ECE695DL: Homework 5

Qilei Zhang

Due Date: Monday, Mar 7, 2022

1 Introduction

The goal of this homework is to understand the usage of skip block network. With skip connections, a network includes shortcut pathways so that the loss calculated at the output can be "felt" more strongly in the earlier layers of the network during backpropagation. A CNN network shall be trained to provide an image's classification and corresponding location. It follows that the convolutional network must use two loss functions, one for classification and the other for regression. The COCO dataset is considered as the training set and validation set.

2 Methodology

2.1 How to Run

The data downloader is embedded in both `hw05_training.py` and `hw05_validation.py`. The downloaded images are not been compressed but keeping the orignal size. Only the image that takes 1/3 width and height will be selected.

2.1.1 To train data

The basic information for the program shall be entered in the `FakeArg` class. `self.root_path` stores the downloaded images for training and the `coco_json_path` is the coco annotations file location. The default value is read from the coco 2017 training set. Here is the example scripts:

```
1 class FakeArg:
2     def __init__(self):
3         self.root_path = './ECE695/coco/Train/'
4         self.coco_json_path =
          ↪  './ECE695/annotations2017/instances_train2017.json'
5         self.class_list = ["cat", "dog", "elephant", "giraffe", "horse",
          ↪  "car", "airplane", "pizza"]
6         self.images_per_class = 500
```

As shown in the example, the default class list has eight categories.

2.1.2 To validate model

Change the `self.root_path` and `coco_json_path` for `FakeArg` class in order to download the validation images. Additionally, image per class may be changed for the validation purpose. Because the 2017 validation set has smaller number of the images, the default value is read from the coco 2013 validation set. The following is an example:

```
1  args.root_path = root_path = './ECE695/coco/Val/'
2  args.coco_json_path = './ECE695/annotations2014/instances_val2014.json'
3  args.images_per_class = 200
```

2.2 To show prediction result

To show the predicted COCO images and annotations, please use the following function in line 1 to provide the output. Here, `img_info` contains the downloaded image information. `cat` refers to the true label. `order` means the image index in a specific category. The maximum `order` value is the `img_per_class`. `cat_list` provides the complete category list/

```
1  plot_detection(img_info, cat, order, cat_list, img_per_class)
```

2.3 Note

In case any unexpected situation, please refers to the colab link if any interruption or warning happens when running the program.

2.4 Main Part

(1). Both skip connections and batch normalization are used in the network design. If not use batch normalization (BN). The probability of the output cross entropy may easily converge, so the balance of BN may be useful.

(2). The illustration for the classification network is shown in Figure 1 below.

(3). The regression network contains six convolutional layers and corresponding ReLU activation layers. At the end of the network, there are three linear layer to provide an `(x_min, y_min, x_max, y_max)` output.

(4). The dataset loader will compress the image and give a uniform image size (128 *times* 128) in a tensor format to the network. When need validation, the boundary box size will re-stretch to draw in the original image.

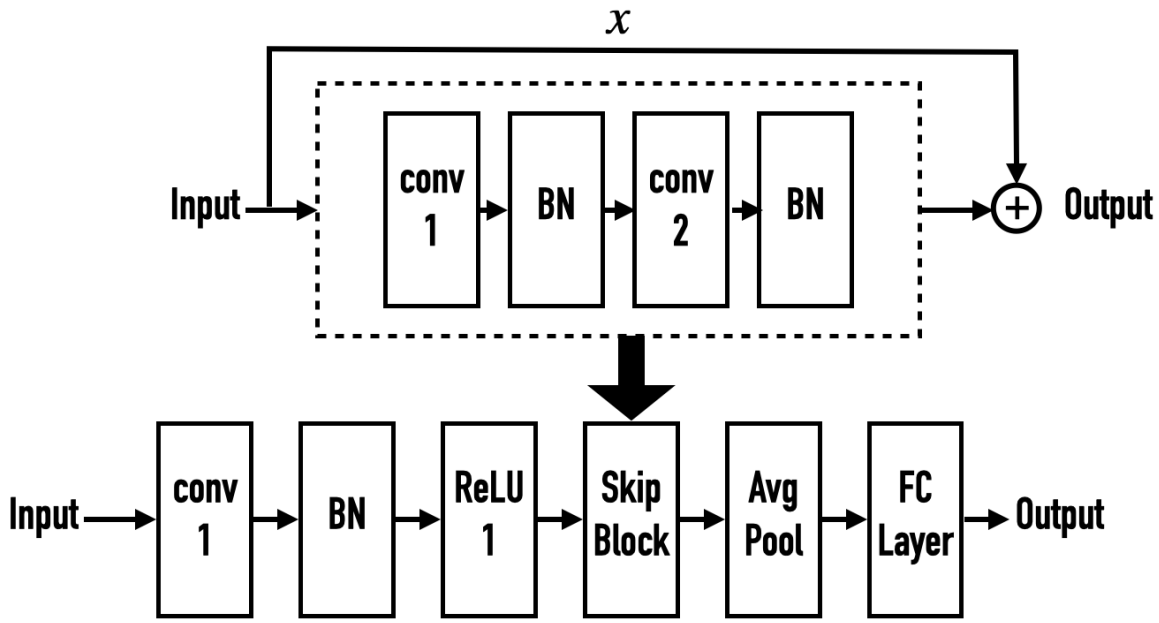

Figure 1: Network Design for Classification including Skip Connection.

