Assignment 4: Build a Supervised Autoencoder.

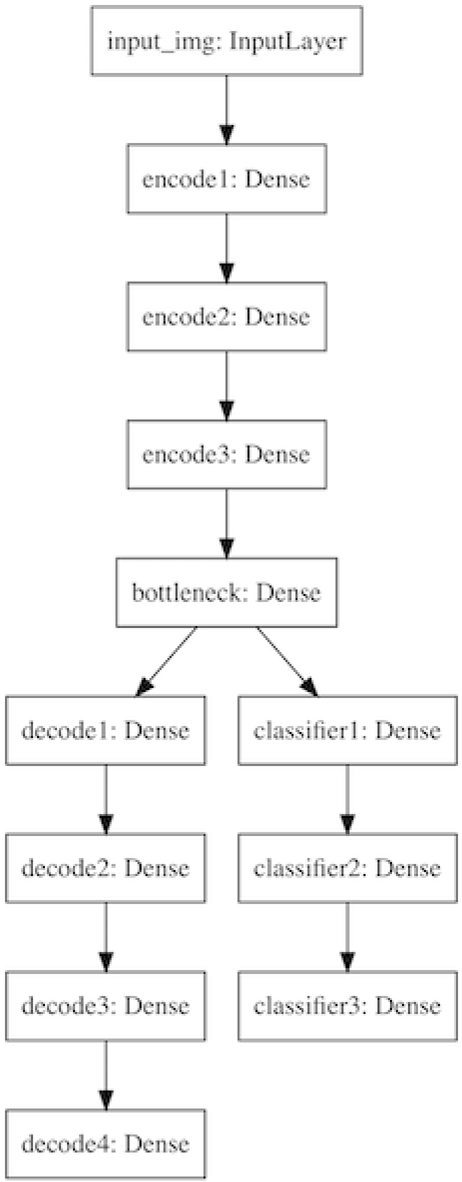
Name: [Bhargavi Katta]

Due Date: Tuesday 5/2/2023 11:59PM

PCA and the standard autoencoder are unsupervised dimensionality reduction methods, and their learned features are not discriminative. If you build a classifier upon the low-dimenional features extracted by PCA and autoencoder, you will find the classification accuracy very poor.

Linear discriminant analysis (LDA) is a traditionally supervised dimensionality reduction method for learning low-dimensional features which are highly discriminative. Likewise, can we extend autoencoder to supervised leanring?

You are required to build and train a supervised autoencoder look like the following. You are required to add other layers properly to alleviate overfitting.



0. You will do the following:

1. Build a standard dense autoencoder, visual the low-dim features and the reconstructions, and evaluate whether the learned low-dim features are discriminative.

2. Repeat the above process by training a supervised autoencoder.

1. Data preparation

1.1. Load data

In [1]:
```
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 28*28).astype('float32') / 255.
x_test = x_test.reshape(10000, 28*28).astype('float32') / 255.

print('Shape of x_train: ' + str(x_train.shape))
print('Shape of x_test: ' + str(x_test.shape))
print('Shape of y_train: ' + str(y_train.shape))
print('Shape of y_test: ' + str(y_test.shape))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
Shape of x_train: (60000, 784)
Shape of x_test: (10000, 784)
Shape of y_train: (60000,)
Shape of y_test: (10000,)
```

1.2. One-hot encode the labels

In the input, a label is a scalar in $\{0, 1, \cdots, 9\}$. One-hot encode transform such a scalar to a 10-dim vector. E.g., a scalar `y_train[j]=3` is transformed to the vector `y_train_vec[j]=[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]` .

1. Define a function `to_one_hot` that transforms an $n \times 1$ array to a $n \times 10$ matrix.

2. Apply the function to `y_train` and `y_test` .

In [2]:
```
import numpy as np

def to_one_hot(y, num_class=10):
```

```
    results = np.zeros((len(y), num_class))
    for i, label in enumerate(y):
        results[i, label] = 1.
    return results

y_train_vec = to_one_hot(y_train)
y_test_vec = to_one_hot(y_test)

print('Shape of y_train_vec: ' + str(y_train_vec.shape))
print('Shape of y_test_vec: ' + str(y_test_vec.shape))

print(y_train[0])
print(y_train_vec[0])
```

```
Shape of y_train_vec: (60000, 10)
Shape of y_test_vec: (10000, 10)
5
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

1.3. Randomly partition the training set to training and validation sets

Randomly partition the 60K training samples to 2 sets:

- a training set containing 10K samples;
- a validation set containing 50K samples. (You can use only 10K to save time.)

In [3]:

```
rand_indices = np.random.permutation(60000)
train_indices = rand_indices[0:10000]
valid_indices = rand_indices[10000:20000]

x_val = x_train[valid_indices, :]
y_val = y_train_vec[valid_indices, :]

x_tr = x_train[train_indices, :]
y_tr = y_train_vec[train_indices, :]

print('Shape of x_tr: ' + str(x_tr.shape))
print('Shape of y_tr: ' + str(y_tr.shape))
print('Shape of x_val: ' + str(x_val.shape))
print('Shape of y_val: ' + str(y_val.shape))
```

```
Shape of x_tr: (10000, 784)
Shape of y_tr: (10000, 10)
```

```
Shape of x_val: (10000, 784)
Shape of y_val: (10000, 10)
```

2. Build an unsupervised autoencoder and tune its hyper-parameters

1. Build a dense autoencoder model
2. Your encoder should contain 3 dense layers and 1 bottlenect layer with 2 as output size.
3. Your decoder should contain 4 dense layers with 784 as output size.
4. You can choose different number of hidden units in dense layers.
5. Do not add other layers (no activation layers), you may add them in later sections.
6. Use the validation data to tune the hyper-parameters (e.g., network structure, and optimization algorithm)
    - Do NOT use test data for hyper-parameter tuning!!!
7. Try to achieve a validation loss as low as possible.
8. Evaluate the model on the test set.
9. Visualize the low-dim features and reconstructions.

2.1. Build the model (20 points)

In [4]:

```
from keras.layers import *
from keras import models

input_img = Input(shape=(784,), name='input_img')


#Model
# encode:128->32->8 -> bottleneck->
#Decoed: bottleneck->8->32->128->784
#tested with the above architecture..finally it gave 93% accuracy
#below implemented one gave 95% accuracy

encode1 = Dense(256, activation='relu', name='encode1')(input_img)
encode2 = Dense(128, activation='relu', name='encode2')(encode1)
encode3 = Dense(64, activation='relu', name='encode3')(encode2)

bottleneck = Dense(2, activation='relu', name='bottleneck')(encode3)

decode1 = Dense(64, activation='relu', name='decode1')(bottleneck)
decode2 = Dense(128, activation='relu', name='decode2')(decode1)
```

```
decode3 = Dense(256, activation='relu', name='decode3')(decode2)
decode4 = Dense(784, activation='relu', name='decode4')(decode3)

ae = models.Model(input_img, decode4)

ae.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_img (InputLayer)      [(None, 784)]             0

 encode1 (Dense)             (None, 256)               200960

 encode2 (Dense)             (None, 128)               32896

 encode3 (Dense)             (None, 64)                8256

 bottleneck (Dense)          (None, 2)                 130

 decode1 (Dense)             (None, 64)                192

 decode2 (Dense)             (None, 128)               8320

 decode3 (Dense)             (None, 256)               33024

 decode4 (Dense)             (None, 784)               201488

=================================================================
Total params: 485,266
Trainable params: 485,266
Non-trainable params: 0
_________________________________________________________________
```

In [5]:

```
# print the network structure to a PDF file

from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot, plot_model

SVG(model_to_dot(ae, show_shapes=False).create(prog='dot', format='svg'))

plot_model(
    model=ae, show_shapes=False,
    to_file='unsupervised_ae.pdf'
```

```
)

# you can find the file "unsupervised_ae.pdf" in the current directory.
```

2.2. Train the model and tune the hyper-parameters (5 points)

In [6]:

```
from tensorflow.keras import optimizers


learning_rate = 1E-2 #  tuned!

ae.compile(loss='mean_squared_error',
           optimizer=optimizers.RMSprop(learning_rate=learning_rate))
```

In [7]:

