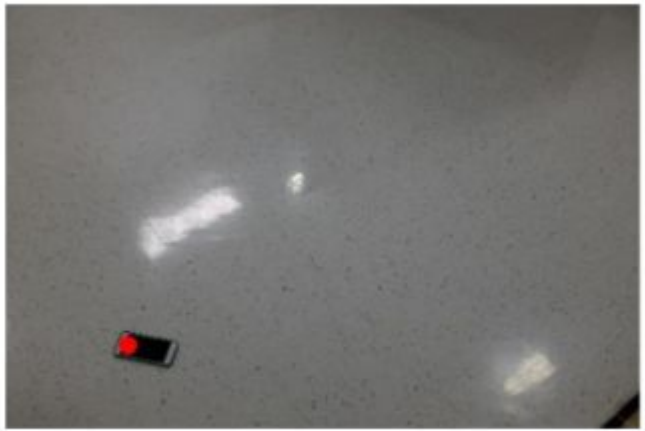
Assignment 1: Object Detection

Tahsin Reasat

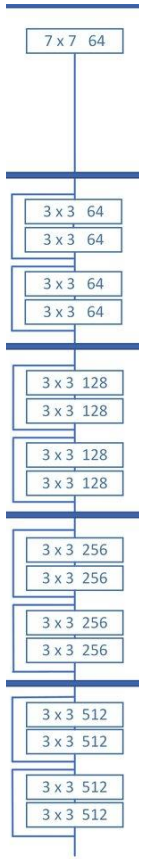
Task

Given an image of a mobile on a background, detect the location of the mobile.

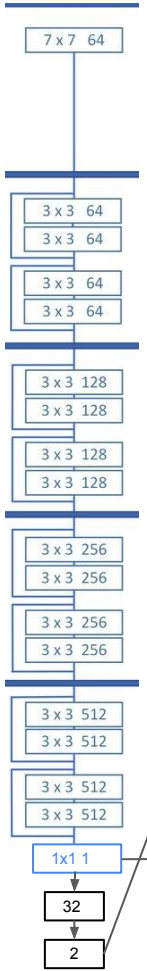


122.jpg

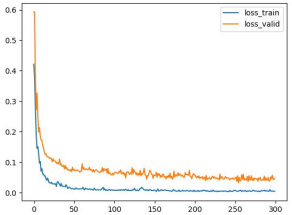
Network Architecture Rationale

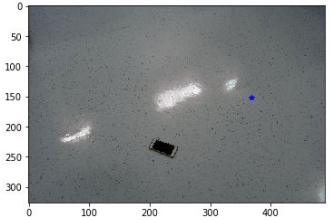


Backbone - ResNet18*

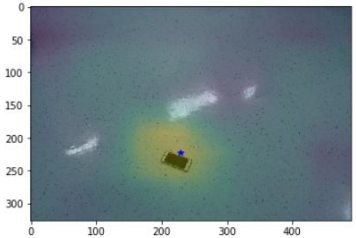


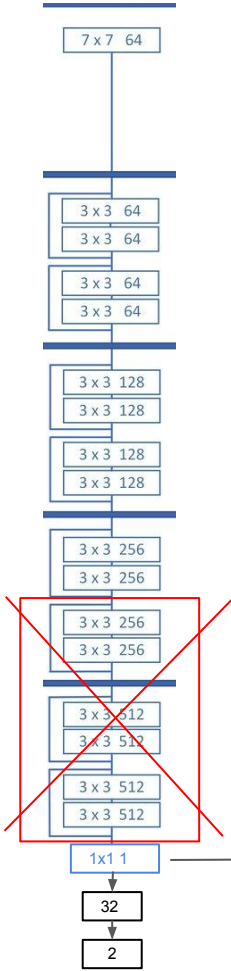
Initial Architecture





16 x 11 activation map





31 x 21 activation map

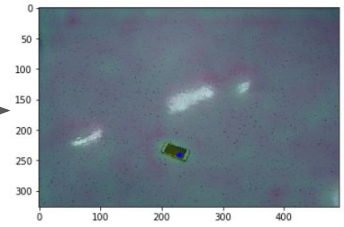


*Fig from Improved Selective Refinement Network for Face Detection. In reality, ResNet18 has 20 conv layers and 1 fc layer