3 Implementation and Results

3.1 Main Program Code

3.1.1 hw05_training.py

```
1  import os
2  import glob
3  import torch
4  import random
5  import requests
6
7  import numpy as np
8  import torch.nn as nn
9  import torch.nn.functional as functional
10 import torchvision.transforms as tvt
11 import matplotlib.pyplot as plt
12
13 from PIL import Image
14 from pycocotools.coco import COCO
15 from torch.utils.data import DataLoader, Dataset
16
```

```
17
18  class FakeArg:
19      """
20      Put the fundamental information
21      """
22      def __init__(self):
23          self.root_path = '/content/drive/MyDrive/ECE695/hw4/coco/Train/'
24          self.coco_json_path =
            ↪  '/content/drive/MyDrive/ECE695/hw4/annotations2017/instances_train2017.json'
25          self.class_list = ["cat", "dog", "elephant", "giraffe", "horse",
            ↪  "car", "airplane", "pizza"]
26          # self.class_list = ["cat", "dog", "elephant"]
27          self.images_per_class = 500
28
29
30  class Downloader:
31      def __init__(self, info):
32          self.root = info.root_path
33          self.json = info.coco_json_path
34          self.class_list = info.class_list
35          self.images_per_class = info.images_per_class
36          self.cat_folder = dict.fromkeys(self.class_list)
37          self.coco = COCO(self.json)
38          self.img_info = []
39
40      def make_folder(self):
41          for cat in self.class_list:
42              folder_root = self.root + cat
43              self.cat_folder[cat] = folder_root
44              if not os.path.exists(folder_root):
45                  os.makedirs(folder_root)
46
47      def get_image(self):
48          cat_index = 0
49          for cat in self.class_list:
50              cat_id = self.coco.getCatIds(cat)
51              img_id = self.coco.getImgIds(catIds=cat_id)
52              imgs = self.coco.loadImgs(img_id)
53              save_number = 0
54              img_index = 0
55              # self.img_info.append([])
56              while save_number < self.images_per_class:
57                  im = imgs[img_index]
58                  annIds = self.coco.getAnnIds(imgIds=im['id'],
                    ↪  catIds=cat_id, iscrowd=False)
59                  anns = self.coco.loadAnns(annIds)
```

```
60                  # if len(anns) == 1:
61                  for ann_index in range(len(anns)):
62                      ann = anns[ann_index]
63                      img_width = im['width']
64                      img_height = im['height']
65                      ann_width = ann['bbox'][2]
66                      ann_height = ann['bbox'][3]
67                      ann_size_check = self.check_ann(img_width, img_height,
                        ↪  ann_width, ann_height)
68                      if ann_size_check:
69                          img_path = self.root + cat + "/" + im['file_name']
70                          save_check = self.save_image(img_path,
                            ↪  im['coco_url'])
71                          if save_check:
72                              self.rgb_image(img_path)
73                              new_img_dict = {'cat': cat, 'order':
                                ↪  save_number, 'bbox': ann['bbox'],
74                                              'image_index': img_index,
                                                ↪  'ann_index': ann_index,
75                                              'img_path': img_path, 'label':
                                                ↪  cat_index,
76                                              'img_size': [img_width,
                                                ↪  img_height]}
77                              #
                                ↪  self.img_info[cat_index].append(new_img_dict)
78                              self.img_info.append(new_img_dict)
79                              save_number += 1
80                              break
81                  img_index += 1
82              cat_index += 1
83              print(cat, 'is downloaded')
84          return self.img_info
85
86      @staticmethod
87      def check_ann(iw, ih, aw, ah):
88          width_ratio = aw/iw
89          height_ratio = ih/ah
90          if (width_ratio > (1/3)) & (height_ratio > (1/3)):
91              return True
92          else:
93              return False
94
95      @staticmethod
96      def rgb_image(img_save_path):
97          image = Image.open(img_save_path)
98          if image.mode != "RGB":
```

```
            image = image.convert(mode="RGB")
        image.save(img_save_path)

    @staticmethod
    def save_image(img_save_path, img_url):
        if not os.path.exists(img_save_path):
            try:
                img_response = requests.get(img_url, timeout=10)
            except requests.exceptions as e:
                return False
            with open(img_save_path, 'wb') as img_f:
                img_f.write(img_response.content)
            return True
        else:
            return True


class QZDatasetClass(Dataset):
    def __init__(self, img_att, trans=None):
        self.transform = trans
        self.img_info = img_att
        # print('is created')

    def __len__(self):
        # print('total is', len(self.img_info))
        return len(self.img_info)

    def __getitem__(self, idx):
        # print('idx', idx)
        img_path = self.img_info[idx]['img_path']
        label = self.img_info[idx]['label']
        bbox = self.img_info[idx]['bbox']
        new_bbox = [bbox[0], bbox[1], bbox[0]+bbox[2], bbox[1]+bbox[3]]
        image = Image.open(img_path)
        image, new_bbox = self.make_square(image, new_bbox)
        bb_tensor = torch.tensor(new_bbox, dtype=torch.float)
        im_ts = self.transform(image)
        return im_ts, label, bb_tensor

    @staticmethod
    def make_square(im, bbox_size, min_size=128):
        w, h = im.size
        w_ratio = w / min_size
        h_ratio = h / min_size
        bbox_size[0] = bbox_size[0] / w_ratio
        bbox_size[1] = bbox_size[1] / h_ratio
```

```
145         bbox_size[2] = bbox_size[2] / w_ratio
146         bbox_size[3] = bbox_size[3] / h_ratio
147         image_resized = im.resize((min_size, min_size), Image.BOX)
148         return image_resized, bbox_size
149
150
151 class Block(nn.Module):
152     def __init__(self, layer, in_ch, out_ch, stride=1, downsample=None):
153         super(Block, self).__init__()
154         self.layer = layer
155         self.conv1 = self.conv3(in_ch, out_ch, stride)
156         self.bn1 = nn.BatchNorm2d(out_ch)
157         self.relu = nn.ReLU(inplace=True)
158         self.conv2 = self.conv3(out_ch, out_ch)
159         self.bn2 = nn.BatchNorm2d(out_ch)
160         self.downsample = downsample
161
162         # if downsample:
163         #     self.downsampler = nn.Conv2d(in_ch, out_ch, 1, stride=2)
164
165     def forward(self, x):
166         identity = x
167         out = self.conv1(x)
168         out = self.bn1(out)
169         out = self.relu(out)
170         out = self.conv2(out)
171         out = self.bn2(out)
172         if self.downsample:
173             identity = self.downsample(x)
174         out += identity
175         out = self.relu(out)
176         return out
177
178     @staticmethod
179     def conv3(in_channels, out_channels, stride=1):
180         return nn.Conv2d(in_channels, out_channels, kernel_size=3,
            ↪ stride=stride, padding=1, bias=False)
181
182
183 class QZhangNet(nn.Module):
184     def __init__(self, class_num, layers=None):
185         super(QZhangNet, self).__init__()
186         if layers is None:
187             # layers = [3, 4, 6, 3]
188             layers = [2, 2, 2, 2]
189         self.in_channels = 16
