

Assignment 1

Daiwei (David) Lu

lud

January 28, 2020

CS 8395

OBJECT LOCALIZATION

DAVID LU



INTRODUCTION

PREPROCESSING AND DATA AUGMENTATION

Determining the location of a phone in images

Return the object coordinates with respect to the image



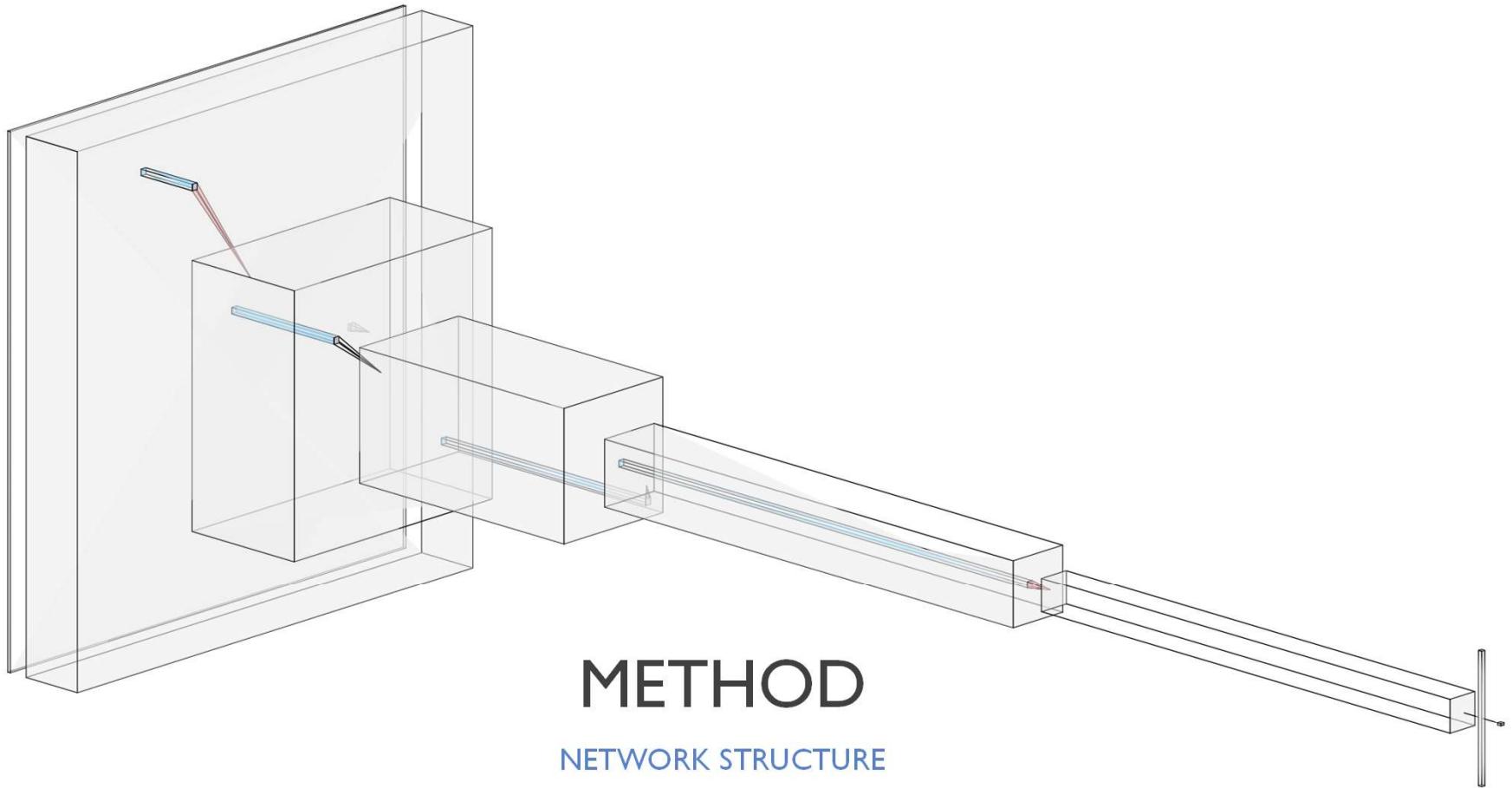
RATIONALE

Coordinate Localization is regressed even in complex detection networks

Retraining feature detectors on Scarce datasets is foolish

Use weights in VGG trained on Imagenet to jumpstart custom model

Update convolutional parameters to specialize network to task





METHOD

INPUT OUTPUT

256x256 input squares
Convolutional and Pooling layers similar to VGG
Fully Connected Layer for regression of two coordinates



