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.

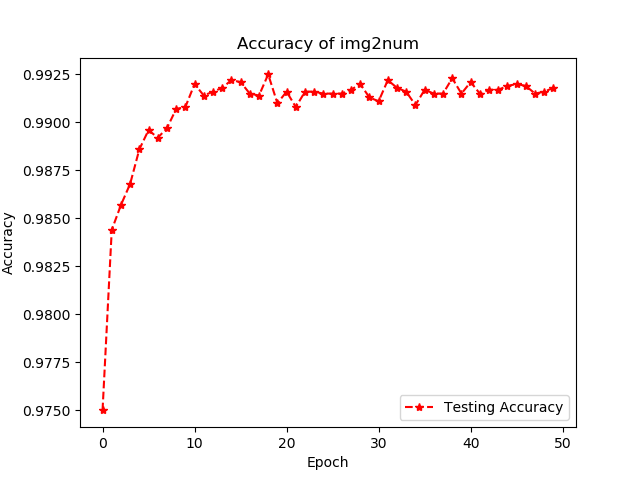


Figure img2num accuracy

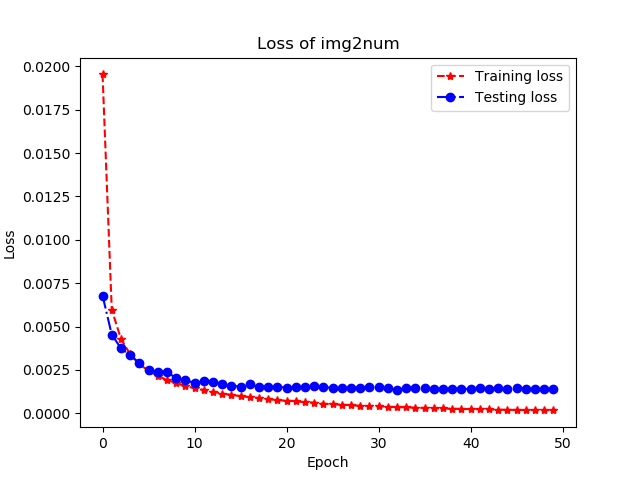


Figure img2num loss

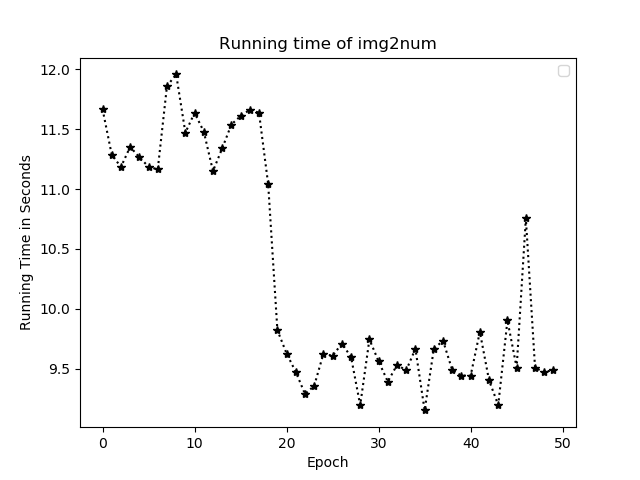


Figure img2num running time

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.