```
history = ae.fit(x_tr, x_tr,
                 batch_size=128,
                 epochs=100,
                 validation_data=(x_val, x_val))
```

```
Epoch 1/100
79/79 [==============================] - 8s 9ms/step - loss: 0.0822 - val_loss: 0.0759
Epoch 2/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0720 - val_loss: 0.0693
Epoch 3/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0672 - val_loss: 0.0753
Epoch 4/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0639 - val_loss: 0.0601
Epoch 5/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0596 - val_loss: 0.0621
Epoch 6/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0569 - val_loss: 0.0551
Epoch 7/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0542 - val_loss: 0.0542
Epoch 8/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0526 - val_loss: 0.0524
Epoch 9/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0516 - val_loss: 0.0525
Epoch 10/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0506 - val_loss: 0.0509
Epoch 11/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0496 - val_loss: 0.0510
Epoch 12/100
```

```
79/79 [==============================] - 0s 5ms/step - loss: 0.0485 - val_loss: 0.0516
Epoch 13/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0479 - val_loss: 0.0479
Epoch 14/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0473 - val_loss: 0.0504
Epoch 15/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0465 - val_loss: 0.0518
Epoch 16/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0458 - val_loss: 0.0497
Epoch 17/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0449 - val_loss: 0.0477
Epoch 18/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0445 - val_loss: 0.0492
Epoch 19/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0438 - val_loss: 0.0478
Epoch 20/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0434 - val_loss: 0.0444
Epoch 21/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0430 - val_loss: 0.0469
Epoch 22/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0430 - val_loss: 0.0466
Epoch 23/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0426 - val_loss: 0.0433
Epoch 24/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0423 - val_loss: 0.0430
Epoch 25/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0420 - val_loss: 0.0452
Epoch 26/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0418 - val_loss: 0.0447
Epoch 27/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0417 - val_loss: 0.0464
Epoch 28/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0414 - val_loss: 0.0432
Epoch 29/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0413 - val_loss: 0.0452
Epoch 30/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0410 - val_loss: 0.0432
Epoch 31/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0407 - val_loss: 0.0457
Epoch 32/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0408 - val_loss: 0.0424
Epoch 33/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0406 - val_loss: 0.0460
Epoch 34/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0405 - val_loss: 0.0426
```

```
Epoch 35/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0402 - val_loss: 0.0465
Epoch 36/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0401 - val_loss: 0.0440
Epoch 37/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0400 - val_loss: 0.0493
Epoch 38/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0402 - val_loss: 0.0432
Epoch 39/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0399 - val_loss: 0.0444
Epoch 40/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0396 - val_loss: 0.0468
Epoch 41/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0397 - val_loss: 0.0437
Epoch 42/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0394 - val_loss: 0.0428
Epoch 43/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0392 - val_loss: 0.0416
Epoch 44/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0390 - val_loss: 0.0437
Epoch 45/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0390 - val_loss: 0.0467
Epoch 46/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0390 - val_loss: 0.0425
Epoch 47/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0389 - val_loss: 0.0427
Epoch 48/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0386 - val_loss: 0.0420
Epoch 49/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0394 - val_loss: 0.0438
Epoch 50/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0393 - val_loss: 0.0410
Epoch 51/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0389 - val_loss: 0.0449
Epoch 52/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0389 - val_loss: 0.0413
Epoch 53/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0386 - val_loss: 0.0417
Epoch 54/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0388 - val_loss: 0.0417
Epoch 55/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0384 - val_loss: 0.0435
Epoch 56/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0384 - val_loss: 0.0446
Epoch 57/100
```

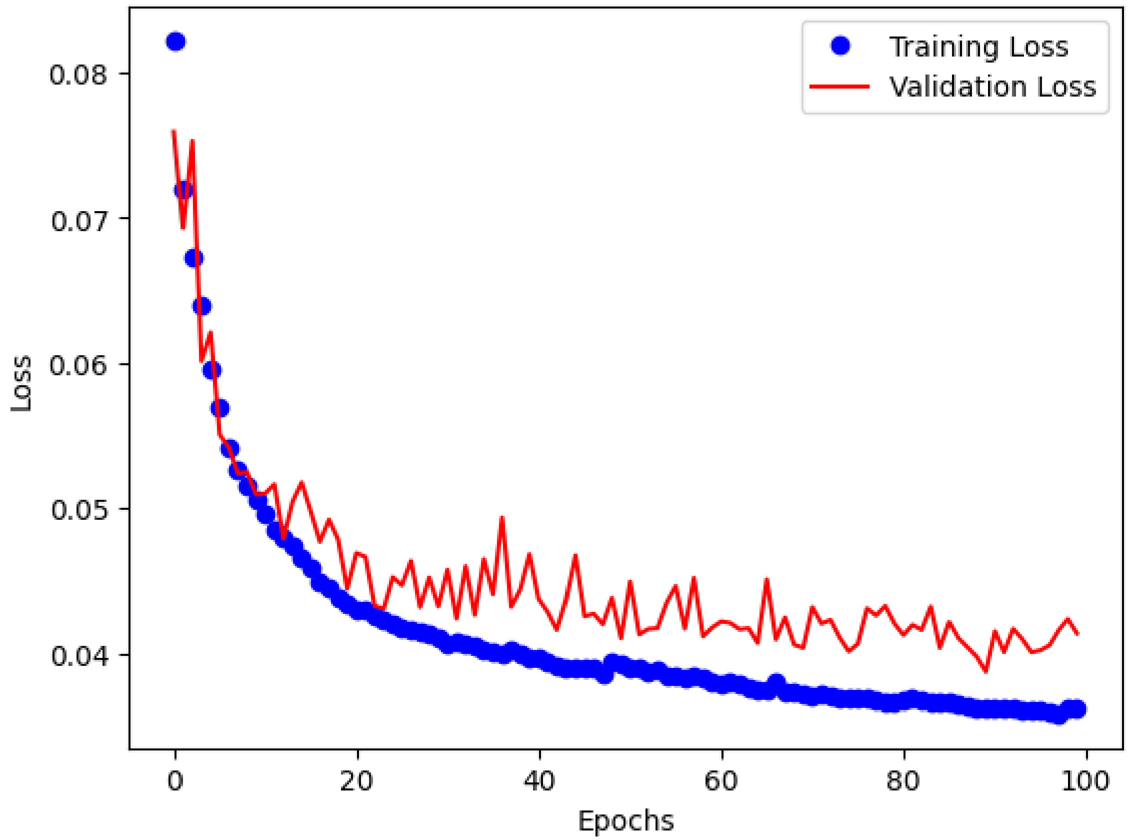
```
79/79 [==============================] - 0s 5ms/step - loss: 0.0383 - val_loss: 0.0417
Epoch 58/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0384 - val_loss: 0.0452
Epoch 59/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0382 - val_loss: 0.0412
Epoch 60/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0380 - val_loss: 0.0418
Epoch 61/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0378 - val_loss: 0.0422
Epoch 62/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0380 - val_loss: 0.0421
Epoch 63/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0378 - val_loss: 0.0416
Epoch 64/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0376 - val_loss: 0.0417
Epoch 65/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0375 - val_loss: 0.0407
Epoch 66/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0374 - val_loss: 0.0451
Epoch 67/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0379 - val_loss: 0.0409
Epoch 68/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0374 - val_loss: 0.0425
Epoch 69/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0374 - val_loss: 0.0406
Epoch 70/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0372 - val_loss: 0.0404
Epoch 71/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0371 - val_loss: 0.0432
Epoch 72/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0372 - val_loss: 0.0420
Epoch 73/100
79/79 [==============================] - 1s 8ms/step - loss: 0.0370 - val_loss: 0.0423
Epoch 74/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0369 - val_loss: 0.0411
Epoch 75/100
79/79 [==============================] - 1s 6ms/step - loss: 0.0369 - val_loss: 0.0401
Epoch 76/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0369 - val_loss: 0.0406
Epoch 77/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0369 - val_loss: 0.0431
Epoch 78/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0368 - val_loss: 0.0426
Epoch 79/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0367 - val_loss: 0.0432
```

