Homework 5 Report

Heng Zhang
0029109755
zhan2614

Introduction

In this homework, we use the nn package to build lenet5 as the model to do two tasks. The first task is the recognize number using MNIST dataset. Another task is to recognize object from 100 classes defined in CIFAR100 dataset. The MNIST dataset has 60000 pictures showing numbers from 0 to 9 and each class in the CIFAR100 has 600 images where 500 are training images and 100 are testing images so in total there are 60000 images in CIFAR100.

Setup

I use img2num.py and img2obj.py to implement these two tasks. Both scripts has a cifar100 class. Additionally img2obj.py has view and cam function which enables the functions of viewing the predicted image using opencv and capture real time image from camera and do the online prediction.

Evaluation

Task 1 number recognition

The results for this task are shown in Figures 1, 2, and 3.

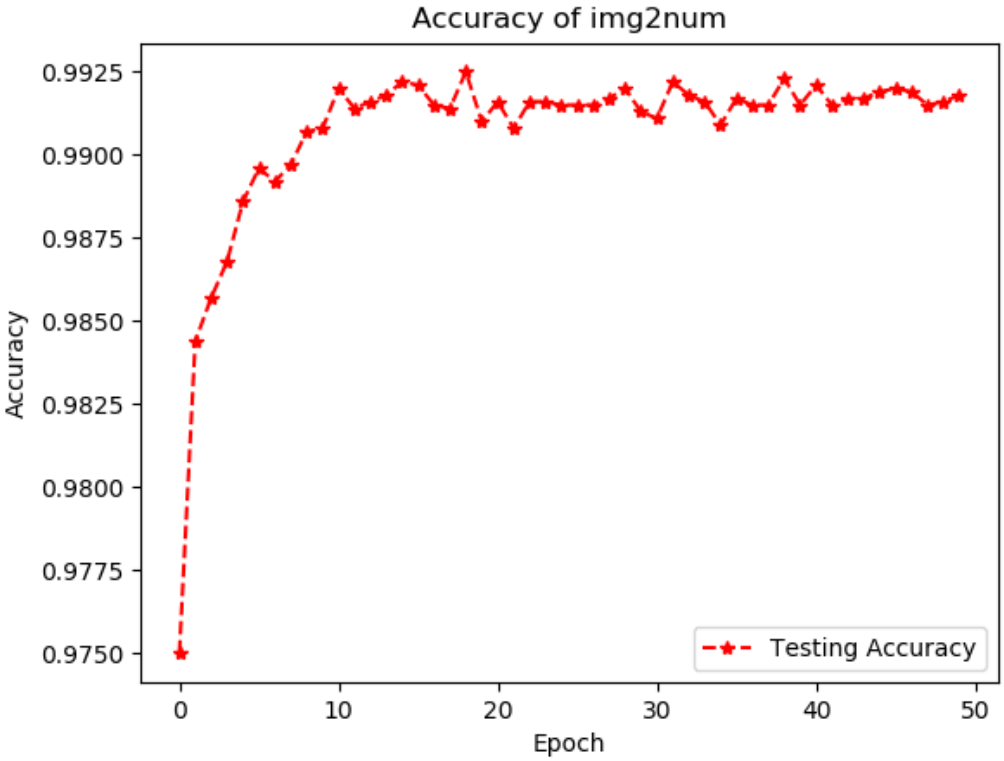

Figure 1 img2num accuracy

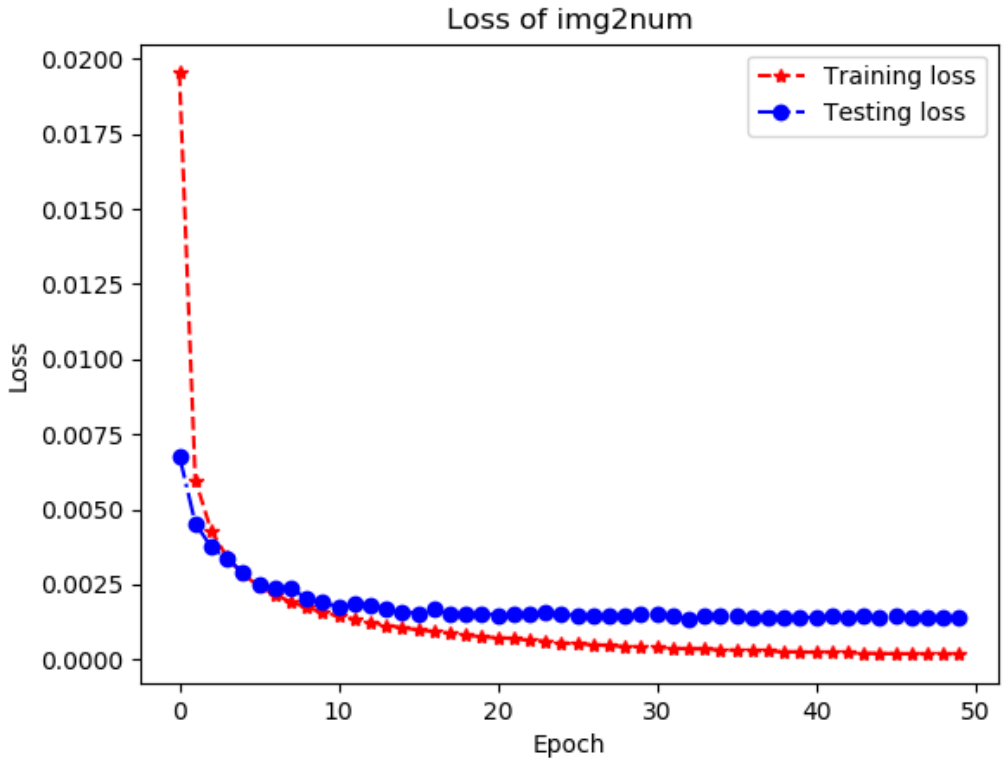

Figure 2 img2num loss

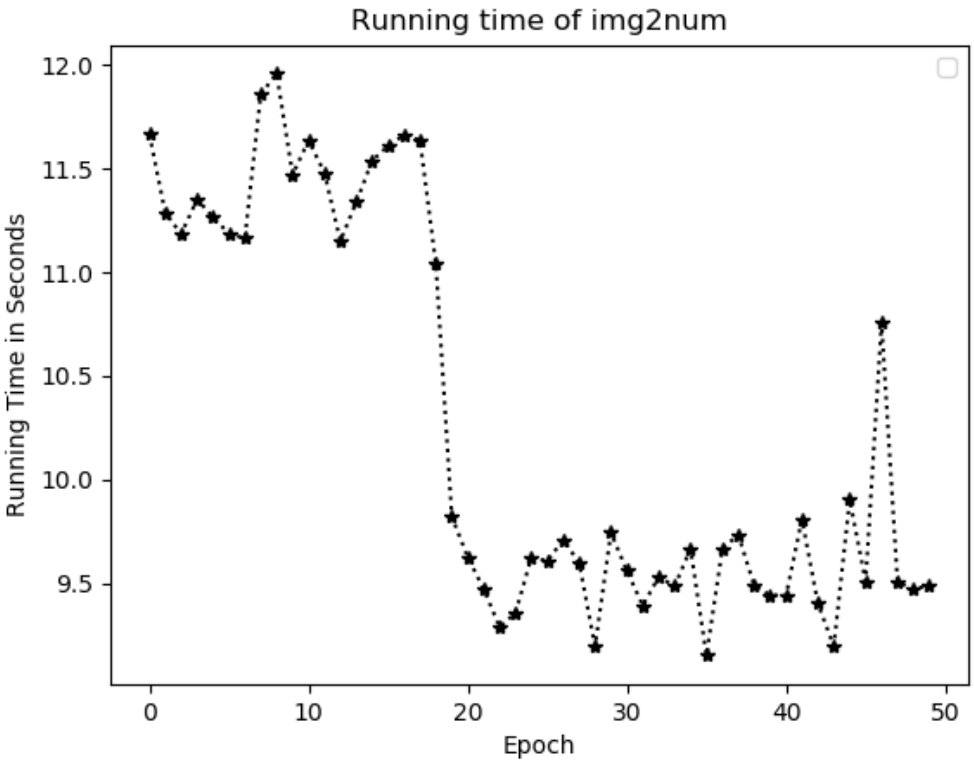

Figure 3 img2num running time

We can see that the accuracy for this task is fairly high to approximately 99%.

Task 2 object observation

The results for this task are represented in Figure 4, 5, and 6.