AD HOC AUGMENTATION DURING TRAINING

PREPROCESSING AND DATA AUGMENTATION

Random Vertical Flipping

Random Horizontal Flipping

Random Brightness Adjustment

Random Contrast Adjustment

Random Saturation Adjustment

Random Hue Adjustment

Normalization



HYPER PARAMETERS AND TRAINING

100 epochs

16 batch

SGD with 0.001 learning rate 0.9 momentum

0.5x / 20 epoch weight decay

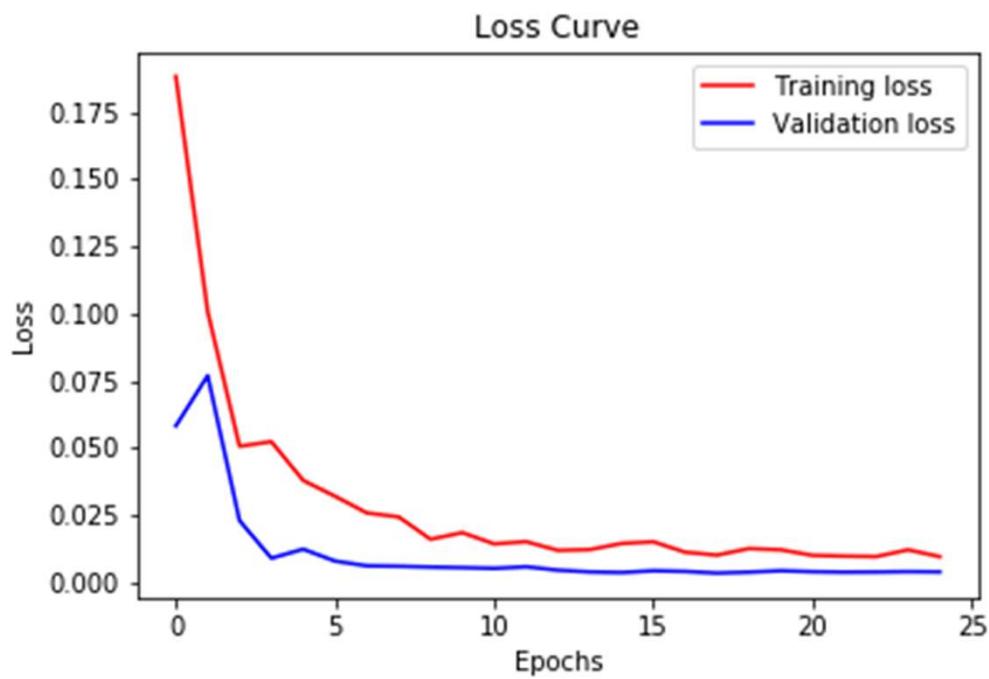
MSE Loss

PARAMETERS AND LAYERS

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 256, 256]	1,792
ReLU-2	[-1, 64, 256, 256]	0
MaxPool2d-3	[-1, 64, 128, 128]	0
Conv2d-4	[-1, 128, 128, 128]	73,856
ReLU-5	[-1, 128, 128, 128]	0
MaxPool2d-6	[-1, 128, 64, 64]	0
Conv2d-7	[-1, 256, 64, 64]	295,168
ReLU-8	[-1, 256, 64, 64]	0
Conv2d-9	[-1, 256, 64, 64]	590,080
ReLU-10	[-1, 256, 64, 64]	0
MaxPool2d-11	[-1, 256, 32, 32]	0
Conv2d-12	[-1, 512, 32, 32]	1,180,160
ReLU-13	[-1, 512, 32, 32]	0
Conv2d-14	[-1, 512, 32, 32]	2,359,808
ReLU-15	[-1, 512, 32, 32]	0
MaxPool2d-16	[-1, 512, 16, 16]	0
Conv2d-17	[-1, 512, 16, 16]	2,359,808
ReLU-18	[-1, 512, 16, 16]	0
Conv2d-19	[-1, 512, 16, 16]	2,359,808
ReLU-20	[-1, 512, 16, 16]	0
MaxPool2d-21	[-1, 512, 8, 8]	0
AdaptiveAvgPool2d-22	[-1, 512, 7, 7]	0
Linear-23	[-1, 64]	1,605,696
Sigmoid-24	[-1, 64]	0
Dropout-25	[-1, 64]	0
Linear-26	[-1, 2]	130

