Computer Vision
Assignment - 3

Abhishek Rajgaria 2017276

Ques1
Perform the following on MNIST dataset to build three new datasets:

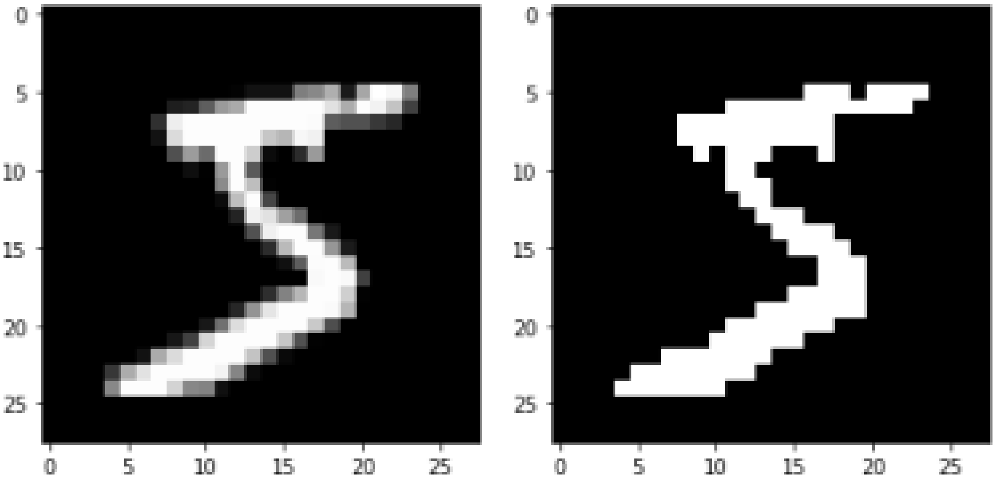
part(a)
Obtain foreground segmentation masks for images in MNIST dataset using TSS-based threshold [Q1, Assignment 1]. In this way, you have rough ground truth masks required to build a new foreground segmentation dataset. [1 Mark]
Note: The pre-existing labels are of no use here. The goal of the dataset is just to extract the foreground.

Steps followed:

- Firstly, I loaded the raw MNIST handwritten dataset using torchvision.datasets and then converted each of the images in the train and test into numpy.array format. ● Then created foreground data which contains foreground image, threshold value and class labels.
- The foreground image and threshold have been evaluated using TSS-based threshold as in Q1-Midsem.

Sample outputs

Raw Image Foreground Image

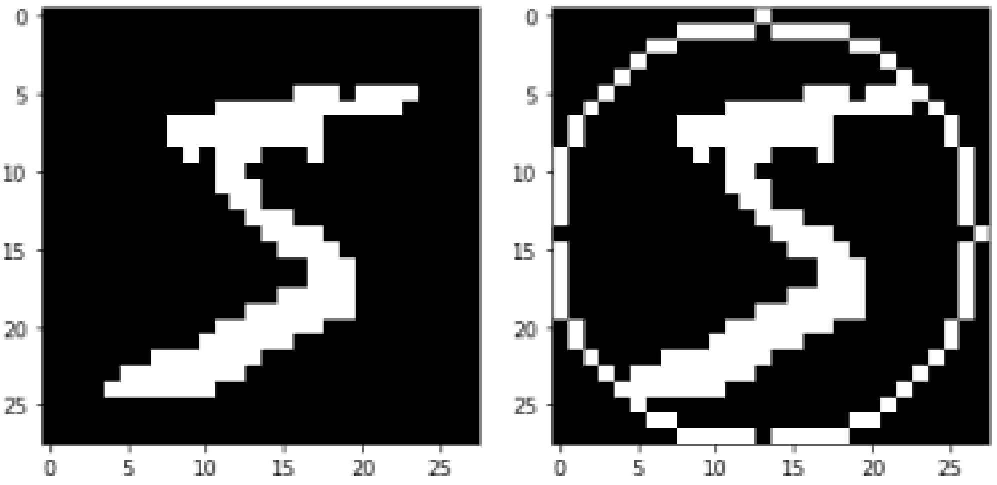


part(b)
Obtain tight groundtruth circles around the foreground segmentation masks obtained in (a). In this way, you can build a new dataset of 10 classes for performing classification with circlization (circular localization). You can use existing libraries for generating the tight circles. [1 Mark]

Steps followed:

- Use the foreground mask and get the coordinates of the foreground and reshape it into shape (nx2).
- I have used Miniball python library for finding the minimum enclosing circle for the foreground, from this we will get circle center coordinates and radius values. ● We have stored the original image, foreground mask, enclosed image, center coordinates , radius and original labels.

Sample output

Input Image Image with minEnclosing circle


part(c)

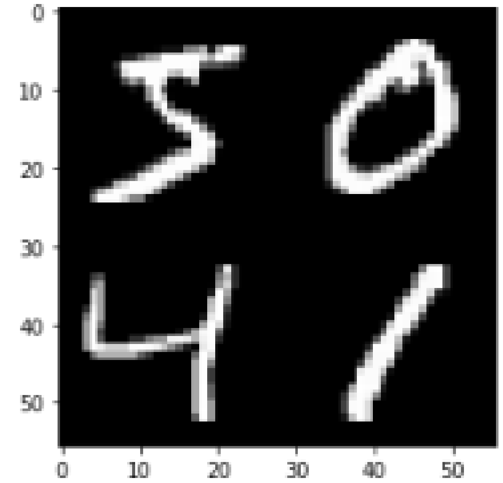
Randomly concatenate 4 images and their corresponding ground truths obtained in (a), along with the pre-existing labels, in a 2x2 manner to develop new images and semantic segmentation ground truths, respectively. In this way, you have a new dataset of 10 classes for performing semantic segmentation. [2 Marks]