```
Epoch 80/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0367 - val_loss: 0.0420
Epoch 81/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0367 - val_loss: 0.0412
Epoch 82/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0369 - val_loss: 0.0419
Epoch 83/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0368 - val_loss: 0.0416
Epoch 84/100
79/79 [==============================] - 0s 5ms/step - loss: 0.0367 - val_loss: 0.0432
Epoch 85/100
79/79 [==============================] - 1s 7ms/step - loss: 0.0367 - val_loss: 0.0404
Epoch 86/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0366 - val_loss: 0.0421
Epoch 87/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0365 - val_loss: 0.0410
Epoch 88/100
79/79 [==============================] - 1s 7ms/step - loss: 0.0364 - val_loss: 0.0404
Epoch 89/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0362 - val_loss: 0.0398
Epoch 90/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0362 - val_loss: 0.0387
Epoch 91/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0362 - val_loss: 0.0415
Epoch 92/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0363 - val_loss: 0.0401
Epoch 93/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0362 - val_loss: 0.0417
Epoch 94/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0361 - val_loss: 0.0410
Epoch 95/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0361 - val_loss: 0.0401
Epoch 96/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0360 - val_loss: 0.0402
Epoch 97/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0359 - val_loss: 0.0405
Epoch 98/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0358 - val_loss: 0.0416
Epoch 99/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0362 - val_loss: 0.0423
Epoch 100/100
79/79 [==============================] - 0s 6ms/step - loss: 0.0363 - val_loss: 0.0414
```

In [8]:

```python
import matplotlib.pyplot as plt
%matplotlib inline
```

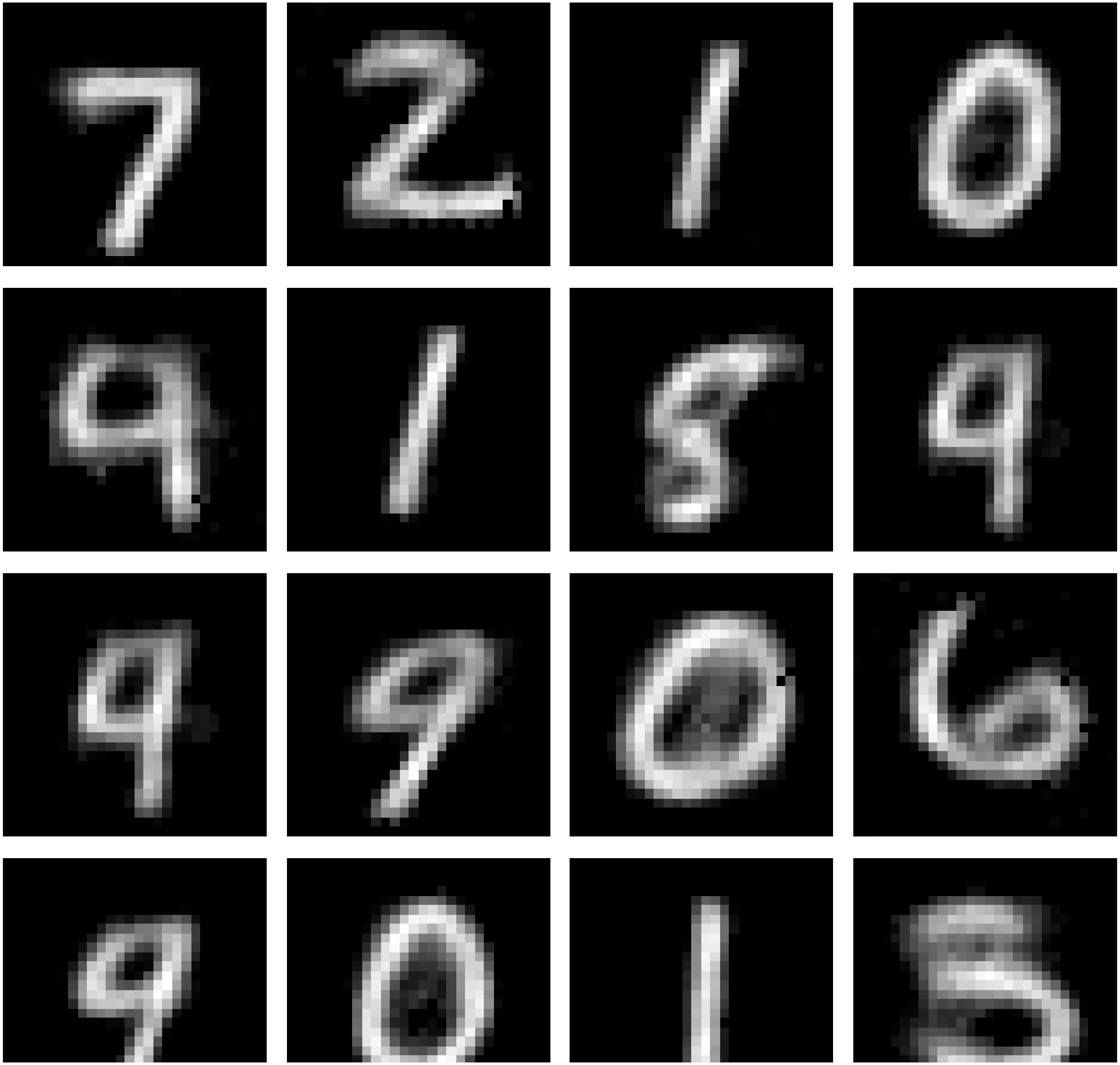
```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



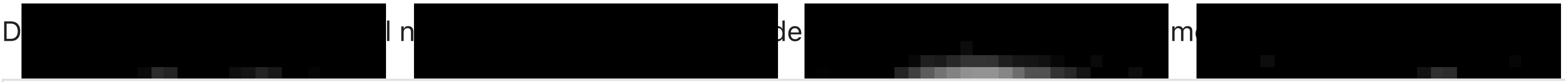
2.3. Visualize the reconstructed test images (5 points)

In [9]:
```
ae_output = ae.predict(x_test).reshape((10000, 28, 28))

ROW = 5
COLUMN = 4

x = ae_output
fname = 'reconstruct_ae.pdf'

fig, axes = plt.subplots(nrows=ROW, ncols=COLUMN, figsize=(8, 10))
for ax, i in zip(axes.flat, np.arange(ROW*COLUMN)):
    image = x[i].reshape(28, 28)
    ax.imshow(image, cmap='gray')
    ax.axis('off')

plt.tight_layout()
plt.savefig(fname)
plt.show()
```

```
313/313 [==============================] - 1s 1ms/step
```



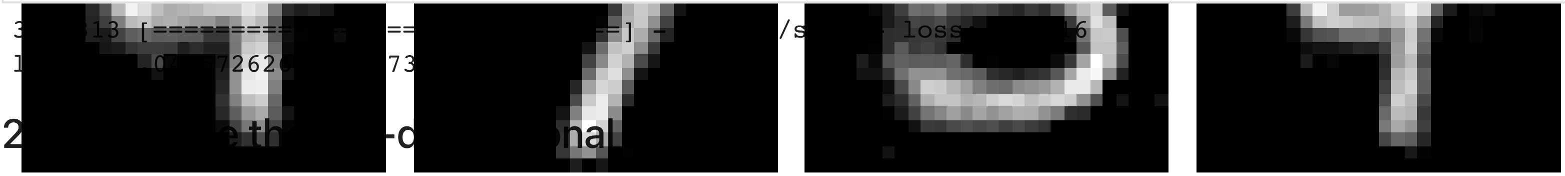
2 [illegible] le



D [illegible] l n [illegible] de [illegible] m [illegible] test set.