```

```
190         self.conv = Block.conv3(3, 16)
191         self.bn = nn.BatchNorm2d(16)
192         self.bn2 = nn.BatchNorm2d(32)
193         self.bn3 = nn.BatchNorm2d(64)
194         self.relu = nn.ReLU(inplace=True)
195         self.layer1 = self.make_layer(1, 16, layers[0])
196         # self.in_channels = self.in_channels * 2
197         self.layer2 = self.make_layer(2, 32, layers[1], 2)
198         self.layer3 = self.make_layer(3, 64, layers[2], 2)
199         self.output = nn.Conv2d(64, 9 * class_num, kernel_size=3,
            ↪ padding=1)
200         self.output_act = nn.Sigmoid()
201         self.avg_pool = nn.AvgPool2d(7)
202         # self.fc = nn.Linear(144 * class_num, class_num)
203         self.fc = nn.Linear(648 * class_num, class_num)
204 
205         # regression
206         self.conv_seqn = nn.Sequential(
207             nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,
                ↪ padding=1),
208             nn.ReLU(inplace=True),
209             nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3,
                ↪ padding=1),
210             nn.ReLU(inplace=True),
211             nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3,
                ↪ padding=1),
212             nn.ReLU(inplace=True),
213             nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3,
                ↪ padding=1),
214             nn.ReLU(inplace=True),
215             nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3,
                ↪ padding=1),
216             nn.ReLU(inplace=True),
217             nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3,
                ↪ padding=1),
218             nn.ReLU(inplace=True) # add
219         )
220         self.fc_seqn = nn.Sequential(
221             nn.Linear(262144, 1024),
222             nn.ReLU(inplace=True),
223             nn.Linear(1024, 512),
224             nn.ReLU(inplace=True),
225             nn.Linear(512, 4)
226         )
227 
228     def make_layer(self, layer, out_channels, blocks, stride=1):
```

```
229         downsample = None
230         if (stride != 1) or (self.in_channels != out_channels):
231             downsample = nn.Sequential(Block.conv3(self.in_channels,
                ↪  out_channels, stride=stride),
232                                        nn.BatchNorm2d(out_channels))
233         layers = []
234         layers.append(Block(layer, self.in_channels, out_channels, stride,
            ↪  downsample))
235         self.in_channels = out_channels
236         for i in range(1, blocks):
237             layers.append(Block(layer, out_channels, out_channels))
238         return nn.Sequential(*layers)
239
240     def forward(self, x):
241         x1 = x.clone()
242         out = self.conv(x1)
243         out = self.bn(out)
244         out = self.relu(out)
245         out = self.layer1(out)
246         # out = self.bn(out)
247         # out = self.layer2(out)
248         # out = self.bn2(out)
249         # out = self.layer3(out)
250         # out = self.bn3(out)
251         # out = self.output(out)
252         # out = self.output_act(out)
253         out = self.avg_pool(out)
254         out = out.view(out.size(0), -1)
255         out = self.fc(out)
256         x2 = nn.MaxPool2d(2, 2)(functional.relu(self.conv(x)))
257         # regression
258         x2 = self.conv_seqn(x2)
259         x2 = x2.view(x.size(0), -1)
260         x2 = self.fc_seqn(x2)
261         return out, x2
262
263
264 def run_code_for_training(net, train_loader, learning_rate=1e-4,
    ↪  momentum_set=0.9, epochs=10, device='cuda:0'):
265     report_fre = 50 # 500
266     net = net.to(device)
267     criterion1 = nn.CrossEntropyLoss()
268     criterion2 = nn.MSELoss()
269     optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate,
        ↪  momentum=momentum_set)
270     loss_running_record_label = []
```

```
271     loss_running_record_bbox = []
272     Iter_record = []
273     for epoch in range(epochs):
274         running_loss_label = 0.0
275         running_loss_bbox = 0.0
276         for i, data in enumerate(train_loader):
277             # print(i)
278             inputs, labels, bbox_gt = data
279             inputs = inputs.to(device)
280             labels = labels.to(device)
281             bbox_gt = bbox_gt.to(device)
282 
283             optimizer.zero_grad()
284             outputs = net(inputs)
285             outputs_label = outputs[0]
286             bbox_pred = outputs[1]
287 
288             loss_label = criterion1(outputs_label, labels)
289             loss_label.backward(retain_graph=True)
290             loss_regression = criterion2(bbox_pred, bbox_gt)
291             loss_regression.backward()
292 
293             optimizer.step()
294             running_loss_label += loss_label.item()
295             running_loss_bbox += loss_regression.item()
296             # print(loss_label.item(), loss_regression.item())
297             if (i + 1) % report_fre == 0:
298                 points1 = running_loss_label / float(report_fre)
299                 points2 = running_loss_bbox / float(report_fre)
300                 loss_running_record_label.append(points1)
301                 loss_running_record_bbox.append(points2)
302                 Iter_record.append(len(loss_running_record_label))
303                 print("\n[epoch:%d, batch:%5d] classification loss: %.3f
                    ↪  regression loss: %.3f"
304                       % (epoch + 1, i + 1, points1, points2))
305                 running_loss_label = 0.0
306                 running_loss_bbox = 0.0
307         net_path = './net.pth'
308         torch.save(net.state_dict(), net_path)
309         print('Finished Training')
310     return loss_running_record_label, loss_running_record_bbox, Iter_record
311 
312 
313 if __name__ == '__main__':
314     if torch.cuda.is_available():
315         myDevice = 'cuda:0'
```

```
316     else:
317         myDevice = 'cpu'
318 
319     args = FakeArg()
320     catList = args.class_list
321     dataFolder = args.root_path
322     catNum = len(catList)
323 
324     # seed = 0
325     # random.seed(seed)
326     # torch.manual_seed(seed)
327     # torch.cuda.manual_seed(seed)
328     # np.random.seed(seed)
329     # os.environ['PYTHONHASHSEED'] = str(seed)
330 
331     torch.cuda.memory_summary(device=None, abbreviated=False)
332 
333     # Download required images
334     myDownloader = Downloader(args)
335     myDownloader.make_folder()
336     print('downloading images')
337     img_info = myDownloader.get_image()
338 
339     # Load and normalize data
340     transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5),
        ↪ (0.5, 0.5, 0.5))])
341     batch_size = 4
342 
343     torch.cuda.empty_cache()
344 
345     trainSet = QZDatasetClass(img_info, transform)
346     trainLoader = DataLoader(dataset=trainSet, batch_size=batch_size,
        ↪ shuffle=True, num_workers=2)
347 
348     print('Data Loader created')
349 
350     net = QZhangNet(class_num=catNum)
351     print('Net Created')
352     netLoss1, netLoss2, iter1 = run_code_for_training(net, trainLoader,
        ↪ learning_rate=1e-6,
353                                                       momentum_set=0.95,
                                                          ↪ epochs=50,
                                                          ↪ device=myDevice)
354     print('Training Done')
355 
356     fig = plt.figure(figsize=(10, 8))
```

```
357     plt.plot(netLoss1, label='Classification')
358     plt.legend()
359     plt.show()
360 
361     fig = plt.figure(figsize=(10, 8))
362     plt.plot(netLoss2, label='Localization')
363     plt.legend()
364     plt.show()
365 
366     print('done4')
```

