Homework 6 Report

Heng Zhang
0029109755
zhan2614

Introduction

In this homework, we use the pretrained alexnet to train the last layer which has only 200 features instead of 1000. We use tiny imagenet dataset. It has 200 classes where each class has 500 training images, 50 validation images. In this homework, we copied the pretrained alexnet weights except the last layer.

Setup

I use train.py and test.py to implement these two tasks. The train.py build the model for 200 classes and the test.py is capable to test it by continuously capturing image from camera and giving real-time predicted result.

Evaluation

Task 1 training results

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

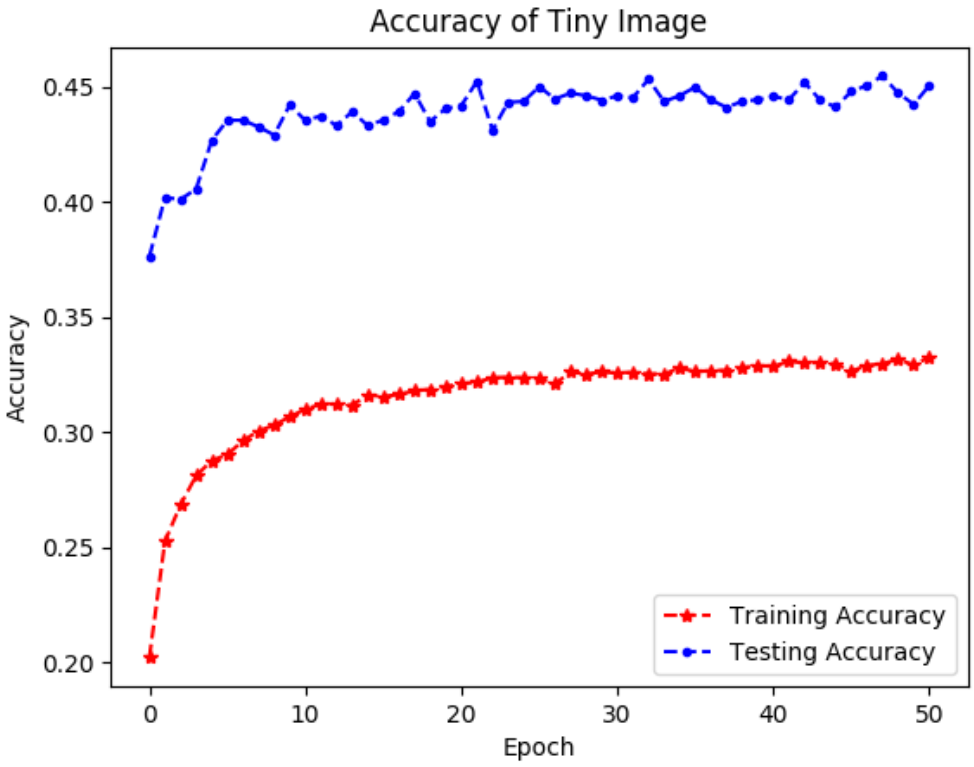

Figure 1 Accuracy comparison

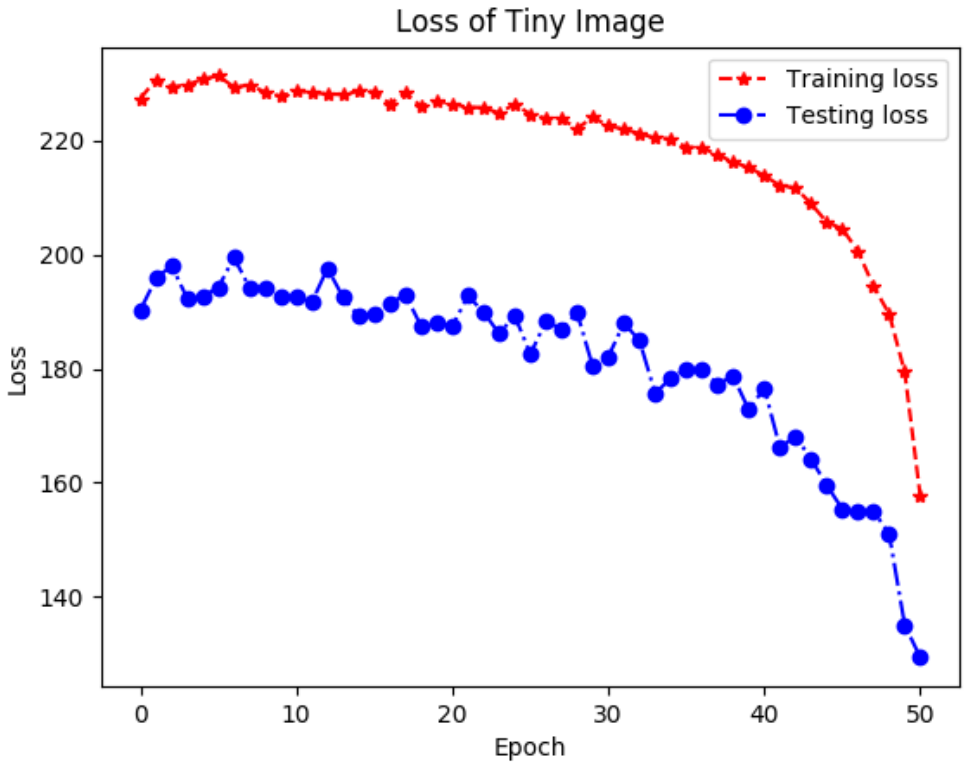

Figure 2 Loss comparison

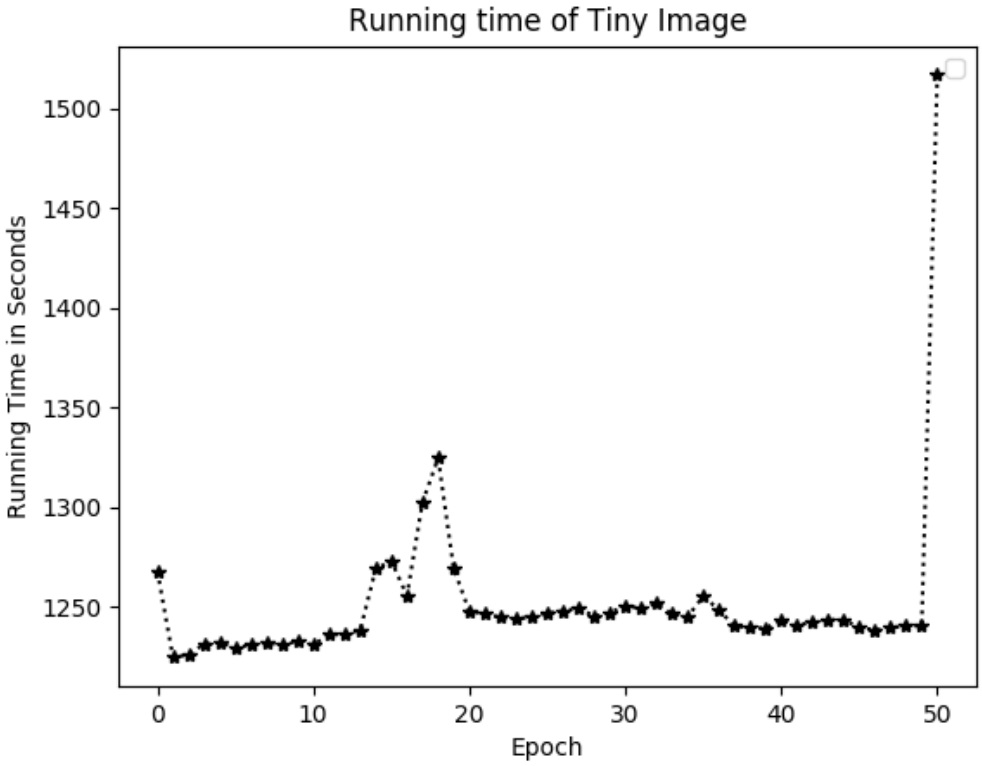

Figure 3 Running time

The accuracy saturates at about 45%. The running time is approximately 20 minutes for one epoch depending the system status.

