Computer Vision using Deep Learning

Foreground extraction, classification with circulation, semantic segmentation using MNIST dataset

Table of Content

QUESTION 1

Perform the following on MNIST dataset to build three new datasets:
1. 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.
2. 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]
3. 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]

Answers

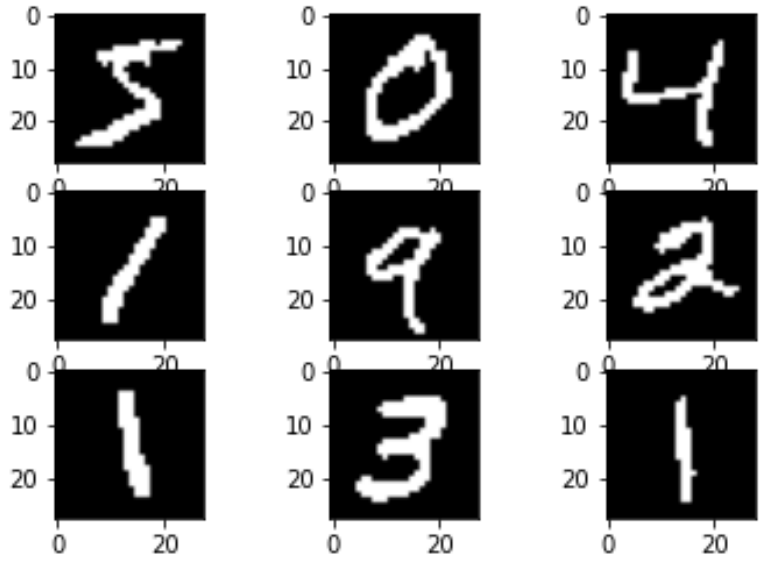
1.1

Firstly let's look at what TSS means
Total sum of squares, TSS = Summation of all (yi - mean(y))^2

TSS thresholding was applied to create the binary masks out of the 28x28 dimension grayscale images. The following are the stats of the masks created that will be used in ques 2.

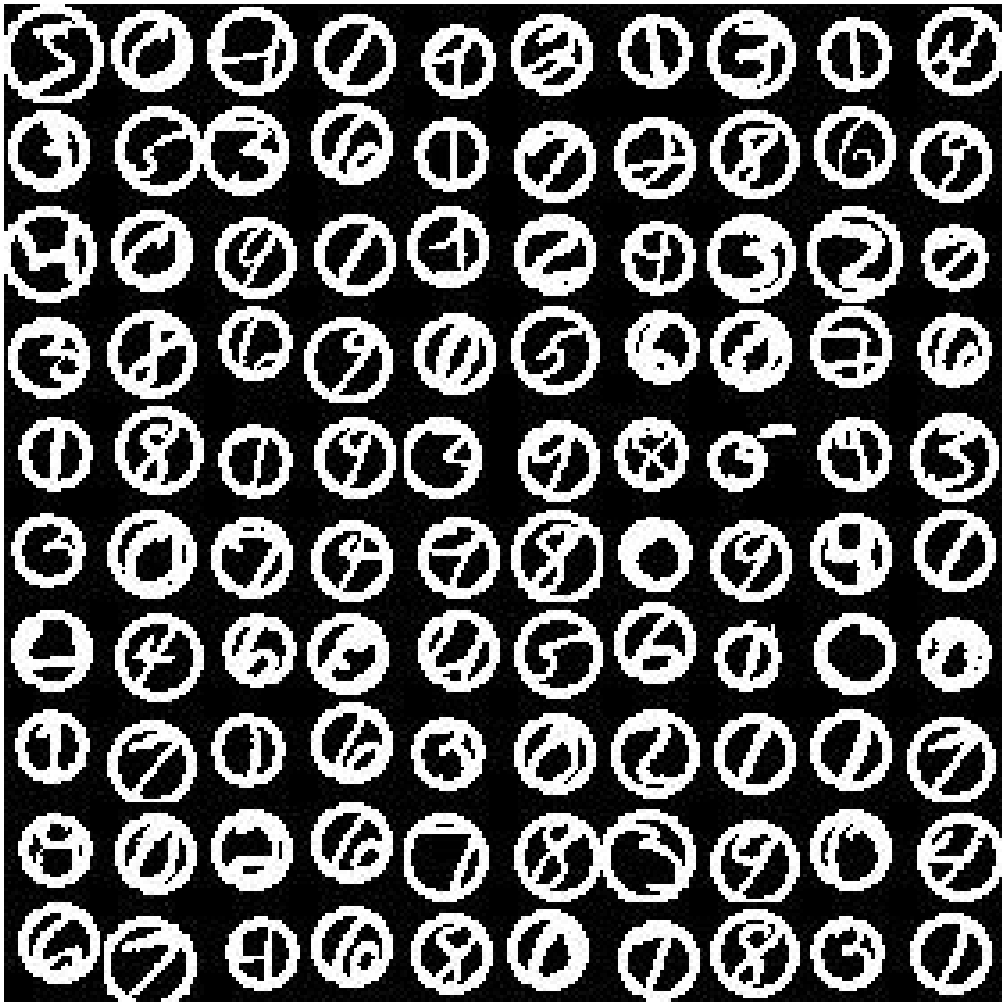
```
Showing shapes of the binary masks created
X_train Binary Masks: (60000, 28, 28)
X_test Binary Masks:  (10000, 28, 28)
```

The binary masks are displayed below. They were multiplied by 255 and then plotted. They are coming out to be the same as the input images in the mnist dataset, ensuring that the masks are formed correctly.



1.2

The minimum enclosing circle of the masks are calculated (using opencv library) and the results of center coordinates and radius are stored for each binary mask. Few results are plotted below



1.3

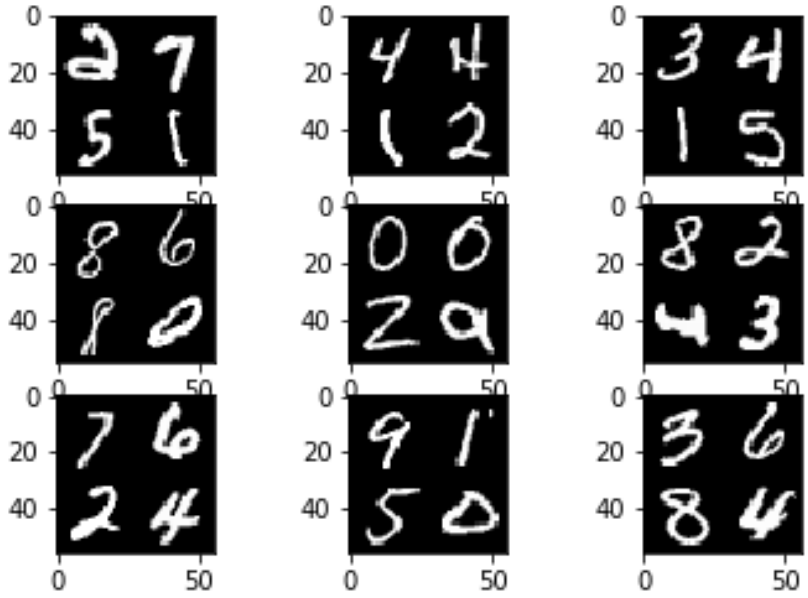
Four random images are concatenated and a new image is formed. Few results are shown below. Due to memory constraints, the size of new 4 D tensor/numpy array, which is basically the train and test data is kept to be
train_size = 5000
test_size = 1000

```
The following were the shape for the corresponding 4D array created
Training data shape =  (5000, 56, 56, 12)
Testing data shape =  (1000, 56, 56, 12)
```

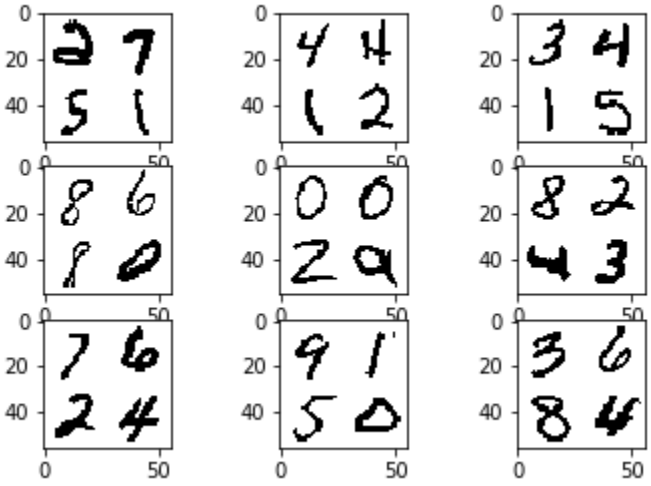
There are 12, 56x56 matrix for each entry. First layer/matrix, 1 out this 12 represents the new composite image, next 10 matrices represent binary mask corresponding to each class, 12th represented the background binary mask.

Out of this the first layer (composite image) will be used as input in ques 4 and rest of the layers as the predictions we want from the network.

Plotting the composite images formed, layer/matrix 1 out of 12.



Plotting the binary mask created for background pixels, layer/matrix 12th out of 12. White pixels highlight the background



QUESTION 2

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]

Answers

- Implemented a modified U-net model architecture for this question as it was proven and tested to be very effective in image segmentation tasks. I used binary cross entropy for the loss as it compares each of the predicted probabilities to actual class output which can be either 0 or 1.

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=["acc"])
```

- I trained the model for 5 epochs, with batch size= 16 and validation_split=0.1. At a moment, the model will train with
  Input: 28*28 grayscale image matrix from the mnist dataset
  Output: 28*28 binary mask matrix for the corresponding image

- After training the model, the model was tested on a test set of size 10000. The outputs were thresholded with value 0.5 to get a binary matrix and Jaccard similarity score (**0.9935**) was calculated for the test set using predicted and true binary masks. The following were the results

