.append(rows)
    labels = np.asarray(labels)

    accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
    stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    plt.figure(figsize=(10, 10))
    sns.heatmap(conf_mat, annot=labels, fmt="", cmap="Blues", cbar=False,
                xticklabels=label_list, yticklabels=label_list)
    plt.ylabel('True label')
    plt.xlabel('Predicted label' + stats_text)
    plt.title('Confusion Matrix')
    file_name = 'net'+str(net_type)+'_confusion_matrix.jpg'
    plt.savefig(file_name)
    plt.show()
```

```python
if __name__ == '__main__':
    print('start')
    if torch.cuda.is_available():
        device = 'cuda:0'
    else:
        device = 'cpu'

    parser = argparse.ArgumentParser(description='HW04 Training')
    parser.add_argument('--root_path', required=True, type=str)
    parser.add_argument('--net_path', required=True, type=str)
    parser.add_argument('--class_list', required=True, nargs='*', type=str)
    args, args_other = parser.parse_known_args()

    seed = 0
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    np.random.seed(seed)
    # torch.backends.cudnn.deterministic = True
    # torch.backends.cudnn.benchmarks = False
    os.environ['PYTHONHASHSEED'] = str(seed)

    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    batch_size = 10
    # catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe", "horse", "refrigerator", "train", "truck"]
    # dataFolder = "./hw04_coco_data/Val/"
    catList = args.class_list
    dataFolder = args.root_path
    netFolder = args.net_path
    catNum = len(catList)
    valSet = qzDatasetClass(dataFolder, catList, transform)
    valLoader = DataLoader(dataset=valSet, batch_size=batch_size, shuffle=True, num_workers=4)

    # net 1
    net1 = QZhangNet(net_type=1, class_num=catNum)
    net1.load_state_dict(torch.load((netFolder+"net1.pth"), map_location=torch.device(device)))
    net1.eval()
    confusion_matrix = validation(net1, valLoader, mat_size=len(catList), device=device)
    plot_confusion_matrix(confusion_matrix, catList, net_type=1)

    # net 2
```

```python
    net2 = QZhangNet(net_type=2, class_num=catNum)
    net2.load_state_dict(torch.load((netFolder + "net2.pth"), map_location=torch.device(device)))
    net2.eval()
    confusion_matrix = validation(net2, valLoader, mat_size=len(catList), device=device)
    plot_confusion_matrix(confusion_matrix, catList, net_type=2)