In [10]:

```
loss = ae.evaluate(x_test, x_test)
print('loss = ' + str(loss))
```



```
3[illegible]13 [========[illegible]==[illegible]==] - [illegible]/s[illegible] loss[illegible]6
1[illegible]0[illegible]7262[illegible]73
```

2 [illegible] e th[illegible]-c[illegible]nal

In [11]:

```
# build the encoder network
ae_encoder = models.Model(input_img, bottleneck)
ae_encoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_img (InputLayer)      [(None, 784)]             0         
                                                                 
 encode1 (Dense)             (None, 256)               200960    
                                                                 
 encode2 (Dense)             (None, 128)               32896     
                                                                 
 encode3 (Dense)             (None, 64)                8256      
                                                                 
 bottleneck (Dense)          (None, 2)                 130       
                                                                 
=================================================================
Total params: 242,242
Trainable params: 242,242
Non-trainable params: 0
_________________________________________________________________
```

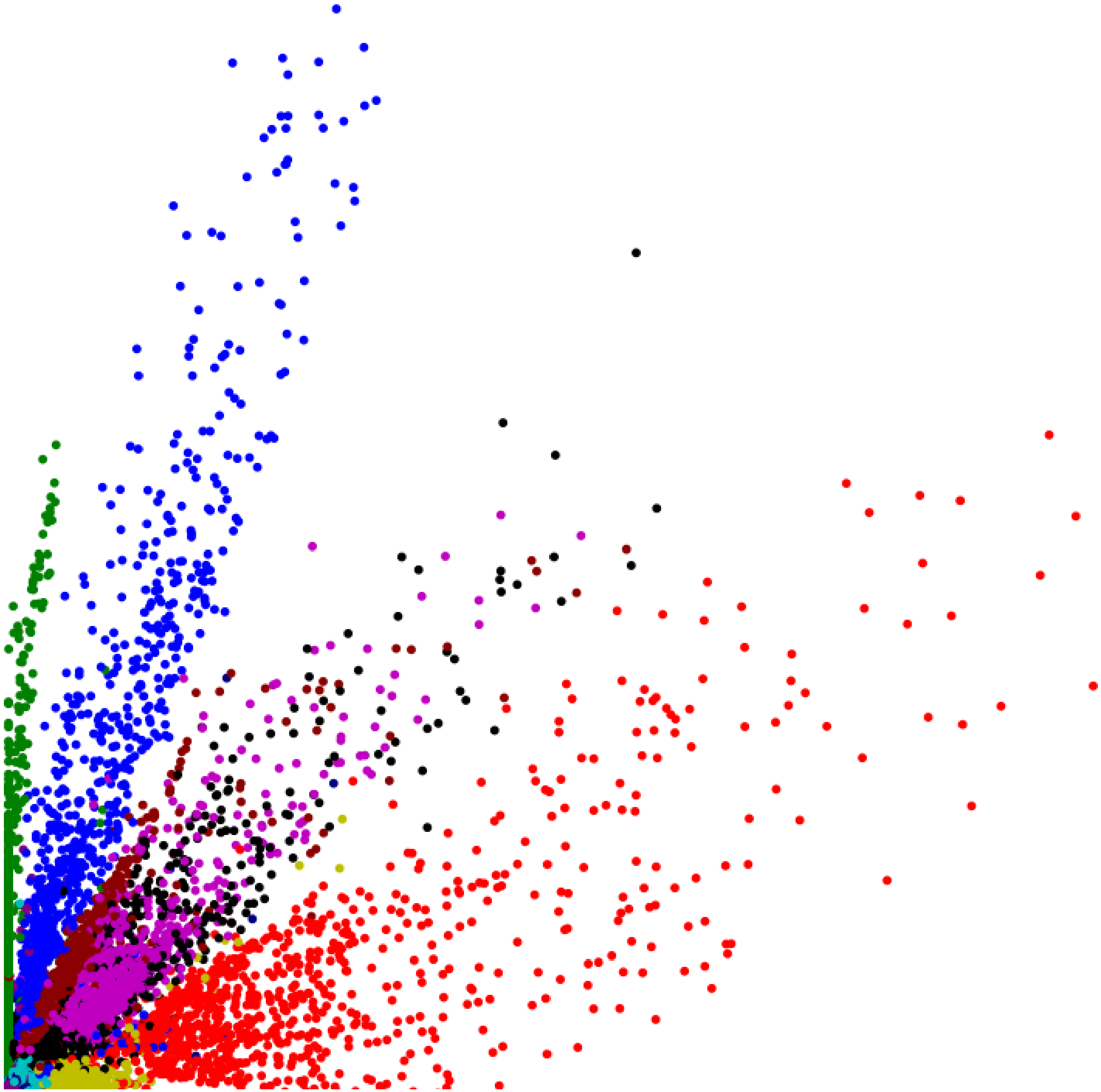
In [12]:

```
# extract low-dimensional features from the test data
encoded_test = ae_encoder.predict(x_test)
print('Shape of encoded_test: ' + str(encoded_test.shape))
```

```
313/313 [==============================] - 0s 1ms/step
Shape of encoded_test: (10000, 2)
```

In [13]:

```
colors = np.array(['r', 'g', 'b', 'm', 'c', 'k', 'y', 'purple', 'darkred', 'navy'])
colors_test = colors[y_test]


import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(8, 8))
plt.scatter(encoded_test[:, 0], encoded_test[:, 1], s=10, c=colors_test, edgecolors=colors_test)
plt.axis('off')
plt.tight_layout()
fname = 'ae_code.pdf'
plt.savefig(fname)
```



Remark:

Judging from the visualization, the low-dim features seems not discriminative, as 2D features from different classes are mixed. Let quantatively find out whether they are discriminative.

3. Are the learned low-dim features discriminative? (10 points)

To find the answer, lets train a classifier on the training set (the extracted 2-dim features) and evaluation on the test set.

In [14]:

```
# extract the 2D features from the training, validation, and test samples
f_tr = ae_encoder.predict(x_tr)
f_val = ae_encoder.predict(x_val)
f_te = ae_encoder.predict(x_test)

print('Shape of f_tr: ' + str(f_tr.shape))
print('Shape of f_te: ' + str(f_te.shape))
```

```
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
Shape of f_tr: (10000, 2)
Shape of f_te: (10000, 2)
```

In [15]:

```
from keras.layers import Dense, Input
from keras import models

input_feat = Input(shape=(2,))

#model:  input->hidden1 layer -> hidden2 -> output
# 2 *128 * 128 *10[classes]

hidden1 = Dense(128, activation='relu')(input_feat)
hidden2 = Dense(128, activation='relu')(hidden1)
output = Dense(10, activation='softmax')(hidden2)

classifier = models.Model(input_feat, output)

classifier.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense (Dense)               (None, 128)               384

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 18,186
Trainable params: 18,186
Non-trainable params: 0
_________________________________________________________________
```