```
No of samples in test set are  10000
Final Jaccard Similarity score for the test set is
0.9934913927361305
```

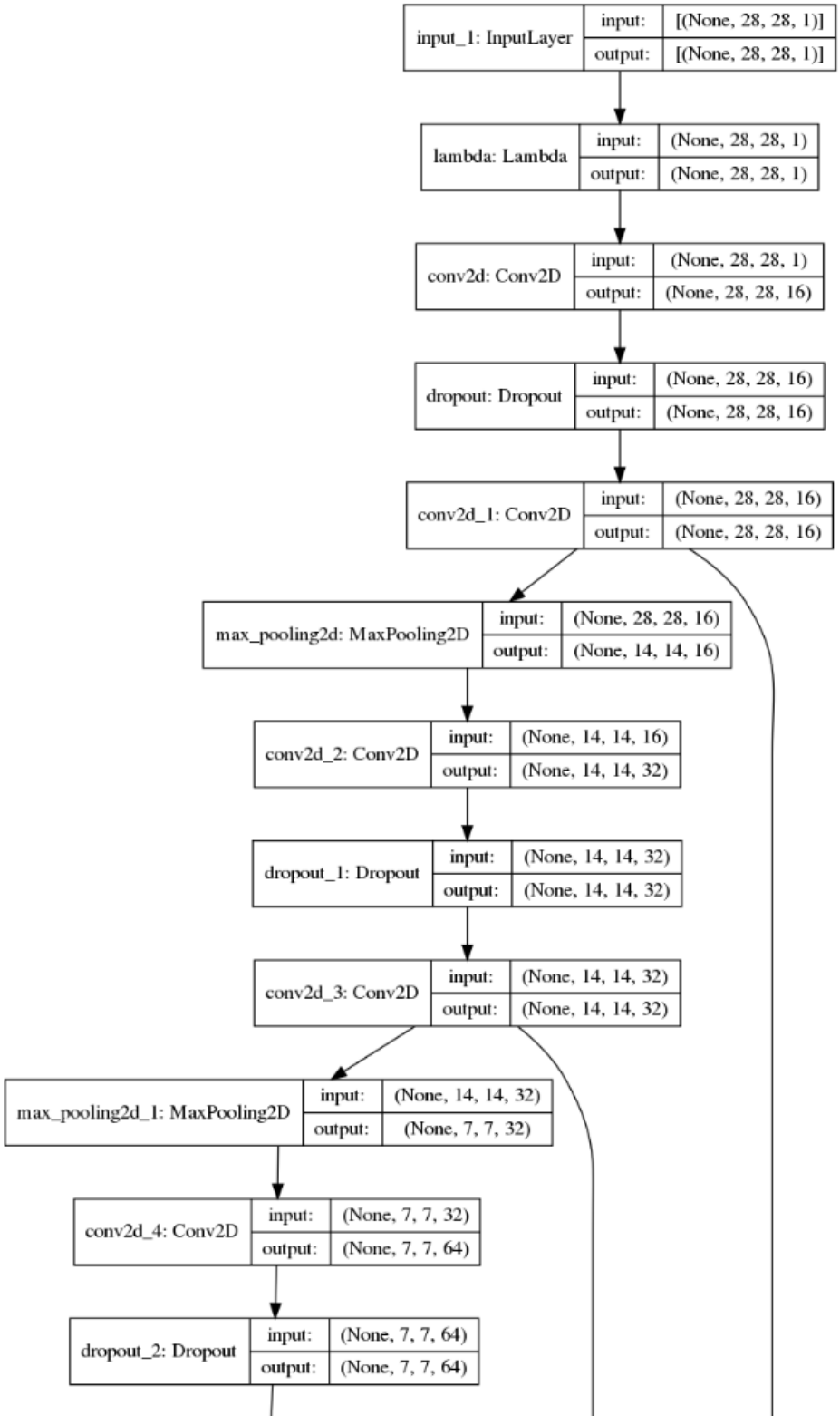
Layers of model

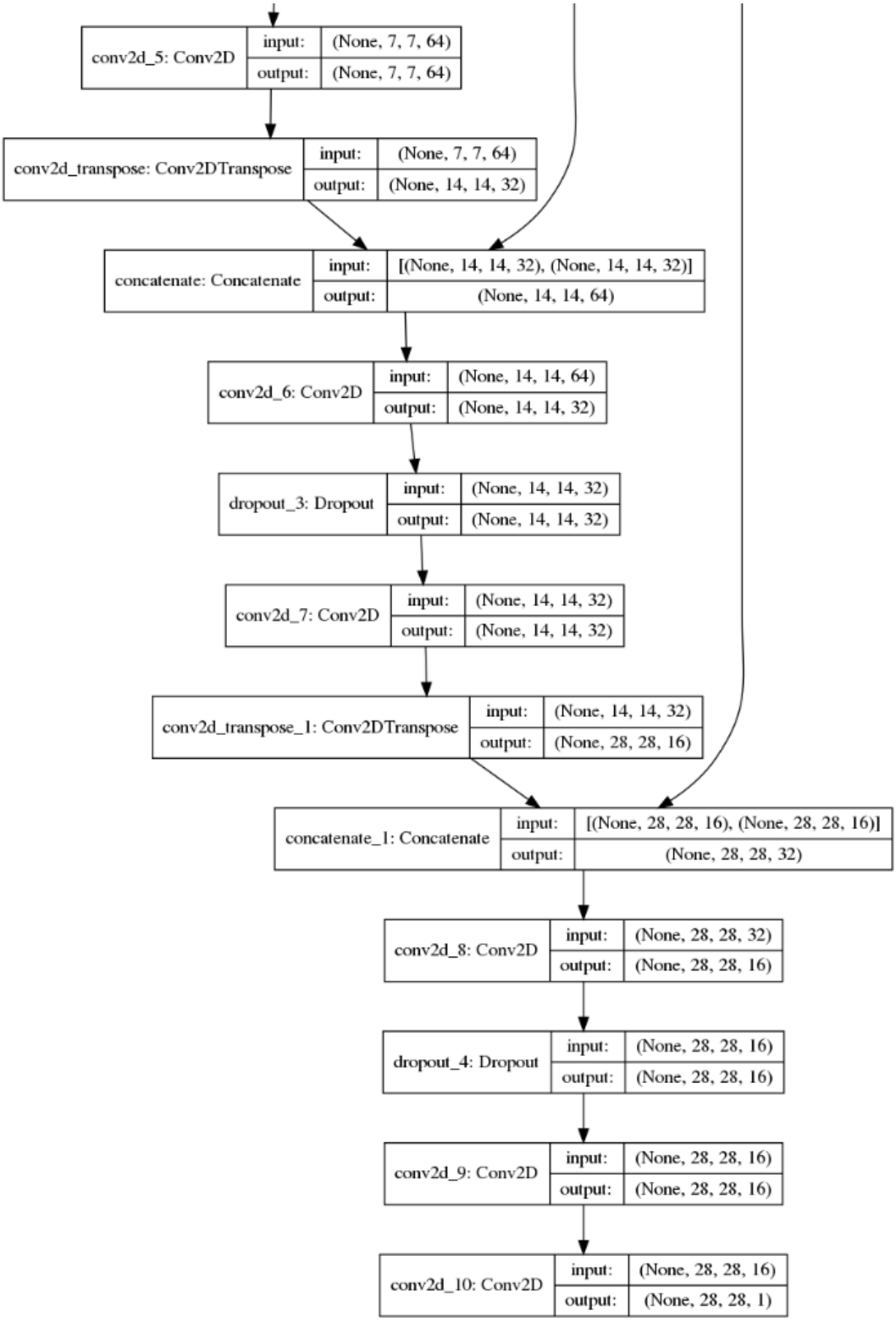
```
Layer (type)                 Output Shape         Param #     Connected to
==================================================================================================
==========
```

```
input_1 (InputLayer)            [(None, 28, 28, 1)]  0
__________________________________________________________________________________________
lambda (Lambda)                 (None, 28, 28, 1)    0           input_1[0][0]
__________________________________________________________________________________________
conv2d (Conv2D)                 (None, 28, 28, 16)   160         lambda[0][0]
__________________________________________________________________________________________
dropout (Dropout)               (None, 28, 28, 16)   0           conv2d[0][0]
__________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 28, 28, 16)   2320        dropout[0][0]
__________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 14, 14, 16)   0           conv2d_1[0][0]
__________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 14, 14, 32)   4640        max_pooling2d[0][0]
__________________________________________________________________________________________
dropout_1 (Dropout)             (None, 14, 14, 32)   0           conv2d_2[0][0]
__________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 14, 14, 32)   9248        dropout_1[0][0]
__________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 7, 7, 32)     0           conv2d_3[0][0]
__________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 7, 7, 64)     18496       max_pooling2d_1[0][0]
__________________________________________________________________________________________
dropout_2 (Dropout)             (None, 7, 7, 64)     0           conv2d_4[0][0]
__________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 7, 7, 64)     36928       dropout_2[0][0]
__________________________________________________________________________________________
conv2d_transpose (Conv2DTranspo (None, 14, 14, 32)   8224        conv2d_5[0][0]
__________________________________________________________________________________________
concatenate (Concatenate)       (None, 14, 14, 64)   0           conv2d_transpose[0][0]
                                                                 conv2d_3[0][0]
__________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 14, 14, 32)   18464       concatenate[0][0]
__________________________________________________________________________________________
```

```
dropout_3 (Dropout)             (None, 14, 14, 32)   0           conv2d_6[0][0]
__________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 14, 14, 32)   9248        dropout_3[0][0]
__________________________________________________________________________________________
conv2d_transpose_1 (Conv2DTrans (None, 28, 28, 16)   2064        conv2d_7[0][0]
__________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 28, 28, 32)   0
conv2d_transpose_1[0][0]
                                                                 conv2d_1[0][0]
__________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 28, 28, 16)   4624        concatenate_1[0][0]
__________________________________________________________________________________________
dropout_4 (Dropout)             (None, 28, 28, 16)   0           conv2d_8[0][0]
__________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 28, 28, 16)   2320        dropout_4[0][0]
__________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 28, 28, 1)    17          conv2d_9[0][0]
==========================================================================================
Total params: 116,753
Trainable params: 116,753
Non-trainable params: 0
```

Visualising the model

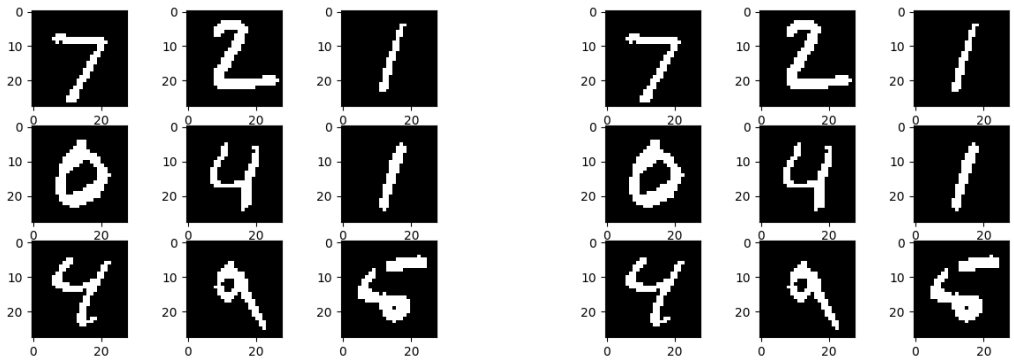

input_1: InputLayer
input: [(None, 28, 28, 1)]
output: [(None, 28, 28, 1)]
lambda: Lambda
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
conv2d: Conv2D
input: (None, 28, 28, 1)
output: (None, 28, 28, 16)
dropout: Dropout
input: (None, 28, 28, 16)
output: (None, 28, 28, 16)
conv2d_1: Conv2D
input: (None, 28, 28, 16)
output: (None, 28, 28, 16)
max_pooling2d: MaxPooling2D
input: (None, 28, 28, 16)
output: (None, 14, 14, 16)
conv2d_2: Conv2D
input: (None, 14, 14, 16)
output: (None, 14, 14, 32)
dropout_1: Dropout
input: (None, 14, 14, 32)
output: (None, 14, 14, 32)
conv2d_3: Conv2D
input: (None, 14, 14, 32)
output: (None, 14, 14, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 14, 14, 32)
output: (None, 7, 7, 32)
conv2d_4: Conv2D
input: (None, 7, 7, 32)
output: (None, 7, 7, 64)
dropout_2: Dropout
input: (None, 7, 7, 64)
output: (None, 7, 7, 64)



Output from Training (0.99 val acc)

```
Epoch 1/10
3375/3375 [==============================] - 201s 59ms/step - loss: 0.0202 - acc: 0.9915 - val_loss: 0.0029 - val_a
cc: 0.9990

Epoch 00001: val_loss improved from inf to 0.00293, saving model to modelq2.h5
Epoch 2/10
3375/3375 [==============================] - 198s 59ms/step - loss: 0.0048 - acc: 0.9979 - val_loss: 0.0027 - val_a
cc: 0.9988

