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:

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:

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

```
1 import argparse
 2 import requests
 3 import os
 4 import torch
 5 import random
 6 import numpy as np
 7
 8 from PIL import Image
 9 from pycocotools.coco import COCO
10 # from requests.exceptions import ConnectionError, ReadTimeout,
    → TooManyRedirects, MissingSchema, InvalidURL
11
12
   # If want to use preset input
13
   # class Fake_args():
          def __init__(self, data_type='val2017'):
14
              self.root_path = '/Users/admin/Downloads/coco/'
15
              self.coco\_json\_path =
16
   #
        '{}/annotations2017/instances_{}.json'.format(self.root_path,data_type)
              #self.class_list = {0: "airplane", 1: "boat"}
17
              self.class_list = ["airplane", "boat"]
18 #
19 #
              self.images\_per\_class = 5
20
21
22 class Downloader:
       def __init__(self, info):
23
           self.root = info.root_path
24
            self.json = info.coco_json_path
25
           self.class_list = info.class_list
26
            self.images_per_class = info.images_per_class
27
            self.cat_folder = dict.fromkeys(self.class_list)
28
29
           self.coco = COCO(self.json)
30
31
        def make_folder(self):
32
           for cat in self.class_list:
                folder_root = self.root + cat
33
34
                self.cat_folder[cat] = folder_root
                if not os.path.exists(folder_root):
35
36
                    os.makedirs(folder_root)
37
38
        def get_image(self):
```

```
39
            for cat in self.class_list:
                cat_id = self.coco.getCatIds(cat)
40
                img_id = self.coco.getImgIds(catIds=cat_id)
41
42
                imgs = self.coco.loadImgs(img_id)
                save_number = 0
43
                img_index = 0
44
45
                while save_number < self.images_per_class:</pre>
                     # for number in range(self.images_per_class):
46
                     img_path = self.root + cat + "/" +
47

    imgs[img_index]['file_name']

48
                    save_check = self.save_image(img_path,

    imgs[img_index]['coco_url'])

49
                    if save_check:
50
                         self.resize_image(img_path)
51
                         save_number += 1
                    img_index += 1
52
53
54
            print("Download Finished")
55
56
        @staticmethod
57
        def resize_image(img_save_path):
            image = Image.open(img_save_path)
58
            if image.mode != "RGB":
59
                image = image.convert(mode="RGB")
60
61
            image_resized = image.resize((64, 64), Image.BOX)
62
            image_resized.save(img_save_path)
63
64
        Ostaticmethod
65
        def save_image(img_save_path, img_url):
66
            try:
67
                img_response = requests.get(img_url, timeout=1)
68
            except requests.exceptions as e:
                return False
69
70
            with open(img_save_path, 'wb') as img_f:
71
                img_f.write(img_response.content)
72
            return True
73
74
75 if __name__ == "__main__":
        seed = 0
76
77
        random.seed(seed)
78
        torch.manual_seed(seed)
79
        torch.cuda.manual_seed(seed)
        np.random.seed(seed)
80
        # torch.backends.cudnn.deterministic = True
81
82
        # torch.backends.cudnn.benchmarks = False
```

```
83
        os.environ['PYTHONHASHSEED'] = str(seed)
84
85
        parser = argparse.ArgumentParser(description='HW04 CocoDownloader')
86
        parser.add_argument('--root_path', required=True, type=str)
87
        parser.add_argument('--coco_json_path', required=True, type=str)
        parser.add_argument('--class_list', required=True, nargs='*', type=str)
88
89
        parser.add_argument('--images_per_class', required=True, type=int)
90
91
        args, args_other = parser.parse_known_args()
92
93
        # args = Fake_args()
94
95
        myDownloader = Downloader(args)
96
        myDownloader.make_folder()
97
        myDownloader.get_image()
    3.1.2 ok.py
 1 import random
 2 import numpy
 3 import os
 4 import argparse
 5 import torchvision.transforms as tvt
 6 import torch.utils.data