3.1.2 hw05_validation.py

```
 1 import copy
 2 import random
 3 import torch
 4 import os
 5 import numpy as np
 6 import torchvision.transforms as tvt
 7 import seaborn as sns
 8 import matplotlib.pyplot as plt
 9 import matplotlib.patches as patches
10 from PIL import Image
11 from hw05_training import QZhangNet, QZDatasetClass, FakeArg, Downloader
12 from torch.utils.data import DataLoader
13 
14 
15 class QZDatasetClassVal(Dataset):
16     def __init__(self, img_att, trans=None):
17         self.transform = trans
18         self.img_info = img_att
19 
20     def __len__(self):
21         return len(self.img_info)
22 
23     def __getitem__(self, idx):
24         img_path = self.img_info[idx]['img_path']
25         label = self.img_info[idx]['label']
26         bbox = self.img_info[idx]['bbox']
27         new_bbox = [bbox[0], bbox[1], bbox[0]+bbox[2], bbox[1]+bbox[3]]
28         image = Image.open(img_path)
29         image, new_bbox = self.make_square(image, new_bbox)
30         bb_tensor = torch.tensor(new_bbox, dtype=torch.float)
31         im_ts = self.transform(image)
32         return im_ts, label, bb_tensor, idx
33 
```

```
34      @staticmethod
35      def make_square(im, bbox_size, min_size=128):
36          w, h = im.size
37          w_ratio = w / min_size
38          h_ratio = h / min_size
39          bbox_size[0] = bbox_size[0] / w_ratio
40          bbox_size[1] = bbox_size[1] / h_ratio
41          bbox_size[2] = bbox_size[2] / w_ratio
42          bbox_size[3] = bbox_size[3] / h_ratio
43          image_resized = im.resize((min_size, min_size), Image.BOX)
44          return image_resized, bbox_size
45
46
47  def validation(net, val_loader, img_info, mat_size, device='cpu'):
48      confusion_mat = np.zeros([mat_size, mat_size], dtype=int)
49      net = copy.deepcopy(net)
50      net = net.to(device)
51      for i, data in enumerate(val_loader):
52          print(i)
53          inputs, labels, bbox_gt, img_idx = data
54          inputs = inputs.to(device)
55          labels = labels.to(device)
56          bbox_gt = bbox_gt.to(device)
57          outputs = net(inputs)
58          # print(outputs)
59          outputs_label = outputs[0]
60          bbox_pred = outputs[1]
61          output_bb = bbox_pred.tolist()
62
63          label_pred = []
64          for output in outputs_label:
65              label_pred.append(torch.argmax(output))
66
67          batch_idx = 0
68          for idx_tensor in img_idx:
69              idx_int = idx_tensor.item()
70              img_info[idx_int]['bbox_pred'] = output_bb[batch_idx]
71              img_info[idx_int]['label_pred'] = label_pred[batch_idx].item()
72              batch_idx += 1
73
74          for record_number in range(len(labels)):
75              confusion_mat[labels[record_number]][label_pred[record_number]]
                ↪  += 1
76      return confusion_mat, img_info
77
78
```

```
79  def plot_confusion_matrix(conf_mat, label_list):
80      size = len(conf_mat)
81      cat_size = np.sum(conf_mat) / size
82      labels = []
83      for row in range(size):
84          rows = []
85          for col in range(size):
86              count = conf_mat[row][col]
87              percent = "%.2f%%" % (count / cat_size * 100)
88              label = str(count) + '\n' + str(percent)
89              rows.append(label)
90          labels.append(rows)
91      labels = np.asarray(labels)
92
93      accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
94      stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
95      plt.figure(figsize=(10, 10))
96      sns.heatmap(conf_mat, annot=labels, fmt="", cmap="Blues", cbar=False,
97                  xticklabels=label_list, yticklabels=label_list)
98      plt.ylabel('True label')
99      plt.xlabel('Predicted label' + stats_text)
100     plt.title('Confusion Matrix')
101     file_name = 'net_confusion_matrix.jpg'
102     plt.savefig(file_name)
103     plt.show()
104
105
106 def plot_detection(img_info, cat, order, cat_list, img_per_class):
107     max_size = 128
108     cat_order = cat_list.index(cat)
109     index = cat_order * img_per_class + order
110     img_dic = img_info[index]
111     img_path = img_dic['img_path']