Task 2 test if predict the object

I intentionally find a picture of jellyfish and put in front of the camera. It successfully predicts the jellyfish as in Figure 4

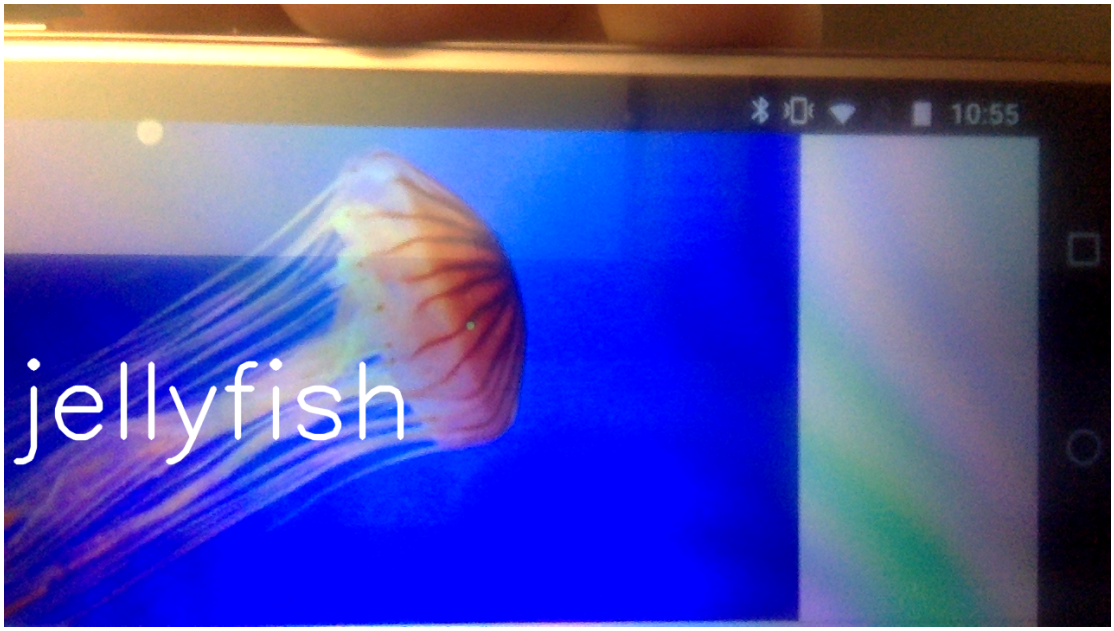

Figure 4 Predicted Jellyfish

I also tested with my keyboard and it correctly predicts it as in Figure 5.