 7 from torch.utils.data import DataLoader
 8 import matplotlib.pyplot as plt
 9
10 from model import QZhangNet
11 from hw04_training import run_code_for_training
12 from hw04_training import qzDatasetClass
13
14 if __name__ == '__main__':
15
        if torch.cuda.is_available():
            device = 'cuda:0'
16
17
        else:
18
            device = 'cpu'
19
20
        parser = argparse.ArgumentParser(description='HW04 Training')
        parser.add_argument('--root_path', required=True, type=str)
21
22
        parser.add_argument('--class_list', required=True, nargs='*', type=str)
23
        args, args_other = parser.parse_known_args()
24
25
        seed = 0
26
        random.seed(seed)
27
        torch.manual_seed(seed)
        torch.cuda.manual_seed(seed)
28
```

```
29
        numpy.random.seed(seed)
        os.environ['PYTHONHASHSEED'] = str(seed)
30
31
32
        # Load and normalize data
        transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5),
33
        \rightarrow (0.5, 0.5, 0.5))])
        batch_size = 10
34
35
36
        # Local Fake
        # catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe",
37
        → "horse", "refrigerator", "train", "truck"]
        # catList = ["airplane", "boat"]
38
        # dataFolder = "./hw04_coco_data/Train/"
39
40
        catList = args.class_list
41
        dataFolder = args.root_path
        catNum = len(catList)
42
43
44
45
        trainSet = qzDatasetClass(dataFolder, catList, transform)
46
        trainLoader = DataLoader(dataset=trainSet, batch_size=batch_size,

    shuffle=True, num_workers=4)

47
        print('first')
48
49
        # First Net
50
        net1 = QZhangNet(net_type=1, class_num=catNum)
51
        print('done1')
52
        net1Loss, iter1 = run_code_for_training(net1, trainLoader,
        → learning_rate=1e-3,
53
                                                 momentum_set=0.9, epochs=10,
                                                  → device=device, net_type=1)
54
        print('done first')
55
56
        # Second Net
57
        net2 = QZhangNet(net_type=2, class_num=catNum)
58
        print('done2')
59
        net2Loss, iter2 = run_code_for_training(net2, trainLoader,

→ learning_rate=1e-3,
60
                                                 momentum_set=0.9, epochs=10,
                                                  → device=device, net_type=2)
61
        print('done second')
62
63
        # Third Net
64
        net3 = QZhangNet(net_type=3, class_num=catNum)
        print('done3')
65
66
        net3Loss, iter3 = run_code_for_training(net3, trainLoader,
        → learning_rate=1e-3,
```

```
67
                                                momentum_set=0.9, epochs=10,
                                                 → device=device, net_type=3)
68
        print('done third')
69
70
        plt.plot(net1Loss, label='Net1')
        plt.plot(net2Loss, label='Net2')
71
72
        plt.plot(net3Loss, label='Net3')
73
        plt.legend()
74
        plt.show()
75
76
        print('done4')
    3.1.3 model.py
 1 # import torch
   import torch.nn as nn
   import torch.nn.functional as functional
 4
 5
 6 class QZhangNet(nn.Module):
 7
        def __init__(self, net_type=1, class_num=10):
            super(QZhangNet, self).__init__()
 8
9
            self.net_type = net_type
            self.conv1 = nn.Conv2d(3, 128, 3) # Default Value
10
            self.conv1_3 = nn.Conv2d(3, 128, 3, padding=1)
11
12
            self.conv2 = nn.Conv2d(128, 128, 3) # (B)
13
            self.pool = nn.MaxPool2d(2, 2)
14
15
            self.fc1_1 = nn.Linear(128 * 31 * 31, 1000)
16
            self.fc1_2 = nn.Linear(128 * 14 * 14, 1000)
17
            self.fc1_3 = nn.Linear(128 * 15 * 15, 1000)
18
            self.fc2 = nn.Linear(1000, class_num)
19
20
        def forward(self, x):
            11 11 11
21
22
            Uncomment the next statement and see what happens to the
23
            performance of your classifier with and without padding.
24
            Note that you will have to change the first arg in the
            call to Linear in line (C) above and in the line (E)
25
26
            shown below. After you have done this experiment, see
            if the statement shown below can be invoked twice with
27
28
            and without padding. How about three times?