    # net 3
    net3 = QZhangNet(net_type=3, class_num=catNum)
    net3.load_state_dict(torch.load((netFolder + "net3.pth"), map_location=torch.device(device)))
    net3.eval()
    confusion_matrix = validation(net3, valLoader, mat_size=len(catList), device=device)
    plot_confusion_matrix(confusion_matrix, catList, net_type=3)
    # print(confusion_matrix)
    print('Valuation finished')
```

3.2 Results

The output images are shown below.

3.3 COCO Downloader

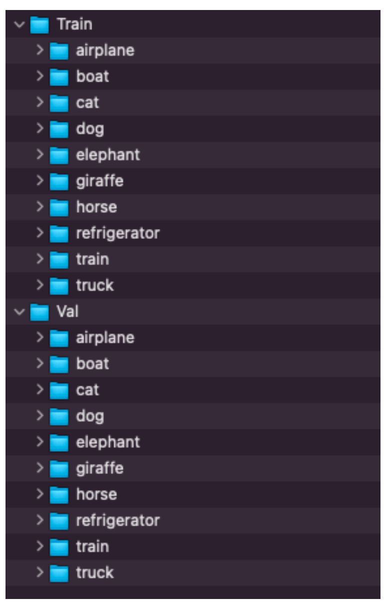

Figure 1: The downloaded folder.

3.4 Training loss

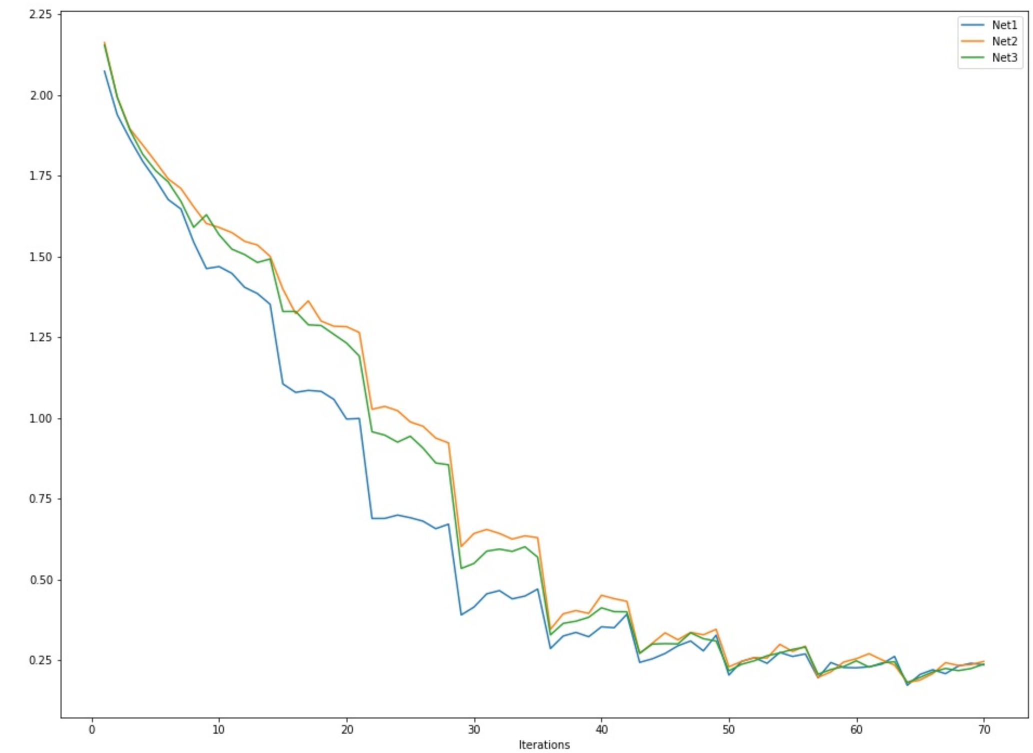

Figure 2: Three different net's training loss.

3.5 Confusion Matrix

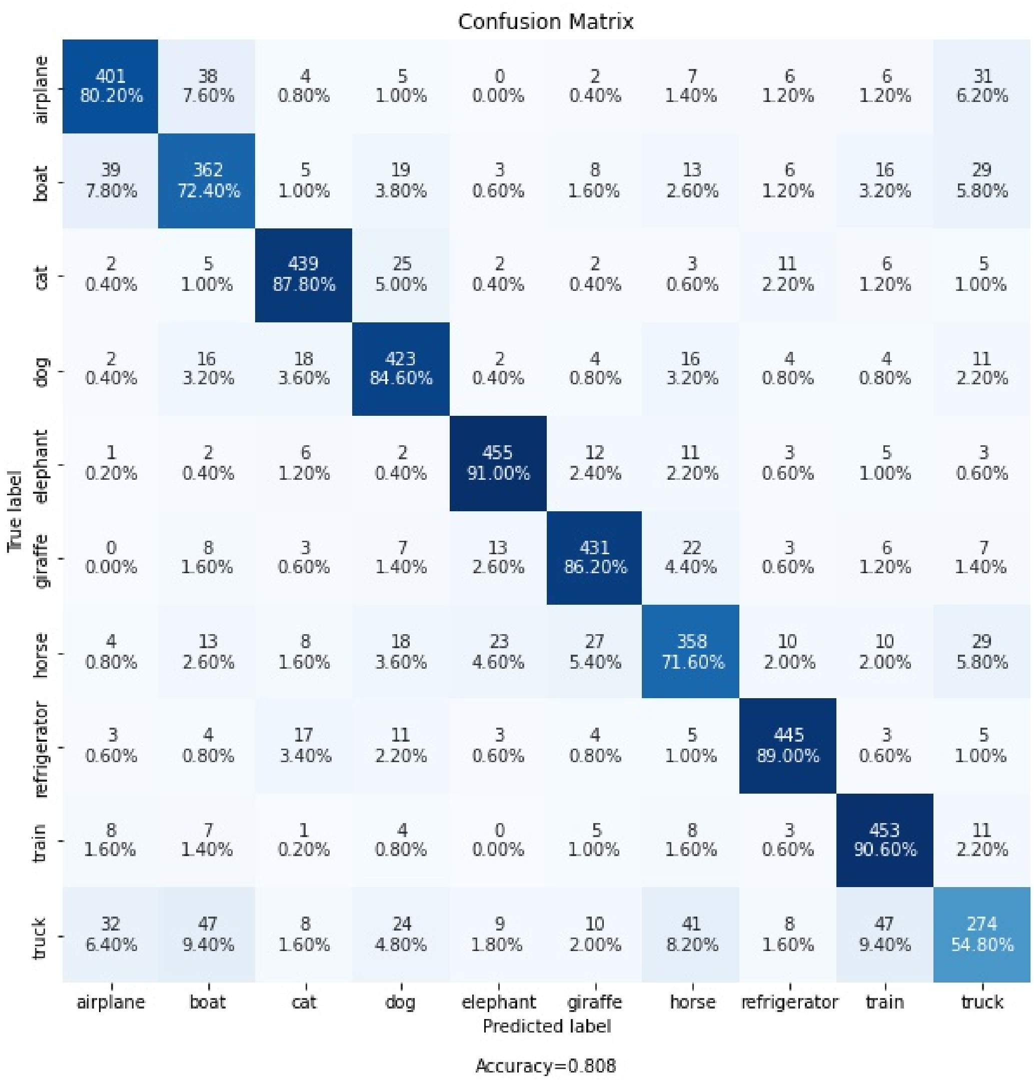

Figure 3: Net1 confusion matrix.