Figure 5 Predicted keyboard

Appendix

train.py

```
1.  import argparse
2.  import matplotlib.pyplot as plt
3.  import os
4.  import shutil
5.  import torch
6.  import torch.nn as nn
7.  import torch.utils.model_zoo as model_zoo
8.  import torch.nn.functional as F
9.  from torchvision import datasets, models, transforms
10. from time import time
11.
12.
13. class AlexNet(nn.Module):
14.     def __init__(self, num_classes=200):
15.         super(AlexNet, self).__init__()
16.         self.features = nn.Sequential(
17.             nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
18.             nn.ReLU(inplace=True),
19.             nn.MaxPool2d(kernel_size=3, stride=2),
20.             nn.Conv2d(64, 192, kernel_size=5, padding=2),
21.             nn.ReLU(inplace=True),
22.             nn.MaxPool2d(kernel_size=3, stride=2),
23.             nn.Conv2d(192, 384, kernel_size=3, padding=1),
24.             nn.ReLU(inplace=True),
25.             nn.Conv2d(384, 256, kernel_size=3, padding=1),
26.             nn.ReLU(inplace=True),
27.             nn.Conv2d(256, 256, kernel_size=3, padding=1),
28.             nn.ReLU(inplace=True),
29.             nn.MaxPool2d(kernel_size=3, stride=2),
30.         )
31.         # the fully connected layer
32.         self.classifier = nn.Sequential(
33.             nn.Dropout(),
34.             nn.Linear(256 * 6 * 6, 4096),
35.             nn.ReLU(inplace=True),
36.             nn.Dropout(),
37.             nn.Linear(4096, 4096),
38.             nn.ReLU(inplace=True),
39.             nn.Linear(4096, num_classes),
```

```
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x

class Model:
    def __init__(self):
        parser = argparse.ArgumentParser()
        parser.add_argument('--data', type=str, help='Directory to thee tiny image set')
        parser.add_argument('--save', type=str, help='Directory to save trained model after completion of training')
        args = parser.parse_args()


        self.train_batch_size =  100
        self.epoch = 51
        self.rate = 0.1
        self.val_batch_size = 10
        def create_val_folder():
            path = os.path.join(args.data, 'val/images')  # path where validation data is present now
            filename = os.path.join(args.data, 'val/val_annotations.txt')  # file where image2class mapping is present
            fp = open(filename, "r")  # open file in read mode
            data = fp.readlines()  # read line by line

            # Create a dictionary with image names as key and corresponding classes as values
            val_img_dict = {}
            for line in data:
                words = line.split("\t")
                val_img_dict[words[0]] = words[1]
            fp.close()

            # Create folder if not present, and move image into proper folder
            for img, folder in val_img_dict.items():
                newpath = (os.path.join(path, folder))
                if not os.path.exists(newpath):  # check if folder exists
```

```
                    os.makedirs(newpath)

                if os.path.exists(os.path.join(path, img)):  # Check if image exists in default directory
                    os.rename(os.path.join(path, img), os.path.join(newpath, img))
        create_val_folder()

        # load data: from https://github.com/pytorch/examples/blob/master/imagenet/main.py

        traindir = os.path.join(args.data, 'train')
        if not os.path.exists(traindir):
            os.makedirs(traindir)
        valdir = os.path.join(args.data, 'val/images')

        if not os.path.exists(valdir):
            os.makedirs(valdir)

        normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        train_dataset = datasets.ImageFolder(
        traindir,
        transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
         normalize,
         ]))

         self.train_loader = torch.utils.data.DataLoader(train_dataset, batch_size = self.train_batch_size, shuffle=True, num_workers = 5)

         val_dataset = datasets.ImageFolder(
         valdir,
             transforms.Compose([
                    transforms.Resize(256),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                normalize,
         ]))

         self.val_loader = torch.utils.data.DataLoader(
```

```
115.             val_dataset, batch_size = self.val_batch_size, shuffle=True, nu
    m_workers = 5)
116.         def get_tiny_classes(class_list):
117.             fp = open(os.path.join(args.data, 'words.txt'))
118.             whole_class_dict = {}
119.             for line in fp.readlines():
120.                 fields = line.split("\t")
121.                 super_label = fields[1].split(',')
122.                 whole_class_dict[fields[0]] = super_label[0].rstrip()
123.             fp.close()
124.
125.             tiny_class = {}
126.             for lab in class_list:
127.                 for k,v in whole_class_dict.items():
128.                     if lab == k:
129.                         tiny_class[k] = v
130.                         continue
131.             return tiny_class
132.
133.         self.classes = train_dataset.classes
134.         self.tiny_classes = get_tiny_classes(self.classes)
135.
136.         pretrained_model = models.alexnet(pretrained=True)
137.         torch.manual_seed(1)
138.         self.model = AlexNet()
139.         # To copy parameters
140.         for i, j in zip(self.model.modules(), pretrained_model.    modules(
    )):  # iterate over both models
141.              if not list(i.children()):
142.                  if len(i.state_dict()) > 0:  # copy weights only     for t
    he convolution and linear layers
143.                      if i.weight.size() == j.weight.size():  # this helps t
    o prevent copying of weights of last layer
144.                          i.weight.data = j.weight.data
145.                          i.bias.data = j.bias.data
146.         for p in self.model.parameters():
147.             p.requires_grad = False
148.         for p in self.model.classifier[6].parameters():
149.             p.requires_grad = True
150.
151.
152.         self.optimizer = torch.optim.Adam(self.model.classifier[6].paramete
    rs(), lr=self.rate)
153.         self.loss_function = nn.CrossEntropyLoss()
```

```
154.
155.         self.check_point_file = os.path.join(args.save, 'alex_checkpoint.ta
     r')
156.         if not os.path.exists(os.path.dirname(self.check_point_file)):
157.             try:
158.                 os.makedirs(os.path.dirname(self.check_point_file))
159.             except OSError as exc: # Guard against race condition
160.                 if exc.errno != errno.EEXIST:
161.                     raise
162.         if os.path.isfile(self.check_point_file):
163.             cp = torch.load(self.check_point_file)