29
30
            if self.net_type == 1:
                x = self.pool(functional.relu(self.conv1(x)))
31
                x = x.view(-1, 128 * 31 * 31) # (E)
32
```

```
33
                x = functional.relu(self.fc1_1(x))
34
            elif self.net_type == 2:
                x = self.pool(functional.relu(self.conv1(x)))
35
36
                x = self.pool(functional.relu(self.conv2(x)))
                x = x.view(-1, 128 * 14 * 14)
37
                x = functional.relu(self.fc1_2(x))
38
            elif self.net_type == 3:
39
                x = self.pool(functional.relu(self.conv1_3(x)))
40
                x = self.pool(functional.relu(self.conv2(x)))
41
42
                x = x.view(-1, 128 * 15 * 15)
43
                x = functional.relu(self.fc1_3(x))
44
            x = self.fc2(x)
45
            return x
```

3.1.4 hw04_training.py

```
1 import random
 2 import numpy
 3 import os
 4 # import sys
 5 import glob
6 import argparse
 7 from PIL import Image
 8 import torch
9 import torchvision.transforms as tvt
10 import torch.utils.data
11 from torch.utils.data import DataLoader, Dataset
12 import matplotlib.pyplot as plt
13 from model import QZhangNet
14
15
16 class qzDatasetClass(Dataset):
17
        def __init__(self, root, category_list, trans=None):
18
19
            Make use of the arguments from the calling
20
            routine to initialise the variables
21
            e.g.image path lists for cat and dog classes
            you could also maintain label_array
22
            0 - - airplane
23
24
            1 - - boat
25
            2 - - cat.
26
27
28
            9 - - truck
29
            Initialise the required transform
30
```

```
31
            self.transform = trans
32
            self.category_list = category_list
33
            self.root = root
34
            self.path_list = []
35
            self.img_info = []
36
37
            for cat in self.category_list:
                cat_path = self.root + cat + "/"
38
                self.path_list.append(cat_path)
39
                cat_label = self.category_list.index(cat)
40
41
                search = cat_path + "*"
                for img in glob.glob(search):
42
                    img_info = [cat_label, img]
43
44
                    self.img_info.append(img_info)
45
46
        def __len__(self):
            11 11 11
47
48
            return the total number of images
49
            refer pytorch documentation for more details
50
51
            return len(self.img_info)
52
        def __getitem__(self, idx):
53
54
            img_path = self.img_info[idx][1]
55
            label = self.img_info[idx][0]
            image = Image.open(img_path)
56
57
            im_ts = self.transform(image)
58
            return im_ts, label
59
60
   def run_code_for_training(net, trainLoader, learning_rate=1e-3,
61

→ momentum_set=0.9, epochs=10,

62
                              device='cuda:0', net_type=1):
63
        net = net.to(device)
64
        criterion = torch.nn.CrossEntropyLoss()
65
        optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate,
        → momentum=momentum_set)
66
        loss_running_record = []
        Iter_record = []
67
        for epoch in range(epochs):
68
            running_loss = 0.0
69
70
            for i, data in enumerate(trainLoader):
71
                print(i)
72
                inputs, labels = data
73
                inputs = inputs.to(device)
74
                labels = labels.to(device)
```

```
75
                 optimizer.zero_grad()
 76
                 outputs = net(inputs)
 77
                 loss = criterion(outputs, labels)
 78
                 loss.backward()
 79
                 optimizer.step()
 80
                 running_loss += loss.item()
                 if (i + 1) \% 500 == 0:
 81
                     points = running_loss / float(500)
 82
 83
                     loss_running_record.append(points)
 84
                     Iter_record.append(len(loss_running_record))
 85
                     print("\n[epoch:%d, batch:%5d] loss: %.3f" % (epoch + 1, i
                      → + 1, points))
                     running_loss = 0.0
 86
 87
         # netpath = './net1.pth'
 88
         netpath = './net' + str(net_type) + '.pth'
 89
         torch.save(net.state_dict(), netpath)
 90
         print('Finished Training')