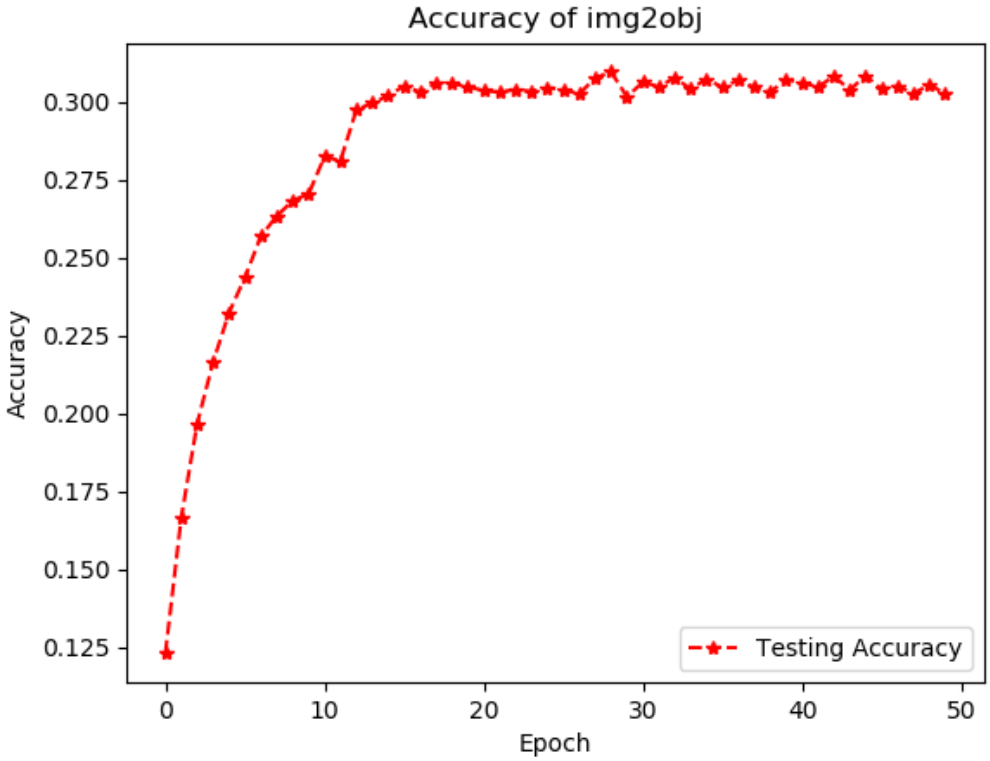

Figure 4 Accuracy of img2obj

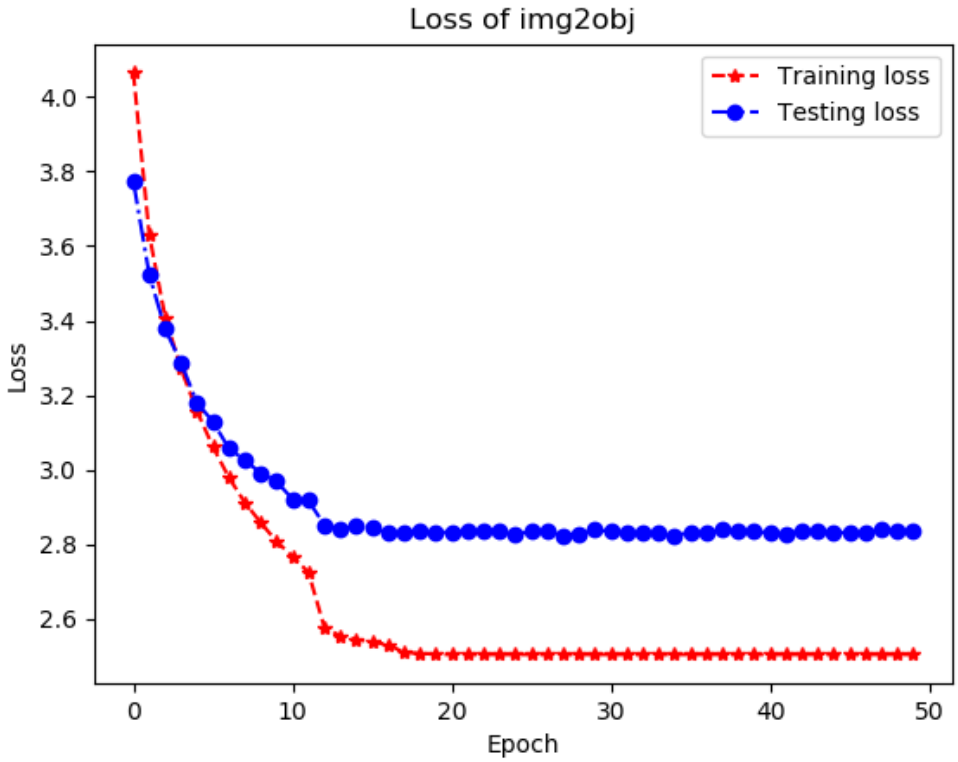

Figure 5 img2obj loss

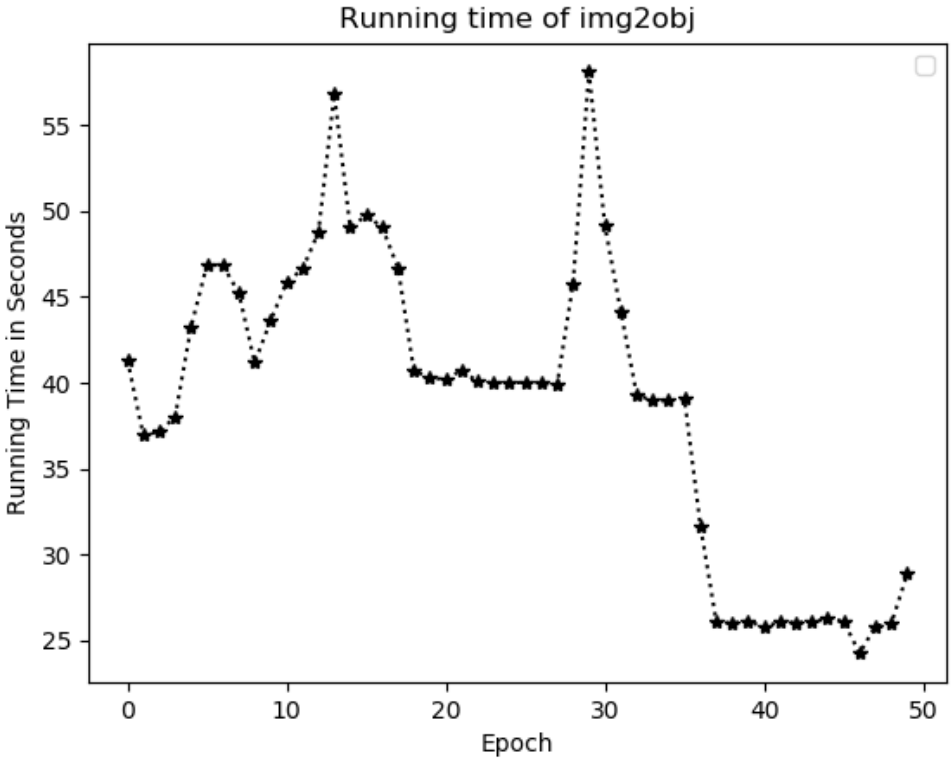

Figure 6 img2obj running time

We see that this task is harder to do and the accuracy approaches about 32%.

In both tasks, the losses are decreasing and the running times are somewhat unstable due to the fact that computers have different available resources on the fly.

Task 3 cam

This task is basically capture a picture from webcam and predicted what is in the picture. Its result is presented in Figure 7. It says the KFC logo and predicts it is a man.

An interesting fact is that sometimes it also predicts it as “cups”, it is also shown in Figure 8.

A finding is that the prediction is not always accurate. In fact most of a time, it is incorrect due to the low prediction accuracy.

Figure 7 opencv capture a man

Figure 8 opencv capture a cup

To regenerate the results in this report, just copy the test.py codes in the Appendix and run it in a machine with opencv and torch installed.