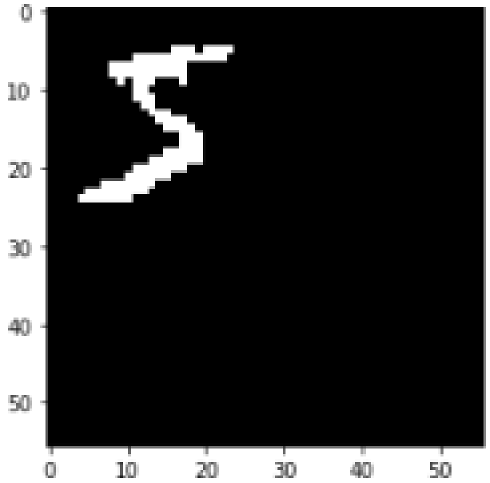
Steps followed:

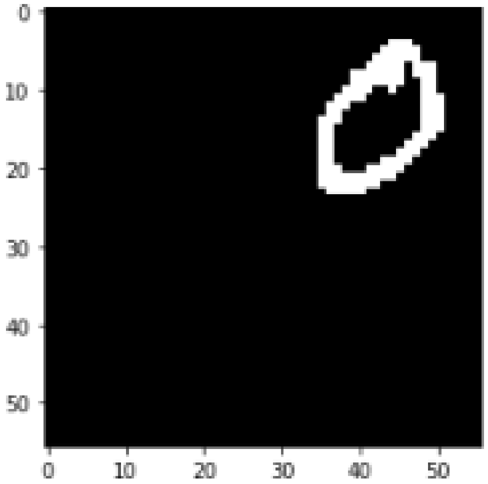
- For this I have used new data obtained from Q1 part(a), using this we have the original image and its corresponding foreground mask and its class label.
- As the data is already shuffled, so we have taken 4 images sequentially. ● Now these 4 images have been merged in 2x2 fashion to create a new input image. Therefore the size of the new input image = 56x56
- For the ground truth we have a layer for each class and background for the semantic segmentation.
- For each image we have masked the foreground in the corresponding layer (layer number = label of that image) with 1 and in the last layer for background. Therefore the ground truth size = 56x56x11.

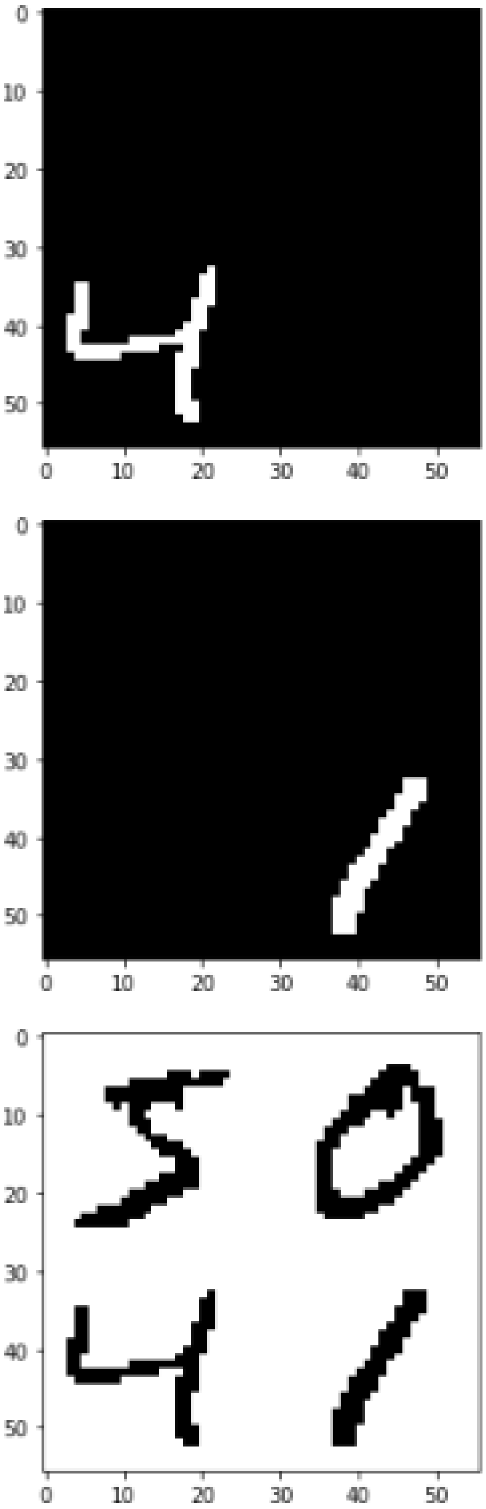
Sample output:

New Input Image Ground Truth Image only concerned layer and
background is shown rest are all zeros.