Preprocessing

- Input RGB normalized according to ImageNet standards

  mean: (0.485, 0.456, 0.406),

  std: (0.229, 0.224, 0.225)

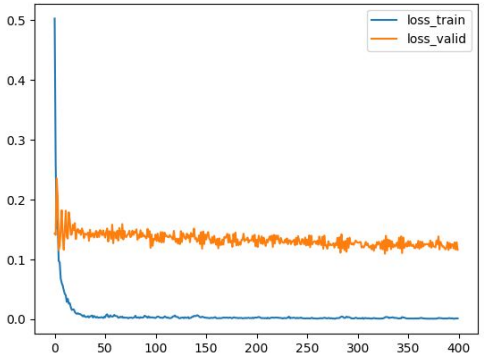
- Output x, y coordinates normalized to image dimension
- Augmentation: None

Parameters

- Optimizer: Adam
- Learning rate : 0.001
- Loss function: Mean Square Error
- Epoch Number: 400
- Batch Size: 32
- OS: Windows
- Specs: RTX 2070, AMD Ryzen 5 2600

Results: Loss Plots

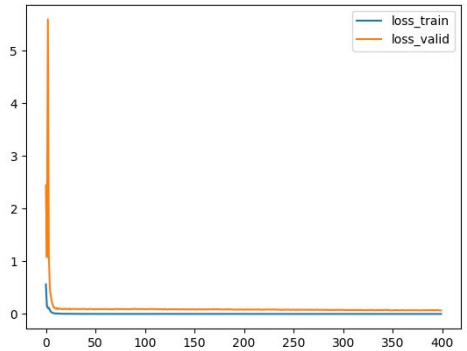
Loss



Epoch

Trained 400 epochs only updating the last conv+fc layers

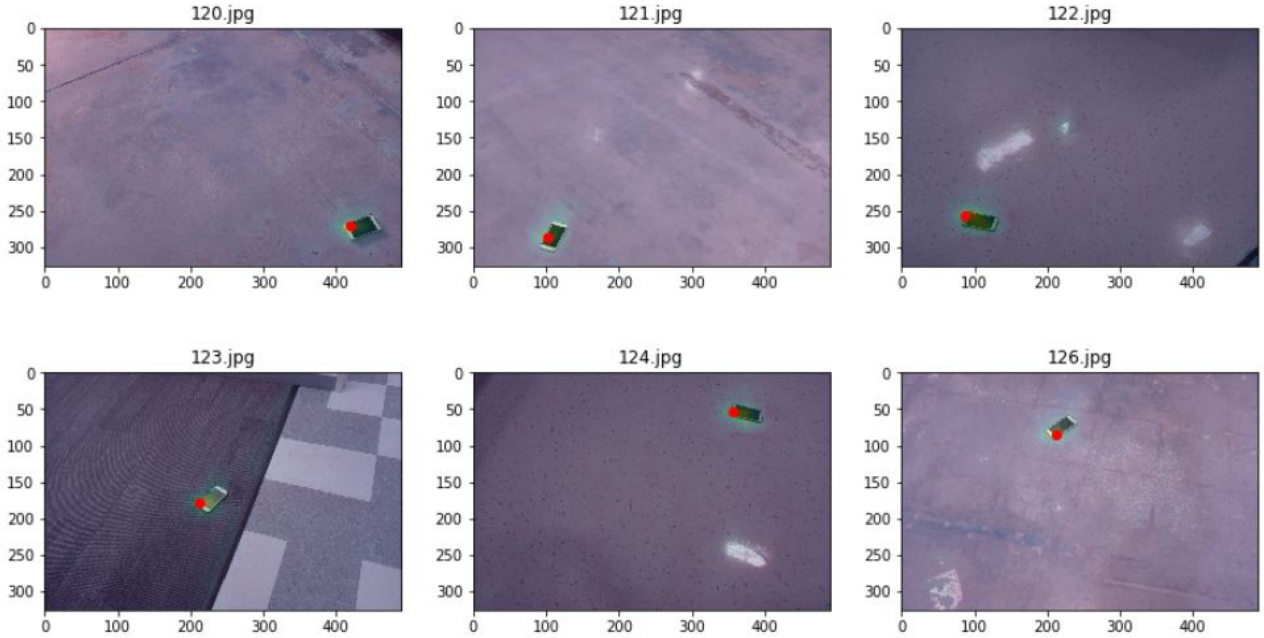
Loss



Epoch

Trained Another 400 epochs updating all the layers

Results: Test Output



Results: Test Coordinates

Name	Coordinate 1	Coordinate 2
120.jpg	0.831288	0.855102
121.jpg	0.880368	0.208163
122.jpg	0.785276	0.177551
123.jpg	0.546012	0.434694
124.jpg	0.165644	0.726531
126.jpg	0.260736	0.434694

Conclusion

The convolution to the fully connected layer mapping is not working properly.

Some sort of centerpooling layer might be better suited for this mapping.