Appendix

img2num.py

```
import os
import shutil
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
from time import time
from torch.autograd import Variable


class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, padding=2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(5*5*16, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

class img2num:

    def __init__(self):
        self.train_batch_size = 60
        self.epoch = 50
        self.labels = 10
        self.rate = 1
        self.input_size = 28 * 28
        self.test_batch_size = 1000
        self.test_loader = torch.utils.data.DataLoader(
            datasets.MNIST('./mnist',
                train=False,
                download=True,
                transform=transforms.Compose([transforms.ToTensor()])),
```

```
                batch_size=self.test_batch_size, shuffle=True, num_workers=1
0)

        self.train_loader = torch.utils.data.DataLoader(
            datasets.MNIST('./mnist',
                train=True,
                download=True,
                transform=transforms.Compose([transforms.ToTensor()])),
                batch_size=self.train_batch_size, shuffle=True, num_workers=
10)

        # input image is 28 * 28 so convert to 1D matrix
        # output labels are 10 [0 - 9]
        torch.manual_seed(1)
        self.model = LeNet()
        self.optimizer = torch.optim.SGD(self.model.parameters(), lr=self.ra
te)
        self.loss_function = nn.MSELoss()

        self.check_point_file = 'img2num_checkpoint.tar'

        if os.path.isfile(self.check_point_file):
            cp = torch.load(self.check_point_file)
            self.start = cp['epoch']
            self.best_acc = cp['best_acc']

            print('checkpoint found at epoch', self.start)
            self.model.load_state_dict(cp['model'])
            self.optimizer.load_state_dict(cp['optimizer'])

            self.training_loss = cp['training_loss']
            self.testing_loss = cp['testing_loss']
            self.testing_acc = cp['testing_acc']
            self.time = cp['time']
        else:
            self.start = 0
            self.best_acc = 0

            self.training_loss = []
            self.testing_loss = []
            self.testing_acc = []
            self.time = []


```

```
        # img is 28*28 bytetensor
    def forward(self, img):
        _3d = torch.unsqueeze(img, 0)
        _4d = torch.unsqueeze(_3d, 0)
        self.model.eval()
        output = self.model(_4d)
        _, result = torch.max(output, 1)
        return result

    def train(self, plot=False):
        print('training')
        def save(state, better, f=self.check_point_file):
            torch.save(state, f)
            if better:
                shutil.copyfile(f, 'img2num_best.tar')

         def onehot_training(target, batch_size):
                output = torch.zeros(batch_size, self.labels)
                for i in range(batch_size):
                    output[i][int(target[i])] = 1.0
                return output

         def training():
             loss = 0
             self.model.train() # set to training mode
             for batch_id, (data, target) in enumerate(self.train_loader):
                 # data.view change the dimension of input to use forward fu
nction
                 forward_pass_output = self.model(data)
                 onehot_target = onehot_training(target, self.train_batch_si
ze)
                 #print(onehot_target.type())
                 cur_loss = self.loss_function(forward_pass_output, onehot_t
arget)
                 loss += cur_loss.data
                 self.optimizer.zero_grad()
                 cur_loss.backward()
                 self.optimizer.step()
             # loss / number of batches
             avg_loss = loss / (len(self.train_loader.dataset) / self.train_
batch_size)
             return avg_loss

         def testing():
```

```
126.             self.model.eval()
127.             loss = 0
128.             correct = 0
129.             for batch_id, (data, target) in enumerate(self.test_loader):
130.                 # data.view change the dimension of input to use forward fu
    nction
131.                 forward_pass_output = self.model(data)
132.                 onehot_target = onehot_training(target, self.test_batch_siz
    e)
133.                 cur_loss = self.loss_function(forward_pass_output, onehot_t
    arget)
134.                 loss += cur_loss.data
135.                 #print(forward_pass_output.size())
136.                 #print(onehot_target.size())
137.                 for i in range(self.test_batch_size):
138.                     val, position = torch.max(forward_pass_output.data[i],
    0)
139.                  #   print('prediction = {}, actual = {}'.format(int(positi
    on), target[i]))
140.                     if position == target[i]:
141.                         correct += 1
142.             # loss / number of batches
143.             avg_loss = loss / (len(self.test_loader.dataset) / self.test_ba
    tch_size)
144.             accuracy = correct / len(self.test_loader.dataset)
145.             return avg_loss, accuracy
146.         for i in range(self.start + 1, self.epoch + 1):
147.             s = time()
148.             train_loss = training()
149.             e = time()
150.             test_loss,accuracy = testing()
151.             print('Epoch {}, training_loss = {}, testing_loss = {}, accurac
    y = {}, time = {}'.format(i, train_loss, test_loss, accuracy, e - s))
152.             self.testing_acc.append(accuracy)
153.             self.training_loss.append(train_loss)
154.             self.testing_loss.append(test_loss)
155.             self.time.append(e-s)
156.             better = False
157.             if accuracy > self.best_acc:
158.                 better = True
159.             self.best_acc = max(self.best_acc, accuracy)
160.             print('Save checkpoint at', i)
161.             state = {
162.                     'epoch': i,
```

```
                    'best_acc': self.best_acc,
                    'model': self.model.state_dict(),
                    'optimizer': self.optimizer.state_dict(),
                    'training_loss': self.training_loss,
                    'testing_loss': self.testing_loss,
                    'testing_acc': self.testing_acc,
                    'time': self.time
                    }
            save(state,better)

        if plot == True:
            return self.time, self.training_loss, self.testing_loss, self.testing_acc

 '''''
        plt.plot(range(self.epoch), acc_list, 'r|--', label='Accuracy')
        plt.plot(range(self.epoch), train_loss_list, 'b*--', label='Training Loss')
        plt.plot(range(self.epoch), test_loss_list, 'yo--', label='Test Loss')
        plt.xlabel('Epoch')
        plt.legend()
        plt.title('Library Neural Network Evaluation')
        plt.savefig('nn_compare.png')
        plt.clf()
    '''