Ques2

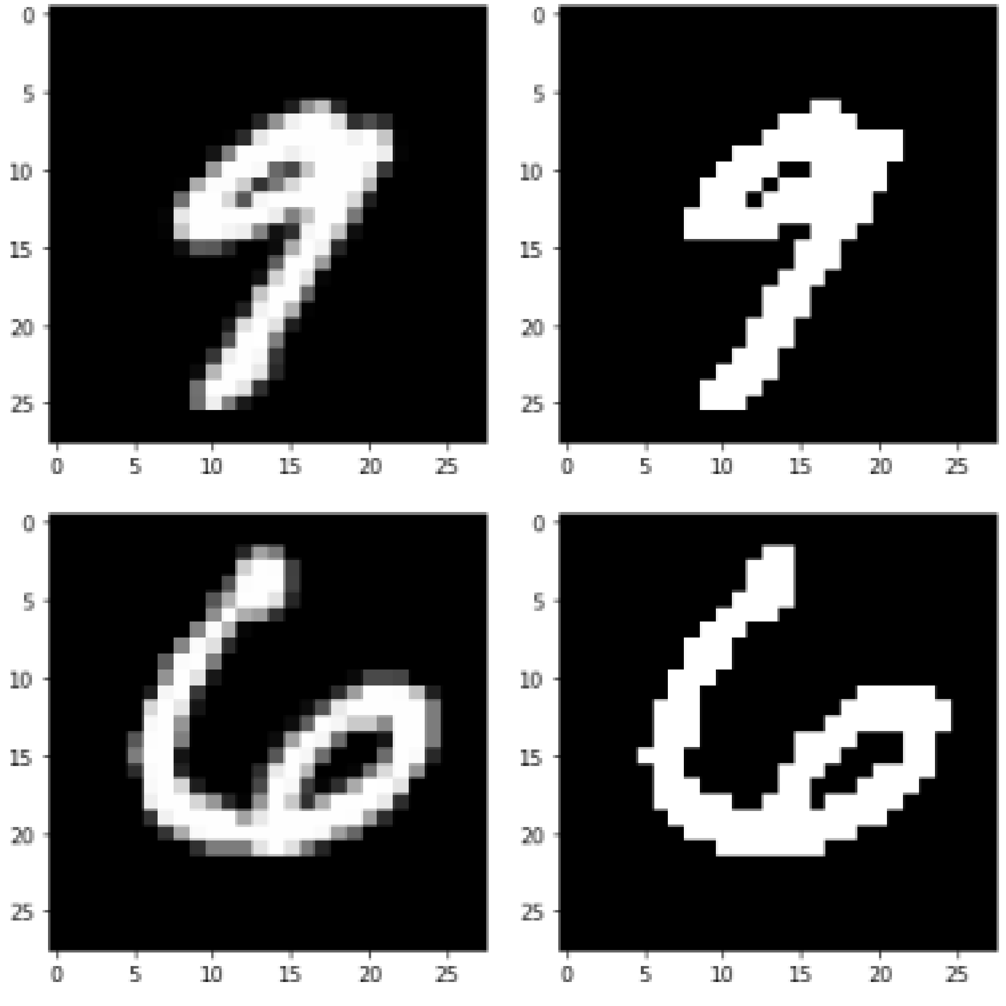
Train a DL network from scratch for performing foreground extraction on the new dataset obtained in Q1 (a). Report your test performance using Jaccard similarity. [3 Marks]

Steps followed:

- Firstly create the target for the model, by making the foreground mask as a binary mask.

- Now use a Deep Learning framework like keras or pytorch.
- Use Con2d layer and Upsampling to extract out the useful information from the image input.
- Optimizer used is **Adam** and loss function used in **binary cross entropy.** ● **Sigmoid** function is used in the final layer, therefore the outputs are between 0-1, so we make them binary and in order to do so, I used a threshold of **0.5** and values above threshold made to 1 else 0.
- Now for the Jaccard Similarity: I used **sklearn.metrics.jaccard_score** and for this I passed true binary mask and predicted binary mask in the flatten form.

Jaccard Similarity Score: **0.9791993266491706**

Input Image Foreground Image


Ques3

Train a DL network from scratch for performing classification with circlization on the new dataset obtained in Q1 (b). Report your test performance using Jaccard Similarity. [4 Marks]

Steps followed:

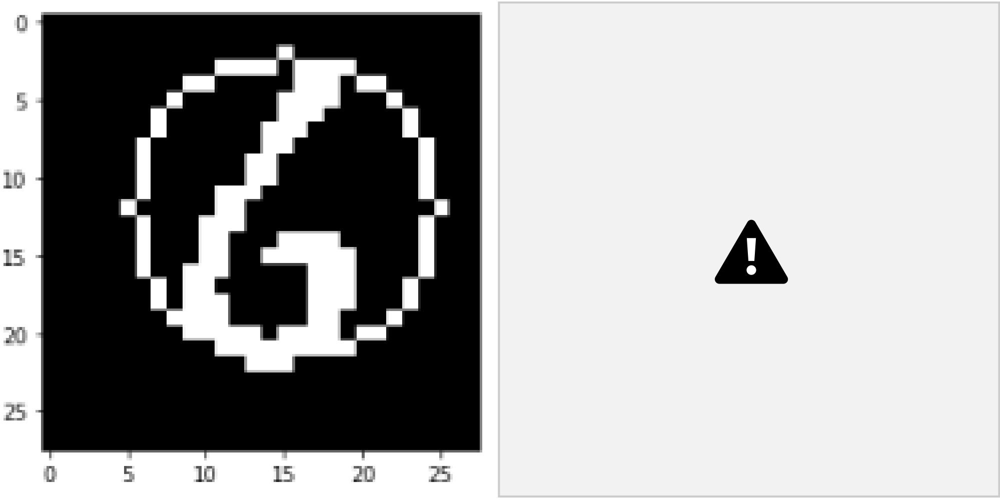
- Firstly create the target for the model, by computing the circle information and labels.

- Now use a Deep Learning framework like keras or pytorch.
- Use Con2d layer and Upsampling to extract out the useful information from the image input. Now in this model we have to perform two tasks namely: (classification and regression).
- Classification head is Fully connected layer which has an output dimension of 10 with the softmax as the activation function.
- Regression head is Fully connected layer which has an output dimension of 3 with the sigmoid as the activation function.
- Optimizer used is **Adam** and loss function used in **sparse_categorical_crossentropy** for classification head and **MSE** regression head.
- Now the predicted value would be in range 0-1, we will first scale it by 28 and then convert to int datatype.
- For the Jaccard Similarity score I have created a binary mask where coordinates inside the circle have value 1 else 0. (same for the ground truth and predicted output) ● Now for the Jaccard Similarity: I used **sklearn.metrics.jaccard_score** and for this I passed true binary mask and predicted binary mask in the flatten form.