Train.py

```
from glob import glob
import os
from albumentations import Compose, Normalize
import albumentations.pytorch as albu_torch
import sys
sys.path.insert(1,r'..\utility')
sys.path.insert(1,r'..\models')
from dataloader import Mobile_Dataset_RAM
from logger import Logger
from loss import loss_l2
from torch.utils.data import DataLoader
from models import ResNet18_conv_fc, ResNet8_conv_fc, ResNet12_conv_fc
import torch.optim as optim
import torch
import time
import argparse
import numpy as np
import pickle

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
TIME_STAMP=time.strftime('%Y-%m-%d-%H-%M-%S')

parser=argparse.ArgumentParser()
parser.add_argument('--dir_project', help='project directory', default=r'..')
parser.add_argument('--dir_lf', help='directory large
files',default=r'D:\Data\cs-8395-dl')
parser.add_argument('--folderData', help='data directory', default='assignment1_data')
parser.add_argument('--encoder',help='encoder',default='resnet12')
parser.add_argument('--lr', help='learning rate', type=float, default=0.001)
parser.add_argument('--batchSize', help='batch size', type=int, default=32)
parser.add_argument('--epoch', help='epoch', type=int, default=400)
parser.add_argument('--resume_from', help='filepath to resume training')
parser.add_argument('--bottleneckFeatures', help='bottleneck the encoder Features',
type=int, default=1)

args=parser.parse_args()

# setting up directories
DIR_LF = args.dir_lf#r'D:\Data\cs-8395-dl'
dir_data = os.path.join(DIR_LF,args.folderData)
#os.path.join(DIR_LF,'assignment1_data')
dir_model = os.path.join(args.dir_lf, 'model',TIME_STAMP)
dir_history = os.path.join(args.dir_project, 'history')
dir_log = os.path.join(args.dir_project, 'log')
dir_config = os.path.join(args.dir_project, 'config')

if os.path.exists(dir_history) is False:
   os.mkdir(dir_history)
```

```
if os.path.exists(dir_log) is False:
   os.mkdir(dir_log)
if os.path.exists(dir_config) is False:
   os.mkdir(dir_config)
if os.path.exists(os.path.join(args.dir_lf, 'model')) is False:
   os.mkdir(os.path.join(args.dir_lf, 'model'))

filepath_hist = os.path.join(dir_history, '{}.bin'.format(TIME_STAMP))
filepath_log = os.path.join(dir_log, '{}.log'.format(TIME_STAMP))
filepath_cfg = os.path.join(dir_config, '{}.cfg'.format(TIME_STAMP))

sys.stdout = Logger(filepath_log)
print(TIME_STAMP)
print(os.path.basename(__file__))
config=vars(args)
config_ls=sorted(list(config.items()))
print('------------------------------------------------------------------------------
-------------------------------------')
for item in config_ls:
   print('{}: {}'.format(item[0],item[1]))
print('------------------------------------------------------------------------------
-------------------------------------')
with open(filepath_cfg, 'w') as file:
   for item in config_ls:
       file.write('{}: {}\n'.format(item[0], item[1]))

if os.path.exists(dir_model)==0:
   print('creating directory to save model at {}'.format(dir_model))
   os.mkdir(dir_model)

filepath_model_best = os.path.join(dir_model, '{}_{}_best.pt'.format(TIME_STAMP,
args.encoder))  ##

dir_data_train = os.path.join(dir_data, 'train')
filepaths_train = glob(os.path.join(dir_data_train, '*.jpg'))
flnames_train = [os.path.basename(path)for path in filepaths_train]

dir_data_valid = os.path.join(dir_data, 'validation')
filepaths_valid = glob(os.path.join(dir_data_valid, '*.jpg'))
flnames_valid = [os.path.basename(path)for path in filepaths_valid]

filepath_labels = os.path.join(dir_data, 'labels', 'labels.txt')
with open(filepath_labels, 'r') as f:
   label_data = f.readlines()
label_dict = {}
for data in label_data:
   name, x, y = data.strip().split(' ')
   label_dict[name] = (float(x), float(y))

# Dataloader
```

```
aug = Compose([
    # Resize(256,256),
    #            RandomRotate90(),
    Normalize(),
    albu_torch.ToTensorV2()
],
)
BATCH_SIZE=args.batchSize
LR = args.lr