```

img2obj.py

```
import numpy as np
import os
import shutil
from pprint import pprint as pp
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
from time import time, sleep
from torch.autograd import Variable
import cv2

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
```

```
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(5*5*16, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 100)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

class img2obj:

    def __init__(self):
        self.train_batch_size = 125
        self.epoch = 50
        self.rate = 0.001
        self.input_size = 32 * 32 * 3 #RGB 3 channels of data
        self.test_batch_size = 1000
        normalize = transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5
, 0.5])
        self.test_loader = torch.utils.data.DataLoader(
            datasets.CIFAR100('./cifar',
                train=False,
                download=True,
                transform=transforms.Compose([transforms.RandomHorizontalFli
p(), transforms.ToTensor(), normalize])),
                batch_size=self.test_batch_size, shuffle=True, num_workers=1
0)

        self.train_loader = torch.utils.data.DataLoader(
            datasets.CIFAR100('./cifar',
                train=True,
                download=True,
                transform=transforms.Compose([transforms.ToTensor(), normali
ze])),
                batch_size=self.train_batch_size, shuffle=True, num_workers=
10)
```

```
55.
56.         self.classes = [
57.                 'beaver', 'dolphin', 'otter', 'seal', 'whale',
58.                 'aquarium fish', 'flatfish', 'ray', 'shark', 'trout',
59.                 'orchids', 'poppies', 'roses', 'sunflowers', 'tulips',
60.                 'bottles', 'bowls', 'cans', 'cups', 'plates',
61.                 'apples', 'mushrooms', 'oranges', 'pears', 'sweet peppers',

62.                 'clock', 'computer keyboard', 'lamp', 'telephone', 'televisi
    on',
63.                 'bed', 'chair', 'couch', 'table', 'wardrobe',
64.                 'bee', 'beetle', 'butterfly', 'categoryrpillar', 'cockroach'
    ,
65.                 'bear', 'leopard', 'lion', 'tiger', 'wolf',
66.                 'bridge', 'castle', 'house', 'road', 'skyscraper',
67.                 'cloud', 'forest', 'mountain', 'plain', 'sea',
68.                 'camel', 'cattle', 'chimpanzee', 'elephant', 'kangaroo',
69.                 'fox', 'porcupine', 'possum', 'raccoon', 'skunk',
70.                 'crab', 'lobster', 'snail', 'spider', 'worm',
71.                 'baby', 'boy', 'girl', 'man', 'woman',
72.                 'crocodile', 'dinosaur', 'lizard', 'snake', 'turtle',
73.                 'hamster', 'mouse', 'rabbit', 'shrew', 'squirrel',
74.                 'maple', 'oak', 'palm', 'pine', 'willow',
75.                 'bicycle', 'bus', 'motorcycle', 'pickup truck', 'train',
76.                 'lawn-mower', 'rocket', 'streetcar', 'tank', 'tractor'
77.                 ]
78.         torch.manual_seed(1)
79.         self.labels = len(self.classes)
80.         # input image is 3*32 * 32 so convert to 1D matrix
81.         self.model = LeNet()
82.         self.optimizer = torch.optim.Adam(self.model.parameters(), lr=self.r
    ate, weight_decay=0.0005)
83.         self.loss_function = nn.CrossEntropyLoss()
84.         self.check_point_file = 'img2obj_checkpoint.tar'
85.
86.         if os.path.isfile(self.check_point_file):
87.             cp = torch.load(self.check_point_file)
88.             self.start = cp['epoch']
89.             self.best_acc = cp['best_acc']
90.
91.             print('checkpoint found at epoch', self.start)
92.             self.model.load_state_dict(cp['model'])
93.             self.optimizer.load_state_dict(cp['optimizer'])
94.
```

```
95.             self.training_loss = cp['training_loss']
96.             self.testing_loss = cp['testing_loss']
97.             self.testing_acc = cp['testing_acc']
98.             self.time = cp['time']
99.         else:
100.             self.start = 0
101.             self.best_acc = 0
102.
103.             self.training_loss = []
104.             self.testing_loss = []
105.             self.testing_acc = []
106.             self.time = []
107.
108.         # img is 28*28 bytetensor
109.     def forward(self, img):
110.         _4d = torch.unsqueeze(img.type(torch.FloatTensor), 0)
111.         self.model.eval()
112.         output = self.model(_4d)
113.         _, result = torch.max(output, 1)
114.         return self.classes[result]
115.
116.     def train(self, plot=False):
117.         print('training')
118.
119.         def save(state, better, f=self.check_point_file):
120.             torch.save(state, f)
121.             if better:
122.                 shutil.copyfile(f, 'img2obj_best.tar')
123.
124.         def training():
125.             loss = 0
126.             self.model.train() # set to training mode
127.             for batch_id, (data, target) in enumerate(self.train_loader):
128.                 # data.view change the dimension of input to use forward fu
    nction
129.                 forward_pass_output = self.model(data)
130.
131.                 cur_loss = self.loss_function(forward_pass_output, target)

132.                 loss += cur_loss.data
133.                 self.optimizer.zero_grad()
134.                 cur_loss.backward()