 91
         return loss_running_record, Iter_record
 92
 93
     if __name__ == '__main__':
 94
         #If want to run directly
 95
 96
         # Training on GPU or CPU
 97
 98
         if torch.cuda.is_available():
 99
             device = 'cuda:0'
100
         else:
             device = 'cpu'
101
102
         seed = 0
103
         random.seed(seed)
104
105
         torch.manual_seed(seed)
106
         torch.cuda.manual_seed(seed)
107
         numpy.random.seed(seed)
         # torch.backends.cudnn.deterministic = True
108
109
         # torch.backends.cudnn.benchmarks = False
         os.environ['PYTHONHASHSEED'] = str(seed)
110
111
112
         # Load and normalize data
         transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5),
113
         \rightarrow (0.5, 0.5, 0.5))])
114
115
         # parser = argparse.ArgumentParser(description='HW04 Training')
         # parser.add_argument('--root_path', required=True, type=str)
116
         # parser.add_argument('--class_list', required=True, nargs='*',
117
          \rightarrow type=str)
```

```
118
         # args, args_other = parser.parse_known_args()
119
120
         # catList = args.class_list
121
         # root = args.root_path
122
123
         batch_size = 10
124
         catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe",
125
         → "horse", "refrigerator", "train", "truck"]
126
         dataFolder = "./hw04_coco_data/Train/"
127
         trainSet = qzDatasetClass(dataFolder, catList, transform)
128
         trainLoader = DataLoader(dataset=trainSet, batch_size=batch_size,

    shuffle=True, num_workers=4)

129
         print('Loader Created')
130
131
         # First Net
         net1 = QZhangNet()
132
133
         print('Net Created')
         net1Loss, iter1 = run_code_for_training(net1, trainLoader,
134
         → learning_rate=1e-3,
135
                                                 momentum_set=0.9, epochs=10,
                                                  → device=device, net_type=1)
         plt.plot(net1Loss, label='Net1')
136
137
         plt.legend()
138
         plt.show()
139
140
         print('Done')
     3.1.5 hw04_validation.py
  1 import copy
  2 import random
  3 import torch
  4 import os
  5 import argparse
  6 import numpy as np
  7 import torchvision.transforms as tvt
  8 import seaborn as sns
  9 import matplotlib.pyplot as plt
 10 from model import QZhangNet
 11 from hw04_training import qzDatasetClass
 12 from torch.utils.data import DataLoader
 13 # from torch.utils.data import DataLoader, Dataset
14
 15
 16 def validation(net, val_loader, mat_size, device='cpu'):
```

```
17
        confusion_mat = np.zeros([mat_size, mat_size], dtype=int)
        net = copy.deepcopy(net)
18
19
        net = net.to(device)
20
        for i, data in enumerate(val_loader):
21
            print(i)
            inputs, labels = data
22
            inputs = inputs.to(device)
23
            labels = labels.to(device)
24
25
            outputs = net(inputs)
26
            # print(outputs)
27
            label_pred = []
            for output in outputs:
28
                label_pred.append(torch.argmax(output))
29
30
            for record_number in range(len(labels)):
31
                confusion_mat[labels[record_number]] [label_pred[record_number]]

→ += 1

32
        return confusion_mat
33
34
35
    def plot_confusion_matrix(conf_mat, label_list, net_type=1):
36
        size = len(conf_mat)
37
        cat_size = np.sum(conf_mat) / size
        labels = []
38
39
        for row in range(size):
40
            rows = []
            for col in range(size):
41
42
                count = conf_mat[row][col]
                percent = \frac{8.2f\%\%}{3} % (count / cat_size * 100)
43
                label = str(count) + '\n' + str(percent)
44
45
                rows.append(label)
            labels.append(rows)
46
47
        labels = np.asarray(labels)
48
49
        accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
        stats_text = "\n\Delta curacy = \underline{-:0.3f''}".format(accuracy)
50
51
        plt.figure(figsize=(10, 10))
        sns.heatmap(conf_mat, annot=labels, fmt="", cmap="Blues", cbar=False,
52
                    xticklabels=label_list, yticklabels=label_list)
53
        plt.ylabel('True label')
54