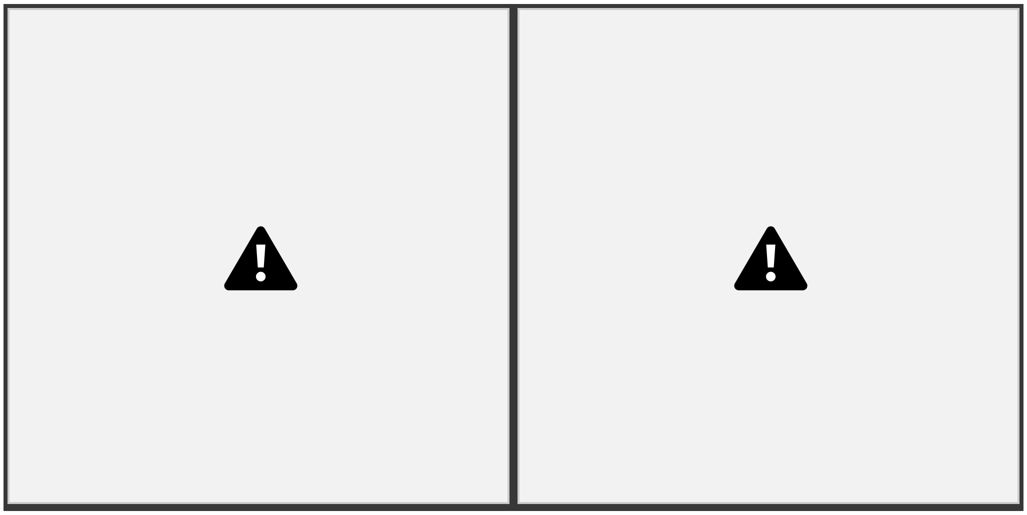
Classification Test Accuracy score: **0.9791**
Jaccard Similarity Score: **0.8899782135076253**

Circle_infor = [circle.x, circle.y, circle.radius]

Original min enclosing circle Predicted min enclosing circle
Circle_info = `[12. 15. 10.]` **pred_Circle_info =** `[11. 14. 11.]`



Circle_info = `[15. 12. 11.]` **pred_Circle_info =** `[15 12 11]`



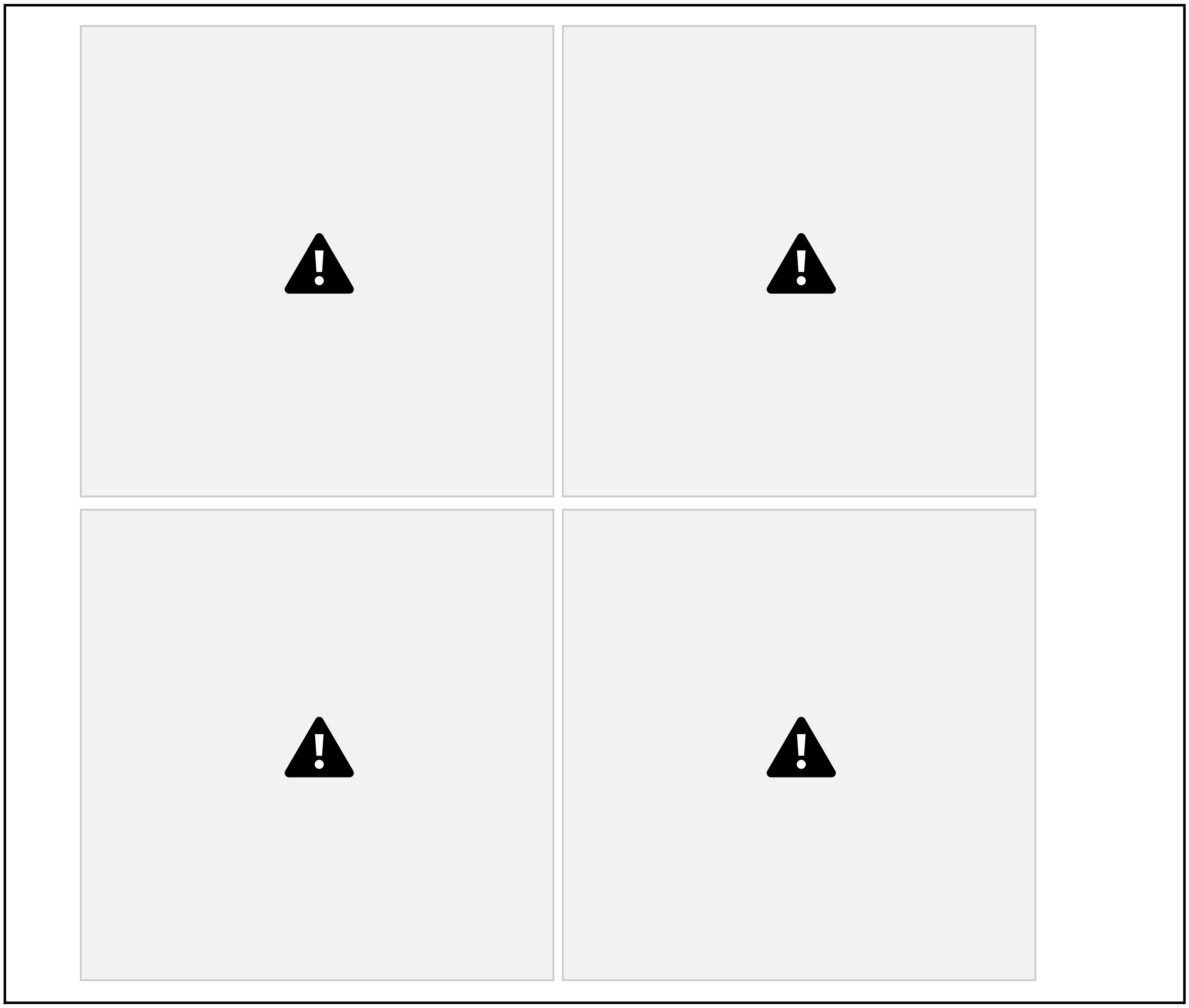
Ques4
Train a DL network from scratch for performing semantic segmentation on the new dataset obtained in Q1 (c). Report your test performance using Jaccard Similarity. [4 Marks]