RESULTS

LOSS CURVE

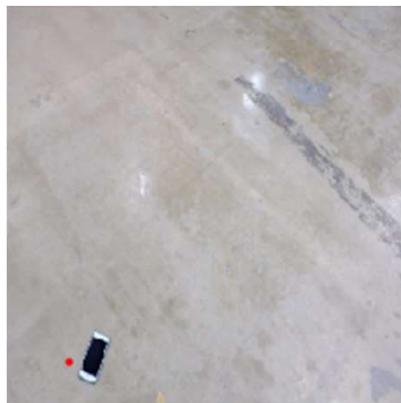


RESULTS

PREDICTED VISUALIZATIONS



I20.jpg



I21.jpg



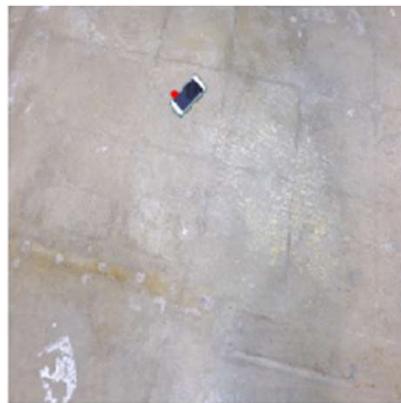
I22.jpg



I23.jpg



I24.jpg



I26.jpg



Name	X	Y
I20.jpg	0.7976	0.7551
I21.jpg	0.1528	0.8774
I22.jpg	0.1555	0.8610
I23.jpg	0.5137	0.5422
I24.jpg	0.7865	0.2362
I26.jpg	0.4059	0.2117

RESULTS

COORDINATES



CONCLUSION

DIFFICULTIES

Generalization to test images
Overfitting with larger structures
Custom dataloader
Sensitive to scale



CONCLUSION

FUTURE STEPS

Multi-scale network for different scale phones

Skip connections to avoid overfitting

Usage of UNET style network to remove background before dense layers

```

1 # Author: Daiwei (David) Lu
2 # Train custom model
3
4 from torch.utils.data import Dataset
5 import torch.nn as nn
6 import torch.optim as optim
7 from torch.optim import lr_scheduler
8 import time
9 import copy
10 from load import *
11 from model import Net
12 from utils import visualize_model
13
14 import warnings
15
16 warnings.filterwarnings("ignore")
17 plt.ion()
18
19
20 def train_model(model, criterion, optimizer, scheduler, dataloaders, device, num_epochs=25):
21     dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
22     class_names = ['x', 'y']
23     since = time.time()
24     best_model_wts = copy.deepcopy(model.state_dict())
25     best_loss = 9001.
26     best_acc = 1.
27     train_loss = []
28     val_loss = []
29     for epoch in range(num_epochs):
30         print('Epoch {}/{}'.format(epoch, num_epochs - 1))
31         print('-' * 10)
32         # Each epoch has a training and validation phase
33         for phase in ['train', 'val']:
34             if phase == 'train':
35                 model.train() # Set model to training mode
36             else:
37                 model.eval() # Set model to evaluate mode
38             running_loss = 0.0
39             running_corrects = 0.0
40             # Iterate over data.
41             for loader in dataloaders[phase]:
42                 inputs, labels = loader['image'], loader['coordinates']
43                 inputs = inputs.float().cuda().to(device)
44                 labels = labels.float().cuda().to(device)
45                 # zero the parameter gradients
46                 optimizer.zero_grad()
47                 # forward
48                 outputs = model(inputs)
49                 loss = criterion(outputs, labels)
50
51                 # backward + optimize only if in training phase
52                 if phase == 'train':
53                     loss.backward()
54                     optimizer.step()
55
56                     running_loss += loss.item() * inputs.size(0)
57                     running_corrects += calc_acc(labels, outputs)