In [16]:

```
classifier.compile(loss='categorical_crossentropy',
                   optimizer=optimizers.RMSprop(learning_rate=1E-4),
                   metrics=['acc'])

history = classifier.fit(f_tr, y_tr,
                         batch_size=32,
                         epochs=30,
                         validation_data=(f_val, y_val))
```

```
Epoch 1/30
313/313 [==============================] - 2s 5ms/step - loss: 1.7912 - acc: 0.3503 - val_loss: 1.5951 - val_ac
c: 0.4142
Epoch 2/30
313/313 [==============================] - 1s 4ms/step - loss: 1.4933 - acc: 0.4588 - val_loss: 1.4300 - val_ac
c: 0.5254
Epoch 3/30
313/313 [==============================] - 1s 4ms/step - loss: 1.3685 - acc: 0.5673 - val_loss: 1.3229 - val_ac
c: 0.5916
Epoch 4/30
313/313 [==============================] - 1s 4ms/step - loss: 1.2690 - acc: 0.6338 - val_loss: 1.2229 - val_ac
c: 0.6576
Epoch 5/30
313/313 [==============================] - 1s 4ms/step - loss: 1.1809 - acc: 0.6829 - val_loss: 1.1368 - val_ac
c: 0.6834
Epoch 6/30
313/313 [==============================] - 1s 5ms/step - loss: 1.1046 - acc: 0.7022 - val_loss: 1.0661 - val_ac
c: 0.7134
```

```
Epoch 7/30
313/313 [==============================] - 1s 4ms/step - loss: 1.0445 - acc: 0.7131 - val_loss: 1.0143 - val_ac
c: 0.7136
Epoch 8/30
313/313 [==============================] - 1s 4ms/step - loss: 0.9995 - acc: 0.7234 - val_loss: 0.9799 - val_ac
c: 0.7057
Epoch 9/30
313/313 [==============================] - 1s 4ms/step - loss: 0.9659 - acc: 0.7321 - val_loss: 0.9470 - val_ac
c: 0.7313
Epoch 10/30
313/313 [==============================] - 1s 4ms/step - loss: 0.9394 - acc: 0.7416 - val_loss: 0.9300 - val_ac
c: 0.7295
Epoch 11/30
313/313 [==============================] - 1s 4ms/step - loss: 0.9202 - acc: 0.7494 - val_loss: 0.9085 - val_ac
c: 0.7476
Epoch 12/30
313/313 [==============================] - 1s 4ms/step - loss: 0.9015 - acc: 0.7588 - val_loss: 0.8980 - val_ac
c: 0.7401
Epoch 13/30
313/313 [==============================] - 1s 4ms/step - loss: 0.8852 - acc: 0.7599 - val_loss: 0.8807 - val_ac
c: 0.7581
Epoch 14/30
313/313 [==============================] - 1s 5ms/step - loss: 0.8721 - acc: 0.7686 - val_loss: 0.8669 - val_ac
c: 0.7606
Epoch 15/30
313/313 [==============================] - 2s 5ms/step - loss: 0.8582 - acc: 0.7720 - val_loss: 0.8569 - val_ac
c: 0.7648
Epoch 16/30
313/313 [==============================] - 1s 4ms/step - loss: 0.8450 - acc: 0.7768 - val_loss: 0.8464 - val_ac
c: 0.7671
Epoch 17/30
313/313 [==============================] - 1s 4ms/step - loss: 0.8338 - acc: 0.7790 - val_loss: 0.8414 - val_ac
c: 0.7652
Epoch 18/30
313/313 [==============================] - 1s 4ms/step - loss: 0.8240 - acc: 0.7823 - val_loss: 0.8254 - val_ac
c: 0.7746
Epoch 19/30
313/313 [==============================] - 1s 4ms/step - loss: 0.8134 - acc: 0.7822 - val_loss: 0.8211 - val_ac
c: 0.7847
Epoch 20/30
313/313 [==============================] - 1s 4ms/step - loss: 0.8038 - acc: 0.7860 - val_loss: 0.8089 - val_ac
c: 0.7806
Epoch 21/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7944 - acc: 0.7873 - val_loss: 0.8080 - val_ac
c: 0.7859
```

```
Epoch 22/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7864 - acc: 0.7898 - val_loss: 0.7962 - val_ac
c: 0.7837
Epoch 23/30
313/313 [==============================] - 1s 5ms/step - loss: 0.7775 - acc: 0.7902 - val_loss: 0.7894 - val_ac
c: 0.7824
Epoch 24/30
313/313 [==============================] - 1s 5ms/step - loss: 0.7704 - acc: 0.7883 - val_loss: 0.7828 - val_ac
c: 0.7865
Epoch 25/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7619 - acc: 0.7920 - val_loss: 0.7741 - val_ac
c: 0.7895
Epoch 26/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7539 - acc: 0.7956 - val_loss: 0.7711 - val_ac
c: 0.7827
Epoch 27/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7471 - acc: 0.7925 - val_loss: 0.7617 - val_ac
c: 0.7888
Epoch 28/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7401 - acc: 0.7939 - val_loss: 0.7603 - val_ac
c: 0.7836
Epoch 29/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7328 - acc: 0.7960 - val_loss: 0.7593 - val_ac
c: 0.7879
Epoch 30/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7269 - acc: 0.7993 - val_loss: 0.7451 - val_ac
c: 0.7903
```

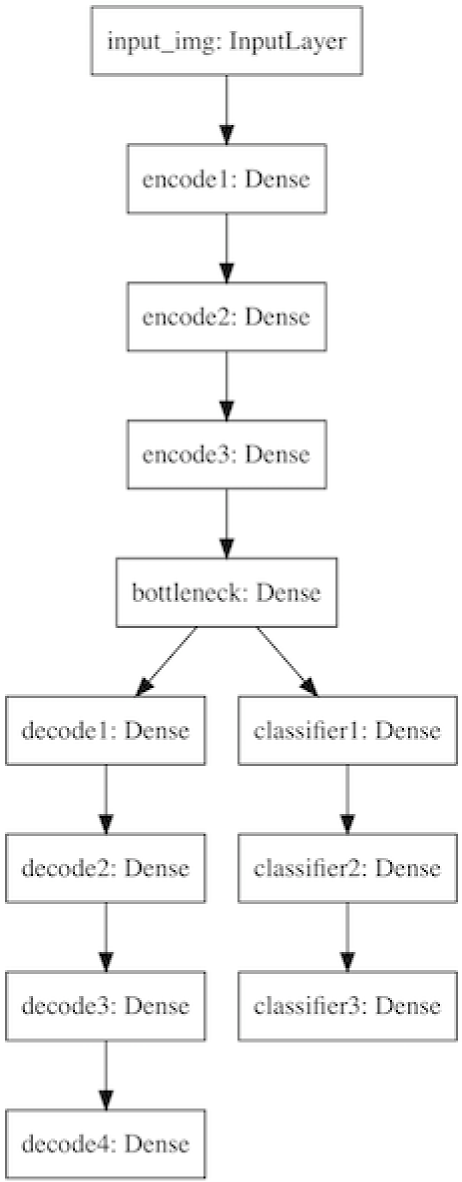
Conclusion

Using the 2D features, the validation accuracy is 60~70%. Recall that using the original data, the accuracy is about 97%. Obviously, the 2D features are not very discriminative.

We are going to build a supervised autoencode model for learning low-dimensional discriminative features.

4. Build a supervised autoencoder model

You are required to build and train a supervised autoencoder look like the following. (Not necessary the same. You can use convolutional layers as well.) You are required to add other layers properly to alleviate overfitting.



4.1. Build the network (30 points)

In [17]:
```
# build the supervised autoencoder network
from keras.layers import *
from keras import models
from keras.regularizers import l2


input_img = Input(shape=(784,), name='input_img')

# encoder network


encode1 = Dense(256, activation='relu', name='encode1')(input_img)
encode2 = Dense(128, activation='relu', name='encode2')(encode1)
encode3 = Dense(64, activation='relu', name='encode3')(encode2)

# The width of the bottleneck layer must be exactly 2.

bottleneck = Dense(2, activation='relu', name='bottleneck')(encode3)

# decoder network
decode1 = Dense(64, activation='relu', name='decode1')(bottleneck)
decode2 = Dense(128, activation='relu', name='decode2')(decode1)
decode3 = Dense(256, activation='relu', name='decode3')(decode2)
decode4 = Dense(784, activation='relu', name='decode4')(decode3)


# build a classifier upon the bottleneck layer
classifier1 = Dense(64, activation='relu', name='classifier1')(bottleneck)
#droputt layer

classifier2 = Dense(128, activation='relu', name='classifier2',kernel_regularizer=l2(0.01))(classifier1)
#droput layer
classifier3 = Dense(10, activation='softmax', name='classifier3')(classifier2)
```

In [18]:
```
# connect the input and the two outputs
sae = models.Model(input_img, [decode4, classifier3])

sae.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
```

```
==========================================================================================
 input_img (InputLayer)         [(None, 784)]        0           []

 encode1 (Dense)                (None, 256)          200960      ['input_img[0][0]']

 encode2 (Dense)                (None, 128)          32896       ['encode1[0][0]']

 encode3 (Dense)                (None, 64)           8256        ['encode2[0][0]']

 bottleneck (Dense)             (None, 2)            130         ['encode3[0][0]']

 decode1 (Dense)                (None, 64)           192         ['bottleneck[0][0]']

 decode2 (Dense)                (None, 128)          8320        ['decode1[0][0]']

 classifier1 (Dense)            (None, 64)           192         ['bottleneck[0][0]']

 decode3 (Dense)                (None, 256)          33024       ['decode2[0][0]']

 classifier2 (Dense)            (None, 128)          8320        ['classifier1[0][0]']

 decode4 (Dense)                (None, 784)          201488      ['decode3[0][0]']

 classifier3 (Dense)            (None, 10)           1290        ['classifier2[0][0]']

==========================================================================================
Total params: 495,068
Trainable params: 495,068
Non-trainable params: 0
__________________________________________________________________________________________
```

In [19]:

```python
# print the network structure to a PDF file

from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot, plot_model

SVG(model_to_dot(sae, show_shapes=False).create(prog='dot', format='svg'))

plot_model(
    model=sae, show_shapes=False,
    to_file='supervised_ae.pdf'
)

# you can find the file "supervised_ae.pdf" in the current directory.
```

4.2. Train the new model and tune the hyper-parameters

The new model has multiple output. Thus we specify **multiple** loss functions and their weights.