112     img_w_o = img_dic['img_size'][0]
113     img_h_o = img_dic['img_size'][1]
114     img_pre_label_idx = img_dic['label_pred']
115     img_pre_label = cat_list[img_pre_label_idx]
116
117     bbox_o = img_dic['bbox']
118     bbox_p = img_dic['bbox_pred']
119     bbox_adj = [bbox_p[0], bbox_p[1], bbox_p[2]-bbox_p[0],
        ↪  bbox_p[3]-bbox_p[1]]
120
121     w_ratio = img_w_o / max_size
122     h_ratio = img_h_o / max_size
123
```

```
124     bbox_p = [bbox_adj[0]*w_ratio, bbox_adj[1]*h_ratio, bbox_adj[2]*w_ratio,
        ↪ bbox_adj[3]*h_ratio]
125
126     image = Image.open(img_path)
127     fig = plt.figure(figsize=(10, 10/img_w_o * img_h_o))
128     ax = fig.add_subplot(1, 1, 1)
129
130     rect1 = plt.Rectangle((bbox_o[0], bbox_o[1]), bbox_o[2], bbox_o[3],
        ↪ fill=False, edgecolor='green', linewidth=1)
131     rect2 = plt.Rectangle((bbox_p[0], bbox_p[1]), bbox_p[2], bbox_p[3],
        ↪ fill=False, edgecolor='red', linewidth=1)
132     ax.add_patch(rect1)
133     ax.add_patch(rect2)
134     plt.imshow(image)
135     plt.text(bbox_p[0] + 10, bbox_p[1] + 40, img_pre_label, fontsize=10,
        ↪ color='red')
136     plt.text(bbox_o[0] + 10, bbox_o[1] + 40, cat, fontsize=10,
        ↪ color='green')
137     plt.savefig(cat+'_example'+str(img_dic['order'])+'.png')
138     plt.show()
139
140
141 if __name__ == '__main__':
142     print('start')
143     if torch.cuda.is_available():
144         device = 'cuda:0'
145     else:
146         device = 'cpu'
147
148     args = FakeArg()
149     args.root_path = root_path =
        ↪ '/content/drive/MyDrive/ECE695/hw4/coco/Val/'
150     args.coco_json_path =
        ↪ '/content/drive/MyDrive/ECE695/hw4/annotations2014/instances_val2014.json'
151     # args.root_path = root_path =
        ↪ '/content/drive/MyDrive/ECE695/hw4/coco/Train/'
152     # args.coco_json_path =
        ↪ '/content/drive/MyDrive/ECE695/hw4/annotations2014/instances_val2014.json'
153     # args.class_list = ["cat", "dog", "elephant"]
154     catList = args.class_list
155     args.images_per_class = 200  # 50
156     images_per_class = args.images_per_class
157     dataFolder = args.root_path
158     catNum = len(catList)
159
160     seed = 0
```

```
161     random.seed(seed)
162     torch.manual_seed(seed)
163     torch.cuda.manual_seed(seed)
164     np.random.seed(seed)
165     # torch.backends.cudnn.deterministic = True
166     # torch.backends.cudnn.benchmarks = False
167     os.environ['PYTHONHASHSEED'] = str(seed)
168
169     myDownloader = Downloader(args)
170     myDownloader.make_folder()
171     print('downloading images')
172     img_info_val = myDownloader.get_image()
173
174     transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5),
        ↪ (0.5, 0.5, 0.5))])
175     batch_size = 10
176
177     valSet = QZDatasetClassVal(img_info_val, transform)
178     valLoader = DataLoader(dataset=valSet, batch_size=batch_size,
        ↪ shuffle=False, num_workers=0)
179
180     valnet = QZhangNet(class_num=catNum)
181     valnet.load_state_dict(torch.load("./net.pth",
        ↪ map_location=torch.device(device)))
182     valnet.eval()
183     confusion_matrix, img_info_val_p = validation(valnet, valLoader,
        ↪ img_info_val, mat_size=len(catList), device=device)
184     plot_confusion_matrix(confusion_matrix, catList)
185
186     # plot_detection(img_info_val_p, 'cat', 0, args.class_list,
        ↪ args.images_per_class)
187     ####
188     # The following part is used to batch output pictures and can be
        ↪ ignored. The example script is shown above.
189     ####
190     # for i in range(len(catList)):
191     #     start_index = args.images_per_class * i
192     #     max_count = 0
193     #     for j in range(args.images_per_class):
194     #         if max_count == 10:
195     #             break
196     #         current_index = start_index + j
197     #         if img_info_val_p[current_index]['label'] ==
        ↪ img_info_val_p[current_index]['label_pred']:
198     #             max_count += 1
199     #             want_cat = catList[i]
```

```
200         #             plot_detection(img_info_val_p, catList[i], j,
            ↪  args.class_list, args.images_per_class)
```