```

```

58     if phase == 'train':
59         scheduler.step()
60
61     epoch_loss = running_loss / dataset_sizes[phase]
62     epoch_acc = running_corrects / dataset_sizes[phase]
63
64     print('{} Loss: {:.6f}'.format(
65         phase, epoch_loss))
66     print('{} Acc: {:.6f}'.format(
67         phase, epoch_acc
68     ))
69     if phase == 'train':
70         train_loss.append(epoch_loss)
71     else:
72         val_loss.append(epoch_loss)
73
74 # deep copy the model
75 if phase == 'val' and epoch_acc < best_acc:
76     best_loss = epoch_loss
77     best_acc = epoch_acc
78     best_model_wts = copy.deepcopy(model.state_dict())
79
80     print()
81
82 time_elapsed = time.time() - since
83 print('Training complete in {:.0f}m {:.0f}s'.format(
84     time_elapsed // 60, time_elapsed % 60))
85 print('Best val Loss: {:.6f}'.format(best_loss))
86 print('Best val Acc: {:.6f}'.format(best_acc))
87
88 # load best model weights
89 model.load_state_dict(best_model_wts)
90 return model, train_loss, val_loss
91
92
93 image_datasets = {'train': PhoneDataset('labels/train.txt',
94                                         '',
95                                         mode='train',
96                                         transform=transforms.Compose([
97                                             Rescale(256),
98                                             RandomVerticalFlip(0.5),
99                                             RandomHorizontalFlip(0.5),
100                                            RandomColorJitter(0.9),
101                                            ToTensor(),
102                                            Normalize()
103                                         ])),
104    'val': PhoneDataset('labels/val.txt',
105                           '',
106                           mode='validation',
107                           transform=transforms.Compose([
108                               Rescale(256),
109                               # RandomVerticalFlip(0.1),
110                               # RandomHorizontalFlip(0.1),
111                               # RandomColorJitter(0.1),
112                               ToTensor(),
113                               Normalize()
114                           ]))}

```

```
115
116
117 def main():
118     dataloaders = {'train': torch.utils.data.DataLoader(image_datasets['train'], batch_size=16,
119                                         shuffle=True),
120                    'val': torch.utils.data.DataLoader(image_datasets['val'], batch_size=4,
121                                         shuffle=True)}
122
123     device = torch.device("cuda")
124
125     model = Net()
126     model = model.to(device)
127
128     criterion = nn.MSELoss()
129
130     optimizer_conv = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
131
132     # Decay LR by a factor of 0.5 every 20 epochs
133     exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=20, gamma=0.5)
134
135     print(model)
136     epochs = 100
137     model, train_loss, val_loss = train_model(model, criterion, optimizer_conv,
138                                              exp_lr_scheduler, dataloaders, device, num_epochs=epochs)
139
140     visualize_model(model, dataloaders, device)
141     plt.ioff()
142     plt.show()
143
144     plt.plot(np.arange(epochs), train_loss, c='red', label='Training loss')
145     plt.plot(np.arange(epochs), val_loss, c='blue', label='Validation loss')
146     plt.legend()
147     plt.title('Loss Curve')
148     plt.xlabel('Epochs')
149     plt.ylabel('Loss')
150     plt.savefig('./Loss Curve')
151
152     torch.save(model.state_dict(), './trainedmodel.pth')
153
154
155 if __name__ == '__main__':
156     main()
```

```

1  # Author: Daiwei (David) Lu
2  # A custom network in VGG style
3
4  import torch
5  from torch import nn
6
7
8  class ConvPool(nn.Module):
9      def __init__(self, channelin, channelout):
10         super(Net, self).__init__()
11         self.convpool = nn.Sequential(
12             nn.Conv2d(channelin, channelout, 3, padding=1),
13             nn.ReLU(inplace=True),
14             nn.MaxPool2d(2, 2),
15         )
16
17     def forward(self, x):
18         return self.convpool(x)
19
20
21  class Conv(nn.Module):
22      def __init__(self, channelin, channelout):
23          super(Net, self).__init__()
24          self.conv = nn.Sequential(
25              nn.Conv2d(channelin, channelout, 3, padding=1),
26              nn.ReLU(inplace=True),
27          )
28
29      def forward(self, x):
30          return self.conv(x)
31
32
33  class Net(nn.Module):
34      def __init__(self):
35          super(Net, self).__init__()
36          self.features = nn.Sequential(
37              nn.Conv2d(3, 64, 3, padding=1),
38              nn.ReLU(inplace=True),
39              nn.MaxPool2d(2, 2),
40              nn.Conv2d(64, 128, 3, padding=1),
41              nn.ReLU(inplace=True),
42              nn.MaxPool2d(2, 2),
43              nn.Conv2d(128, 256, 3, padding=1),
44              nn.ReLU(inplace=True),
45              nn.Conv2d(256, 256, 3, padding=1),
46              nn.ReLU(inplace=True),
47              nn.MaxPool2d(2, 2),
48              nn.Conv2d(256, 512, 3, padding=1),
49              nn.ReLU(inplace=True),
50              nn.Conv2d(512, 512, 3, padding=1),
51              nn.ReLU(inplace=True),
52              nn.MaxPool2d(2, 2),
53              nn.Conv2d(512, 512, 3, padding=1),
54              nn.ReLU(inplace=True),
55              nn.Conv2d(512, 512, 3, padding=1),
56              nn.ReLU(inplace=True),
57              nn.MaxPool2d(2, 2))