Epoch 00002: val_loss improved from 0.00293 to 0.00271, saving model to modelq2.h5
Epoch 3/10
3375/3375 [==============================] - 202s 60ms/step - loss: 0.0038 - acc: 0.9984 - val_loss: 0.0023 - val_a
cc: 0.9990

Epoch 00003: val_loss improved from 0.00271 to 0.00232, saving model to modelq2.h5
Epoch 4/10
3375/3375 [==============================] - 201s 59ms/step - loss: 0.0031 - acc: 0.9987 - val_loss: 0.0024 - val_a
cc: 0.9990

Epoch 00004: val_loss did not improve from 0.00232
Epoch 5/10
3375/3375 [==============================] - 212s 63ms/step - loss: 0.0027 - acc: 0.9989 - val_loss: 0.0021 - val_a
cc: 0.9991
```

Plots of results predicted (few pred and true binary images from test set)



QUESTION 3

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]
Note: If the classification is already wrong, the Jaccard Similarity score will become zero.

Answers

- I implemented a deep learning model with convolution layers, few max pooling layers, average pooling layer (before branching out for results) and dense layers. The output is branched out in two ways.
    a. One predicts the label of the image (0-9). As we know that in our dataset the label exists and no image is blank, we are not predicting whether the image is blank or not. The size of classification output of the last layer is taken as 10 (for labels 0 to 9) and uses 'softmax' as the activation function.
    b. Other branch works for regression and predicts a continuous value between 0 to 1 (as data regarding center and radius is normalised). The last layer of this branch uses 'sigmoid' for activation and is of size 3.

    Losses and loss weights given to the model are shown below

```
model = Model(inputs=[inputs], outputs=[classifier_head, reg_head])

losses = {'label': 'categorical_crossentropy', 'bbox': 'mse'}

loss_weights = {'label': 1.0, 'bbox': 1.0}

model.compile('adam', loss=losses, loss_weights=loss_weights,
metrics=['acc','mse'])
```

- I trained the model for 15 epochs with validation_split=0.1 and batch_size=16. At a moment, the model will train with
  Input: 28*28 grayscale image matrix from the mnist dataset

Output: a list of 2 arrays, one of size 10 for classification having 1 for at the place of label index, and other of size 3 for center and radius normalised values

- After training the model, the model was tested on a test set of size 10000. The masks were created in case of right predictions and the jaccard similarity score was calculated.

Jaccard Similarity (**0.71**)

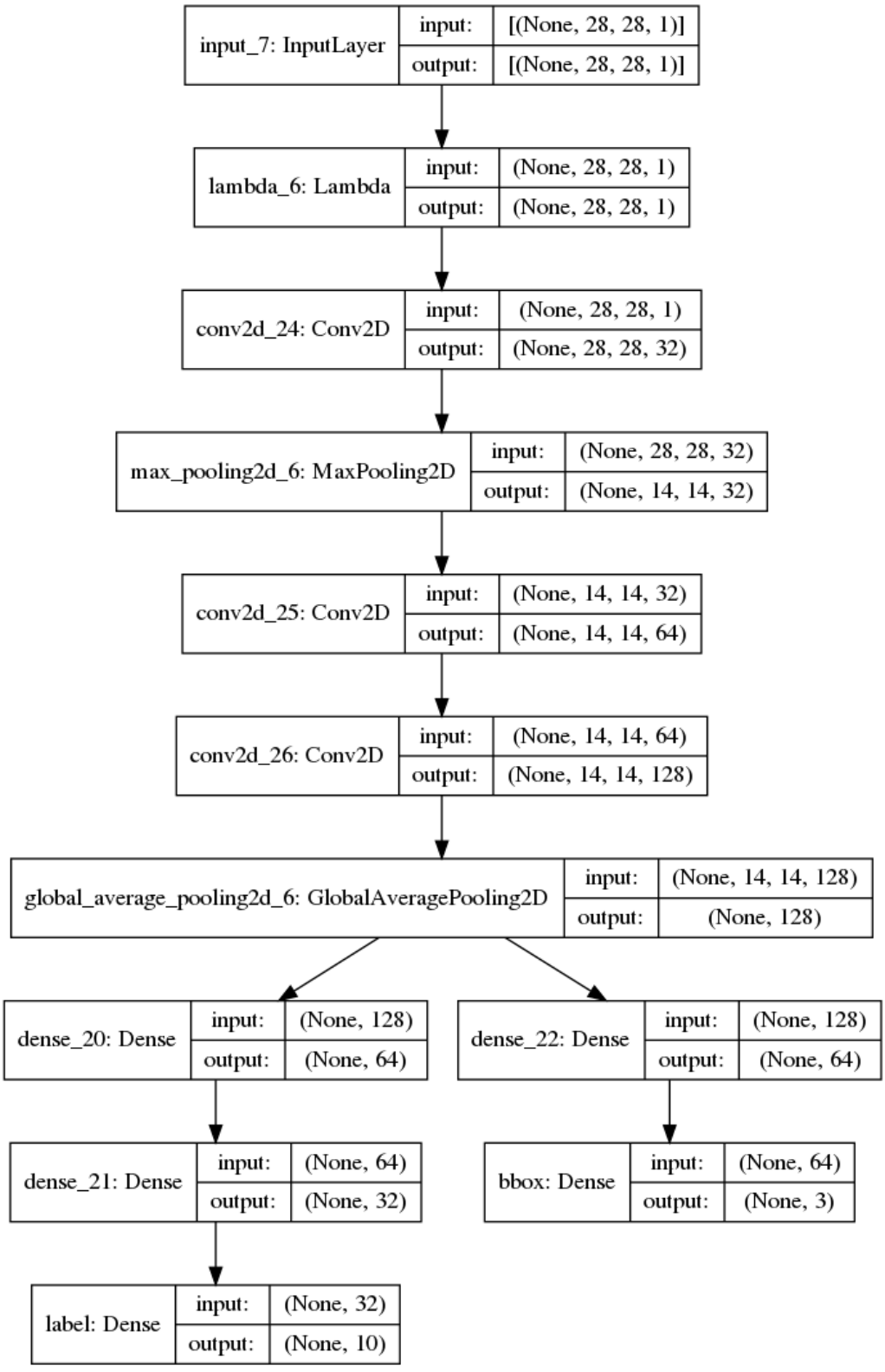
```
Printing the result....
The no of samples in test set are  10000
No of right classifications =  9896
No of wrong classifications =  104
Jaccard score is  0.7141385316421728
```

Layers of model

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================
==============
input_7 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________________________
_______________
lambda_6 (Lambda)            (None, 28, 28, 1)         0           input_7[0][0]
_________________________________________________________________________________
_______________
conv2d_24 (Conv2D)           (None, 28, 28, 32)        320         lambda_6[0][0]
_________________________________________________________________________________
_______________
max_pooling2d_6 (MaxPooling2D)  (None, 14, 14, 32)     0           conv2d_24[0][0]
_________________________________________________________________________________
_______________
conv2d_25 (Conv2D)           (None, 14, 14, 64)        18496
max_pooling2d_6[0][0]
_________________________________________________________________________________
_______________
conv2d_26 (Conv2D)           (None, 14, 14, 128)       32896       conv2d_25[0][0]
_________________________________________________________________________________
_______________
global_average_pooling2d_6 (Glo (None, 128)            0           conv2d_26[0][0]
_________________________________________________________________________________
```

```
_________________
dense_20 (Dense)                (None, 64)           8256
global_average_pooling2d_6[0][0]
__________________________________________________________________________________________
_________________
dense_21 (Dense)                (None, 32)           2080        dense_20[0][0]
__________________________________________________________________________________________
_________________
dense_22 (Dense)                (None, 64)           8256
global_average_pooling2d_6[0][0]
__________________________________________________________________________________________
_________________
label (Dense)                   (None, 10)           330         dense_21[0][0]
__________________________________________________________________________________________
_________________
bbox (Dense)                    (None, 3)            195         dense_22[0][0]
==========================================================================================
===============
Total params: 70,829
Trainable params: 70,829
Non-trainable params: 0
```

Visualising the model



Output from Training (val_label_acc: 0.9912, val_bbox_mse: 0.0023)

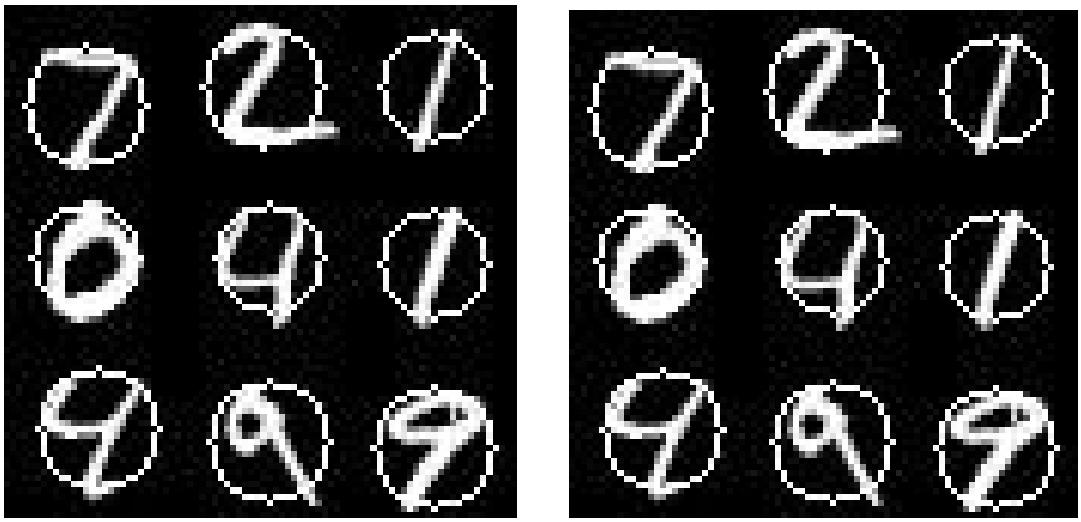
```
Epoch 1/15
3375/3375 [==============================] - 60s 18ms/step - loss: 0.9941 - label_loss: 0.9910 - bbox_loss: 0.0031
- label_acc: 0.6444 - label_mse: 0.0427 - bbox_acc: 0.5855 - bbox_mse: 0.0031 - val_loss: 0.1376 - val_label_loss:
0.1346 - val_bbox_loss: 0.0030 - val_label_acc: 0.9600 - val_label_mse: 0.0060 - val_bbox_acc: 0.7125 - val_bbox_ms
e: 0.0030

Epoch 00001: val_loss improved from inf to 0.13758, saving model to modelq3.h5
Epoch 2/15
3375/3375 [==============================] - 63s 19ms/step - loss: 0.1601 - label_loss: 0.1577 - bbox_loss: 0.0024
- label_acc: 0.9511 - label_mse: 0.0075 - bbox_acc: 0.7080 - bbox_mse: 0.0024 - val_loss: 0.0756 - val_label_loss:
0.0727 - val_bbox_loss: 0.0029 - val_label_acc: 0.9777 - val_label_mse: 0.0033 - val_bbox_acc: 0.7170 - val_bbox_ms
e: 0.0029

Epoch 00002: val_loss improved from 0.13758 to 0.07559, saving model to modelq3.h5
Epoch 3/15
3375/3375 [==============================] - 64s 19ms/step - loss: 0.1104 - label_loss: 0.1080 - bbox_loss: 0.0023
- label_acc: 0.9669 - label_mse: 0.0051 - bbox_acc: 0.7324 - bbox_mse: 0.0023 - val_loss: 0.0817 - val_label_loss:
0.0789 - val_bbox_loss: 0.0029 - val_label_acc: 0.9787 - val_label_mse: 0.0033 - val_bbox_acc: 0.6733 - val_bbox_ms
e: 0.0029

Epoch 00003: val_loss did not improve from 0.07559
Epoch 4/15
3375/3375 [==============================] - 64s 19ms/step - loss: 0.0814 - label_loss: 0.0792 - bbox_loss: 0.0022
- label_acc: 0.9756 - label_mse: 0.0037 - bbox_acc: 0.7447 - bbox_mse: 0.0022 - val_loss: 0.0492 - val_label_loss:
0.0464 - val_bbox_loss: 0.0027 - val_label_acc: 0.9862 - val_label_mse: 0.0021 - val_bbox_acc: 0.6768 - val_bbox_ms
e: 0.0027

Epoch 00004: val_loss improved from 0.07559 to 0.04919, saving model to modelq3.h5
Epoch 5/15
3375/3375 [==============================] - 64s 19ms/step - loss: 0.0645 - label_loss: 0.0625 - bbox_loss: 0.0020
- label_acc: 0.9806 - label_mse: 0.0030 - bbox_acc: 0.7473 - bbox_mse: 0.0020 - val_loss: 0.0540 - val_label_loss:
0.0514 - val_bbox_loss: 0.0026 - val_label_acc: 0.9853 - val_label_mse: 0.0023 - val_bbox_acc: 0.7698 - val_bbox_ms
e: 0.0026
```