3.2 Results

The output images are shown below.

3.3 Training loss

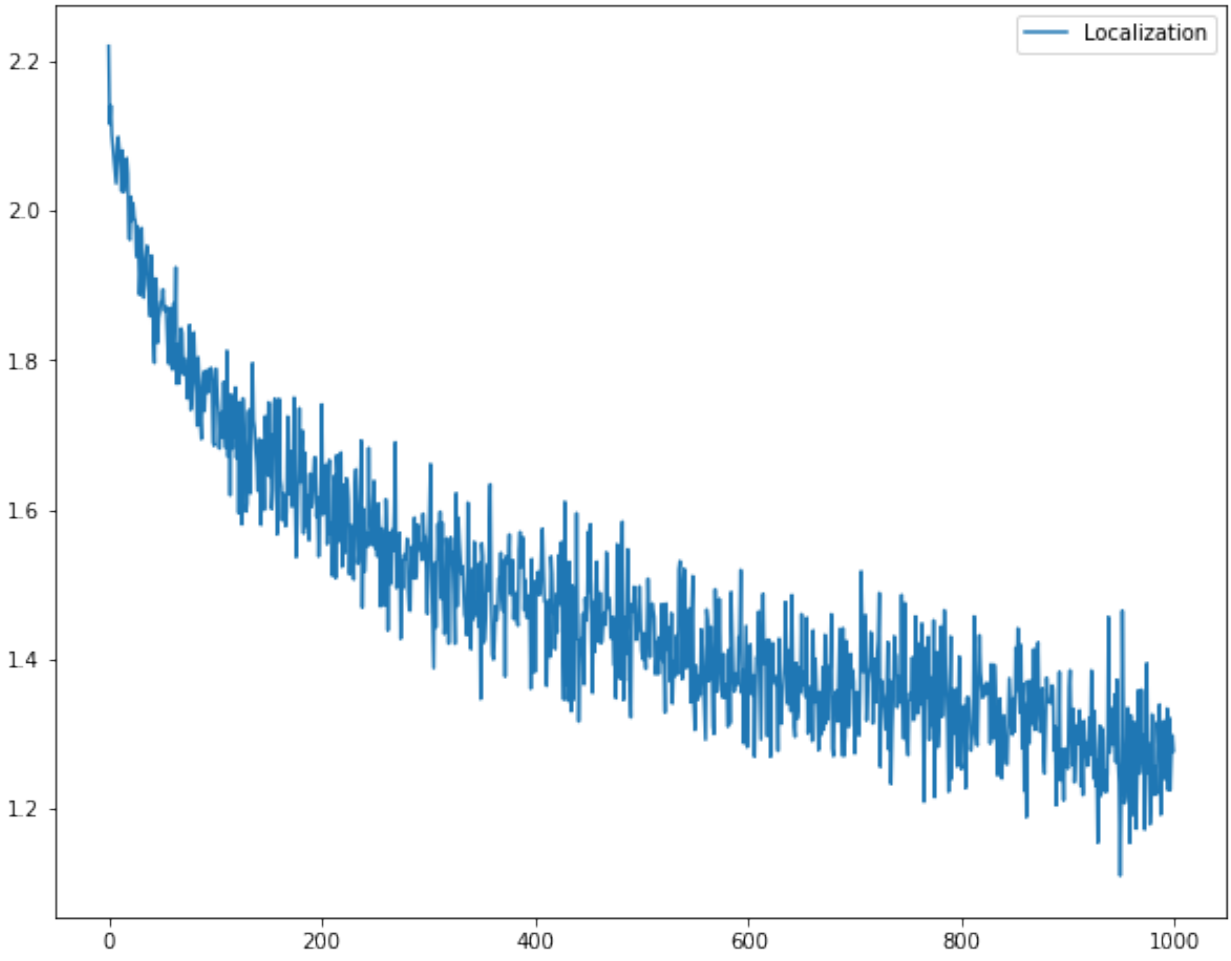

Figure 2: Net's training loss for classification

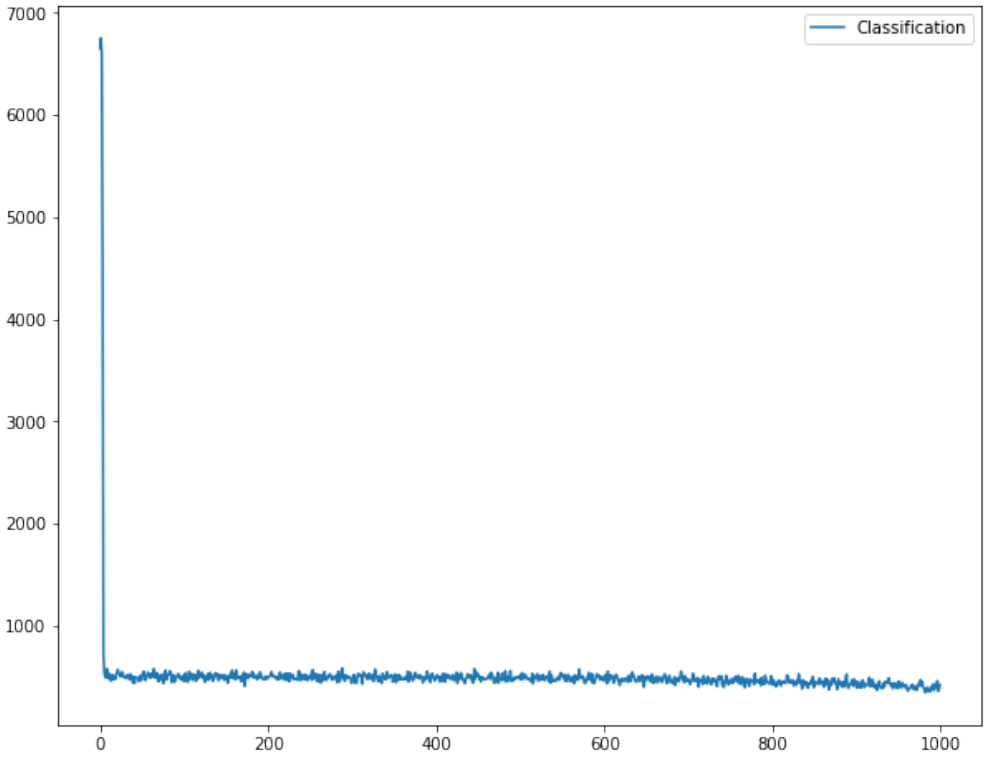

Figure 3: Net's training loss for localization

Because the epochs is large, the first several epochs for localization regression loss for is drawn to show the decrease part.