```

```
58     self.avgpool = nn.AdaptiveAvgPool2d(output_size=(7, 7))
59     self.classifier = nn.Sequential(nn.Linear(512 * 7 * 7, 64),
60                                     nn.Sigmoid(),
61                                     nn.Dropout(0.25),
62                                     nn.Linear(64, 2))
63
64 def forward(self, x):
65     # ModuleList can act as an iterable, or be indexed using ints
66     x1 = self.features(x)
67     x2 = self.avgpool(x1)
68     x3 = torch.flatten(x2, 1)
69     x4 = self.classifier(x3)
70     return x4
71
```

```

1 # Author: Daiwei (David) Lu
2 # A fully custom dataloader for the cellphone dataset
3
4 import os
5 import torch
6 import pandas as pd
7 from PIL import Image
8 from skimage import io, transform
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from torch.utils.data import Dataset
12 from torchvision import transforms
13 import torchvision.transforms.functional as TF
14 import random
15
16 import warnings
17
18 warnings.filterwarnings("ignore")
19 plt.ion()
20
21
22 class PhoneDataset(Dataset):
23
24     def __init__(self, file, root, mode='/train', transform=None, test=False):
25         self.data = pd.read_csv(file, sep=" ", header=None)
26         if test:
27             self.data.columns = ["names"]
28         else:
29             self.data.columns = ["names", "x", "y"]
30         self.root = root
31         self.transform = transform
32         self.mode = mode
33
34     def __len__(self):
35         return len(self.data)
36
37     def __getitem__(self, idx):
38         if torch.is_tensor(idx):
39             idx = idx.tolist()
40         img_name = os.path.join(self.root + self.mode, self.data.names[idx])
41         image = io.imread(img_name)
42         coords = np.array([self.data.x[idx], self.data.y[idx]])
43         sample = {'image': image, 'coordinates': coords}
44         if self.transform:
45             sample = self.transform(sample)
46         return sample
47
48
49 class Rescale(object):
50
51     def __init__(self, output_size):
52         assert isinstance(output_size, (int, tuple))
53         self.output_size = output_size
54
55     def __call__(self, sample):
56         image, coordinates = sample['image'], sample['coordinates']
57         new_h, new_w = self.output_size, self.output_size

```

```

58     img = transform.resize(image, (new_h, new_w))
59     return {'image': img, 'coordinates': coordinates}
60
61
62 class Normalize(object):
63     def __init__(self, inplace=False):
64         self.mean = (0.5692824, 0.55365936, 0.5400631)
65         self.std = (0.1325967, 0.1339596, 0.14305606)
66         self.inplace = inplace
67
68     def __call__(self, sample):
69         image, coordinates = sample['image'], sample['coordinates']
70         return {'image': TF.normalize(image, self.mean, self.std, self.inplace), 'coordinates':
71                 coordinates,
72                 'original': image}
73
74
75 class ToTensor(object):
76
77     def __call__(self, sample):
78         dtype = torch.FloatTensor if torch.cuda.is_available() else torch.FloatTensor
79         image, coordinates = sample['image'], sample['coordinates']
80         image = image.transpose((2, 0, 1))
81         return {'image': torch.from_numpy(image).type(dtype),
82                 'coordinates': torch.from_numpy(coordinates).type(dtype)}
83
84
85 class RandomHorizontalFlip(object):
86     def __init__(self, p=0.5):
87         self.p = p
88
89     def __call__(self, sample):
90         image, coordinates = sample['image'], sample['coordinates']
91         if random.random() < self.p:
92             image *= 255
93             image = Image.fromarray(np.uint8(image))
94             image = TF.hflip(image)
95             image = np.array(image)
96             image = np.float32(image) / 255.
97             coordinates[0] = 1.0 - coordinates[0]
98         return {'image': image, 'coordinates': coordinates}
99
100
101 class RandomVerticalFlip(object):
102     def __init__(self, p=0.5):
103         self.p = p
104
105     def __call__(self, sample):
106         image, coordinates = sample['image'], sample['coordinates']
107         if random.random() < self.p:
108             image *= 255.
109             image = Image.fromarray(np.uint8(image))
110             image = TF.vflip(image)
111             image = np.array(image)
112             image = np.float32(image) / 255.
113             coordinates[1] = 1.0 - coordinates[1]
114
115         return {'image': image, 'coordinates': coordinates}