Steps followed:

- Firstly create the target for the model, by making the foreground mask as a binary mask.
- Now use a Deep Learning framework like keras or pytorch.
- Use Con2d layer and Upsampling to extract out the useful information from the image input.
- Optimizer used is **Adam** and loss function used in **binary cross entropy. ● Sigmoid** function is used in the final layer, therefore the outputs are between 0-1, so we make them binary and in order to do so, I used a threshold of **0.5** and values above threshold made to 1 else 0.
- For the inference I have also converted the predicted output into 56x56x3, by giving each coordinate a different rgb color according to the channel number and 1 or 0 at that coordinate.
- Now for the Jaccard Similarity: I used **sklearn.metrics.jaccard_score** and for this I passed true binary mask and predicted binary mask in the flatten form.

Jaccard Similarity Score: **0.9604184124533751**

New Input Image Segmented Image



Code

Ques	Code

import

```
import pickle
 import cv2
 import copy
 import numpy as np
 import torch
 import math
 import torch.nn as nn
 import torch.nn.functional as F
 import torch.utils.data
 from torch.autograd import Variable
```

```
import torchvision
import torchvision.datasets as datasets
import matplotlib.pyplot as plt
mnist_trainset = datasets.MNIST(root='./data',
train=True, download=True, transform=None)
mnist_testset = datasets.MNIST(root='./data',
train=False, download=True, transform=None)
train_len = len(mnist_trainset)
test_len = len(mnist_testset)
raw_train = []
raw_test = []
print(train_len)
print(test_len)
for i in range(train_len):
    # print(mnist_trainset[i])

raw_train.append([np.array(mnist_trainset[i][0]),mnist_trainset
[i ][1]])

for i in range(test_len):

raw_test.append([np.array(mnist_testset[i][0]),mnist_testset[i]
[1 ]])

raw_train_file = open('raw_train_pkl', 'wb')
pickle.dump(raw_train, raw_train_file)
raw_train_file.close()

raw_test_file = open('raw_test_pkl', 'wb')
pickle.dump(raw_test, raw_test_file)
raw_test_file.close()
```

Q1 a

```
def get_fg(dataset):
    data = []
    n = len(dataset)
    for i in range(n):
        timage = dataset[i][0]
        label = dataset[i][1]
        th_value,th_image =
cv2.threshold(timage,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

```
        data.append([timage,th_image,th_value,label])
    return data

train_fg = get_fg(raw_train)
test_fg = get_fg(raw_test)
print(train_fg[0][0].shape)
fgdata_train_file = open('fgdata_train_pkl', 'wb')
pickle.dump(train_fg, fgdata_train_file)
fgdata_train_file.close()

fgdata_test_file = open('fgdata_test_pkl', 'wb')
pickle.dump(test_fg, fgdata_test_file)
fgdata_test_file.close()
plt.imshow(th_image,cmap="gray")
plt.show()
```

Q1 b

```
!pip install miniball
import miniball
def get_cirl(dataset):
    data = []
    n = len(dataset)
    for i in range(n):
        print(i)
        image = dataset[i][0]
        fg_image = copy.deepcopy(dataset[i][1])
        label = dataset[i][3]
        mask = np.array(np.where(fg_image == 255))
        C,r =
        miniball.get_bounding_ball(mask.transpose()) c =
        (math.floor(C[1]), math.floor(C[0]))
        # cy = C[1]
        r = math.ceil(np.sqrt(r))
        circ_image = cv2.circle(fg_image, c, r, 255,
thickness=1, lineType=8, shift=0)
        data.append([image, circ_image, dataset[i][1],
label, c[1], c[0], r])
        # plt.imshow(circ_image,cmap="gray")
        # plt.show()
        # plt.imshow(dataset[i][1],cmap="gray")
        # plt.show()
    return data
```

Q1 c

```python
train_circ = get_cirl(train_fg)
test_circ = get_cirl(test_fg)

circ_train_file = open('circ_train_pkl', 'wb')
pickle.dump(train_circ, circ_train_file)
circ_train_file.close()

circ_test_file = open('circ_test_pkl', 'wb')
pickle.dump(test_circ, circ_test_file)
circ_test_file.close()
circ_train_file = open('circ_train_pkl', 'wb')
pickle.dump(train_circ, circ_train_file)
circ_train_file.close()

circ_test_file = open('circ_test_pkl', 'wb')
pickle.dump(test_circ, circ_test_file)
circ_test_file.close()

np.set_printoptions(threshold=np.inf)
def get_semt(dataset):
    data = []
    n = len(dataset)
    for i in range(0,n,4):
        image1 = dataset[i][0]
        image2 = dataset[i+1][0]
        image3 = dataset[i+2][0]
        image4 = dataset[i+3][0]
        # print(type(image1))
        cmb_image = np.vstack([np.hstack([image1, image2]), np.hstack([image3, image4])])
        # print(cmb_image.shape)
        # plt.imshow(cmb_image,cmap="gray")
        # plt.show()
        row,col = cmb_image.shape

        label1 = dataset[i][3]
        label2 = dataset[i+1][3]
        label3 = dataset[i+2][3]
        label4 = dataset[i+3][3]
        # print(label1,label2,label3)
```