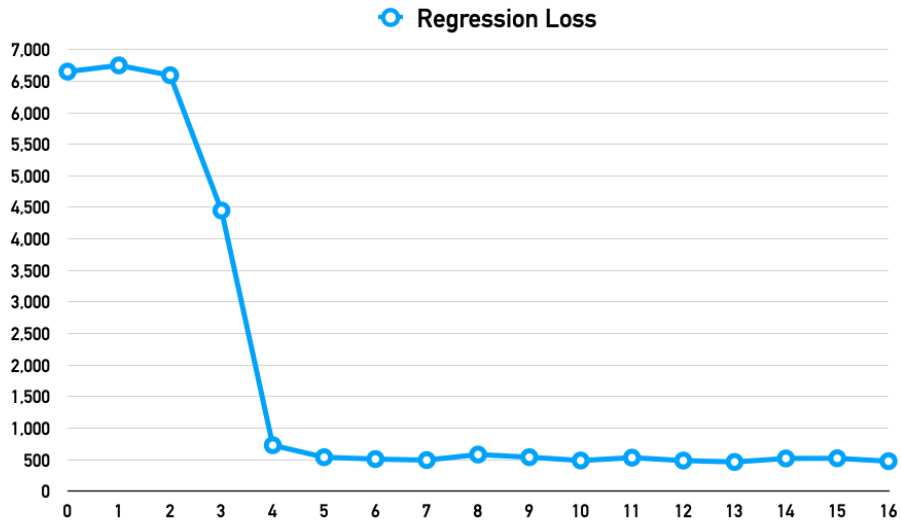

Figure 4: Net's training loss for localization

3.4 Confusion Matrix

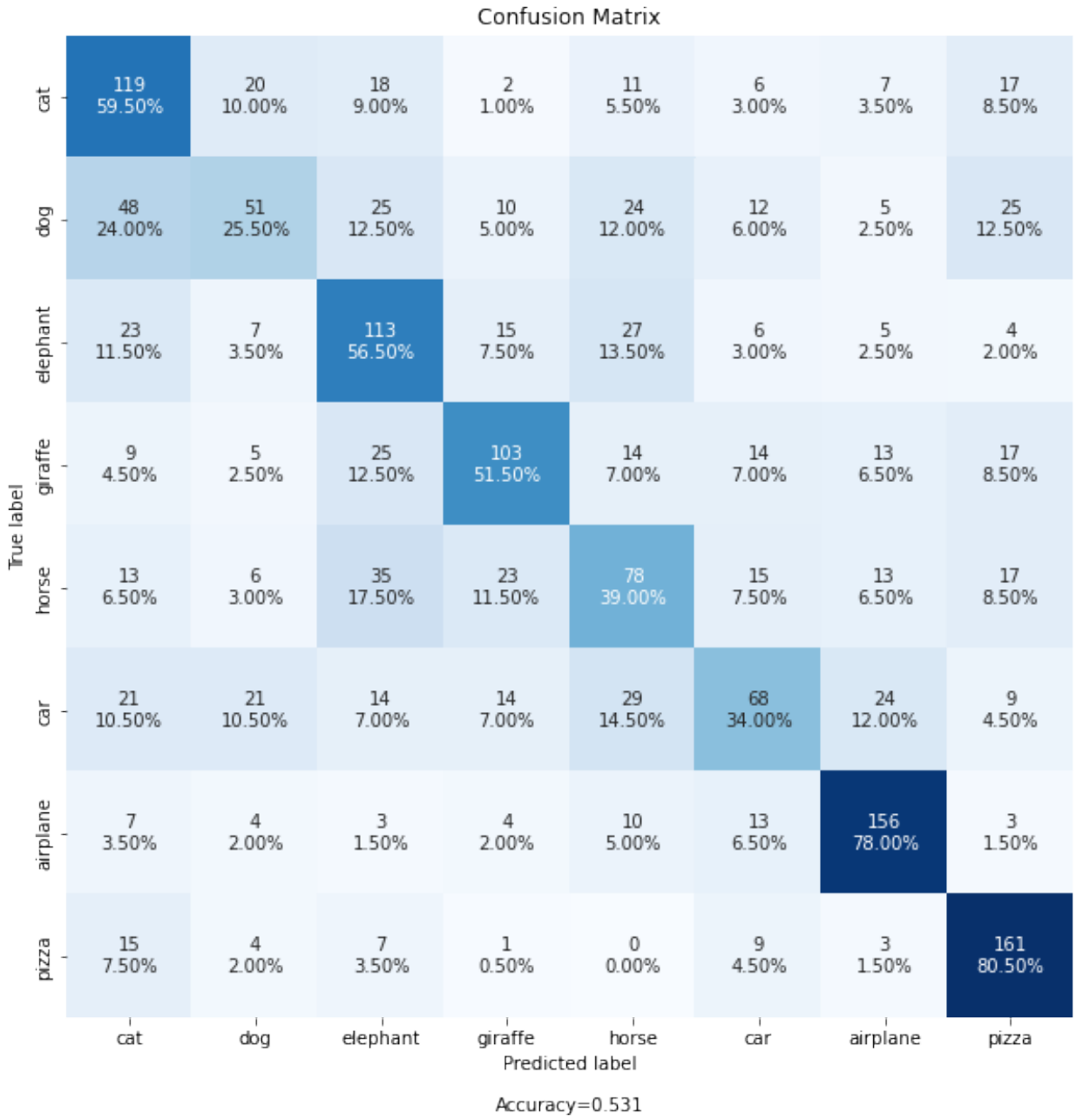

Figure 5: Net confusion matrix.

3.5 Annotated and Predicted Image Examples

3.5.1 Airplane

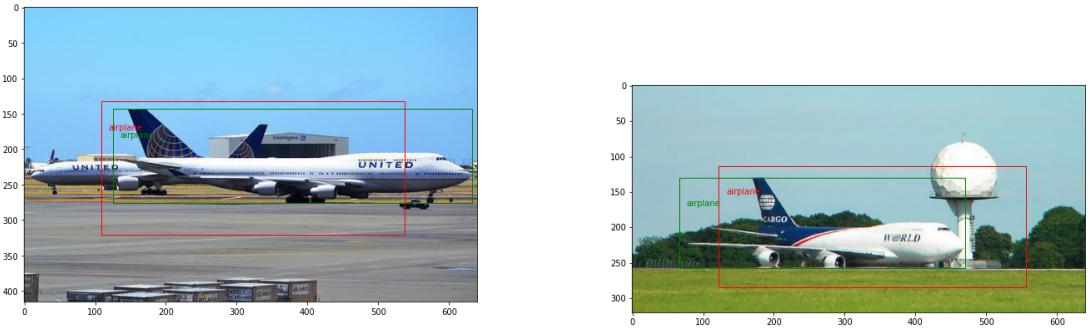

Figure 6: Airplane.

3.5.2 Car

Figure 7: Car.

3.5.3 Cat

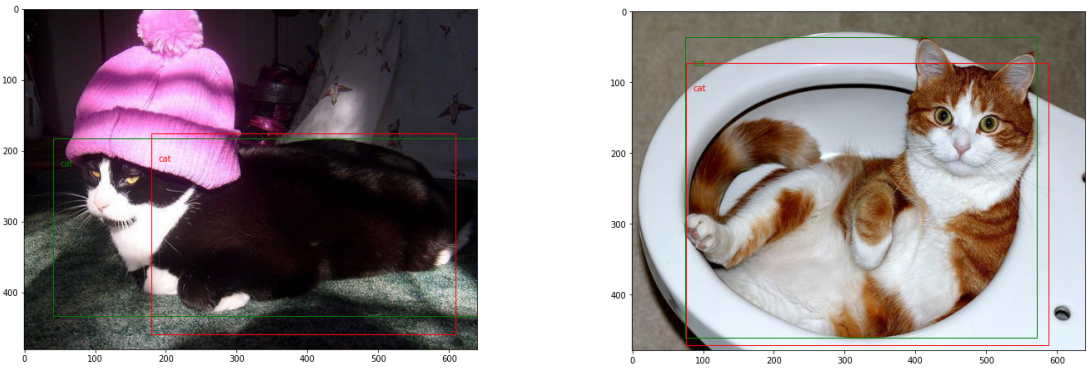

Figure 8: Cat.