EPOCH=args.epoch
Dataset_train =
Mobile_Dataset_RAM(dir_data=dir_data_train,files=flnames_train,label_dict=label_dict,t
ransform=aug)
loader_train=DataLoader(Dataset_train,batch_size=BATCH_SIZE, shuffle=True)
print('train samples {}'.format(len(Dataset_train)))
Dataset_valid =
Mobile_Dataset_RAM(dir_data=dir_data_valid,files=flnames_valid,label_dict=label_dict,t
ransform=aug)
loader_valid=DataLoader(Dataset_valid,batch_size=BATCH_SIZE, shuffle=False)
print('validation samples {}'.format(len(Dataset_valid)))
# Model
if args.encoder == 'resnet18':
    model = ResNet18_conv_fc(pretrained=True,
bottleneckFeatures=args.bottleneckFeatures).to(device)
if args.encoder == 'resnet8':
    model = ResNet8_conv_fc(pretrained=True,
bottleneckFeatures=args.bottleneckFeatures).to(device)
if args.encoder == 'resnet12':
    model = ResNet12_conv_fc(pretrained=True,
bottleneckFeatures=args.bottleneckFeatures).to(device)

print(model)

# Optimizer
optimizer = optim.Adam(model.parameters(), lr=LR, betas=(0.9, 0.999), eps=1e-08,
weight_decay=0,
                               amsgrad=False)

# Train
if args.resume_from is not None:
    # Resume?
    print('resuming training from {}'.format(args.resume_from))
    train_states = torch.load(args.resume_from)
    model.load_state_dict(train_states['model_state_dict'])
    optimizer.load_state_dict(train_states['optimizer_state_dict'])
    epoch_range = np.arange(train_states['epoch']+1, train_states['epoch']+1+EPOCH)
else:
    train_states = {
                'epoch': 0,
                'model_state_dict': model.state_dict(),
```

```
                'optimizer_state_dict': optimizer.state_dict(),
                'model_save_criteria': np.inf,
            }
    epoch_range = np.arange(1,EPOCH+1)

loss_train=[]
loss_valid=[]
for epoch in epoch_range:
    running_loss = 0
    model.train()
    for i, sample in enumerate(loader_train):
        optimizer.zero_grad()
        img = sample[0].to(device)
        target = sample[1].to(device)
        output = model(img)
        loss = loss_l2(target,output)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        mean_loss = running_loss / (i + 1)
        print('train >>> epoch: {}/{}, batch: {}/{}, mean_loss: {:.4f}'.format(
            epoch,
            epoch_range[-1],
            i+1,
            len(loader_train),
            mean_loss

        ))
    loss_train.append(mean_loss)
    model.eval()
    running_loss = 0
    with torch.no_grad():
        for i, sample in enumerate(loader_valid):
            img = sample[0].to(device)
            target = sample[1].to(device)
            output = model(img)
            loss = loss_l2(target, output)
            # print(loss.item())
            running_loss += loss.item()
            mean_loss = running_loss / (i + 1)
            # img_r = reverse_transform(img.cpu().squeeze())
            # print(img_r.shape)
            # plt.imshow(img_r)
            # plt.plot(target.cpu().squeeze()[0] * img_r.shape[1],
target.cpu().squeeze()[1]* img_r.shape[0], 'r*')
            # plt.plot(output.cpu().squeeze()[0] * img_r.shape[1],
output.cpu().squeeze()[1] * img_r.shape[0], 'b*')
            # plt.show()

    print('valid >>> epoch: {}/{}, mean_loss: {:.4f}'.format(
```

```
        epoch,
        epoch_range[-1],
        mean_loss
    ))
    loss_valid.append(mean_loss)

    log = {
        'loss_train':loss_train,
        'loss_valid':loss_valid
        }
    with open(filepath_hist, 'wb') as pfile:
        pickle.dump(log, pfile)
    if mean_loss<train_states['model_save_criteria']:
        print('criteria decreased from {:.4f} to {:.4f}, saving best model at
{}'.format(train_states['model_save_criteria'],

mean_loss,

filepath_model_best))
        train_states = {
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'model_save_criteria': mean_loss,
        }
        torch.save(train_states, filepath_model_best)

print(TIME_STAMP)
```