Jumping to last 5 epochs

```
Epoch 00010: val_loss improved from 0.04891 to 0.04216, saving model to modelq3.h5
Epoch 11/15
3375/3375 [==============================] - 65s 19ms/step - loss: 0.0299 - label_loss: 0.0281 - bbox_loss: 0.0018
- label_acc: 0.9912 - label_mse: 0.0014 - bbox_acc: 0.7716 - bbox_mse: 0.0018 - val_loss: 0.0350 - val_label_loss:
0.0327 - val_bbox_loss: 0.0023 - val_label_acc: 0.9907 - val_label_mse: 0.0015 - val_bbox_acc: 0.7693 - val_bbox_ms
e: 0.0023

Epoch 00011: val_loss improved from 0.04216 to 0.03496, saving model to modelq3.h5
Epoch 12/15
3375/3375 [==============================] - 66s 19ms/step - loss: 0.0273 - label_loss: 0.0255 - bbox_loss: 0.0017
- label_acc: 0.9917 - label_mse: 0.0013 - bbox_acc: 0.7809 - bbox_mse: 0.0017 - val_loss: 0.0373 - val_label_loss:
0.0350 - val_bbox_loss: 0.0022 - val_label_acc: 0.9900 - val_label_mse: 0.0016 - val_bbox_acc: 0.7802 - val_bbox_ms
e: 0.0022

Epoch 00012: val_loss did not improve from 0.03496
Epoch 13/15
3375/3375 [==============================] - 64s 19ms/step - loss: 0.0245 - label_loss: 0.0227 - bbox_loss: 0.0018
- label_acc: 0.9927 - label_mse: 0.0011 - bbox_acc: 0.7824 - bbox_mse: 0.0018 - val_loss: 0.0352 - val_label_loss:
0.0329 - val_bbox_loss: 0.0023 - val_label_acc: 0.9913 - val_label_mse: 0.0013 - val_bbox_acc: 0.7768 - val_bbox_ms
e: 0.0023

Epoch 00013: val_loss did not improve from 0.03496
Epoch 14/15
3375/3375 [==============================] - 65s 19ms/step - loss: 0.0221 - label_loss: 0.0204 - bbox_loss: 0.0017
- label_acc: 0.9932 - label_mse: 0.0010 - bbox_acc: 0.7841 - bbox_mse: 0.0017 - val_loss: 0.0411 - val_label_loss:
0.0387 - val_bbox_loss: 0.0023 - val_label_acc: 0.9892 - val_label_mse: 0.0017 - val_bbox_acc: 0.7818 - val_bbox_ms
e: 0.0023

Epoch 00014: val_loss did not improve from 0.03496
Epoch 15/15
3375/3375 [==============================] - 65s 19ms/step - loss: 0.0206 - label_loss: 0.0189 - bbox_loss: 0.0017
- label_acc: 0.9940 - label_mse: 9.0381e-04 - bbox_acc: 0.7806 - bbox_mse: 0.0017 - val_loss: 0.0389 - val_label_lo
ss: 0.0365 - val_bbox_loss: 0.0023 - val_label_acc: 0.9912 - val_label_mse: 0.0014 - val_bbox_acc: 0.7948 - val_bbo
x_mse: 0.0023
```

Plots of results predicted (few true and pred min enclosing circle for images from test set)

Truth Values Predicted



Stats printed on Terminal for above plots

```
Q3TrueValues
In plot:Sample  0  has x, y, r are   12 15 10
In plot:Sample  1  has x, y, r are   16 12 11
In plot:Sample  2  has x, y, r are   14 13 10
In plot:Sample  3  has x, y, r are   13 13 9
In plot:Sample  4  has x, y, r are   14 14 9
In plot:Sample  5  has x, y, r are   14 14 9
In plot:Sample  6  has x, y, r are   15 13 10
In plot:Sample  7  has x, y, r are   17 15 10
In plot:Sample  8  has x, y, r are   15 17 10


Q3Predictions
In plot:Sample  0  has x, y, r are   13 16 10
In plot:Sample  1  has x, y, r are   14 13 10
In plot:Sample  2  has x, y, r are   14 13 9
In plot:Sample  3  has x, y, r are   13 13 9
In plot:Sample  4  has x, y, r are   15 13 9
In plot:Sample  5  has x, y, r are   14 14 9
In plot:Sample  6  has x, y, r are   15 13 10
In plot:Sample  7  has x, y, r are   15 15 10
In plot:Sample  8  has x, y, r are   14 16 10
```

QUESTION 4

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]

Answer

- Implemented a modified U-net model architecture for this question as it was proven and tested to be very effective in image segmentation tasks. I used binary cross entropy for the loss as it compares each of the predicted probabilities to actual class output which can be either 0 or 1.

