

# 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 ( $64 \times 64$ ) 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:

- 1 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
- 2 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
- 3 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

```
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():
14 #     def __init__(self, data_type='val2017'):
15 #         self.root_path = '/Users/admin/Downloads/coco/'
16 #         self.coco_json_path =
#             → '{}/annotations2017/instances_{}.json'.format(self.root_path, data_type)
17 #         self.class_list = {0: "airplane", 1: "boat"}
18 #         self.class_list = ["airplane", "boat"]
19 #         self.images_per_class = 5
20
21
22 class Downloader:
23     def __init__(self, info):
24         self.root = info.root_path
25         self.json = info.coco_json_path
26         self.class_list = info.class_list
27         self.images_per_class = info.images_per_class
28         self.cat_folder = dict.fromkeys(self.class_list)
29         self.coco = COCO(self.json)
30
31     def make_folder(self):
32         for cat in self.class_list:
33             folder_root = self.root + cat
34             self.cat_folder[cat] = folder_root
35             if not os.path.exists(folder_root):
36                 os.makedirs(folder_root)
37
38     def get_image(self):
```

```

39         for cat in self.class_list:
40             cat_id = self.coco.getCatIds(cat)
41             img_id = self.coco.getImgIds(catIds=cat_id)
42             imgs = self.coco.loadImgs(img_id)
43             save_number = 0
44             img_index = 0
45             while save_number < self.images_per_class:
46                 # for number in range(self.images_per_class):
47                 img_path = self.root + cat + "/" +
48                     imgs[img_index]['file_name']
49                 save_check = self.save_image(img_path,
50                     imgs[img_index]['coco_url'])
51                 if save_check:
52                     self.resize_image(img_path)
53                     save_number += 1
54                     img_index += 1
55
56     @staticmethod
57     def resize_image(img_save_path):
58         image = Image.open(img_save_path)
59         if image.mode != "RGB":
60             image = image.convert(mode="RGB")
61         image_resized = image.resize((64, 64), Image.BOX)
62         image_resized.save(img_save_path)
63
64     @staticmethod
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:
69             return False
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__":
76     seed = 0
77     random.seed(seed)
78     torch.manual_seed(seed)
79     torch.cuda.manual_seed(seed)
80     np.random.seed(seed)
81     # torch.backends.cudnn.deterministic = True
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)
88     parser.add_argument('--class_list', required=True, nargs='*', type=str)
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():
16         device = 'cuda:0'
17     else:
18         device = 'cpu'
19
20     parser = argparse.ArgumentParser(description='HW04 Training')
21     parser.add_argument('--root_path', required=True, type=str)
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)
28     torch.cuda.manual_seed(seed)

```

```

29     numpy.random.seed(seed)
30     os.environ['PYTHONHASHSEED'] = str(seed)
31
32     # Load and normalize data
33     transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5),
34         → (0.5, 0.5, 0.5))])
35     batch_size = 10
36
37     # Local Fake
38     # catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe",
39     → "horse", "refrigerator", "train", "truck"]
40     # catList = ["airplane", "boat"]
41     # dataFolder = "./hw04_coco_data/Train/"
42     catList = args.class_list
43     dataFolder = args.root_path
44     catNum = len(catList)
45
46     trainSet = qzDatasetClass(dataFolder, catList, transform)
47     trainLoader = DataLoader(dataset=trainSet, batch_size=batch_size,
48         → shuffle=True, num_workers=4)
49     print('first')
50
51     # First Net
52     net1 = QZhangNet(net_type=1, class_num=catNum)
53     print('done1')
54     net1Loss, iter1 = run_code_for_training(net1, trainLoader,
55         → learning_rate=1e-3,
56                                         momentum_set=0.9, epochs=10,
57                                         → device=device, net_type=1)
58     print('done first')
59
60     # Second Net
61     net2 = QZhangNet(net_type=2, class_num=catNum)
62     print('done2')
63     net2Loss, iter2 = run_code_for_training(net2, trainLoader,
64         → learning_rate=1e-3,
65                                         momentum_set=0.9, epochs=10,
66                                         → device=device, net_type=2)
67     print('done second')
68
69     # Third Net
70     net3 = QZhangNet(net_type=3, class_num=catNum)
71     print('done3')
72     net3Loss, iter3 = run_code_for_training(net3, trainLoader,
73         → learning_rate=1e-3,