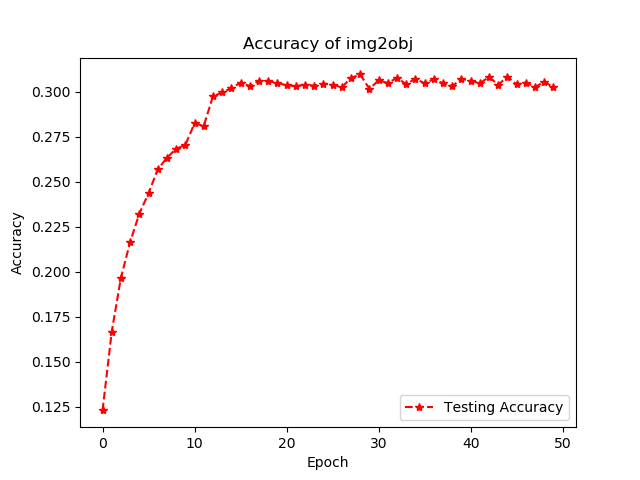


Figure Accuracy of img2obj

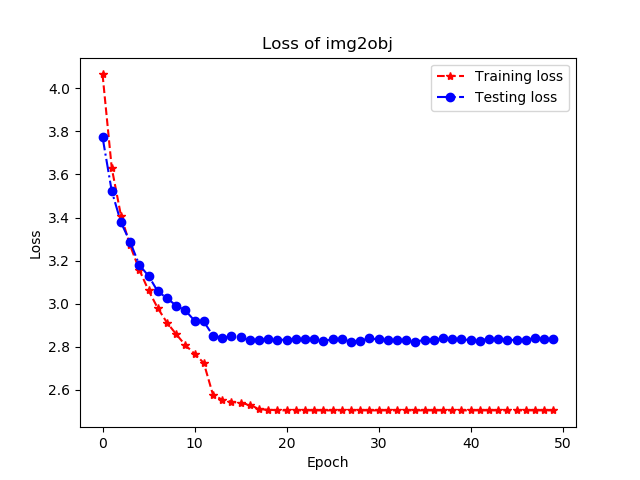


Figure img2obj loss

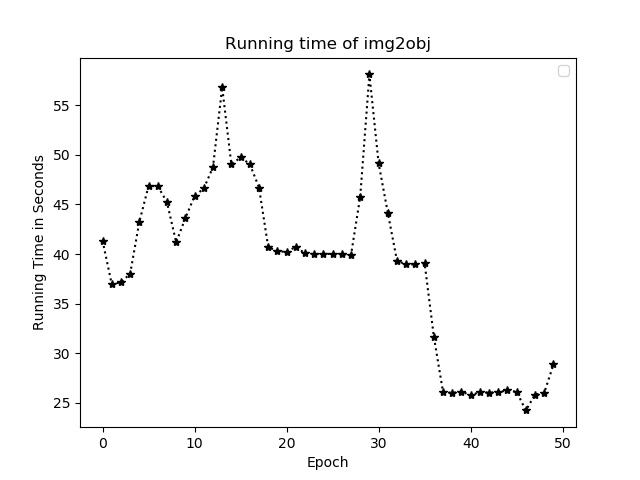


Figure img2obj running time

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 opencv capture a man



Figure 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

1. **import** os
2. **import** shutil
3. **import** torch
4. **import** torch.nn as nn
5. **import** torch.nn.functional as F
6. **from** torchvision **import** datasets, transforms
7. **from** time **import** time
8. **from** torch.autograd **import** Variable

11. **class** LeNet(nn.Module):
12. **def** \_\_init\_\_(self):
13. super(LeNet, self).\_\_init\_\_()
14. self.conv1 = nn.Conv2d(1, 6, 5, padding=2)
15. self.conv2 = nn.Conv2d(6, 16, 5)
16. self.fc1 = nn.Linear(5\*5\*16, 120)
17. self.fc2 = nn.Linear(120, 84)
18. self.fc3 = nn.Linear(84, 10)
20. **def** forward(self, x):
21. x = F.relu(self.conv1(x))
22. x = F.max\_pool2d(x, 2)
23. x = F.relu(self.conv2(x))
24. x = F.max\_pool2d(x, 2)
25. x = x.view(x.size(0), -1)
26. x = F.relu(self.fc1(x))
27. x = F.relu(self.fc2(x))
28. x = self.fc3(x)
29. **return** x
31. **class** img2num:
33. **def** \_\_init\_\_(self):
34. self.train\_batch\_size = 60
35. self.epoch = 50
36. self.labels = 10
37. self.rate = 1
38. self.input\_size = 28 \* 28
39. self.test\_batch\_size = 1000
40. self.test\_loader = torch.utils.data.DataLoader(
41. datasets.MNIST('./mnist',
42. train=False,
43. download=True,
44. transform=transforms.Compose([transforms.ToTensor()])),
45. batch\_size=self.test\_batch\_size, shuffle=True, num\_workers=10)
47. self.train\_loader = torch.utils.data.DataLoader(
48. datasets.MNIST('./mnist',
49. train=True,
50. download=True,
51. transform=transforms.Compose([transforms.ToTensor()])),
52. batch\_size=self.train\_batch\_size, shuffle=True, num\_workers=10)
54. # input image is 28 \* 28 so convert to 1D matrix
55. # output labels are 10 [0 - 9]
56. torch.manual\_seed(1)
57. self.model = LeNet()
58. self.optimizer = torch.optim.SGD(self.model.parameters(), lr=self.rate)
59. self.loss\_function = nn.MSELoss()
61. self.check\_point\_file = 'img2num\_checkpoint.tar'
63. **if** os.path.isfile(self.check\_point\_file):
64. cp = torch.load(self.check\_point\_file)
65. self.start = cp['epoch']
66. self.best\_acc = cp['best\_acc']
68. **print**('checkpoint found at epoch', self.start)
69. self.model.load\_state\_dict(cp['model'])
70. self.optimizer.load\_state\_dict(cp['optimizer'])
72. self.training\_loss = cp['training\_loss']
73. self.testing\_loss = cp['testing\_loss']
74. self.testing\_acc = cp['testing\_acc']
75. self.time = cp['time']
76. **else**:
77. self.start = 0
78. self.best\_acc = 0
80. self.training\_loss = []
81. self.testing\_loss = []
82. self.testing\_acc = []
83. self.time = []