Test.py

```
import os
from matplotlib import pyplot as plt
import argparse
from glob import glob
import sys
sys.path.insert(1,r'..\utility')
sys.path.insert(1,r'..\models')
from dataloader import  Mobile_Dataset_RAM, reverse_transform
from albumentations import Compose, Normalize
import albumentations.pytorch as albu_torch
from torch.utils.data import DataLoader
from models import ResNet12_conv_fc
import torch
import torch.nn as nn
from skimage import io, transform
import numpy as np

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
parser=argparse.ArgumentParser()
parser.add_argument('--filepath',required=True)
parser.add_argument('--filepath_model',default=r'..\model_weight\2020-01-27-20-33-41_r
esnet12_best.pt')
args = parser.parse_args()
dir_img = os.path.dirname(args.filepath)
flnames = os.path.basename(args.filepath)
aug = Compose([
   Normalize(),
   albu_torch.ToTensorV2()
],
)
Dataset = Mobile_Dataset_RAM(dir_data=dir_img, files=[flnames], label_dict=None,
transform=aug)
print('number of samples {}'.format(len(Dataset)))
loader=DataLoader(Dataset,batch_size=1, shuffle=False)
model = ResNet12_conv_fc(pretrained=False,bottleneckFeatures=False).to(device)
res_last_conv = nn.Sequential(*list(model.children())[:-2])
train_states=torch.load(args.filepath_model)
print('loading model from epoch {}, with criteria
{}'.format(train_states['epoch'],train_states['model_save_criteria']))
model.load_state_dict(train_states['model_state_dict'])
model.eval()
with torch.no_grad():
   for i, sample in enumerate(loader):
       img = sample[0].to(device)
       output = res_last_conv(img)
       img_r = reverse_transform(img.cpu().squeeze())
       am_np=output.squeeze().cpu().numpy()
       am_np_rz=transform.resize(
               am_np,
               img_r.shape[:2])
       x,y=np.unravel_index(am_np_rz.argmax(), am_np_rz.shape)
       print(x / img_r.shape[0], y / img_r.shape[1])
       print('row, column ==> {:.4f}, {:.4f}'.format(x / img_r.shape[0], y /
img_r.shape[1]))
       plt.imshow(img_r)
       plt.imshow(
           am_np_rz,
           alpha=0.3
       )
       plt.plot(y,x,'ro')
       plt.show()
```