```

```

67                                     momentum_set=0.9, epochs=10,
68                                     ↵ device=device, net_type=3)
69
70     print('done third')
71
72     plt.plot(net1Loss, label='Net1')
73     plt.plot(net2Loss, label='Net2')
74     plt.plot(net3Loss, label='Net3')
75     plt.legend()
76     plt.show()
77
78     print('done4')

```

### 3.1.3 model.py

```

1  # import torch
2  import torch.nn as nn
3  import torch.nn.functional as functional
4
5
6  class QZhangNet(nn.Module):
7      def __init__(self, net_type=1, class_num=10):
8          super(QZhangNet, self).__init__()
9          self.net_type = net_type
10         self.conv1 = nn.Conv2d(3, 128, 3) # Default Value
11         self.conv1_3 = nn.Conv2d(3, 128, 3, padding=1)
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):
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
25             call to Linear in line (C) above and in the line (E)
26             shown below. After you have done this experiment, see
27             if the statement shown below can be invoked twice with
28             and without padding. How about three times?
29             """
30
31         if self.net_type == 1:
32             x = self.pool(functional.relu(self.conv1(x)))
33             x = x.view(-1, 128 * 31 * 31) # (E)

```

```

33         x = functional.relu(self.fc1_1(x))
34     elif self.net_type == 2:
35         x = self.pool(functional.relu(self.conv1(x)))
36         x = self.pool(functional.relu(self.conv2(x)))
37         x = x.view(-1, 128 * 14 * 14)
38         x = functional.relu(self.fc1_2(x))
39     elif self.net_type == 3:
40         x = self.pool(functional.relu(self.conv1_3(x)))
41         x = self.pool(functional.relu(self.conv2(x)))
42         x = x.view(-1, 128 * 15 * 15)
43         x = functional.relu(self.fc1_3(x))
44     x = self.fc2(x)
45
46     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
22             you could also maintain label_array
23             0 -- airplane
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:
38         cat_path = self.root + cat + "/"
39         self.path_list.append(cat_path)
40         cat_label = self.category_list.index(cat)
41         search = cat_path + "*"
42         for img in glob.glob(search):
43             img_info = [cat_label, img]
44             self.img_info.append(img_info)
45
46     def __len__(self):
47         """
48             return the total number of images
49             refer pytorch documentation for more details
50         """
51         return len(self.img_info)
52
53     def __getitem__(self, idx):
54         img_path = self.img_info[idx][1]
55         label = self.img_info[idx][0]
56         image = Image.open(img_path)
57         im_ts = self.transform(image)
58         return im_ts, label
59
60
61     def run_code_for_training(net, trainLoader, learning_rate=1e-3,
62         → momentum_set=0.9, epochs=10,
63             → device='cuda:0', net_type=1):
64         net = net.to(device)
65         criterion = torch.nn.CrossEntropyLoss()
66         optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate,
67             → momentum=momentum_set)
68         loss_running_record = []
69         Iter_record = []
70         for epoch in range(epochs):
71             running_loss = 0.0
72             for i, data in enumerate(trainLoader):
73                 print(i)
74                 inputs, labels = data
75                 inputs = inputs.to(device)
76                 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()
81     if (i + 1) % 500 == 0:
82         points = running_loss / float(500)
83         loss_running_record.append(points)
84         Iter_record.append(len(loss_running_record))
85         print("\n[epoch:{}d, batch:{}d] loss: {:.3f}" % (epoch + 1, i
86             + 1, points))
87         running_loss = 0.0
88     # netpath = './net1.pth'
89     netpath = './net' + str(net_type) + '.pth'
90     torch.save(net.state_dict(), netpath)
91     print('Finished Training')
92     return loss_running_record, Iter_record
93
94 if __name__ == '__main__':
95     #If want to run directly
96
97     # Training on GPU or CPU
98     if torch.cuda.is_available():
99         device = 'cuda:0'