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=["acc"])
```

- I trained the model for 10 epochs, with batch size= 16 and validation_split=0.1. At a moment, the model will train with
  Input: 56*56 grayscale composite image matrix (formed from 4 images)
  Output: 56*56*11 binary masks (10 for labels and 1 for) matrix for the corresponding image

- After training the model, the model was tested on a test set of size 1000. The model gave a 56*56*11 matrix for each image with values between 0 to 1. For a single cell, let's say i,j in 56*56, the label was taken as 1 for which the value was maximum in the 11 channels/layers for the cell i,j. Using this binary masks of size 56*56*11 was created which were used for Jaccard similarity (**0.97**). The following were the results

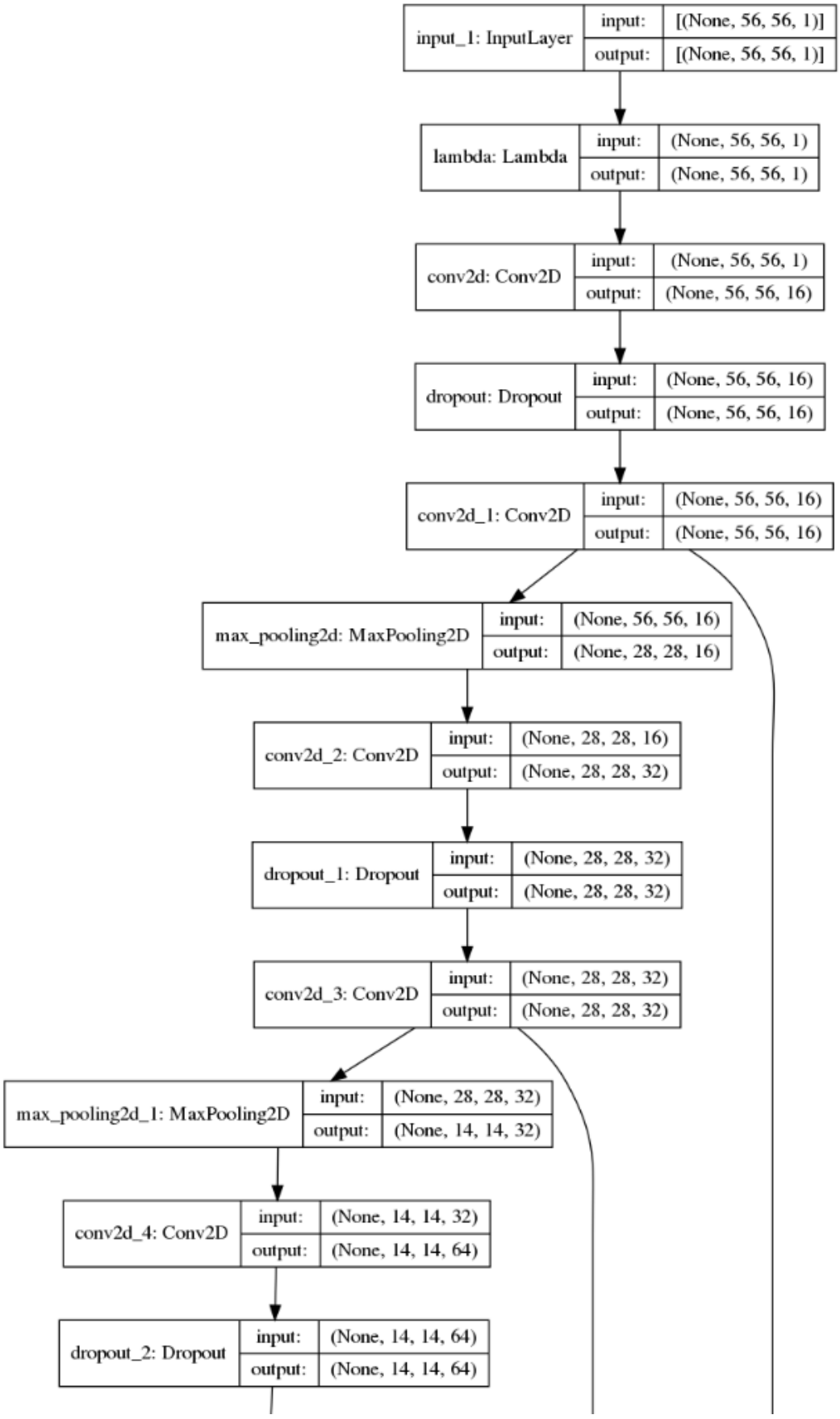
```
Shape of predicted test samples are  (1000, 56, 56, 11)
No of samples in test set are  1000
Jaccard index is coming out to be  0.9712379303272716
```

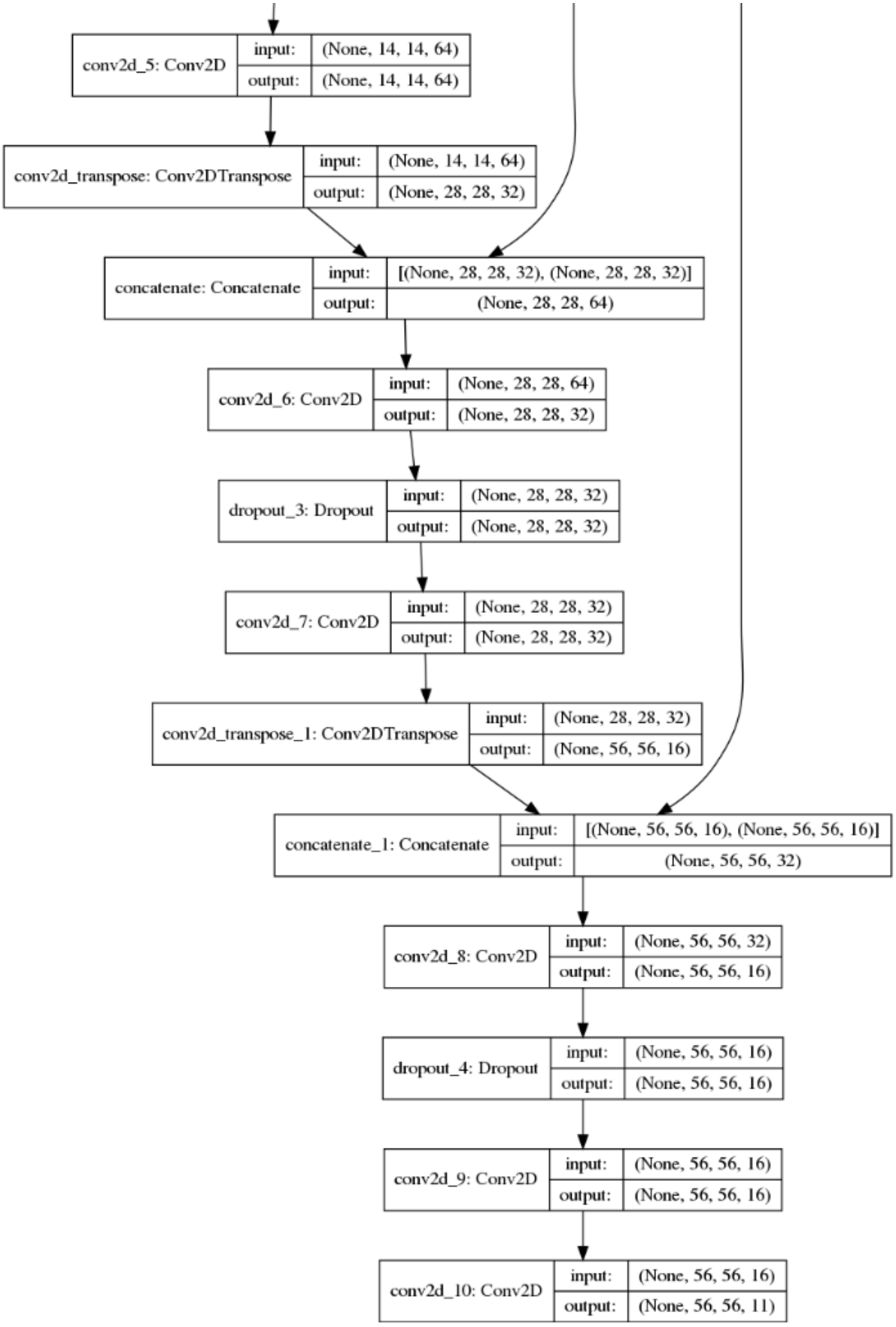
Layers of model

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 56, 56, 1)]       0
_________________________________________________________________________________________________
lambda (Lambda)              (None, 56, 56, 1)         0         input_1[0][0]
_________________________________________________________________________________________________
conv2d (Conv2D)              (None, 56, 56, 16)        160       lambda[0][0]
_________________________________________________________________________________________________
dropout (Dropout)            (None, 56, 56, 16)        0         conv2d[0][0]
_________________________________________________________________________________________________
conv2d_1 (Conv2D)            (None, 56, 56, 16)        2320      dropout[0][0]
_________________________________________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 28, 28, 16)        0         conv2d_1[0][0]
_________________________________________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 32)        4640
max_pooling2d[0][0]
_________________________________________________________________________________________________
dropout_1 (Dropout)          (None, 28, 28, 32)        0         conv2d_2[0][0]
_________________________________________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 32)        9248      dropout_1[0][0]
_________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 14, 14, 32)      0         conv2d_3[0][0]
_________________________________________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        18496
max_pooling2d_1[0][0]
_________________________________________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 64)        0         conv2d_4[0][0]
_________________________________________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        36928     dropout_2[0][0]
_________________________________________________________________________________________________
conv2d_transpose (Conv2DTranspo (None, 28, 28, 32)     8224      conv2d_5[0][0]
_________________________________________________________________________________________________
```

```
concatenate (Concatenate)       (None, 28, 28, 64)   0
conv2d_transpose[0][0]
                                                                 conv2d_3[0][0]
__________________________________________________________________________________________
_____________
conv2d_6 (Conv2D)               (None, 28, 28, 32)   18464       concatenate[0][0]
__________________________________________________________________________________________
_____________
dropout_3 (Dropout)             (None, 28, 28, 32)   0           conv2d_6[0][0]
__________________________________________________________________________________________
_____________
conv2d_7 (Conv2D)               (None, 28, 28, 32)   9248        dropout_3[0][0]
__________________________________________________________________________________________
_____________
conv2d_transpose_1 (Conv2DTrans (None, 56, 56, 16)   2064        conv2d_7[0][0]
__________________________________________________________________________________________
_____________
concatenate_1 (Concatenate)     (None, 56, 56, 32)   0
conv2d_transpose_1[0][0]
                                                                 conv2d_1[0][0]
__________________________________________________________________________________________
_____________
conv2d_8 (Conv2D)               (None, 56, 56, 16)   4624
concatenate_1[0][0]
__________________________________________________________________________________________
_____________
dropout_4 (Dropout)             (None, 56, 56, 16)   0           conv2d_8[0][0]
__________________________________________________________________________________________
_____________
conv2d_9 (Conv2D)               (None, 56, 56, 16)   2320        dropout_4[0][0]
__________________________________________________________________________________________
_____________
conv2d_10 (Conv2D)              (None, 56, 56, 11)   187         conv2d_9[0][0]
==========================================================================================
==============
Total params: 116,923
Trainable params: 116,923
Non-trainable params: 0
```

Visualising the model

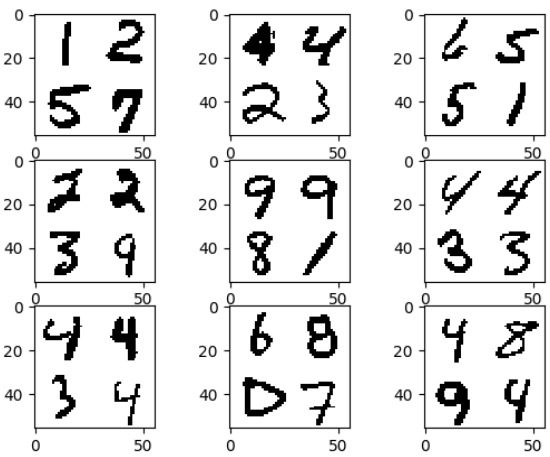
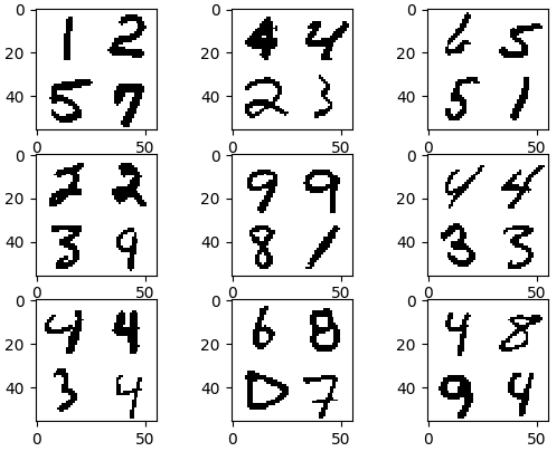

input_1: InputLayer
input: [(None, 56, 56, 1)]
output: [(None, 56, 56, 1)]
lambda: Lambda
input: (None, 56, 56, 1)
output: (None, 56, 56, 1)
conv2d: Conv2D
input: (None, 56, 56, 1)
output: (None, 56, 56, 16)
dropout: Dropout
input: (None, 56, 56, 16)
output: (None, 56, 56, 16)
conv2d_1: Conv2D
input: (None, 56, 56, 16)
output: (None, 56, 56, 16)
max_pooling2d: MaxPooling2D
input: (None, 56, 56, 16)
output: (None, 28, 28, 16)
conv2d_2: Conv2D
input: (None, 28, 28, 16)
output: (None, 28, 28, 32)
dropout_1: Dropout
input: (None, 28, 28, 32)
output: (None, 28, 28, 32)
conv2d_3: Conv2D
input: (None, 28, 28, 32)
output: (None, 28, 28, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 28, 28, 32)
output: (None, 14, 14, 32)
conv2d_4: Conv2D
input: (None, 14, 14, 32)
output: (None, 14, 14, 64)
dropout_2: Dropout
input: (None, 14, 14, 64)
output: (None, 14, 14, 64)


conv2d_5: Conv2D
input: (None, 14, 14, 64)
output: (None, 14, 14, 64)
conv2d_transpose: Conv2DTranspose
input: (None, 14, 14, 64)
output: (None, 28, 28, 32)
concatenate: Concatenate
input: [(None, 28, 28, 32), (None, 28, 28, 32)]
output: (None, 28, 28, 64)
conv2d_6: Conv2D
input: (None, 28, 28, 64)
output: (None, 28, 28, 32)
dropout_3: Dropout
input: (None, 28, 28, 32)
output: (None, 28, 28, 32)
conv2d_7: Conv2D
input: (None, 28, 28, 32)
output: (None, 28, 28, 32)
conv2d_transpose_1: Conv2DTranspose
input: (None, 28, 28, 32)
output: (None, 56, 56, 16)
concatenate_1: Concatenate
input: [(None, 56, 56, 16), (None, 56, 56, 16)]
output: (None, 56, 56, 32)
conv2d_8: Conv2D
input: (None, 56, 56, 32)
output: (None, 56, 56, 16)
dropout_4: Dropout
input: (None, 56, 56, 16)
output: (None, 56, 56, 16)
conv2d_9: Conv2D
input: (None, 56, 56, 16)
output: (None, 56, 56, 16)
conv2d_10: Conv2D
input: (None, 56, 56, 16)
output: (None, 56, 56, 11)