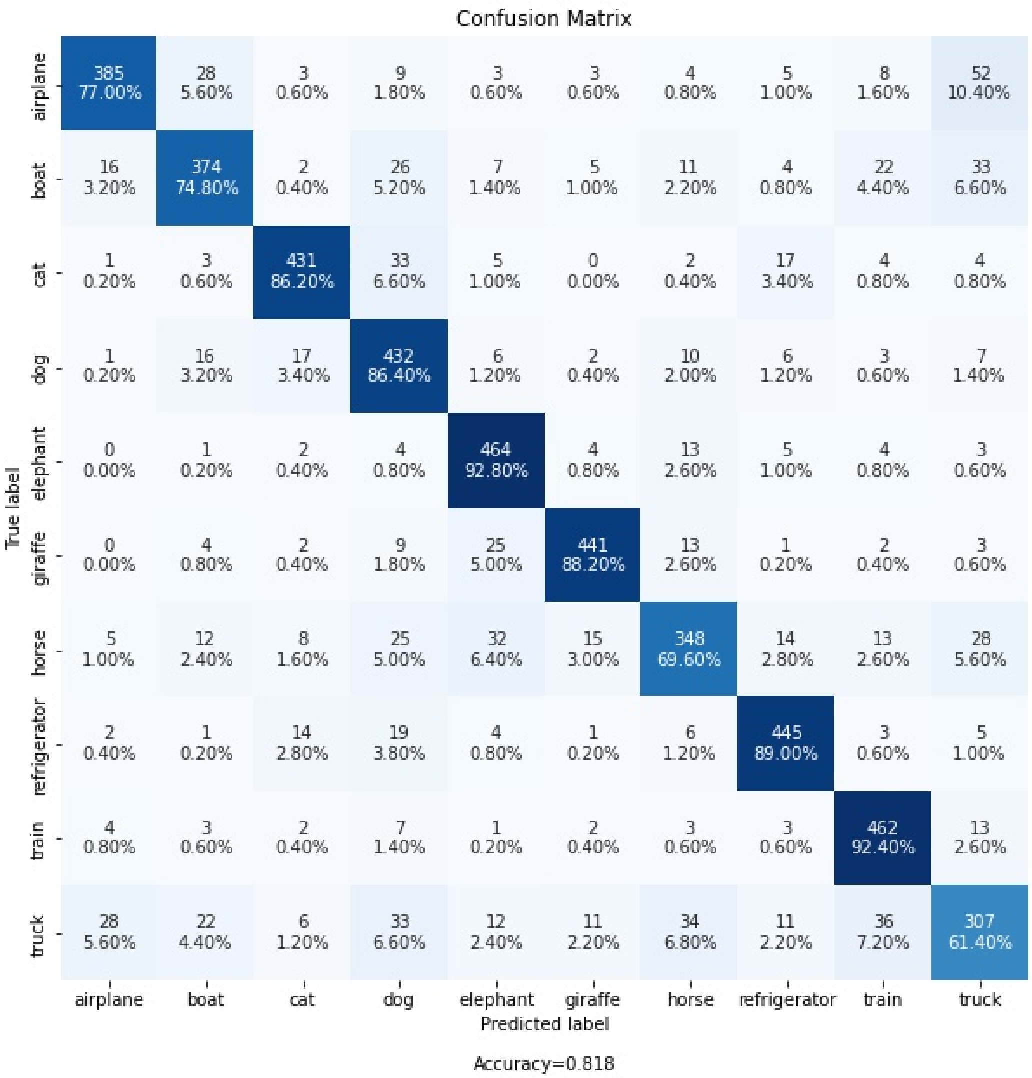

Figure 4: Net2 confusion matrix.

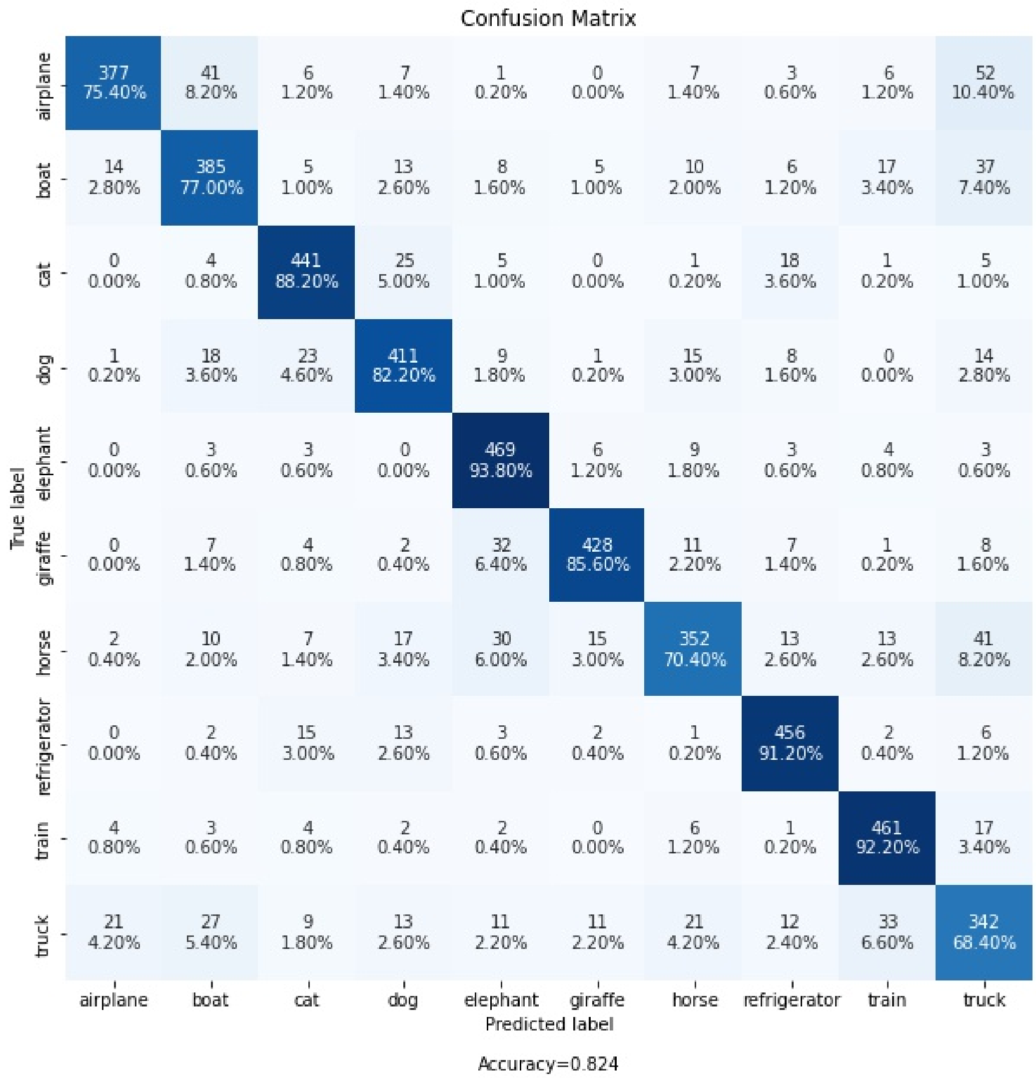

Figure 5: Net3 confusion matrix.

4 Lessons Learned

1. The simpler network structure may have better convergence speed due to the less number of the parameters, such as `net1`.
2. The purpose for the script `torch.backends.cudnn.deterministic = True` and `torch.backends.cudnn.benchmark = False` may make the program be slower and more reproducible.
3. Some multiprocess error may occur if not put the main code under the `if __name__ ==`

`'__main__':`

4. Passing the complete dataset once in the neural network is not enough, thus we need to have multiple epochs to pass the complete dataset several times in the same neural network.
5. It is better to specify the `map_location` when load the network file. Otherwise, there maybe a error occurs.
6. The `load_state_dict()` function takes a dictionary object, not a path to a saved object. This means that the saved `state_dict` must bedeserialized before pass it to the `load_state_dict()` function.
7. Call `model.eval()` to set dropout and batch normalization layers to evaluation mode before running inference.

5 Suggested Enhancements

1. Try different parameters such as batch size and learning rate to see the if there is more improvement could be made.
2. Construct a deeper convolutional layer in the network.
3. More comments in code should be added.
4. Give a more flexible adjustments to the path input.