100    else:
101        device = 'cpu'
102
103    seed = 0
104    random.seed(seed)
105    torch.manual_seed(seed)
106    torch.cuda.manual_seed(seed)
107    numpy.random.seed(seed)
108    # torch.backends.cudnn.deterministic = True
109    # torch.backends.cudnn.benchmarks = False
110    os.environ['PYTHONHASHSEED'] = str(seed)
111
112    # Load and normalize data
113    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5),
114        (0.5, 0.5, 0.5))])
115
116    # parser = argparse.ArgumentParser(description='HW04 Training')
117    # parser.add_argument('--root_path', required=True, type=str)
118    # parser.add_argument('--class_list', required=True, nargs='*',
119        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
125     catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe",
126                ↳ "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,
129                               ↳ shuffle=True, num_workers=4)
129     print('Loader Created')
130
131     # First Net
132     net1 = QZhangNet()
133     print('Net Created')
134     net1Loss, iter1 = run_code_for_training(net1, trainLoader,
135                                              ↳ learning_rate=1e-3,
136                                              momentum_set=0.9, epochs=10,
137                                              ↳ device=device, net_type=1)
136     plt.plot(net1Loss, label='Net1')
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)
18     net = copy.deepcopy(net)
19     net = net.to(device)
20     for i, data in enumerate(val_loader):
21         print(i)
22         inputs, labels = data
23         inputs = inputs.to(device)
24         labels = labels.to(device)
25         outputs = net(inputs)
26         # print(outputs)
27         label_pred = []
28         for output in outputs:
29             label_pred.append(torch.argmax(output))
30         for record_number in range(len(labels)):
31             confusion_mat[labels[record_number]][label_pred[record_number]] =
32             += 1
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
38         labels = []
39         for row in range(size):
40             rows = []
41             for col in range(size):
42                 count = conf_mat[row][col]
43                 percent = "%.{2f}%" % (count / cat_size * 100)
44                 label = str(count) + '\n' + str(percent)
45                 rows.append(label)
46             labels.append(rows)
47         labels = np.asarray(labels)
48
49         accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
50         stats_text = "\n\nAccuracy=-{:0.3f}".format(accuracy)
51         plt.figure(figsize=(10, 10))
52         sns.heatmap(conf_mat, annot=labels, fmt="", cmap="Blues", cbar=False,
53                     xticklabels=label_list, yticklabels=label_list)
54         plt.ylabel('True label')
55         plt.xlabel('Predicted label' + stats_text)
56         plt.title('Confusion Matrix')
57         file_name = 'net'+str(net_type)+ '_confusion_matrix.jpg'
58         plt.savefig(file_name)
59         plt.show()
60
61

```

```

62 if __name__ == '__main__':
63     print('start')
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
81 # torch.backends.cudnn.benchmarks = False
82 os.environ['PYTHONHASHSEED'] = str(seed)
83
84 transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5),
85             (0.5, 0.5, 0.5))])
86 batch_size = 10
87 # catList = ["airplane", "boat", "cat", "dog", "elephant", "giraffe",
88 #             "horse", "refrigerator", "train", "truck"]
89 # dataFolder = "./hw04_coco_data/Val/"
90 catList = args.class_list
91 dataFolder = args.root_path
92 netFolder = args.net_path
93 catNum = len(catList)
94 valSet = qzDatasetClass(dataFolder, catList, transform)
95 valLoader = DataLoader(dataset=valSet, batch_size=batch_size,
96             shuffle=True, num_workers=4)
97
98 # net 1
99 net1 = QZhangNet(net_type=1, class_num=catNum)
100 net1.load_state_dict(torch.load(netFolder+"net1.pth"),
101             map_location=torch.device(device)))
102 net1.eval()
103 confusion_matrix = validation(net1, valLoader, mat_size=len(catList),
104             device=device)
105 plot_confusion_matrix(confusion_matrix, catList, net_type=1)
106
107 # net 2