        plt.xlabel('Predicted label' + stats_text)
55
        plt.title('Confusion Matrix')
56
57
        file_name = 'net'+str(net_type)+'_confusion_matrix.jpg'
        plt.savefig(file_name)
58
59
        plt.show()
60
61
```

```
62 if __name__ == '__main__':
         print('start')
63
64
         if torch.cuda.is_available():
65
             device = 'cuda:0'
66
         else:
67
             device = 'cpu'
68
69
         parser = argparse.ArgumentParser(description='HW04 Training')
70
         parser.add_argument('--root_path', required=True, type=str)
71
         parser.add_argument('--net_path', required=True, type=str)
72
         parser.add_argument('--class_list', required=True, nargs='*', type=str)
73
         args, args_other = parser.parse_known_args()
74
75
         seed = 0
76
         random.seed(seed)
77
         torch.manual seed(seed)
78
         torch.cuda.manual_seed(seed)
79
         np.random.seed(seed)
80
         # torch.backends.cudnn.deterministic = True
         # torch.backends.cudnn.benchmarks = False
81
82
         os.environ['PYTHONHASHSEED'] = str(seed)
83
84
         transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5),
         \rightarrow (0.5, 0.5, 0.5))])
85
         batch_size = 10
         # catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe",
86
         → "horse", "refrigerator", "train", "truck"]
         # dataFolder = "./hw04_coco_data/Val/"
87
88
         catList = args.class_list
         dataFolder = args.root_path
89
         netFolder = args.net_path
90
91
         catNum = len(catList)
         valSet = qzDatasetClass(dataFolder, catList, transform)
92
93
         valLoader = DataLoader(dataset=valSet, batch_size=batch_size,

    shuffle=True, num_workers=4)

94
95
         # n.e.t. 1
96
         net1 = QZhangNet(net_type=1, class_num=catNum)
         net1.load_state_dict(torch.load((netFolder+"net1.pth")),
97

→ map_location=torch.device(device)))
         net1.eval()
98
99
         confusion_matrix = validation(net1, valLoader, mat_size=len(catList),
         → device=device)
         plot_confusion_matrix(confusion_matrix, catList, net_type=1)
100
101
102
         # net 2
```

```
net2 = QZhangNet(net_type=2, class_num=catNum)
103
         net2.load_state_dict(torch.load((netFolder + "net2.pth"),
104
             map_location=torch.device(device)))
105
         net2.eval()
         confusion_matrix = validation(net2, valLoader, mat_size=len(catList),
106
         → device=device)
         plot_confusion_matrix(confusion_matrix, catList, net_type=2)
107
108
109
         # net 3
110
         net3 = QZhangNet(net_type=3, class_num=catNum)
         net3.load_state_dict(torch.load((netFolder + "net3.pth"),
111

→ map_location=torch.device(device)))
         net3.eval()
112
113
         confusion_matrix = validation(net3, valLoader, mat_size=len(catList),
         → device=device)
         plot_confusion_matrix(confusion_matrix, catList, net_type=3)
114
         # print(confusion_matrix)
115
116
         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

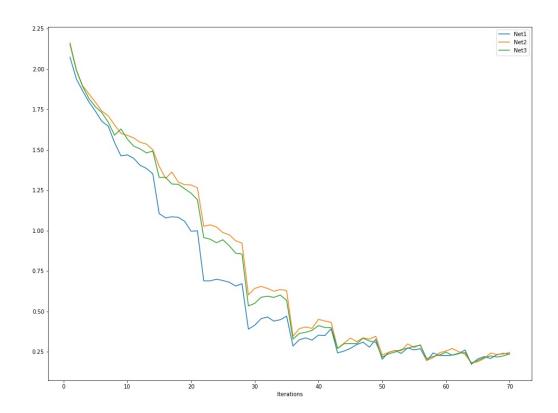


Figure 2: Three different net's training loss.

3.5 Confusion Matrix

Confusion Matrix											
airplane	401 80.20%	38 7.60%	4 0.80%	5 1.00%	0 0.00%	2 0.40%	7 1.40%	6 1.20%	6 1.20%	31 6.20%	
boat	39	362	5	19	3	8	13	6	16	29	
	7.80%	72.40%	1.00%	3.80%	0.60%	1.60%	2.60%	1.20%	3.20%	5.80%	
- at	2	5	439	25	2	2	3	11	6	5	
	0.40%	1.00%	87.80%	5.00%	0.40%	0.40%	0.60%	2.20%	1.20%	1.00%	
dog -	2	16	18	423	2	4	16	4	4	11	
	0.40%	3.20%	3.60%	84.60%	0.40%	0.80%	3.20%	0.80%	0.80%	2.20%	
True label	1	2	6	2	455	12	11	3	5	3	
ffe elephant	0.20%	0.40%	1.20%	0.40%	91.00%	2.40%	2.20%	0.60%	1.00%	0.60%	
True giraffe '	0 0.00%	8 1.60%	3 0.60%	7 1.40%	13 2.60%	431 86.20%	22 4.40%	3 0.60%	6 1.20%	7 1.40%	
horse	4	13	8	18	23	27	358	10	10	29	
	0.80%	2.60%	1.60%	3.60%	4.60%	5.40%	71.60%	2.00%	2.00%	5.80%	
refrigerator	3	4	17	11	3	4	5	445	3	5	
,	0.60%	0.80%	3.40%	2.20%	0.60%	0.80%	1.00%	89.00%	0.60%	1.00%	
train -	8	7	1	4	0	5	8	3	453	11	
	1.60%	1.40%	0.20%	0.80%	0.00%	1.00%	1.60%	0.60%	90.60%	2.20%	
ruck	32 6.40%	47 9.40%	8	24 4.80%	9 1.80%	10 2.00%	41 8.20%	8 1.60%	47 9.40%	274 54.80%	
	airplane	boat	cat	dog	elephant giraffe horse refrigerator train t Predicted label						
	Accuracy=0.808										

Figure 3: Net1 confusion matrix.