135.                 self.optimizer.step()
136.             # loss / number of batches
```

```
137.             avg_loss = loss / (len(self.train_loader.dataset) / self.train_
    batch_size)
138.             return avg_loss
139.
140.         def testing():
141.             self.model.eval()
142.             loss = 0
143.             correct = 0
144.             for batch_id, (data, target) in enumerate(self.test_loader):
145.                 # data.view change the dimension of input to use forward fu
    nction
146.                 forward_pass_output = self.model(data)
147.                 cur_loss = self.loss_function(forward_pass_output, target)

148.                 loss += cur_loss.data
149.                 #print(forward_pass_output.size())
150.                 #print(onehot_target.size())
151.                 for i in range(self.test_batch_size):
152.                     val, position = torch.max(forward_pass_output.data[i],
    0)
153.                  #   print('prediction = {}, actual = {}'.format(int(positi
    on), target[i]))
154.                     if position == target[i]:
155.                         correct += 1
156.             # loss / number of batches
157.             avg_loss = loss / (len(self.test_loader.dataset) / self.test_ba
    tch_size)
158.             accuracy = correct / len(self.test_loader.dataset)
159.             return avg_loss, accuracy
160.         last_acc = 0
161.         for i in range(self.start + 1, self.epoch + 1):
162.             s = time()
163.             train_loss = training()
164.             e = time()
165.             test_loss, accuracy = testing()
166.             print('Epoch {}, training_loss = {}, testing_loss = {}, accurac
    y = {}, time = {}'.format(i, train_loss, test_loss, accuracy, e - s))
167.
168.             if last_acc > accuracy:
169.                 for g in self.optimizer.param_groups:
170.                     g['lr'] = g['lr']/10
171.                 print('learning rate changed to', g['lr'])
172.             last_acc = accuracy
173.             self.testing_acc.append(accuracy)
```

```
174.                 self.training_loss.append(train_loss)
175.                 self.testing_loss.append(test_loss)
176.                 self.time.append(e-s)
177.                 better = False
178.                 if accuracy > self.best_acc:
179.                     better = True
180.                 self.best_acc = max(self.best_acc, accuracy)
181.                 print('Save checkpoint at', i)
182.                 state = {
183.                         'epoch': i,
184.                         'best_acc': self.best_acc,
185.                         'model': self.model.state_dict(),
186.                         'optimizer': self.optimizer.state_dict(),
187.                         'training_loss': self.training_loss,
188.                         'testing_loss': self.testing_loss,
189.                         'testing_acc': self.testing_acc,
190.                         'time': self.time
191.                         }
192.                 save(state,better)
193.
194.         #label = self.classes[self.train_loader.dataset[20][1]]
195.         #print("actual label is", label)
196.         #self.view(self.train_loader.dataset[20][0])
197.
198.         if plot == True:
199.             return self.time, self.training_loss, self.testing_loss, self.t
    esting_acc
200.
201.     def view(self, img):
202.         category = self.forward(img)
203.         print('Prediction is', category)
204.         img = img.type(torch.FloatTensor) / 2 + 0.5
205.         img_numpy = np.transpose(img.numpy(), (1,2,0))
206.
207.         cv2.namedWindow(category, cv2.WINDOW_NORMAL)
208.         cv2.resizeWindow(category, 640, 480)
209.         cv2.imshow(category, img_numpy)
210.         cv2.waitKey(0)
211.         cv2.destroyAllWindows()
212.
213.     def cam(self, idx = 0):
214.
215.         def prepare(img_origin):
```

```
            img_scaled = cv2.resize(img_origin, (32, 32), interpolation=cv2.INTER_LINEAR)
            # Convert to Tensor and Normalize
            prepare = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])])

            return prepare(img_scaled)

        cam = cv2.VideoCapture(idx)
        cam.set(3, 1280)
        cam.set(4, 720)
        cv2.namedWindow("test")
        img_counter = 0
        print('Press e/E to exit, c/C to capture a picture\n')
        while True:
            ret, frame = cam.read()
            if not ret:
                break
            norm_img_tensor = prepare(frame)
            predicted_category = self.forward(norm_img_tensor)
            print(predicted_category)
            cv2.putText(frame, predicted_category, (10,500), cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 255, 255), 5, cv2.LINE_AA)
            cv2.imshow('Capturing', frame)

            k = cv2.waitKey(1) & 0xFF
            if k == ord('e'):
                # e pressed
                print("E hit, closing...")
                break
            elif k == ord('c'):
                # c pressed
                img_name = "opencv_frame_{}.png".format(img_counter)
                cv2.imwrite(img_name, frame)
                print("{} written!".format(img_name))
                img_counter += 1

        cam.release()

        cv2.destroyAllWindows()
```