```

```

103     net2 = QZhangNet(net_type=2, class_num=catNum)
104     net2.load_state_dict(torch.load(netFolder + "net2.pth"),
105                           map_location=torch.device(device)))
106     net2.eval()
107     confusion_matrix = validation(net2, valLoader, mat_size=len(catList),
108                                     device=device)
109     plot_confusion_matrix(confusion_matrix, catList, net_type=2)
110
111     # net 3
112     net3 = QZhangNet(net_type=3, class_num=catNum)
113     net3.load_state_dict(torch.load(netFolder + "net3.pth"),
114                           map_location=torch.device(device)))
115     net3.eval()
116     confusion_matrix = validation(net3, valLoader, mat_size=len(catList),
117                                     device=device)
118     plot_confusion_matrix(confusion_matrix, catList, net_type=3)
119     # print(confusion_matrix)
120     print('Valuation finished')

```

### 3.2 Results

The output images are shown below.

### 3.3 COCO Downloader

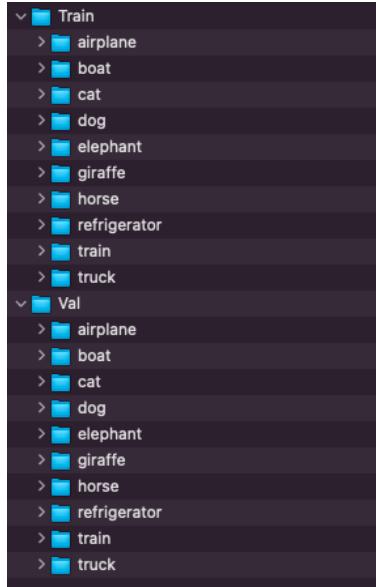


Figure 1: The downloaded folder.

### 3.4 Training loss

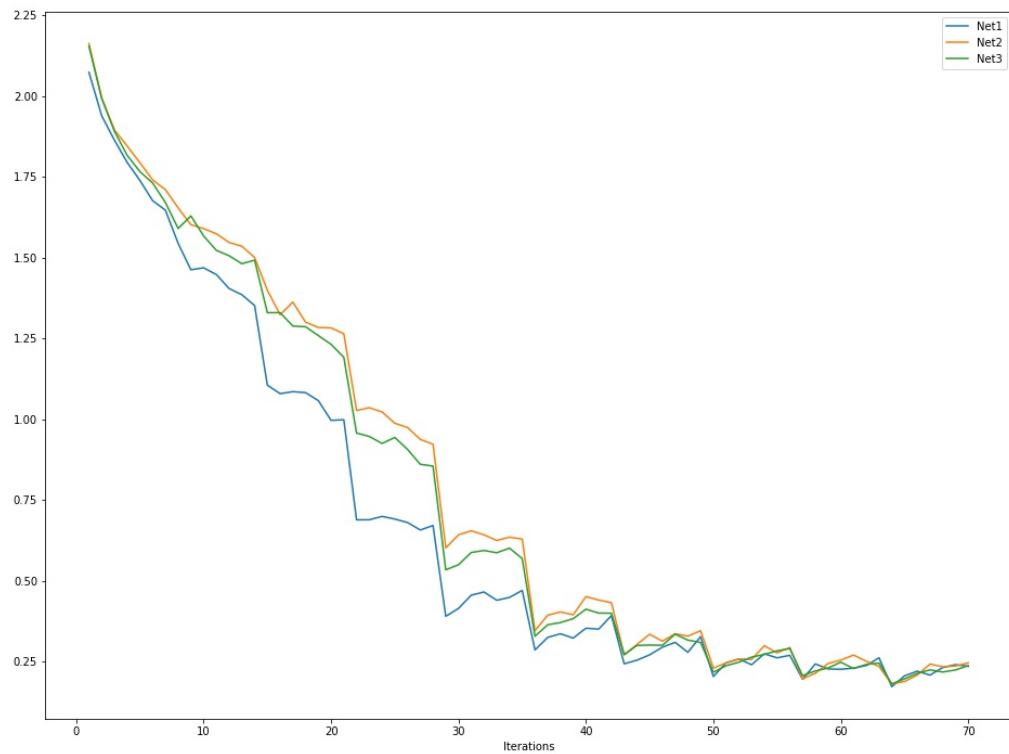


Figure 2: Three different net's training loss.