	Confusion Matrix									
airplane	385	28	3	9	3	3	4	5	8	52
	77.00%	5.60%	0.60%	1.80%	0.60%	0.60%	0.80%	1.00%	1.60%	10.40%
boat	16	374	2	26	7	5	11	4	22	33
	3.20%	74.80%	0.40%	5.20%	1.40%	1.00%	2.20%	0.80%	4.40%	6.60%
- at	1	3	431	33	5	0	2	17	4	4
	0.20%	0.60%	86.20%	6.60%	1.00%	0.00%	0.40%	3.40%	0.80%	0.80%
Бор	1	16	17	432	6	2	10	6	3	7
-	0.20%	3.20%	3.40%	86.40%	1.20%	0.40%	2.00%	1.20%	0.60%	1.40%
True label giraffe elephant '	0 0.00%	1 0.20%	2 0.40%	4 0.80%	464 92.80%	4 0.80%	13 2.60%	5 1.00%	4 0.80%	3 0.60%
True giraffe '	0.00%	4 0.80%	2 0.40%	9 1.80%	25 5.00%	441 88.20%	13 2.60%	1 0.20%	2 0.40%	3 0.60%
horse	5	12	8	25	32	15	348	14	13	28
	1.00%	2.40%	1.60%	5.00%	6.40%	3.00%	69.60%	2.80%	2.60%	5.60%
refrigerator	2	1	14	19	4	1	6	445	3	5
	0.40%	0.20%	2.80%	3.80%	0.80%	0.20%	1.20%	89.00%	0.60%	1.00%
tain -	4	3	2	7	1	2	3	3	462	13
	0.80%	0.60%	0.40%	1.40%	0.20%	0.40%	0.60%	0.60%	92.40%	2.60%
tuck -	28	22	6	33	12	11	34	11	36	307
	5.60%	4.40%	1.20%	6.60%	2.40%	2.20%	6.80%	2.20%	7.20%	61.40%
	airplane	rplane boat cat dog elephant giraffe Predicted label Accuracy=0.818						refrigerator	train	truck

Figure 4: Net2 confusion matrix.

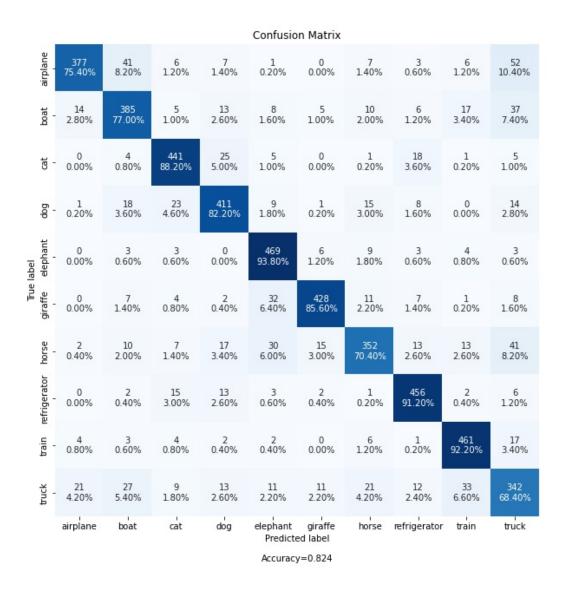


Figure 5: Net3 confusion matrix.

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 be described 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.