86. # img is 28\*28 bytetensor
87. **def** forward(self, img):
88. \_3d = torch.unsqueeze(img, 0)
89. \_4d = torch.unsqueeze(\_3d, 0)
90. self.model.eval()
91. output = self.model(\_4d)
92. \_, result = torch.max(output, 1)
93. **return** result
95. **def** train(self, plot=False):
96. **print**('training')
97. **def** save(state, better, f=self.check\_point\_file):
98. torch.save(state, f)
99. **if** better:
100. shutil.copyfile(f, 'img2num\_best.tar')
102. **def** onehot\_training(target, batch\_size):
103. output = torch.zeros(batch\_size, self.labels)
104. **for** i **in** range(batch\_size):
105. output[i][int(target[i])] = 1.0
106. **return** output
108. **def** training():
109. loss = 0
110. self.model.train() # set to training mode
111. **for** batch\_id, (data, target) **in** enumerate(self.train\_loader):
112. # data.view change the dimension of input to use forward function
113. forward\_pass\_output = self.model(data)
114. onehot\_target = onehot\_training(target, self.train\_batch\_size)
115. #print(onehot\_target.type())
116. cur\_loss = self.loss\_function(forward\_pass\_output, onehot\_target)
117. loss += cur\_loss.data
118. self.optimizer.zero\_grad()
119. cur\_loss.backward()
120. self.optimizer.step()
121. # loss / number of batches
122. avg\_loss = loss / (len(self.train\_loader.dataset) / self.train\_batch\_size)
123. **return** avg\_loss
125. **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 function
131. forward\_pass\_output = self.model(data)
132. onehot\_target = onehot\_training(target, self.test\_batch\_size)
133. cur\_loss = self.loss\_function(forward\_pass\_output, onehot\_target)
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(position), 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\_batch\_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 = {}, accuracy = {}, 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,
163. 'best\_acc': self.best\_acc,
164. 'model': self.model.state\_dict(),
165. 'optimizer': self.optimizer.state\_dict(),
166. 'training\_loss': self.training\_loss,
167. 'testing\_loss': self.testing\_loss,
168. 'testing\_acc': self.testing\_acc,
169. 'time': self.time
170. }
171. save(state,better)
173. **if** plot == True:
174. **return** self.time, self.training\_loss, self.testing\_loss, self.testing\_acc
176. '''''
177. plt.plot(range(self.epoch), acc\_list, 'r|--', label='Accuracy')
178. plt.plot(range(self.epoch), train\_loss\_list, 'b\*--', label='Training Loss')
179. plt.plot(range(self.epoch), test\_loss\_list, 'yo--', label='Test Loss')
180. plt.xlabel('Epoch')
181. plt.legend()
182. plt.title('Library Neural Network Evaluation')
183. plt.savefig('nn\_compare.png')
184. plt.clf()
185. '''