Output from training (val acc is 0.99)

```
Epoch 1/10
282/282 [==============================] - 69s 239ms/step - loss: 0.1514 - acc: 0.7902 - val_loss: 0.0443 - val_ac
c: 0.8827

Epoch 00001: val_loss improved from inf to 0.04429, saving model to modelq4.h5
Epoch 2/10
282/282 [==============================] - 62s 220ms/step - loss: 0.0433 - acc: 0.8857 - val_loss: 0.0341 - val_ac
c: 0.9110

Epoch 00002: val_loss improved from 0.04429 to 0.03408, saving model to modelq4.h5
Epoch 3/10
282/282 [==============================] - 63s 223ms/step - loss: 0.0333 - acc: 0.9155 - val_loss: 0.0234 - val_ac
c: 0.9524

Epoch 00003: val_loss improved from 0.03408 to 0.02335, saving model to modelq4.h5
Epoch 4/10
282/282 [==============================] - 63s 222ms/step - loss: 0.0230 - acc: 0.9522 - val_loss: 0.0115 - val_ac
c: 0.9838

Epoch 00004: val_loss improved from 0.02335 to 0.01152, saving model to modelq4.h5
Epoch 5/10
282/282 [==============================] - 63s 223ms/step - loss: 0.0142 - acc: 0.9765 - val_loss: 0.0073 - val_ac
c: 0.9893

Epoch 00005: val_loss improved from 0.01152 to 0.00726, saving model to modelq4.h5
Epoch 6/10
282/282 [==============================] - 63s 224ms/step - loss: 0.0095 - acc: 0.9844 - val_loss: 0.0050 - val_ac
c: 0.9921

Epoch 00006: val_loss improved from 0.00726 to 0.00500, saving model to modelq4.h5
Epoch 7/10
282/282 [==============================] - 62s 221ms/step - loss: 0.0069 - acc: 0.9882 - val_loss: 0.0043 - val_ac
c: 0.9930

Epoch 00007: val_loss improved from 0.00500 to 0.00430, saving model to modelq4.h5
Epoch 8/10
282/282 [==============================] - 62s 221ms/step - loss: 0.0055 - acc: 0.9906 - val_loss: 0.0042 - val_ac
c: 0.9931

Epoch 00008: val_loss improved from 0.00430 to 0.00415, saving model to modelq4.h5
Epoch 9/10
282/282 [==============================] - 62s 220ms/step - loss: 0.0050 - acc: 0.9913 - val_loss: 0.0030 - val_ac
c: 0.9952

Epoch 00009: val_loss improved from 0.00415 to 0.00304, saving model to modelq4.h5
Epoch 10/10
282/282 [==============================] - 62s 220ms/step - loss: 0.0041 - acc: 0.9928 - val_loss: 0.0033 - val_ac
c: 0.9946
```

Plots of results predicted (few pred and true binary background images from test set (last mat/layer of 11 layers given as output by the model))

Codes

Code for ques 1

```
#!/usr/bin/env python
# coding: utf-8

# # Data Preparation

# In[1]:


from keras.datasets import mnist
from matplotlib import pyplot
import numpy as np
import cv2
import random


# Functions

# In[6]:


# for part 1

def plotFewSamples(samples):
    for i in range(9):
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(samples[i], cmap=pyplot.get_cmap('gray'))
    pyplot.show()

def plotFewBinarySamples(samples,filename):
    for i in range(9):
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(setMatRange(samples[i]),
cmap=pyplot.get_cmap('gray'))
    pyplot.savefig(filename)
```

```
    pyplot.show()

def plotFewCompositeSamples(samples,filename,index):
    for i in range(9):
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(setMatRange(samples[i,:,:,index]),
cmap=pyplot.get_cmap('gray'))
    pyplot.savefig(filename)
    pyplot.show()

def plotImagesWithCircles(samples,no_of_samples,no_in_row):
    images = []
    img = []
    color = (255,0,0)
    for i in range(no_of_samples):
        tempimg = samples[i,:,:]
        drawing = setMatRange(tempimg)
        tempimg = np.array(tempimg).astype(np.uint8)
        drawing = np.array(drawing).astype(np.uint8)

        _, contours, hierarchy = cv2.findContours(tempimg,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        (x,y), r = cv2.minEnclosingCircle(contours[0])
#         print('Len of contours = ',len(contours))
#         if len(contours)>1:
#             for c in contours:
#                 print(len(c))
        contours_poly = cv2.approxPolyDP(contours[0], 3, True)
        cv2.drawContours(drawing, contours_poly, 0, color)
        cv2.circle(drawing, (int(x), int(y)), int(r), color, 2)

        if(img == []):
            img = drawing
        else :
            img = np.hstack((img, drawing))

        if (img.shape[1] /28 == no_in_row):
            if(images == []):
                images = img.copy()
                img = []
            else:
                images = np.vstack((images, img))
                img = []
```

```
    cv2.imwrite('q3_data.jpg', images)
    cv2.imshow('q3_data', images)
    cv2.waitKey()
    cv2.destroyAllWindows()

def applyTSS(mat):
    #       print('In TSS image, ',mat.shape)

    MIN_PIXEL_SUM = 10000000000
    OTSU_THRESH = -1
    unique_values = np.unique(mat)

    for i in range(len(unique_values)):
        temp_thresh = unique_values[i]
        pos1 = mat>=temp_thresh
        pos2 = mat<temp_thresh

        l1 = np.array([])
        l2 = np.array([])

        if pos1.any():
            l1 = np.array(mat[pos1]).ravel()
        if pos2.any():
            l2 = np.array(mat[pos2]).ravel()

        val=1000000

        if len(l1)!=0 and len(l2)!=0:
            val = np.var(l1)*l1.shape[0] + np.var(l2)*l2.shape[0]
        elif len(l1)!=0:
            val = np.var(l1)*l1.shape[0]
        else:
            val = np.var(l2)*l2.shape[0]

#         if len(l1)!=0 and len(l2)!=0:
#             val = np.var(l1)*l1.shape[0] + np.var(l2)*l2.shape[0]
#         elif len(l1)!=0:
#             val = np.var(l1)*l1.shape[0]
#         else:
#             val = np.var(l2)*l2.shape[0]
        # print(val,MIN_PIXEL_SUM)

        if(val<MIN_PIXEL_SUM):
```

```
            OTSU_THRESH=temp_thresh
            MIN_PIXEL_SUM =val

    binary_mask = np.zeros(mat.shape)
    binary_mask[mat>=OTSU_THRESH] = 1
    return binary_mask

def computeBinaryMasks(samples):
    binary_masks = np.zeros(samples.shape)
    for i in range(samples.shape[0]):
        binary_masks[i,:,:] = applyTSS(samples[i,:,:])
    return binary_masks

def setMatRange(mat):
    max_value = np.max(mat)
    mat = mat / max_value * 255
    return mat

def saveNumpyArray(a,filename):
    np.save(filename, a)

def loadNumpyArray(filename):
    return np.load(filename+'.npy')

# for part 2

def findMinEnclosingCircle(samples):
    circle_details = np.zeros((samples.shape[0],3))
    for i in range(samples.shape[0]):
        curr_img = np.array(samples[i,:,:]).astype(np.uint8)
        _, contours, hierarchy = cv2.findContours(curr_img,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        (x,y), r = cv2.minEnclosingCircle(contours[0])
#         print('Len of contour is ',len(contours))
        circle_details[i,0] = x
        circle_details[i,1] = y
        circle_details[i,2] = r
    return circle_details


# for part 3

def getConcatenatedMat(samples,ind1,ind2,ind3,ind4):
```

```
    topimg = np.hstack((samples[ind1,:,:],samples[ind2,:,:]))
    botimg = np.hstack((samples[ind3,:,:],samples[ind4,:,:]))
    return np.vstack((topimg,botimg))

def getBackgroundMat(mat):
    mat = mat + 1
    mat[mat==2] = 0
    return mat

def form4imagesDataset(samples,binary_masks,truth_labels,total_no):
    sizex = samples.shape[1]
    composite_images = np.zeros((total_no,sizex*2,sizex*2,12))
    datasize = samples.shape[0]

    for i in range(total_no):
        img1_index = random.randint(0,datasize-1)
        img2_index = random.randint(0,datasize-1)
        img3_index = random.randint(0,datasize-1)
        img4_index = random.randint(0,datasize-1)

        # 0 index for original image
        # 1 to 10 for foreground
        # 11 index for background

        # input image, grayscale one
        composite_images[i,:,:,0] =
getConcatenatedMat(samples,img1_index,img2_index,img3_index,img4_index)

        img1_label = truth_labels[img1_index]
        img2_label = truth_labels[img2_index]
        img3_label = truth_labels[img3_index]
        img4_label = truth_labels[img4_index]

        composite_images[i,0:sizex,0:sizex,img1_label+1] =
binary_masks[img1_index,:,:]
        composite_images[i,0:sizex,sizex:2*sizex,img2_label+1] =
binary_masks[img2_index,:,:]
        composite_images[i,sizex:2*sizex,0:sizex,img3_label+1] =
binary_masks[img3_index,:,:]
        composite_images[i,sizex:2*sizex,sizex:2*sizex,img4_label+1] =
binary_masks[img4_index,:,:]

        # background pixels
```

```
        background =
getConcatenatedMat(binary_masks,img1_index,img2_index,img3_index,img4_index
)
        composite_images[i,:,:,11] = getBackgroundMat(background)
    return composite_images


# In[3]:


(train_X, train_y), (test_X, test_y) = mnist.load_data()

print('Showing some stats for the data..')
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_y.shape))


# ## 1.1 Prepare Data For Q2

# In[5]:


print('Computing the binary masks for training data....')
binary_masks_trainX = computeBinaryMasks(train_X)
print('Computing the binary masks for testing data....')
binary_masks_testX = computeBinaryMasks(test_X)

print('\nPlotting some results for the visualistion..')
plotFewBinarySamples(binary_masks_trainX,'q2_dataset_plots')

print('saving the results')
saveNumpyArray(binary_masks_trainX,'q2dataset_train')
saveNumpyArray(binary_masks_testX,'q2dataset_test')


# ## 1.2 Prepare Data For Q3

# In[10]:


print('Loading the data from 1.1')
```

```
binary_masks_trainX = loadNumpyArray('q2dataset_train')
binary_masks_testX = loadNumpyArray('q2dataset_test')
print(binary_masks_trainX.shape)
print(binary_masks_testX.shape)


print('Calculating center and radius of binary masks of train X..')
cen_rad_trainX = findMinEnclosingCircle(binary_masks_trainX)
print('Calculating center and radius of binary masks of test X..')
cen_rad_testX = findMinEnclosingCircle(binary_masks_testX)


# saving the results
print('saving the results')
saveNumpyArray(cen_rad_trainX,'q3dataset_cen_rad_train')
saveNumpyArray(cen_rad_testX,'q3dataset_cen_rad_test')


# In[44]:


# displaying the results
print(binary_masks_trainX.shape)
print(binary_masks_testX.shape)
plotImagesWithCircles(binary_masks_trainX.copy(),100,10)


# ## 1.3 Prepare Data for Q4

# In[7]:


print('Loading the data from 1.1 and printing shapes of train and test')
binary_masks_trainX = loadNumpyArray('q2dataset_train')
binary_masks_testX = loadNumpyArray('q2dataset_test')
print(binary_masks_trainX.shape)
print(binary_masks_testX.shape)
print()

train_size = 5000
test_size = 1000

print('Finding the composite images')
```

```
composite_images_trainX = form4imagesDataset(train_X,binary_masks_trainX,
train_y,train_size)
print('Training data shape = ',composite_images_trainX.shape)
composite_images_testX = form4imagesDataset(test_X,binary_masks_testX,
test_y,test_size)
print('Testing data shape = ',composite_images_testX.shape)


#plotting the results
plotFewCompositeSamples(composite_images_trainX,'q4_dataset.png',0)
plotFewCompositeSamples(composite_images_trainX,'q4_dataset_background.png'
,-1)

# saving the results
print('saving the results')
saveNumpyArray(composite_images_trainX,'q4dataset_composite_imgs_train')
saveNumpyArray(composite_images_testX,'q4dataset_composite_imgs_test')


# # Done
#
```