```

```

File - C:\Users\david\Documents\Spring 2020\8395\HW1\load.py
114
115
116 class RandomColorJitter(object):
117     def __init__(self, p=0.2, brightness=(0.5, 1.755), contrast=(0.5, 1.5), saturation=(0.5, 1.5), hue
118         =(-0.2, 0.2)):
119         self.p = p
120         self.brightness = brightness
121         self.contrast = contrast
122         self.saturation = saturation
123         self.hue = hue
124
125     def __call__(self, sample):
126         image, coordinates = sample['image'], sample['coordinates']
127         if random.random() < self.p:
128             image *= 255.
129             image = Image.fromarray(np.uint8(image))
130             modifications = []
131             brightness_factor = random.uniform(self.brightness[0], self.brightness[1])
132             modifications.append(transforms.Lambda(lambda img: TF.adjust_brightness(image,
133             brightness_factor)))
134             contrast_factor = random.uniform(self.contrast[0], self.contrast[1])
135             modifications.append(transforms.Lambda(lambda img: TF.adjust_contrast(image,
136             contrast_factor)))
137             saturation_factor = random.uniform(self.saturation[0], self.saturation[1])
138             modifications.append(transforms.Lambda(lambda img: TF.adjust_saturation(image,
139             saturation_factor)))
140             hue_factor = random.uniform(self.hue[0], self.hue[1])
141             modifications.append(transforms.Lambda(lambda img: TF.adjust_hue(image, hue_factor
142             )))
143             random.shuffle(modifications)
144             modification = transforms.Compose(modifications)
145             image = modification(image)
146
147             image = np.array(image)
148             image = np.float32(image) / 255.
149             return {'image': image, 'coordinates': coordinates}
150
151
152     def calc_acc(input, output):
153         sum = 0.0
154         for i in range(input.shape[0]):
155             sum += torch.sqrt((input[i][0] - output[i][0]) ** 2 + (input[i][1] - output[i][1]) ** 2)
156         return sum / input.shape[0]
157

```

```
1 # Author: Daiwei (David) Lu
2 # Find the PhoneDataset mean and std for normalization
3
4 from load import *
5 from torch.utils.data import DataLoader
6
7 dataset = PhoneDataset('data/labels/train.txt',
8     'data',
9     mode='/train',
10    transform=transforms.Compose([
11        Rescale(256),
12        ToTensor(),
13        Normalize()
14    ]))
15 loader = DataLoader(
16    dataset,
17    batch_size=10,
18    num_workers=1,
19    shuffle=False
20 )
21
22 pixel_mean = np.zeros(3)
23 pixel_std = np.zeros(3)
24 k = 1
25 for load in loader:
26    imgs = load['image']
27    imgs = np.array(imgs)
28    print(imgs.shape)
29    for i in range(imgs.shape[0]):
30        image = imgs[i]
31        pixels = image.reshape((-1, image.shape[2]))
32
33        for pixel in pixels:
34            diff = pixel - pixel_mean
35            pixel_mean += diff / k
36            pixel_std += diff * (pixel - pixel_mean)
37            k += 1
38
39 pixel_std = np.sqrt(pixel_std / (k - 2))
40 print(pixel_mean)
41 print(pixel_std)
42
```

```

1 # Author: Daiwei (David) Lu
2 # Make predictions on test images
3
4 import torch
5 from skimage import io, transform
6 from torch.utils.data import Dataset
7 from torchvision import transforms
8 from model import Net
9 from utils import TestFile, Rescale, ToTensor, show_dot
10 import argparse
11
12 MODEL_PATH = './model.pth'
13
14
15 def test_model(path, viz=False):
16     image = io.imread(path)
17     image = transform.resize(image, (256, 256))
18
19     device = torch.device("cuda")
20
21     model = Net()
22     model = model.to(device)
23
24     model.load_state_dict(torch.load(MODEL_PATH))
25     model.eval()
26
27     with torch.no_grad():
28         set = TestFile(path,
29                         transform=transforms.Compose([
30                             Rescale(256),
31                             ToTensor()
32                         ]))
33
34         loader = torch.utils.data.DataLoader(set)
35         for i, input in enumerate(loader):
36             image = input['image'].float().cuda().to(device)
37             coordinates = model(image).data
38             coordinates = coordinates.cpu().numpy()
39             print('{:.4f} {:.4f}'.format(coordinates[0][0], coordinates[0][1]))
40             if viz:
41                 show_dot(input['original'], coordinates)
42
43
44 def main():
45     # Training settings
46     parser = argparse.ArgumentParser(description='Test Prediction')
47     parser.add_argument('path', metavar='P', type=str,
48                         help='path of file for prediction')
49     parser.add_argument('viz', metavar='V', type=bool, default=False,
50                         help='visualize prediction')
51     args = parser.parse_args()
52     test_model(args.path, args.viz)
53
54
55 if __name__ == '__main__':
56     main()
57