img2obj.py

1. **import** numpy as np
2. **import** os
3. **import** shutil
4. **from** pprint **import** pprint as pp
5. **import** torch
6. **import** torch.nn as nn
7. **import** torch.nn.functional as F
8. **from** torchvision **import** datasets, transforms
9. **from** time **import** time, sleep
10. **from** torch.autograd **import** Variable
11. **import** cv2
13. **class** LeNet(nn.Module):
14. **def** \_\_init\_\_(self):
15. super(LeNet, self).\_\_init\_\_()
16. self.conv1 = nn.Conv2d(3, 6, 5)
17. self.conv2 = nn.Conv2d(6, 16, 5)
18. self.fc1 = nn.Linear(5\*5\*16, 120)
19. self.fc2 = nn.Linear(120, 84)
20. self.fc3 = nn.Linear(84, 100)
22. **def** forward(self, x):
23. x = F.relu(self.conv1(x))
24. x = F.max\_pool2d(x, 2)
25. x = F.relu(self.conv2(x))
26. x = F.max\_pool2d(x, 2)
27. x = x.view(x.size(0), -1)
28. x = F.relu(self.fc1(x))
29. x = F.relu(self.fc2(x))
30. x = self.fc3(x)
31. **return** x
33. **class** img2obj:
35. **def** \_\_init\_\_(self):
36. self.train\_batch\_size = 125
37. self.epoch = 50
38. self.rate = 0.001
39. self.input\_size = 32 \* 32 \* 3 #RGB 3 channels of data
40. self.test\_batch\_size = 1000
41. normalize = transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
42. self.test\_loader = torch.utils.data.DataLoader(
43. datasets.CIFAR100('./cifar',
44. train=False,
45. download=True,
46. transform=transforms.Compose([transforms.RandomHorizontalFlip(), transforms.ToTensor(), normalize])),
47. batch\_size=self.test\_batch\_size, shuffle=True, num\_workers=10)
49. self.train\_loader = torch.utils.data.DataLoader(
50. datasets.CIFAR100('./cifar',
51. train=True,
52. download=True,
53. transform=transforms.Compose([transforms.ToTensor(), normalize])),
54. batch\_size=self.train\_batch\_size, shuffle=True, num\_workers=10)
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', 'television',
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.rate, weight\_decay=0.0005)
83. self.loss\_function = nn.CrossEntropyLoss()
84. self.check\_point\_file = 'img2obj\_checkpoint.tar'
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']
91. **print**('checkpoint found at epoch', self.start)
92. self.model.load\_state\_dict(cp['model'])
93. self.optimizer.load\_state\_dict(cp['optimizer'])
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
103. self.training\_loss = []
104. self.testing\_loss = []
105. self.testing\_acc = []
106. self.time = []
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]
116. **def** train(self, plot=False):
117. **print**('training')
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')
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 function
129. forward\_pass\_output = self.model(data)
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
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 function
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(position), 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\_batch\_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 = {}, accuracy = {}, time = {}'.format(i, train\_loss, test\_loss, accuracy, e - s))
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)
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])
198. **if** plot == True:
199. **return** self.time, self.training\_loss, self.testing\_loss, self.testing\_acc
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))
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()
213. **def** cam(self, idx = 0):
215. **def** prepare(img\_origin):
216. img\_scaled = cv2.resize(img\_origin, (32, 32), interpolation=cv2.INTER\_LINEAR)
217. # Convert to Tensor and Normalize
218. prepare = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])])
220. **return** prepare(img\_scaled)
222. cam = cv2.VideoCapture(idx)
223. cam.set(3, 1280)
224. cam.set(4, 720)
225. cv2.namedWindow("test")
226. img\_counter = 0
227. **print**('Press e/E to exit, c/C to capture a picture\n')
228. **while** True:
229. ret, frame = cam.read()
230. **if** **not** ret:
231. **break**
232. norm\_img\_tensor = prepare(frame)
233. predicted\_category = self.forward(norm\_img\_tensor)
234. **print**(predicted\_category)
235. cv2.putText(frame, predicted\_category, (10,500), cv2.FONT\_HERSHEY\_SIMPLEX, 4, (255, 255, 255), 5, cv2.LINE\_AA)
236. cv2.imshow('Capturing', frame)
238. k = cv2.waitKey(1) & 0xFF
239. **if** k == ord('e'):
240. # e pressed
241. **print**("E hit, closing...")
242. **break**
243. **elif** k == ord('c'):
244. # c pressed
245. img\_name = "opencv\_frame\_{}.png".format(img\_counter)
246. cv2.imwrite(img\_name, frame)
247. **print**("{} written!".format(img\_name))
248. img\_counter += 1
250. cam.release()
252. cv2.destroyAllWindows()

test.py

1. **import** matplotlib.pyplot as plt
2. **from** img2num **import** img2num
3. **from** img2obj **import** img2obj
4. **def** graph(time, train\_loss, test\_loss, accuracy, name):
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()
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()
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)