Code for ques 2

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import sys
import random
import warnings

import numpy as np
import pandas as pd
import matplotlib
```

```
matplotlib.use('agg')
import matplotlib.pyplot as plt


from tqdm import tqdm
from itertools import chain
from skimage.io import imread, imshow, imread_collection,
concatenate_images
from skimage.transform import resize
from skimage.morphology import label

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dropout, Lambda
from tensorflow.keras.layers import Conv2D, Conv2DTranspose
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import backend as K
import tensorflow.keras
import tensorflow as tf
import cv2

from sklearn.metrics import jaccard_score
from tensorflow.keras.datasets import mnist


# In[2]:


# functions

def loadNumpyArray(filename):
    return np.load(filename+'.npy')

def jaccardSimilarity(mask1,mask2):
    return jaccard_score(mask1.ravel(),mask2.ravel())


# Loading the data

# In[3]:
```

```
(train_X, train_y), (test_X, test_y) = mnist.load_data()

print('Showing some stats for the data..')
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_y.shape))

print('Loading the data from 1.1')
binary_masks_trainX = loadNumpyArray('q2dataset_train')
binary_masks_testX = loadNumpyArray('q2dataset_test')


# Preparing the data

# In[4]:


# Set some parameters
IMG_WIDTH = 28
IMG_HEIGHT = 28
IMG_CHANNELS = 1
MODEL_NAME = 'modelq2.h5'

seed = 42
random.seed = seed
np.random.seed = seed

print("Preparing the data")

no_of_samples = train_X.shape[0]
X_train = np.zeros((no_of_samples, IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
dtype=np.uint8)
Y_train = np.zeros((no_of_samples, IMG_HEIGHT, IMG_WIDTH, 1),
dtype=np.bool)
print("X_train",X_train.shape)
print("Y_train",Y_train.shape)
print('Getting and resizing train images and masks ... ')

for i in range(no_of_samples):
    X_train[i,:,:,0] = train_X[i]
    Y_train[i,:,:,0] = binary_masks_trainX[i]
```

```
# test data
test_samples_no = test_X.shape[0]
X_test = np.zeros((test_samples_no, IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
dtype=np.uint8)
Y_test = np.zeros((test_samples_no, IMG_HEIGHT, IMG_WIDTH, 1),
dtype=np.bool)

sizes_test = []

for i in range(test_samples_no):
    X_test[i,:,:,0] = test_X[i]
    Y_test[i,:,:,0] = binary_masks_testX[i]

print('Done!')


# Defining the model

# In[5]:


# Build U-Net model
inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
s = Lambda(lambda x: x / 255) (inputs)

c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (s)
c1 = Dropout(0.1) (c1)
c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)

c2 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (p1)
c2 = Dropout(0.1) (c2)
c2 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c2)
p2 = MaxPooling2D((2, 2)) (c2)

# c3 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (p2)
# c3 = Dropout(0.2) (c3)
```

```
# c3 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c3)
# p3 = MaxPooling2D((2, 2)) (c3)

# c4 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p3)
# c4 = Dropout(0.2) (c4)
# c4 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c4)
# p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

c5 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (p2)
c5 = Dropout(0.3) (c5)
c5 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c5)

# u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same') (c5)
# u6 = concatenate([u6, c4])
# c6 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (u6)
# c6 = Dropout(0.2) (c6)
# c6 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c6)

# u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c6)
# u7 = concatenate([u7, c3])
# c7 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (u7)
# c7 = Dropout(0.2) (c7)
# c7 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c7)

u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c5)
u8 = concatenate([u8, c2])
c8 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (u8)
c8 = Dropout(0.1) (c8)
c8 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c8)

u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
```

```
c9 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (u9)
c9 = Dropout(0.1) (c9)
c9 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c9)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)

model = Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=["acc"])
model.summary()


# In[6]:



tf.keras.utils.plot_model(
    model, to_file='q2model.png', show_shapes=True, show_layer_names=True,
    rankdir='TB', expand_nested=True, dpi=96
)


# Training the model

# In[ ]:


# Fit model
earlystopper = EarlyStopping(patience=10, verbose=1)
checkpointer = ModelCheckpoint(MODEL_NAME, verbose=1, save_best_only=True)
epochs =  10
results = model.fit(X_train, Y_train, validation_split=0.1, batch_size=16,
epochs=epochs,callbacks=[earlystopper, checkpointer])


# Testing the model

# In[7]:


model = load_model(MODEL_NAME)
```

```
preds_test = model.predict(X_test, verbose=1)
preds_test_t = (preds_test > 0.5).astype(np.uint8)


# Finding the Jaccard Similarity

# In[8]:


no_of_test_samples = X_test.shape[0]

total_jac_score = 0
for i in range(no_of_test_samples):
    total_jac_score+=
jaccardSimilarity(preds_test_t[i,:,:,0],binary_masks_testX[i])
total_jac_score = total_jac_score /no_of_test_samples

print('No of samples in test set are ',no_of_test_samples)
print('Final Jaccard Similarity score for the test set is ',
total_jac_score)


# Plotting

# In[11]:


from matplotlib import pyplot
def setMatRange(mat):
    max_value = np.max(mat)
    mat = mat / max_value * 255
    return mat

def plotFewBinarySamples(samples,filename):
    for i in range(9):
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(setMatRange(samples[i]),
cmap=pyplot.get_cmap('gray'))
    pyplot.savefig(filename)
    pyplot.show()

plotFewBinarySamples(preds_test_t[:,:,:,0],'q2testpred.png')
plotFewBinarySamples(binary_masks_testX,'q2testtrue.png')
```

```
# # Thank You!
```

Code for ques 3

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import sys
import random
import warnings

import numpy as np
import pandas as pd
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt


from tqdm import tqdm
from itertools import chain
from skimage.io import imread, imshow, imread_collection,
```

```
concatenate_images
from skimage.transform import resize
from skimage.morphology import label

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Dense, LocallyConnected1D
from tensorflow.keras.layers import Dropout, Lambda
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, Conv1D
from tensorflow.keras.layers import MaxPooling2D, MaxPooling1D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import backend as K
import tensorflow.keras
import tensorflow as tf
import cv2

from sklearn.metrics import jaccard_score
from tensorflow.keras.datasets import mnist
import math


# Loading the data

# In[2]:


# functions

def loadNumpyArray(filename):
    return np.load(filename+'.npy')

def jaccardSimilarity(mask1,mask2):
    return jaccard_score(mask1.ravel(),mask2.ravel())

def dis(i,j,x,y):
    return math.sqrt((i-x)**2 + (j-y)**2)

def computeMask(cenrad,max_coord,max_radius):
    x = cenrad[0]*max_coord
    y = cenrad[1]*max_coord
    rad = cenrad[2]*max_radius
```

```
    mask = np.zeros((28,28))

    for i in range(mask.shape[0]):
        for j in range(mask.shape[1]):
            if dis(i,j,x,y) <= rad:
                mask[i][j]=1
    return mask



# In[18]:


(train_X, train_y), (test_X, test_y) = mnist.load_data()

print('Loading the data from 1.2')
cenrad_trainX=loadNumpyArray('q3dataset_cen_rad_train')
cenrad_testX = loadNumpyArray('q3dataset_cen_rad_test')


print('Showing some stats for the data..')
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_y.shape))


# Preparing the data

# In[19]:


# Set some parameters
IMG_WIDTH = 28
IMG_HEIGHT = 28
IMG_CHANNELS = 1
OUTPUT_CHANNELS = 1
MODEL_NAME = 'modelq3.h5'

NO_OF_OUTPUT_CLASSSES =10
# OUTPUT_SIZE = NO_OF_OUTPUT_CLASSSES + 3 # x,y,radius
OUTPUT_SIZE = NO_OF_OUTPUT_CLASSSES # x,y,radius
```

```
MAX_COORD = 28
MAX_RADIUS = 14 * math.sqrt(2) #28/2

seed = 42
random.seed = seed
np.random.seed = seed

print("Preparing the data")

no_of_samples = train_X.shape[0]
X_train = np.zeros((no_of_samples, IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
dtype=np.uint8)
# Y_train = np.zeros((no_of_samples, IMG_HEIGHT, IMG_WIDTH,
OUTPUT_CHANNELS), dtype=np.bool)
Y_train_label = np.zeros((no_of_samples,OUTPUT_SIZE))
Y_train_cenrad = cenrad_trainX
Y_train_cenrad[:,:-1] = Y_train_cenrad[:,:-1]/ MAX_COORD
Y_train_cenrad[:,-1] = Y_train_cenrad[:,-1] / MAX_RADIUS


print("X_train",X_train.shape)
print("Y_train for label",Y_train_label.shape)
print("Y_train for bbox",Y_train_cenrad.shape)

for i in range(no_of_samples):
    X_train[i,:,:,0] = train_X[i,:,:]
    Y_train_label[i,train_y[i]] = 1

# test data
test_samples_no = test_X.shape[0]
X_test = np.zeros((test_samples_no, IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
dtype=np.uint8)
# Y_test = np.zeros((test_samples_no, IMG_HEIGHT, IMG_WIDTH,
OUTPUT_CHANNELS), dtype=np.bool)
# Y_test = np.zeros((test_samples_no, 1,1,OUTPUT_SIZE))
Y_test_label = np.zeros((test_samples_no,OUTPUT_SIZE))
Y_test_cenrad = cenrad_testX
Y_test_cenrad[:,:-1] = Y_test_cenrad[:,:-1]/ MAX_COORD
Y_test_cenrad[:,-1] = Y_test_cenrad[:,-1] / MAX_RADIUS

for i in range(test_samples_no):
    X_test[i,:,:,0] = test_X[i,:,:]
    Y_test_label[i,test_y[i]] = 1
```

```
print('Showing some stats for the data..')
print('X_train: ' + str(X_train.shape))
print('Y_train label: ' + str(Y_train_label.shape))
print('X_test:  '  + str(X_test.shape))
print('Y_test label:  '  + str(Y_test_label.shape))
print('Done!')


# Defining the model

# In[20]:


inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
s = Lambda(lambda x: x / 255) (inputs)
x = Conv2D(32, (3,3), activation='elu', kernel_initializer='he_normal',
padding='same')(s)
x = Conv2D(32, (3,3), activation='elu', kernel_initializer='he_normal',
padding='same')(s)
x = MaxPooling2D((2,2))(x)
# x = Conv2D(32, (3,3), activation='relu')(x)
# x = MaxPooling2D((2,2))(x)
x = Conv2D(64, (3,3), activation='elu', kernel_initializer='he_normal',
padding='same')(x)
x = Conv2D(128, (2,2), activation='elu', kernel_initializer='he_normal',
padding='same')(x)

# u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (x)
# # u8 = concatenate([u8, c2])
# c8 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (u8)
# c8 = Dropout(0.1) (c8)
# x = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c8)

# u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c8)
# u9 = concatenate([u9, c1], axis=3)
# c9 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (u9)
# c9 = Dropout(0.1) (c9)
# c9 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c9)
```

```
x = GlobalAveragePooling2D()(x)

# classifier_head = Dropout(0.3)(x)
classifier_head = Dense(64, activation='relu')(x)
classifier_head = Dense(32, activation='relu')(classifier_head)
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)

model = Model(inputs=[inputs], outputs=[classifier_head, reg_head])

losses = {'label': 'categorical_crossentropy',
          'bbox': 'mse'}

loss_weights = {'label': 1.0,
                'bbox': 1.0}

model.compile('adam', loss=losses, loss_weights=loss_weights,
metrics=['acc','mse'])
model.summary()


# In[21]:


tf.keras.utils.plot_model(
    model, to_file='q3model.png', show_shapes=True, show_layer_names=True,
    rankdir='TB', expand_nested=True, dpi=96
)


# training the model

# In[22]:
```

```
# Fit model
earlystopper = EarlyStopping(patience=10, verbose=1)
checkpointer = ModelCheckpoint(MODEL_NAME, verbose=1, save_best_only=True)
epochs =  15
results = model.fit(X_train, [Y_train_label, Y_train_cenrad],
validation_split=0.1, batch_size=16, epochs=epochs,callbacks=[earlystopper,
checkpointer])


# Testing the model

# In[23]:


model = load_model(MODEL_NAME)
preds_test = model.predict(X_test, verbose=1)


# In[25]:


pred_labels = preds_test[0]
pred_bbox = preds_test[1]

wrong_classification=0
right_classification=0
total_jac_score = 0

# for plotting
samples_to_plot = 9
samples_in_row = 3
images_to_plot = np.zeros((samples_to_plot,28,28,1))
true_cenrad_plot = np.zeros((samples_to_plot,3))
pred_cenrad_plot = np.zeros((samples_to_plot,3))
plot_count = 0

for i in range(pred_labels.shape[0]):
    temp = np.zeros(pred_labels.shape[1])
    max_index = np.argmax(pred_labels[i,:])
    pred_labels[i,:] = temp
    pred_labels[i,max_index] = 1
    if test_y[i]!=max_index:
```

```
        wrong_classification+=1
    else:
        #compute jacc index
        right_classification+=1
        true_mask = computeMask(Y_train_cenrad[i],MAX_COORD,MAX_RADIUS)
        pred_mask = computeMask(pred_bbox[i],MAX_COORD,MAX_RADIUS)
        total_jac_score+= jaccardSimilarity(pred_mask,true_mask)

        if plot_count<samples_to_plot:
            images_to_plot[plot_count] = X_test[i]
            true_cenrad_plot[plot_count] = Y_test_cenrad[i]
            pred_cenrad_plot[plot_count] = pred_bbox[i]
            plot_count+=1

print('Printing the result....')
print('The no of samples in test set are ',preds_test[0].shape[0])
print('No of right classifications = ',right_classification)
print('No of wrong classifications = ',wrong_classification)
print('Jaccard score is ',total_jac_score/preds_test[0].shape[0])
# print('Jaccard score is ',total_jac_score/right_classification)



# Plotting the result

# In[14]:


def
plotImagesWithCircles(samples,cenrad,no_of_samples,no_in_row,filename,max_c
oord,max_rad):
    images = []
    img = []
    color = (255,0,0)
    print(filename)
    for i in range(no_of_samples):
        tempimg = samples[i,:,:,0]
        x = cenrad[i,0]*max_coord
        y = cenrad[i,1]*max_coord
        r = cenrad[i,2]*max_rad
        print('In plot:Sample ',i,' has x, y, r are
',int(x),int(y),int(r))
        tempimg = np.array(tempimg).astype(np.uint8)
```

```
        drawing = tempimg
        cv2.circle(drawing, (int(x), int(y)), int(r), color, 1)

        if(img == []):
            img = drawing
        else :
            img = np.hstack((img, drawing))

        if (img.shape[1] /28 == no_in_row):
            if(images == []):
                images = img.copy()
                img = []
            else:
                images = np.vstack((images, img))
                img = []
    print()
    cv2.imwrite(filename+'.jpg', images)
    cv2.imshow(filename, images)
    cv2.waitKey()
    cv2.destroyAllWindows()

plotImagesWithCircles(images_to_plot,true_cenrad_plot,samples_to_plot,sampl
es_in_row,'Q3TrueValues',MAX_COORD,MAX_RADIUS)
plotImagesWithCircles(images_to_plot,pred_cenrad_plot,samples_to_plot,sampl
es_in_row,'Q3Predictions',MAX_COORD,MAX_RADIUS)


# # Thank You
```