In [20]:
```
from tensorflow.keras import optimizers

sae.compile(loss=['mean_squared_error', 'categorical_crossentropy'],
            loss_weights=[1, 0.5], # to be tuned
            optimizer=optimizers.RMSprop(learning_rate=1E-3))

history = sae.fit(x_tr, [x_tr, y_tr],
                  batch_size=32,
                  epochs=100,
                  validation_data=(x_val, [x_val, y_val]))
```

```
Epoch 1/100
313/313 [==============================] - 5s 8ms/step - loss: 0.9931 - decode4_loss: 0.0765 - classifier3_los
s: 1.4264 - val_loss: 0.6434 - val_decode4_loss: 0.0666 - val_classifier3_loss: 0.9827
Epoch 2/100
313/313 [==============================] - 2s 6ms/step - loss: 0.4987 - decode4_loss: 0.0643 - classifier3_los
s: 0.7268 - val_loss: 0.4410 - val_decode4_loss: 0.0623 - val_classifier3_loss: 0.6387
Epoch 3/100
313/313 [==============================] - 2s 7ms/step - loss: 0.3308 - decode4_loss: 0.0611 - classifier3_los
s: 0.4373 - val_loss: 0.3906 - val_decode4_loss: 0.0602 - val_classifier3_loss: 0.5748
Epoch 4/100
313/313 [==============================] - 2s 7ms/step - loss: 0.2506 - decode4_loss: 0.0589 - classifier3_los
s: 0.3098 - val_loss: 0.3683 - val_decode4_loss: 0.0586 - val_classifier3_loss: 0.5595
Epoch 5/100
313/313 [==============================] - 2s 7ms/step - loss: 0.2062 - decode4_loss: 0.0577 - classifier3_los
s: 0.2439 - val_loss: 0.2837 - val_decode4_loss: 0.0581 - val_classifier3_loss: 0.4062
Epoch 6/100
313/313 [==============================] - 2s 7ms/step - loss: 0.1695 - decode4_loss: 0.0567 - classifier3_los
s: 0.1838 - val_loss: 0.2860 - val_decode4_loss: 0.0571 - val_classifier3_loss: 0.4200
Epoch 7/100
313/313 [==============================] - 2s 6ms/step - loss: 0.1486 - decode4_loss: 0.0557 - classifier3_los
s: 0.1513 - val_loss: 0.2475 - val_decode4_loss: 0.0551 - val_classifier3_loss: 0.3540
Epoch 8/100
313/313 [==============================] - 2s 6ms/step - loss: 0.1337 - decode4_loss: 0.0547 - classifier3_los
s: 0.1292 - val_loss: 0.2518 - val_decode4_loss: 0.0557 - val_classifier3_loss: 0.3662
Epoch 9/100
313/313 [==============================] - 2s 6ms/step - loss: 0.1203 - decode4_loss: 0.0541 - classifier3_los
s: 0.1075 - val_loss: 0.2345 - val_decode4_loss: 0.0543 - val_classifier3_loss: 0.3370
Epoch 10/100
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.1104 - decode4_loss: 0.0538 - classifier3_los
s: 0.0910 - val_loss: 0.2569 - val_decode4_loss: 0.0537 - val_classifier3_loss: 0.3866
Epoch 11/100
313/313 [==============================] - 2s 7ms/step - loss: 0.1041 - decode4_loss: 0.0534 - classifier3_los
s: 0.0823 - val_loss: 0.2454 - val_decode4_loss: 0.0537 - val_classifier3_loss: 0.3645
Epoch 12/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0974 - decode4_loss: 0.0533 - classifier3_los
s: 0.0706 - val_loss: 0.2512 - val_decode4_loss: 0.0536 - val_classifier3_loss: 0.3785
Epoch 13/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0932 - decode4_loss: 0.0531 - classifier3_los
s: 0.0647 - val_loss: 0.2414 - val_decode4_loss: 0.0536 - val_classifier3_loss: 0.3603
Epoch 14/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0852 - decode4_loss: 0.0527 - classifier3_los
s: 0.0508 - val_loss: 0.2540 - val_decode4_loss: 0.0529 - val_classifier3_loss: 0.3883
Epoch 15/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0804 - decode4_loss: 0.0523 - classifier3_los
s: 0.0431 - val_loss: 0.2617 - val_decode4_loss: 0.0520 - val_classifier3_loss: 0.4071
Epoch 16/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0797 - decode4_loss: 0.0521 - classifier3_los
s: 0.0433 - val_loss: 0.2721 - val_decode4_loss: 0.0522 - val_classifier3_loss: 0.4279
Epoch 17/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0767 - decode4_loss: 0.0517 - classifier3_los
s: 0.0396 - val_loss: 0.2504 - val_decode4_loss: 0.0518 - val_classifier3_loss: 0.3868
Epoch 18/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0753 - decode4_loss: 0.0514 - classifier3_los
s: 0.0385 - val_loss: 0.2673 - val_decode4_loss: 0.0533 - val_classifier3_loss: 0.4186
Epoch 19/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0736 - decode4_loss: 0.0512 - classifier3_los
s: 0.0356 - val_loss: 0.3223 - val_decode4_loss: 0.0512 - val_classifier3_loss: 0.5337
Epoch 20/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0718 - decode4_loss: 0.0509 - classifier3_los
s: 0.0332 - val_loss: 0.2675 - val_decode4_loss: 0.0507 - val_classifier3_loss: 0.4248
Epoch 21/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0708 - decode4_loss: 0.0506 - classifier3_los
s: 0.0323 - val_loss: 0.2822 - val_decode4_loss: 0.0512 - val_classifier3_loss: 0.4536
Epoch 22/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0690 - decode4_loss: 0.0506 - classifier3_los
s: 0.0291 - val_loss: 0.2894 - val_decode4_loss: 0.0509 - val_classifier3_loss: 0.4692
Epoch 23/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0681 - decode4_loss: 0.0503 - classifier3_los
s: 0.0285 - val_loss: 0.2717 - val_decode4_loss: 0.0505 - val_classifier3_loss: 0.4355
Epoch 24/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0684 - decode4_loss: 0.0503 - classifier3_los
s: 0.0294 - val_loss: 0.2741 - val_decode4_loss: 0.0502 - val_classifier3_loss: 0.4412
Epoch 25/100
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.0627 - decode4_loss: 0.0498 - classifier3_los
s: 0.0197 - val_loss: 0.2703 - val_decode4_loss: 0.0498 - val_classifier3_loss: 0.4356
Epoch 26/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0652 - decode4_loss: 0.0495 - classifier3_los
s: 0.0259 - val_loss: 0.2918 - val_decode4_loss: 0.0501 - val_classifier3_loss: 0.4777
Epoch 27/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0649 - decode4_loss: 0.0494 - classifier3_los
s: 0.0255 - val_loss: 0.2986 - val_decode4_loss: 0.0497 - val_classifier3_loss: 0.4920
Epoch 28/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0656 - decode4_loss: 0.0492 - classifier3_los
s: 0.0272 - val_loss: 0.2744 - val_decode4_loss: 0.0500 - val_classifier3_loss: 0.4430
Epoch 29/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0646 - decode4_loss: 0.0490 - classifier3_los
s: 0.0257 - val_loss: 0.2784 - val_decode4_loss: 0.0492 - val_classifier3_loss: 0.4527
Epoch 30/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0599 - decode4_loss: 0.0485 - classifier3_los
s: 0.0177 - val_loss: 0.2880 - val_decode4_loss: 0.0493 - val_classifier3_loss: 0.4732
Epoch 31/100
313/313 [==============================] - 3s 8ms/step - loss: 0.0636 - decode4_loss: 0.0483 - classifier3_los
s: 0.0255 - val_loss: 0.3320 - val_decode4_loss: 0.0493 - val_classifier3_loss: 0.5605
Epoch 32/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0623 - decode4_loss: 0.0481 - classifier3_los
s: 0.0236 - val_loss: 0.2955 - val_decode4_loss: 0.0485 - val_classifier3_loss: 0.4887
Epoch 33/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0591 - decode4_loss: 0.0476 - classifier3_los
s: 0.0187 - val_loss: 0.2930 - val_decode4_loss: 0.0488 - val_classifier3_loss: 0.4838