```
mask1_pnts = np.array(np.where(dataset[i][1] == 255))
mask2_pnts = np.array(np.where(dataset[i+1][1] ==
255)) mask2_pnts[1] = mask2_pnts[1]+28
mask3_pnts = np.array(np.where(dataset[i+2][1] ==
255)) mask3_pnts[0] = mask3_pnts[0]+28
mask4_pnts = np.array(np.where(dataset[i+3][1] ==
255)) mask4_pnts[0] = mask4_pnts[0]+28
mask4_pnts[1] = mask4_pnts[1]+28

bg_mask1_pnts = np.array(np.where(dataset[i][1] == 0))
bg_mask2_pnts = np.array(np.where(dataset[i+1][1] ==
0)) bg_mask2_pnts[1] = bg_mask2_pnts[1]+28
bg_mask3_pnts = np.array(np.where(dataset[i+2][1] ==
0)) bg_mask3_pnts[0] = bg_mask3_pnts[0]+28
bg_mask4_pnts = np.array(np.where(dataset[i+3][1] ==
0)) bg_mask4_pnts[0] = bg_mask4_pnts[0]+28
bg_mask4_pnts[1] = bg_mask4_pnts[1]+28

grnd_truth = np.zeros((row,col,11))
grnd_truth[mask1_pnts[0],mask1_pnts[1],label1]  =
1  grnd_truth[mask2_pnts[0],mask2_pnts[1],label2]
=                                              1
grnd_truth[mask3_pnts[0],mask3_pnts[1],label3]  =
1  grnd_truth[mask4_pnts[0],mask4_pnts[1],label4]
= 1

grnd_truth[bg_mask1_pnts[0],bg_mask1_pnts[1],10]  =
1  grnd_truth[bg_mask2_pnts[0],bg_mask2_pnts[1],10]
=                                                  1
grnd_truth[bg_mask3_pnts[0],bg_mask3_pnts[1],10]  =
1  grnd_truth[bg_mask4_pnts[0],bg_mask4_pnts[1],10]
= 1




 # print(grnd_truth[:,:,label1])
# plt.imshow(grnd_truth[:,:,label1],cmap="gray")
 # plt.show()

# plt.imshow(grnd_truth[:,:,label2],cmap="gray")
 # plt.show()
# plt.imshow(grnd_truth[:,:,label3],cmap="gray")
```

```
        # plt.show()
        # plt.imshow(grnd_truth[:,:,label4],cmap="gray")
```

```
        # plt.show()

        # plt.imshow(grnd_truth[:,:,10],cmap="gray")
        # plt.show()

        # plt.imshow(grnd_truth)
        # plt.show()

        # return
        print(i)
        data.append([cmb_image, grnd_truth])
    return data



train_semt = get_semt(train_fg)[:5000]
test_semt = get_semt(test_fg)
semt_train_file = open('semt_train_pkl', 'wb')
pickle.dump(train_semt, semt_train_file)
semt_train_file.close()

semt_test_file = open('semt_test_pkl', 'wb')
pickle.dump(test_semt, semt_test_file)
semt_test_file.close()
```

Q2

```
semt_train_file = open('semt_train_pkl', 'wb')
pickle.dump(train_semt, semt_train_file)
semt_train_file.close()

semt_test_file = open('semt_test_pkl', 'wb')
pickle.dump(test_semt, semt_test_file)
semt_test_file.close()
def get_fgdata_x_y(dataset):
    x = []
    y = []

    n = len(dataset)
    for i in range(n):
        # print(i)

x.append(dataset[i][0].reshape((28,28,1)).astype(np.float32))
```

```
y.append((dataset[i][1].reshape((28,28,1))/255).astype(np.float
32 ))

    x = np.array(x)
    y = np.array(y)
    print(x.shape)
    print(y.shape)

    return x,y


train_fgdata_x, train_fgdata_y =
get_fgdata_x_y(train_fgdata[:30000])
test_fgdata_x, test_fgdata_y =
get_fgdata_x_y(test_fgdata) import tensorflow as tf
from tensorflow.keras import datasets, layers,
models tf.keras.backend.clear_session()
model = models.Sequential()
model.add(layers.Conv2D(filters=16, kernel_size=(3,
3), activation='relu',
input_shape=train_fgdata_x.shape[1:], padding='same'))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
# model.add(layers.MaxPooling2D(pool_size=(2, 2)))
# model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
# model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
# model.add(layers.UpSampling2D(size=(2, 2)))
# model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.UpSampling2D(size=(2, 2)))
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3),
activation='relu', padding='same'))
model.add(layers.Conv2D(filters=16, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.Conv2D(filters=train_fgdata_y.shape[-1]
, kernel_size=(3, 3), activation='sigmoid',
padding='same')) model.summary()
# loss_function =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tru
e) model.compile(optimizer='adam',
             loss=tf.keras.losses.BinaryCrossentropy(),
             metrics=[tf.keras.metrics.BinaryAccuracy(),
                      tf.keras.metrics.Recall(),
                      tf.keras.metrics.Precision()])
history = model.fit(train_fgdata_x, train_fgdata_y,
epochs=10, batch_size=128,
                   validation_data=(test_fgdata_x,
test_fgdata_y))
model.save("fg_extraction_model")
from tensorflow.keras.models import load_model

model = load_model("fg_extraction_model")
test_output_fgdata = model.predict(test_fgdata_x)
print(test_output_fgdata.shape)
def apply_thresh(dataset,th):
    data = []
    n = len(dataset)
    for i in range(n):
        dataset[i][dataset[i] >th] = 1
        dataset[i] = dataset[i].astype(np.uint8)
        # segment_image = np.zeros((m,n,3))
        # for j in range(num_class):
        # segment_image[dataset[i][:,:,j]>th] = dic[j] #
            segment_image[dataset[i][:,:,j]<=th] = dic[10]
        # data.append(segment_image.astype(np.uint8))