Code for ques 4

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import sys
import random
import warnings

import numpy as np
import pandas as pd
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt


from tqdm import tqdm
from itertools import chain
from skimage.io import imread, imshow, imread_collection,
concatenate_images
from skimage.transform import resize
from skimage.morphology import label

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dropout, Lambda
from tensorflow.keras.layers import Conv2D, Conv2DTranspose
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import backend as K
```

```
import tensorflow.keras
import tensorflow as tf
import cv2

from sklearn.metrics import jaccard_score
from tensorflow.keras.datasets import mnist


# Loading the data

# In[2]:


# functions

def loadNumpyArray(filename):
    return np.load(filename+'.npy')

def jaccardSimilarity(mask1,mask2):
    return jaccard_score(mask1.ravel(),mask2.ravel())


# In[3]:


(train_X, train_y), (test_X, test_y) = mnist.load_data()

print('Loading the data from 1.3')
composite_images_trainX=loadNumpyArray('q4dataset_composite_imgs_train')
composite_images_testX = loadNumpyArray('q4dataset_composite_imgs_test')

train_X = composite_images_trainX
test_X = composite_images_testX

print('Showing some stats for the data..')
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_y.shape))

# print('Loading the data from 1.1')
# binary_masks_trainX = loadNumpyArray('q2dataset_train')
# binary_masks_testX = loadNumpyArray('q2dataset_test')
```

```
# Preparing the data

# In[4]:


# Set some parameters
IMG_WIDTH = 56
IMG_HEIGHT = 56
IMG_CHANNELS = 1
OUTPUT_CHANNELS = 11
MODEL_NAME = 'modelq4.h5'

seed = 42
random.seed = seed
np.random.seed = seed

print("Preparing the data")

no_of_samples = train_X.shape[0]
X_train = np.zeros((no_of_samples, IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
dtype=np.uint8)
Y_train = np.zeros((no_of_samples, IMG_HEIGHT, IMG_WIDTH, OUTPUT_CHANNELS),
dtype=np.bool)
print("X_train",X_train.shape)
print("Y_train",Y_train.shape)
# print('Getting and resizing train images and masks ... ')

for i in range(no_of_samples):
    X_train[i,:,:,0] = train_X[i,:,:,0]
    Y_train[i,:,:,:] = train_X[i,:,:,1:]

# test data
test_samples_no = test_X.shape[0]
X_test = np.zeros((test_samples_no, IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
dtype=np.uint8)
Y_test = np.zeros((test_samples_no, IMG_HEIGHT, IMG_WIDTH,
OUTPUT_CHANNELS), dtype=np.bool)

sizes_test = []

for i in range(test_samples_no):
```

```
    X_test[i,:,:,0] = test_X[i,:,:,0]
    Y_test[i,:,:,:] = test_X[i,:,:,1:]

print('Done!')


# Defining the model

# In[5]:


# Build U-Net model
inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
s = Lambda(lambda x: x / 255) (inputs)

c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (s)
c1 = Dropout(0.1) (c1)
c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)

c2 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (p1)
c2 = Dropout(0.1) (c2)
c2 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c2)
p2 = MaxPooling2D((2, 2)) (c2)

# c3 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (p2)
# c3 = Dropout(0.2) (c3)
# c3 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c3)
# p3 = MaxPooling2D((2, 2)) (c3)

# c4 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (p3)
# c4 = Dropout(0.2) (c4)
# c4 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c4)
# p4 = MaxPooling2D(pool_size=(2, 2)) (c4)
```

```
c5 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (p2)
c5 = Dropout(0.3) (c5)
c5 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c5)

# u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same') (c5)
# u6 = concatenate([u6, c4])
# c6 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (u6)
# c6 = Dropout(0.2) (c6)
# c6 = Conv2D(128, (3, 3), activation='elu',
kernel_initializer='he_normal', padding='same') (c6)

# u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c6)
# u7 = concatenate([u7, c3])
# c7 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (u7)
# c7 = Dropout(0.2) (c7)
# c7 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c7)

u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c5)
u8 = concatenate([u8, c2])
c8 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (u8)
c8 = Dropout(0.1) (c8)
c8 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c8)

u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (u9)
c9 = Dropout(0.1) (c9)
c9 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal',
padding='same') (c9)

outputs = Conv2D(11, (1, 1), activation='sigmoid') (c9)

model = Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=["acc"])
```

```
model.summary()


# In[6]:


tf.keras.utils.plot_model(
    model, to_file='q4model.png', show_shapes=True, show_layer_names=True,
    rankdir='TB', expand_nested=True, dpi=96
)


# Training the model

# In[7]:


# Fit model
earlystopper = EarlyStopping(patience=10, verbose=1)
checkpointer = ModelCheckpoint(MODEL_NAME, verbose=1, save_best_only=True)
epochs =  10
results = model.fit(X_train, Y_train, validation_split=0.1, batch_size=16,
epochs=epochs,callbacks=[earlystopper, checkpointer])


# Testing the model

# In[8]:


model = load_model(MODEL_NAME)
preds_test = model.predict(X_test, verbose=1)
# preds_test_t = (preds_test > 0.5).astype(np.uint8)


# In[9]:


size0,sizex,sizey,sizechannels = preds_test.shape

binary_preds_test = np.zeros(preds_test.shape)

total_jac_index = 0
```

```
for k in range(size0):
    for i in range(sizex):
        for j in range(sizey):
            max_index=np.argmax(preds_test[k,i,j,:])
            binary_preds_test[k,i,j,max_index]=1


for k in range(size0):
    jac_index = 0
    count=0
    for ch in range(sizechannels):
        if np.sum(np.sum(Y_test[k,:,:,ch]))!=0 and
np.sum(np.sum(binary_preds_test[k,:,:,ch]))!=0:
            count+=1
            jac_index+=
jaccardSimilarity(binary_preds_test[k,:,:,ch],Y_test[k,:,:,ch])
    jac_index = jac_index/count
    total_jac_index += jac_index

total_jac_index = total_jac_index/size0



# for k in range(size0):
#     union_count = np.zeros(sizechannels)
#     intersection_count = np.zeros(sizechannels)
#     for i in range(sizex):
#         for j in range(sizey):
#             max_index=np.argmax(preds_test[k,i,j,:])
#             if Y_test[k,i,j,max_index]==1:
#                 union_count[max_index] +=1
#                 intersection_count[max_index] +=1
#             else:
#                 union_count[max_index] +=1
#                 union_count[Y_test[k,i,j,:]==1] +=1

#     # calc jac index
#     jac_index = 0
#     deno=0
#     for index in range(union_count.size):
#         if union_count[index]!=0:
```

```
#             deno+=1
#             jac_index += intersection_count[index] / union_count[index]
#     jac_index = jac_index/deno
#     total_jac_index +=jac_index

# total_jac_index = total_jac_index/size0

print('Shape of predicted test samples are ',preds_test.shape)
print('No of samples in test set are ',size0)
print('Jaccard index is coming out to be ',total_jac_index)




# In[12]:


from matplotlib import pyplot

def setMatRange(mat):
    max_value = np.max(mat)
    mat = mat / max_value * 255
    return mat

def plotFewCompositeSamples(samples,filename,index):
    for i in range(9):
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(setMatRange(samples[i,:,:,index]),
cmap=pyplot.get_cmap('gray'))
    pyplot.savefig(filename)
    pyplot.show()

plotFewCompositeSamples(binary_preds_test,'q4testpred.png',-1)
plotFewCompositeSamples(Y_test,'q4testtrue.png',-1)


# # Thank You!
```