Models.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
from torchvision import models
from torch import nn
from torch.nn import functional as F
import torch
from torchvision import models
import torchvision


class ResNet18(nn.Module):
    def __init__(self, pretrained,bottleneckFeatures=1):
        super(ResNet18,self).__init__()
        self.resnet18 = models.resnet18(pretrained=pretrained)
        self.resnet18_fc_stripped = nn.Sequential(*list(self.resnet18.children())[:-1])
        if bottleneckFeatures ==1:
            print('freezing feature extracting layers')
            for param in self.resnet18_fc_stripped.parameters():
                param.requires_grad = False
        self.fc1 = nn.Linear(in_features=512, out_features=2)

    def forward(self, x):
        x = self.resnet18_fc_stripped(x)
        x = x.reshape(x.size(0), -1)
        # print(x.shape)
        x = self.fc1(x)
        return x
class ResNet18_conv_fc(nn.Module):
    def __init__(self, pretrained,bottleneckFeatures=1):
        super(ResNet18_conv_fc,self).__init__()
        resnet18 = models.resnet18(pretrained=pretrained)
        self.resnet18_fc_stripped = nn.Sequential(*list(resnet18.children())[:-2])
        if bottleneckFeatures ==1:
            print('freezing feature extracting layers')
            for param in self.resnet18_fc_stripped.parameters():
                param.requires_grad = False
        self.conv_last = nn.Conv2d(512,1,kernel_size=(1,1),stride=(1,1))
        self.fc1 = nn.Linear(in_features=16*11, out_features=32)
        self.fc2 = nn.Linear(in_features=32, out_features=2)

    def forward(self, x):
        x = self.resnet18_fc_stripped(x)
        x = self.conv_last(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

class ResNet8_conv_fc(nn.Module):
    def __init__(self, pretrained,bottleneckFeatures=1):
        super(ResNet8_conv_fc,self).__init__()
        resnet18 = models.resnet18(pretrained=pretrained)
```

```
        self.resnet_partial =
nn.Sequential(nn.Sequential(*list(resnet18.children())[:5],

list(resnet18.children())[5][0]))
        if bottleneckFeatures ==1:
            print('freezing feature extracting layers')
            for param in self.resnet_partial.parameters():
                param.requires_grad = False
        self.conv_last = nn.Conv2d(128,1,kernel_size=(1,1),stride=(1,1))
        self.fc1 = nn.Linear(in_features=62*41, out_features=32)
        self.fc2 = nn.Linear(in_features=32, out_features=2)

    def forward(self, x):
        x = self.resnet_partial(x)
        # print(x.shape)
        x = self.conv_last(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

class ResNet12_conv_fc(nn.Module):
    def __init__(self, pretrained,bottleneckFeatures=1):
        super(ResNet12_conv_fc,self).__init__()
        resnet18 = models.resnet18(pretrained=pretrained)
        # print(resnet18)
        self.resnet_partial = nn.Sequential(*list(resnet18.children())[:6],
*list(list(resnet18.children())[6][0].children())[:-1])
        if bottleneckFeatures ==1:
            print('freezing feature extracting layers')
            for param in self.resnet_partial.parameters():
                param.requires_grad = False
        self.conv_last = nn.Conv2d(256,1,kernel_size=(1,1),stride=(1,1))
        self.fc1 = nn.Linear(in_features=31*21, out_features=32)
        self.fc2 = nn.Linear(in_features=32, out_features=2)

    def forward(self, x):
        x = self.resnet_partial(x)
        # print(x.shape)
        x = self.conv_last(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x
if __name__ =='__main__':
    model=ResNet12_conv_fc(pretrained=False)
    print(model)
    data=torch.rand(2,3,490,326)
    print(data.shape)
    output=model(data)
```