```

```

1 # Author: Daiwei (David) Lu
2 # Useful Utils
3
4 import torch
5 from skimage import io, transform
6 from torch.utils.data import Dataset
7 import matplotlib.pyplot as plt
8 from torchvision import utils
9 import torchvision.transforms.functional as TF
10
11
12 class TestFile(Dataset):
13
14     def __init__(self, file, transform=None):
15         self.file = file
16         self.transform = transform
17
18     def __len__(self):
19         return 1
20
21     def __getitem__(self, idx):
22         sample = io.imread(self.file)
23         if self.transform:
24             sample = self.transform(sample)
25         return sample
26
27
28 class Rescale(object):
29
30     def __init__(self, output_size):
31         assert isinstance(output_size, (int, tuple))
32         self.output_size = output_size
33
34     def __call__(self, sample):
35         new_h, new_w = self.output_size, self.output_size
36         img = transform.resize(sample, (new_h, new_w))
37         return img
38
39
40 class Normalize(object):
41     def __init__(self, inplace=False):
42         self.mean = (0.5692824, 0.55365936, 0.5400631)
43         self.std = (0.1325967, 0.1339596, 0.14305606)
44         self.inplace = inplace
45
46     def __call__(self, sample):
47         image = sample['image']
48         return {'image': TF.normalize(image, self.mean, self.std, self.inplace),
49                 'original': image}
50
51
52 class ToTensor(object):
53
54     def __call__(self, sample):
55         dtype = torch.FloatTensor if torch.cuda.is_available() else torch.FloatTensor
56         image = sample.transpose((2, 0, 1))
57         return torch.from_numpy(image).type(dtype)

```

```

58
59
60 def show_dot(image, coordinates):
61     plt.imshow(image)
62     plt.scatter(image.shape[1] * coordinates[0][0], image.shape[0] * coordinates[0][1], marker='.', c
63 = 'r')
64     plt.pause(0.001)
65
66 def batch_show(sample_batched):
67     """Show image for a batch of samples."""
68     images_batch, coordinates_batch = \
69         sample_batched['original'], sample_batched['coordinates']
70     batch_size = len(images_batch)
71     im_size = images_batch.size(2)
72     grid_border_size = 2
73     grid = utils.make_grid(images_batch)
74     plt.imshow(grid.numpy().transpose((1, 2, 0)))
75
76     for i in range(batch_size):
77         plt.scatter(coordinates_batch[i, 0].cpu().numpy() * 256 + i * im_size + (i + 1) *
78                     grid_border_size,
79                     coordinates_batch[i, 1].cpu().numpy() * 256 + grid_border_size,
80                     marker='.', c='r')
81
82
83 def visualize_model(model, dataloaders, device):
84     was_training = model.training
85     model.eval()
86     fig = plt.figure()
87
88     with torch.no_grad():
89         for i, batch in enumerate(dataloaders['val']):
90             inputs, labels = batch['image'], batch['coordinates']
91             inputs = inputs.float().cuda().to(device)
92             print('Label:', batch['coordinates'].data)
93             batch['coordinates'].data = model(inputs).data
94             plt.figure()
95             batch_show(batch)
96             plt.axis('off')
97             plt.ioff()
98             plt.show()
99
100    model.train(mode=was_training)
101

```