164.             self.start = cp['epoch']
165.             self.best_acc = cp['best_acc']
166.
167.             print('checkpoint found at epoch', self.start)
168.             self.model.load_state_dict(cp['model'])
169.             self.optimizer.load_state_dict(cp['optimizer'])
170.
171.             self.training_loss = cp['training_loss']
172.             self.training_acc = cp['training_acc']
173.             self.testing_loss = cp['testing_loss']
174.             self.testing_acc = cp['testing_acc']
175.             self.time = cp['time']
176.         else:
177.             self.start = 0
178.             self.best_acc = 0
179.
180.             self.training_loss = []
181.             self.training_acc = []
182.             self.testing_loss = []
183.             self.testing_acc = []
184.             self.time = []
185.     def train(self, plot=False):
186.         def save(state, better, f=self.check_point_file):
187.             torch.save(state, f)
188.             if better:
189.                 shutil.copyfile(f, os.path.join(args.save, 'alexnet_best.ta
     r'))
190.         def training():
191.             correct = 0
192.             loss = 0
193.             self.model.train() # set to training mode
194.             for batch_id, (data, target) in enumerate(self.train_loader):
```

```
195.                 # data.view change the dimension of input to use forward fu
    nction
196.                 forward_pass_output = self.model(data)
197.                 #print(onehot_target.type())
198.                 cur_loss = self.loss_function(forward_pass_output, target)

199.                 loss += cur_loss.data
200.                 self.optimizer.zero_grad()
201.                 cur_loss.backward()
202.                 self.optimizer.step()
203.
204.                 val, position = torch.max(forward_pass_output.data, 1)
205.                 for i in range(self.train_batch_size):
206.                     if position[i] == target.data[i]:
207.                         correct += 1
208.             # loss / number of batches
209.             avg_loss = loss / (len(self.train_loader.dataset) / self.train_
    batch_size)
210.             accuracy = correct / len(self.train_loader.dataset)
211.
212.             return avg_loss, accuracy
213.
214.         def testing():
215.             self.model.eval()
216.             loss = 0
217.             correct = 0
218.             for batch_id, (data, target) in enumerate(self.val_loader):
219.                 # data.view change the dimension of input to use forward fu
    nction
220.                 forward_pass_output = self.model(data)
221.                 cur_loss = self.loss_function(forward_pass_output, target)

222.                 loss += cur_loss.data
223.                 #print(forward_pass_output.size())
224.                 #print(onehot_target.size())
225.                 val, position = torch.max(forward_pass_output.data, 1)
226.                 for i in range(self.val_batch_size):
227.                  #   print('prediction = {}, actual = {}'.format(int(positi
    on), target[i]))
228.                     if position[i] == target[i]:
229.                         correct += 1
230.             # loss / number of batches
231.             avg_loss = loss / (len(self.val_loader.dataset) / self.val_batc
    h_size)
```

```
232.             accuracy = correct / len(self.val_loader.dataset)
233.             return avg_loss, accuracy
234.
235.         for i in range(self.start + 1, self.epoch + 1):
236.             print('Epoch {}'.format(i))
237.             s = time()
238.             print('Training\n')
239.             train_loss, train_accuracy = training()
240.             e = time()
241.             print('Training Done. Testing....\n')
242.             test_loss, test_accuracy = testing()
243.             self.testing_acc.append(test_accuracy)
244.             self.training_acc.append(train_accuracy)
245.             self.training_loss.append(train_loss)
246.             self.testing_loss.append(test_loss)
247.             self.time.append(e-s)
248.             better = False
249.             if test_accuracy > self.best_acc:
250.                 better = True
251.             self.best_acc = max(self.best_acc, test_accuracy)
252.             print('training_loss = {}, testing_loss = {}, training accuracy
    = {}, testing accuracy = {}, current best test accuracy = {}, time = {}, be
    tter = {}'.format(train_loss, test_loss, train_accuracy, test_accuracy, self
    .best_acc, e - s, better))
253.             print('Saved checkpoint at', i)
254.             state = {
255.                     'epoch': i,
256.                     'best_acc': self.best_acc,
257.                     'model': self.model.state_dict(),
258.                     'optimizer': self.optimizer.state_dict(),
259.                     'training_loss': self.training_loss,
260.                     'testing_loss': self.testing_loss,
261.                     'testing_acc': self.testing_acc,
262.                     'training_acc': self.training_acc,
263.                     'time': self.time,
264.                     'classes': self.classes,
265.                     'tiny_class': self.tiny_classes
266.
267.                     }
268.             save(state,better)
269.         if plot == True:
270.             return self.time, self.training_loss, self.testing_loss, self.t
    raining_acc, self.testing_acc
271.
```

```
272.
273. def graph(time, train_loss, test_loss, train_accuracy, test_accuracy, name)
    :
274.
275.     plt.plot(time, 'k*:')
276.     plt.ylabel('Running Time in Seconds')
277.     plt.legend()
278.     plt.xlabel('Epoch')
279.     plt.title("Running time of " + name)
280.     plt.savefig(name+'_time.png')
281.     plt.clf()
282.
283.     plt.plot(range(len(train_accuracy)), train_accuracy, 'r*--
    ', label='Training Accuracy')
284.     plt.plot(range(len(test_accuracy)), test_accuracy, 'b.--
    ', label='Testing Accuracy')
285.     plt.xlabel('Epoch')
286.     plt.ylabel('Accuracy')
287.     plt.title("Accuracy of " + name)
288.     plt.legend()
289.     plt.savefig(name + '_acc.png')
290.     plt.clf()
291.     plt.plot(range(len(train_loss)), train_loss, 'r*--
    ', label='Training loss')
292.     plt.plot(range(len(test_loss)), test_loss, 'bo-.', label='Testing loss'
    )
293.     plt.xlabel('Epoch')
294.     plt.legend()
295.     plt.title("Loss of "+name)
296.     plt.ylabel('Loss')
297.     plt.savefig(name+'_loss.png')
298.     plt.clf()
299.
300.
301. if __name__ == "__main__":
302.     m = Model()
303.     time, train_loss, test_loss, train_accuracy, test_accuracy = m.train(Tr
    ue)
304.     graph(time, train_loss, test_loss, train_accuracy, test_accuracy,  'Tin
    y Image')
```