Epoch 34/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0581 - decode4_loss: 0.0473 - classifier3_los
s: 0.0173 - val_loss: 0.3359 - val_decode4_loss: 0.0483 - val_classifier3_loss: 0.5709
Epoch 35/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0605 - decode4_loss: 0.0471 - classifier3_los
s: 0.0227 - val_loss: 0.3388 - val_decode4_loss: 0.0483 - val_classifier3_loss: 0.5763
Epoch 36/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0596 - decode4_loss: 0.0470 - classifier3_los
s: 0.0211 - val_loss: 0.3392 - val_decode4_loss: 0.0486 - val_classifier3_loss: 0.5769
Epoch 37/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0608 - decode4_loss: 0.0467 - classifier3_los
s: 0.0242 - val_loss: 0.2601 - val_decode4_loss: 0.0479 - val_classifier3_loss: 0.4210
Epoch 38/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0560 - decode4_loss: 0.0464 - classifier3_los
s: 0.0157 - val_loss: 0.3082 - val_decode4_loss: 0.0477 - val_classifier3_loss: 0.5176
Epoch 39/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0603 - decode4_loss: 0.0462 - classifier3_los
s: 0.0248 - val_loss: 0.3460 - val_decode4_loss: 0.0474 - val_classifier3_loss: 0.5937
Epoch 40/100
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.0588 - decode4_loss: 0.0460 - classifier3_los
s: 0.0217 - val_loss: 0.3072 - val_decode4_loss: 0.0470 - val_classifier3_loss: 0.5171
Epoch 41/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0560 - decode4_loss: 0.0457 - classifier3_los
s: 0.0171 - val_loss: 0.3228 - val_decode4_loss: 0.0469 - val_classifier3_loss: 0.5484
Epoch 42/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0574 - decode4_loss: 0.0456 - classifier3_los
s: 0.0198 - val_loss: 0.3186 - val_decode4_loss: 0.0478 - val_classifier3_loss: 0.5382
Epoch 43/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0557 - decode4_loss: 0.0454 - classifier3_los
s: 0.0170 - val_loss: 0.2996 - val_decode4_loss: 0.0468 - val_classifier3_loss: 0.5024
Epoch 44/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0608 - decode4_loss: 0.0452 - classifier3_los
s: 0.0275 - val_loss: 0.3044 - val_decode4_loss: 0.0464 - val_classifier3_loss: 0.5120
Epoch 45/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0570 - decode4_loss: 0.0450 - classifier3_los
s: 0.0207 - val_loss: 0.3438 - val_decode4_loss: 0.0475 - val_classifier3_loss: 0.5888
Epoch 46/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0561 - decode4_loss: 0.0448 - classifier3_los
s: 0.0191 - val_loss: 0.3363 - val_decode4_loss: 0.0471 - val_classifier3_loss: 0.5747
Epoch 47/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0581 - decode4_loss: 0.0449 - classifier3_los
s: 0.0230 - val_loss: 0.2947 - val_decode4_loss: 0.0462 - val_classifier3_loss: 0.4942
Epoch 48/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0561 - decode4_loss: 0.0448 - classifier3_los
s: 0.0193 - val_loss: 0.3122 - val_decode4_loss: 0.0461 - val_classifier3_loss: 0.5285
Epoch 49/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0554 - decode4_loss: 0.0445 - classifier3_los
s: 0.0187 - val_loss: 0.2886 - val_decode4_loss: 0.0466 - val_classifier3_loss: 0.4813
Epoch 50/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0545 - decode4_loss: 0.0444 - classifier3_los
s: 0.0170 - val_loss: 0.3599 - val_decode4_loss: 0.0463 - val_classifier3_loss: 0.6242
Epoch 51/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0553 - decode4_loss: 0.0444 - classifier3_los
s: 0.0189 - val_loss: 0.3471 - val_decode4_loss: 0.0459 - val_classifier3_loss: 0.5991
Epoch 52/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0535 - decode4_loss: 0.0442 - classifier3_los
s: 0.0155 - val_loss: 0.3411 - val_decode4_loss: 0.0460 - val_classifier3_loss: 0.5870
Epoch 53/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0561 - decode4_loss: 0.0440 - classifier3_los
s: 0.0212 - val_loss: 0.3406 - val_decode4_loss: 0.0460 - val_classifier3_loss: 0.5860
Epoch 54/100
313/313 [==============================] - 3s 8ms/step - loss: 0.0532 - decode4_loss: 0.0439 - classifier3_los
s: 0.0155 - val_loss: 0.3422 - val_decode4_loss: 0.0458 - val_classifier3_loss: 0.5895
Epoch 55/100
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.0576 - decode4_loss: 0.0439 - classifier3_los
s: 0.0246 - val_loss: 0.3424 - val_decode4_loss: 0.0459 - val_classifier3_loss: 0.5900
Epoch 56/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0531 - decode4_loss: 0.0436 - classifier3_los
s: 0.0159 - val_loss: 0.4244 - val_decode4_loss: 0.0460 - val_classifier3_loss: 0.7540
Epoch 57/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0521 - decode4_loss: 0.0436 - classifier3_los
s: 0.0140 - val_loss: 0.4563 - val_decode4_loss: 0.0450 - val_classifier3_loss: 0.8201
Epoch 58/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0537 - decode4_loss: 0.0436 - classifier3_los
s: 0.0172 - val_loss: 0.3227 - val_decode4_loss: 0.0453 - val_classifier3_loss: 0.5524
Epoch 59/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0515 - decode4_loss: 0.0434 - classifier3_los
s: 0.0135 - val_loss: 0.3272 - val_decode4_loss: 0.0457 - val_classifier3_loss: 0.5598
Epoch 60/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0621 - decode4_loss: 0.0433 - classifier3_los
s: 0.0350 - val_loss: 0.3255 - val_decode4_loss: 0.0463 - val_classifier3_loss: 0.5557
Epoch 61/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0546 - decode4_loss: 0.0434 - classifier3_los
s: 0.0198 - val_loss: 0.3062 - val_decode4_loss: 0.0451 - val_classifier3_loss: 0.5197
Epoch 62/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0532 - decode4_loss: 0.0431 - classifier3_los
s: 0.0174 - val_loss: 0.3497 - val_decode4_loss: 0.0449 - val_classifier3_loss: 0.6066
Epoch 63/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0584 - decode4_loss: 0.0431 - classifier3_los
s: 0.0279 - val_loss: 0.3689 - val_decode4_loss: 0.0456 - val_classifier3_loss: 0.6444
Epoch 64/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0531 - decode4_loss: 0.0429 - classifier3_los
s: 0.0178 - val_loss: 0.3484 - val_decode4_loss: 0.0460 - val_classifier3_loss: 0.6024
Epoch 65/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0548 - decode4_loss: 0.0428 - classifier3_los
s: 0.0215 - val_loss: 0.3590 - val_decode4_loss: 0.0452 - val_classifier3_loss: 0.6251
Epoch 66/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0501 - decode4_loss: 0.0427 - classifier3_los
s: 0.0124 - val_loss: 0.3540 - val_decode4_loss: 0.0447 - val_classifier3_loss: 0.6163
Epoch 67/100
313/313 [==============================] - 3s 8ms/step - loss: 0.0535 - decode4_loss: 0.0428 - classifier3_los
s: 0.0187 - val_loss: 0.3552 - val_decode4_loss: 0.0445 - val_classifier3_loss: 0.6189
Epoch 68/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0515 - decode4_loss: 0.0426 - classifier3_los
s: 0.0152 - val_loss: 0.3384 - val_decode4_loss: 0.0450 - val_classifier3_loss: 0.5846
Epoch 69/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0507 - decode4_loss: 0.0425 - classifier3_los
s: 0.0141 - val_loss: 0.3548 - val_decode4_loss: 0.0447 - val_classifier3_loss: 0.6178
Epoch 70/100
```