### 3.5 Confusion Matrix

Confusion Matrix										
	airplane	boat	cat	dog	elephant	giraffe	horse	refrigerator	train	truck
True label	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%
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 7.80%	362 72.40%	5 1.00%	19 3.80%	3 0.60%	8 1.60%	13 2.60%	6 1.20%	16 3.20%	29 5.80%
cat	2 0.40%	5 1.00%	439 87.80%	25 5.00%	2 0.40%	2 0.40%	3 0.60%	11 2.20%	6 1.20%	5 1.00%
dog	2 0.40%	16 3.20%	18 3.60%	423 84.60%	2 0.40%	4 0.80%	16 3.20%	4 0.80%	4 0.80%	11 2.20%
elephant	1 0.20%	2 0.40%	6 1.20%	2 0.40%	455 91.00%	12 2.40%	11 2.20%	3 0.60%	5 1.00%	3 0.60%
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 0.80%	13 2.60%	8 1.60%	18 3.60%	23 4.60%	27 5.40%	358 71.60%	10 2.00%	10 2.00%	29 5.80%
refrigerator	3 0.60%	4 0.80%	17 3.40%	11 2.20%	3 0.60%	4 0.80%	5 1.00%	445 89.00%	3 0.60%	5 1.00%
train	8 1.60%	7 1.40%	1 0.20%	4 0.80%	0 0.00%	5 1.00%	8 1.60%	3 0.60%	453 90.60%	11 2.20%
truck	32 6.40%	47 9.40%	8 1.60%	24 4.80%	9 1.80%	10 2.00%	41 8.20%	8 1.60%	47 9.40%	274 54.80%
Predicted label										
Accuracy=0.808										

Figure 3: Net1 confusion matrix.

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

Accuracy=0.818

Figure 4: Net2 confusion matrix.

Confusion Matrix										
	airplane	boat	cat	dog	elephant	giraffe	horse	refrigerator	train	truck
True label	377 75.40%	41 8.20%	6 1.20%	7 1.40%	1 0.20%	0 0.00%	7 1.40%	3 0.60%	6 1.20%	52 10.40%
airplane	14 2.80%	385 77.00%	5 1.00%	13 2.60%	8 1.60%	5 1.00%	10 2.00%	6 1.20%	17 3.40%	37 7.40%
boat	0 0.00%	4 0.80%	441 88.20%	25 5.00%	5 1.00%	0 0.00%	1 0.20%	18 3.60%	1 0.20%	5 1.00%
cat	1 0.20%	18 3.60%	23 4.60%	411 82.20%	9 1.80%	1 0.20%	15 3.00%	8 1.60%	0 0.00%	14 2.80%
dog	0 0.00%	3 0.60%	3 0.60%	0 0.00%	469 93.80%	6 1.20%	9 1.80%	3 0.60%	4 0.80%	3 0.60%
elephant	0 0.00%	7 1.40%	4 0.80%	2 0.40%	32 6.40%	428 85.60%	11 2.20%	7 1.40%	1 0.20%	8 1.60%
giraffe	2 0.40%	10 2.00%	7 1.40%	17 3.40%	30 6.00%	15 3.00%	352 70.40%	13 2.60%	13 2.60%	41 8.20%
horse	0 0.00%	2 0.40%	15 3.00%	13 2.60%	3 0.60%	2 0.40%	1 0.20%	456 91.20%	2 0.40%	6 1.20%
refrigerator	4 0.80%	3 0.60%	4 0.80%	2 0.40%	2 0.40%	0 0.00%	6 1.20%	1 0.20%	461 92.20%	17 3.40%
train	21 4.20%	27 5.40%	9 1.80%	13 2.60%	11 2.20%	11 2.20%	21 4.20%	12 2.40%	33 6.60%	342 68.40%
truck	airplane	boat	cat	dog	elephant	giraffe	horse	refrigerator	train	truck
Predicted label										

Accuracy=0.824

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.