    # return data

apply_thresh(test_output_fgdata,0.5)

from sklearn.metrics import jaccard_score
n_test = len(test_output_fgdata)
```

Q3

```
jc_score1 = 0
for i,j in zip(test_fgdata_y,test_output_fgdata):
    jc_score1+= jaccard_score(i.flatten(),j.flatten())

jc_score1 /= n_test
print(jc_score1)

plt.imshow(test_fgdata_x[201].reshape((28,28)), cmap =
"gray") plt.show()


plt.imshow(test_output_fgdata[201].reshape((28,28)), cmap
= "gray")
plt.show()

circ_train_file = open('circ_train_pkl', 'rb')
circ_test_file = open('circ_test_pkl', 'rb')
train_circ = pickle.load(circ_train_file)
test_circ = pickle.load(circ_test_file)

circ_train_file.close()
circ_test_file.close()
def get_circ_x_y(dataset):
    x = []
    y = []
    b = []
    # [image, circ_image, dataset[i][1], label, c[1], c[0],
    r] n = len(dataset)
    for i in range(n):
        # print(i)
        x.append(dataset[i][0].reshape((28,28,1)))
        y.append(dataset[i][3])

b.append([dataset[i][4]/28,dataset[i][5]/28,dataset[i][6]/28])


    x = np.array(x)
    y = np.array(y)
    b = np.array(b)
    print(x.shape)
    print(y.shape)
    print(b.shape)

    return x,{'label':y,'bbox':b}


train_circ_x, train_circ_y =
get_circ_x_y(train_circ[:30000]) test_circ_x, test_circ_y =
get_circ_x_y(test_circ) import tensorflow as tf
from tensorflow.keras import datasets, layers,
models from tensorflow.keras.layers import *
from tensorflow.keras.models import *

tf.keras.backend.clear_session()

def get_model():
    inputs = Input(shape=(28,28,1))
    x = Conv2D(filters=16, kernel_size=(3, 3),
activation='relu',padding="same")(inputs)
    x = Conv2D(filters=32, kernel_size=(3, 3),
activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Conv2D(filters=32, kernel_size=(3, 3),
activation='relu', padding='same')(x)
    x = Conv2D(filters=64, kernel_size=(3, 3),
activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = GlobalAveragePooling2D()(x)

    classifier_head = Dropout(0.3)(x)
    classifier_head = Dense(10, activation='softmax',
name='label')(classifier_head)

    reg_head = Dense(64, activation='relu')(x)
    reg_head = Dense(32, activation='relu')(reg_head)
    reg_head = Dense(3, activation='sigmoid',
name='bbox')(reg_head)

    return Model(inputs=[inputs],
outputs=[classifier_head, reg_head])
model = get_model()
model.summary()
```

```
                 losses = {'label':
 'sparse_categorical_crossentropy', 'bbox': 'mse'}

loss_weights = {'label': 1.0,
                'bbox': 1.0}
model.compile('adam', loss=losses,
loss_weights=loss_weights, metrics=['acc'])
history = model.fit(train_circ_x, train_circ_y,
epochs=20, batch_size=32,
                   validation_data=(test_circ_x, test_circ_y))
model.save("class_circ_model")
from tensorflow.keras.models import load_model
model = load_model("class_circ_model")
test_output_circ = model.predict(test_circ_x)
print(len(test_output_circ))
def get_label_circ(output):
    output_labels = output[0]
    output_circ = output[1]
    labels = np.argmax(output_labels, axis=1)
    print(labels.shape)
    output_circ = (output_circ*28).astype(np.uint8)

    # output_circ[:,0:2] =
    np.floor(output_circ[:,0:2]) # output_circ[:,2] =
    np.ceil(output_circ[:,2])

    return labels, output_circ

pred_label, pred_circ =
get_label_circ(test_output_circ) print(test_circ_y)
def get_circ_ext(dataset):
    mask = []
    enc = []
    y = []
    # [image, circ_image, dataset[i][1], label, c[1], c[0],
    r] n = len(dataset)
    for i in range(n):
        # print(i)
        mask.append(dataset[i][2])
        enc.append(dataset[i][1])
        y.append(dataset[i][3])
    mask = np.array(mask)
    enc = np.array(enc)
    y = np.array(y)
    print(mask.shape)
    print(enc.shape)
    print(y.shape)

    return mask, enc, y

test_circ_mask, test_circ_enc, test_circ_label =
get_circ_ext(test_circ)
def do_enc(mask, circ):
    data = []
    n = len(mask)
    for i in range(n):
        circ_image = cv2.circle(mask[i],
(circ[i][1],circ[i][0]), circ[i][2], 255, thickness=1,
lineType=8, shift=0) data.append(circ_image)

    return data

pred_enc_circ = do_enc(test_circ_mask, pred_circ)
from sklearn.metrics import jaccard_score

class Point:
    def __init__(self,x,y):
        self.x = x
        self.y = y

class Circle:
    def __init__(self,c,r):
        self.c = c
        self.r = r

def dist(a,b):
    return np.sqrt((a.x -b.x)**2 + (a.y-b.y)**2)

def inside(c,p):
    if(dist(c.c,p)<=c.r):
        return True
    return False
```