3.5.4 Dog

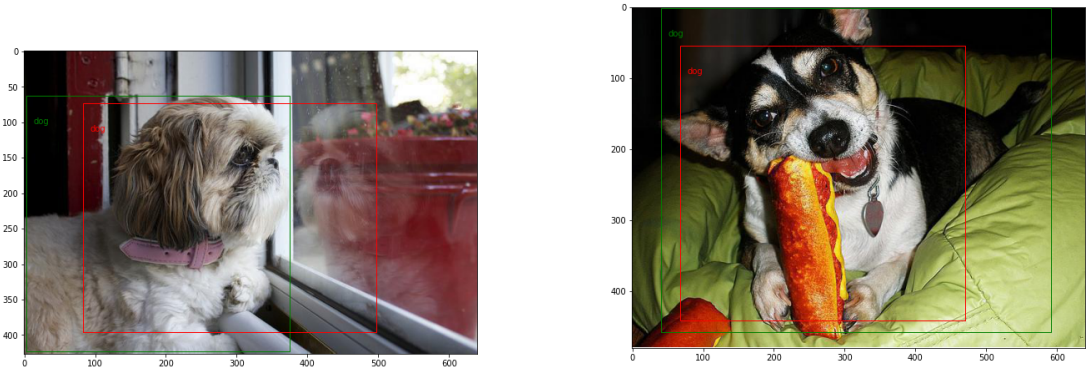

Figure 9: Dog.

3.5.5 Elephant

Figure 10: Elephant.

3.5.6 Giraffe

Figure 11: Giraffe.

3.5.7 Horse

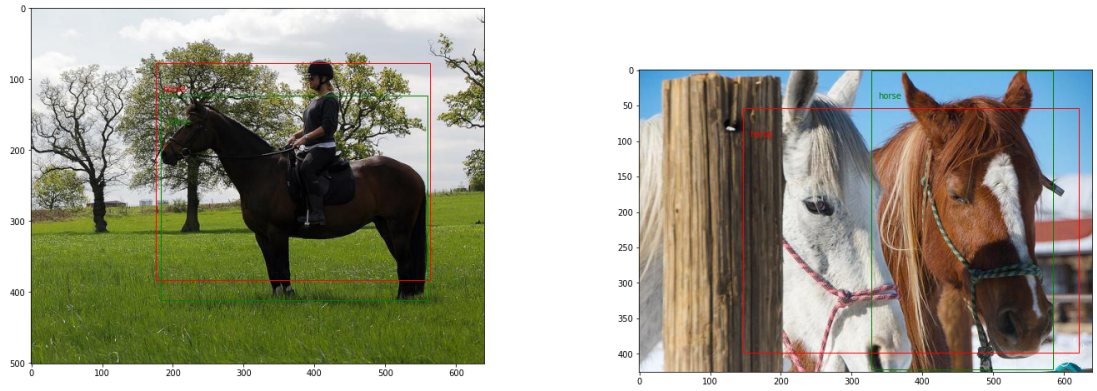

Figure 12: Horse.

3.5.8 Pizza

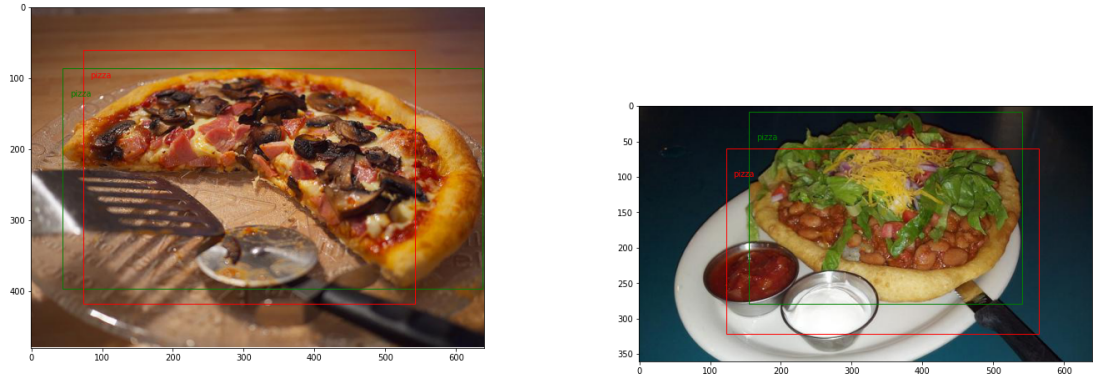

Figure 13: Pizza.

4 Lessons Learned

1. Objects of the same type are easily misidentified, such as cats and dogs, both of which are furry.
2. The gradients can explode if the learning rate is set too high.
3. Too many unreasonable layers may brings a large amount of parameters. It may cause CUDA out of memory.
4. Remember to turn off the random seed, otherwise it may cause the path of the SGD to be fixed on a certain route all the time, and then the gradient is not drop.

5. Increase the convolution layers may helps increase the accuracy for localization prediction.
6. The data in the training set constructs the gradient plane, so despite training with different parameters, the convergence loss may be approximately the same.
7. Too small input may cause that the train training effect is not ideal.

5 Suggested Enhancements

1. Try separate learning rate for classification and detection.
2. Try design different skip connection network and compare them.
3. Try to add more category to train the data.
4. Change the learning rate based on number of epochs.
5. Intersection over Union (IoU) metrics may be used instead of the MSE loss.
6. Recognition of multiple objects. Sometimes when multiple objects appear at the same time, the predicted marquee will cover all the objects, and the largest one that you actually want to predict is only a small part.