test.py

```
1.  import argparse
```

```
import cv2
import os
import torch
from torchvision import transforms

parser = argparse.ArgumentParser()
parser.add_argument('--model', type=str, help='Directory to the saved model')

args = parser.parse_args()

from train import AlexNet
class TestClass:
    def __init__(self):
        self.model = AlexNet()
        # load from model
        self.check_point_file = os.path.join(args.model, 'alex_checkpoint.tar')
        if not os.path.exists(os.path.dirname(self.check_point_file)):
            try:
                os.makedirs(os.path.dirname(self.check_point_file))
            except OSError as exc: # Guard against race condition
                if exc.errno != errno.EEXIST:
                    raise
        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.tiny_class = cp['tiny_class']
            self.classes = cp['classes']
        else:
            print('No model found. Exit!!!')
            exit()

    def forward(self, img):
        _4d = torch.unsqueeze(img.type(torch.FloatTensor), 0)
        self.model.eval()
        output = self.model(_4d)
        _, result = torch.max(output, 1)

        print('predicted', result)
```

```
44.         label = self.tiny_class[self.classes[result.data[0]]]
45.         return label
46.
47.
48.     def cam(self, idx = 0):
49.
50.             def prepare(img_origin):
51.                 # Convert to Tensor and Normalize
52.                 transformer = transforms.Compose([
53.                     transforms.ToPILImage(),
54.                     transforms.Scale(256),
55.                     transforms.CenterCrop(224),
56.                     transforms.ToTensor(),
57.                     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.
    229, 0.224, 0.225])])
58.
59.                 return transformer(img_origin)
60.
61.             cam = cv2.VideoCapture(idx)
62.             cam.set(3, 1280)
63.             cam.set(4, 720)
64.             cv2.namedWindow("test")
65.             img_counter = 0
66.             print('Press e/E to exit, c/C to capture a picture\n')
67.             while True:
68.                 ret, frame = cam.read()
69.                 if not ret:
70.                     break
71.                 norm_img_tensor = prepare(frame)
72.                 predicted_category = self.forward(norm_img_tensor)
73.                 print(predicted_category)
74.                 cv2.putText(frame, predicted_category, (10,500), cv2.FONT_HE
    RSHEY_SIMPLEX, 4, (255, 255, 255), 5, cv2.LINE_AA)
75.                 cv2.imshow('Capturing', frame)
76.
77.                 k = cv2.waitKey(1) & 0xFF
78.                 if k == ord('e'):
79.                     # e pressed
80.                     print("E hit, closing...")
81.                     break
82.                 elif k == ord('c'):
83.                     # c pressed
84.                     img_name = "opencv_frame_{}.png".format(img_counter)
85.                     cv2.imwrite(img_name, frame)
```

```
                    print("{} written!".format(img_name))
                    img_counter += 1

            cam.release()
            cv2.destroyAllWindows()

if __name__ == "__main__":
    md = TestClass()
    md.cam()
```