```
def compute_jc_score2(test_circ_y, pred_label,
    pred_circ): n_test = len(test_circ_y)
    # test_circ_y = test_circ_y.astype(np.uint8)
    jc_score = 0
    true_label = test_circ_y['label']
    true_circ =
    (test_circ_y['bbox']*28).astype(np.uint8) for i in
    range(n_test):
        if(true_label[i]!=pred_label[i]):
            continue
        true_mask = np.zeros((28,28))
        pred_mask = np.zeros((28,28))
        tc =
Circle(Point(true_circ[i][1],true_circ[i][0]),true_circ[i][2
        ]) pc =
Circle(Point(pred_circ[i][1],pred_circ[i][0]),pred_circ[i][2
        ]) for j in range(28):
            for k in range(28):
                p = Point(j,k)
                if(inside(tc,p)):
                    true_mask[j][k] = 1
                if(inside(pc,p)):
                    pred_mask[j][k] = 1
        jc_score+=
jaccard_score(true_mask.flatten(),pred_mask.flatten(

    )) return jc_score/n_test

jc_score2 = compute_jc_score2(test_circ_y, pred_label,
pred_circ) print(jc_score2)
sample = 0
print("circle information")
print(test_circ_y['bbox'][sample]*28)
print(pred_circ[sample])

plt.imshow(test_circ_enc[sample],cmap = "gray")
plt.show()
plt.imshow(pred_enc_circ[sample],cmap = "gray")
plt.show()
from sklearn.metrics import accuracy_score
print("Test Accuracy score: ", accuracy_score(test_circ_label,
pred_label))
```

Q4

```
semt_train_file = open('semt_train_pkl', 'rb')
semt_test_file = open('semt_test_pkl', 'rb')
train_semt = pickle.load(semt_train_file)
test_semt = pickle.load(semt_test_file)

semt_train_file.close()
semt_test_file.close()
def get_semt_x_y(dataset):
    x = []
    y = []

    n = len(dataset)
    for i in range(n):
        # print(i)
        x.append(dataset[i][0].reshape((56,56,1)))
        y.append(dataset[i][1])

    x = np.array(x)
    y = np.array(y)
    print(x.shape)
    print(y.shape)

    return x,y


train_semt_x, train_semt_y =
get_semt_x_y(train_semt) test_semt_x, test_semt_y =
get_semt_x_y(test_semt)
import tensorflow as tf
from tensorflow.keras import datasets, layers,
models tf.keras.backend.clear_session()

model = models.Sequential()
model.add(layers.Conv2D(filters=16, kernel_size=(3,
3), activation='relu',
input_shape=train_semt_x.shape[1:], padding='same'))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3),
activation='relu', padding='same'))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.UpSampling2D(size=(2, 2)))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.UpSampling2D(size=(2, 2)))
model.add(layers.Conv2D(filters=32, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.Conv2D(filters=16, kernel_size=(3,
3), activation='relu', padding='same'))
model.add(layers.Conv2D(filters=train_semt_y.shape[-1],
kernel_size=(3, 3), activation='sigmoid',
padding='same')) model.summary()
model.compile(optimizer='adam',
             loss=tf.keras.losses.BinaryCrossentropy(),
             metrics=[tf.keras.metrics.BinaryAccuracy(),
                      tf.keras.metrics.Recall(),
                      tf.keras.metrics.Precision()])
history = model.fit(train_semt_x, train_semt_y,
epochs=15, batch_size=32,
                   validation_data=(test_semt_x, test_semt_y))
model.save("segmented_model")
from tensorflow.keras.models import load_model

model = load_model("segmented_model")
test_output_semt = model.predict(test_semt_x)
print(test_output_semt.shape)
from sklearn.metrics import jaccard_score

n_test = len(test_output_semt)
jc_score3 = 0
test_output_copy = copy.deepcopy(test_output_semt)
test_output_copy[test_output_copy >= 0.5] = 1
```

```
test_output_copy[test_output_copy < 0.5] = 0
test_output_copy =
test_output_copy.astype(np.uint8) for i,k in
zip(test_semt_y,test_output_copy):
    jc_score3 += jaccard_score(i.flatten(),k.flatten())

jc_score3/=n_test
print(jc_score3)
from sklearn.metrics import jaccard_score

n_test = len(test_output_semt)
jc_score3 = 0
for i,k in zip(test_semt_y,test_output_semt):
    n_layer = i.shape[-1]
    l_jc_score = 0
    j = copy.deepcopy(k)
    j[j > 0.5] = 1
    j[j<=0.5] = 0
    for l in range(n_layer):
        l_jc_score +=
jaccard_score(i[:,:,l].flatten(),j[:,:,l].flatten(
    )) jc_score3+= (l_jc_score/n_layer)


jc_score3 /= n_test
print(jc_score3)

def draw_image(dataset,th):
    data = []
    n = len(dataset)
    dic = {0: [255,0,0], 1: [0,0,255], 2: [0,255,0], 3:
[255,255,0], 4: [255,0,255], 5: [0,255,255], 6: [255,255,255],
7: [25,50,40], 8: [0,80,90], 9: [250,120,120], 10: [0,0,0] }
for i in range(n):
        m,n,num_class = dataset[i].shape[:3]
        segment_image = np.zeros((m,n,3))
        for j in range(num_class):
            segment_image[dataset[i][:,:,j]>th] = dic[j]
            # segment_image[dataset[i][:,:,j]<=th] = dic[10]
        data.append(segment_image.astype(np.uint8))
    return data
```

```
final_semt_data = draw_image(test_output_semt,0.5)
plt.imshow(test_semt_x[20].reshape((56,56)))
plt.show()
plt.imshow(final_semt_data[20])
plt.show()
```