```
    print(output.shape)
```

Dataloader.py

```
from torch.utils.data import Dataset
from PIL import Image
from tqdm import tqdm
import os
import random
import numpy as np
import cv2
import torch
from glob import glob
from skimage import io
from scipy.ndimage import gaussian_filter
from albumentations import (
    Compose,
    Normalize,
)
import albumentations.pytorch as albu_torch

from matplotlib import pyplot as plt

def reverse_transform(img_t,mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]):
    img_r = np.array(img_t)
    img_r = img_r.transpose([1,2,0])
    img_r = img_r*std+mean
    img_r *=255
    img_r=img_r.astype(np.uint8)
    img_r = np.squeeze(img_r)
    return img_r

class Mobile_Dataset_RAM(Dataset):
    def __init__(self, dir_data, files, label_dict, transform=None):
        self.dir_data = dir_data
        self.transform = transform
        self.files = files
        self.image_all=[]
        self.label_dict=label_dict
        print('loading images to RAM')
        for file in tqdm(self.files):
            # file = self.files[idx]
            path_img = os.path.join(self.dir_data, file)
            image = cv2.imread(path_img)
            self.image_all.append(image)

    def __len__(self):
        size = len(self.files)
        return size
```

```
    def __getitem__(self, idx):
        image=self.image_all[idx]
        if 'test' in self.dir_data:
            target=[0.5,0.5]
        else:
            target=self.label_dict[self.files[idx]]
        # print(self.files[idx],image.shape)
        transformed=self.transform(image=image)
        img = transformed['image']
        return img,torch.tensor(target)


if __name__=='__main__':

    dir_data = r'D:\Data\cs-8395-dl\assignment1_data'
    dir_data_train = os.path.join(dir_data,'train')
    filepaths_train = glob(os.path.join(dir_data_train, '*.jpg'))
    filepaths_train_label = os.path.join(dir_data, 'labels', 'labels.txt')
    with open(filepaths_train_label, 'r') as f:
        label_data = f.readlines()
    label_dict = {}
    for data in label_data:
        name, x, y = data.strip().split(' ')
        label_dict[name] = (float(x), float(y))

    aug = Compose([
        # Resize(256,256),
        #            RandomRotate90(),
        Normalize(),
        albu_torch.ToTensorV2()
    ],
    )
    files = list(label_dict.keys())

    Mobile_Dataset =
Mobile_Dataset_HM_RAM(dir_data=dir_data_train,files=files,label_dict=label_dict,transf
orm=aug)
    sample = Mobile_Dataset[0]
    img = sample[0]
    target = sample[1].cpu().numpy()
    img = reverse_transform(img)
    plt.imshow(img)
    plt.imshow(target, alpha=0.3)
    plt.show()
```

Logger.py

```
import sys
class Logger(object):
    def __init__(self,path):
```

```
        self.terminal = sys.stdout
        self.log = open(path, "a+")

    def write(self, message):
        self.terminal.write(message)
        self.log.write(message)

    def flush(self):
        #this flush method is needed for python 3 compatibility.
        #this handles the flush command by doing nothing.
        #you might want to specify some extra behavior here.
        pass
```

loss.py

```
import torch

def loss_l2(y,y_p, gamma=0):
    #calculate loss per sample
    y=y.double()
    y_p=y_p.double()
    loss = (((y-y_p)**gamma)*((y-y_p)*(y-y_p))).sum()/y.shape[0]
    return loss
```