test.py

```
import matplotlib.pyplot as plt
from img2num import img2num
```

```
3.  from img2obj import img2obj
4.  def graph(time, train_loss, test_loss, accuracy, name):
5.
6.      plt.plot(time, 'k*:')
7.      plt.ylabel('Running Time in Seconds')
8.      plt.legend()
9.      plt.xlabel('Epoch')
10.     plt.title("Running time of " + name)
11.     plt.savefig(name+'_time.png')
12.     plt.clf()
13.
14.     plt.plot(range(len(accuracy)), accuracy, 'r*--
    ', label='Testing Accuracy')
15.     plt.xlabel('Epoch')
16.     plt.ylabel('Accuracy')
17.     plt.title("Accuracy of " + name)
18.     plt.legend()
19.     plt.savefig(name + '_acc.png')
20.     plt.clf()
21.
22.     plt.plot(range(len(train_loss)), train_loss, 'r*--
    ', label='Training loss')
23.     plt.plot(range(len(test_loss)), test_loss, 'bo-.', label='Testing loss')

24.     plt.xlabel('Epoch')
25.     plt.legend()
26.     plt.title("Loss of "+name)
27.     plt.ylabel('Loss')
28.     plt.savefig(name+'_loss.png')
29.     plt.clf()
30. print('img2Num testing')
31. img = img2num()
32. time, train_loss, test_loss, accuracy = img.train(True)
33. graph(time, train_loss, test_loss, accuracy, 'img2num')
34. print('img2obj testing')
35. img = img2obj()
36. time, train_loss,test_loss, accuracy = img.train(True)
37. graph(time, train_loss, test_loss, accuracy, 'img2obj')
38. img.cam(0)
```