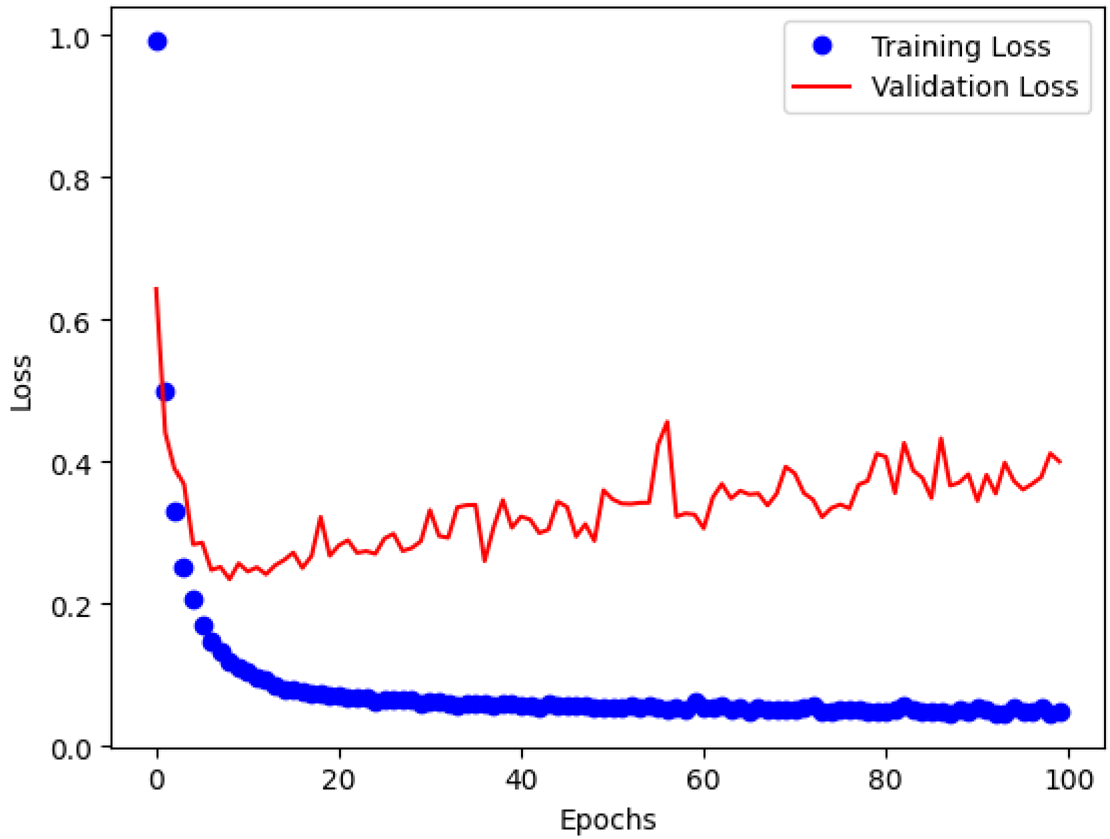
```
313/313 [==============================] - 2s 7ms/step - loss: 0.0526 - decode4_loss: 0.0424 - classifier3_loss: 0.0181 - val_loss: 0.3930 - val_decode4_loss: 0.0444 - val_classifier3_loss: 0.6945
Epoch 71/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0512 - decode4_loss: 0.0422 - classifier3_loss: 0.0156 - val_loss: 0.3835 - val_decode4_loss: 0.0446 - val_classifier3_loss: 0.6750
Epoch 72/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0543 - decode4_loss: 0.0422 - classifier3_loss: 0.0219 - val_loss: 0.3558 - val_decode4_loss: 0.0447 - val_classifier3_loss: 0.6198
Epoch 73/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0568 - decode4_loss: 0.0422 - classifier3_loss: 0.0267 - val_loss: 0.3466 - val_decode4_loss: 0.0444 - val_classifier3_loss: 0.6022
Epoch 74/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0489 - decode4_loss: 0.0419 - classifier3_loss: 0.0118 - val_loss: 0.3221 - val_decode4_loss: 0.0472 - val_classifier3_loss: 0.5474
Epoch 75/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0475 - decode4_loss: 0.0418 - classifier3_loss: 0.0091 - val_loss: 0.3346 - val_decode4_loss: 0.0447 - val_classifier3_loss: 0.5779
Epoch 76/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0525 - decode4_loss: 0.0418 - classifier3_loss: 0.0191 - val_loss: 0.3398 - val_decode4_loss: 0.0444 - val_classifier3_loss: 0.5888
Epoch 77/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0508 - decode4_loss: 0.0418 - classifier3_loss: 0.0157 - val_loss: 0.3342 - val_decode4_loss: 0.0440 - val_classifier3_loss: 0.5786
Epoch 78/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0510 - decode4_loss: 0.0417 - classifier3_loss: 0.0166 - val_loss: 0.3676 - val_decode4_loss: 0.0442 - val_classifier3_loss: 0.6446
Epoch 79/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0489 - decode4_loss: 0.0415 - classifier3_loss: 0.0124 - val_loss: 0.3730 - val_decode4_loss: 0.0441 - val_classifier3_loss: 0.6558
Epoch 80/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0499 - decode4_loss: 0.0415 - classifier3_loss: 0.0144 - val_loss: 0.4114 - val_decode4_loss: 0.0442 - val_classifier3_loss: 0.7317
Epoch 81/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0483 - decode4_loss: 0.0414 - classifier3_loss: 0.0116 - val_loss: 0.4069 - val_decode4_loss: 0.0441 - val_classifier3_loss: 0.7231
Epoch 82/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0519 - decode4_loss: 0.0413 - classifier3_loss: 0.0191 - val_loss: 0.3559 - val_decode4_loss: 0.0440 - val_classifier3_loss: 0.6218
Epoch 83/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0581 - decode4_loss: 0.0413 - classifier3_loss: 0.0316 - val_loss: 0.4265 - val_decode4_loss: 0.0439 - val_classifier3_loss: 0.7628
Epoch 84/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0519 - decode4_loss: 0.0413 - classifier3_loss: 0.0192 - val_loss: 0.3878 - val_decode4_loss: 0.0438 - val_classifier3_loss: 0.6856
Epoch 85/100
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.0482 - decode4_loss: 0.0411 - classifier3_los
s: 0.0123 - val_loss: 0.3773 - val_decode4_loss: 0.0453 - val_classifier3_loss: 0.6621
Epoch 86/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0493 - decode4_loss: 0.0410 - classifier3_los
s: 0.0145 - val_loss: 0.3488 - val_decode4_loss: 0.0438 - val_classifier3_loss: 0.6082
Epoch 87/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0497 - decode4_loss: 0.0410 - classifier3_los
s: 0.0152 - val_loss: 0.4330 - val_decode4_loss: 0.0469 - val_classifier3_loss: 0.7697
Epoch 88/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0473 - decode4_loss: 0.0410 - classifier3_los
s: 0.0102 - val_loss: 0.3663 - val_decode4_loss: 0.0433 - val_classifier3_loss: 0.6437
Epoch 89/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0528 - decode4_loss: 0.0410 - classifier3_los
s: 0.0211 - val_loss: 0.3707 - val_decode4_loss: 0.0438 - val_classifier3_loss: 0.6510
Epoch 90/100
313/313 [==============================] - 3s 9ms/step - loss: 0.0496 - decode4_loss: 0.0408 - classifier3_los
s: 0.0153 - val_loss: 0.3824 - val_decode4_loss: 0.0473 - val_classifier3_loss: 0.6681
Epoch 91/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0535 - decode4_loss: 0.0408 - classifier3_los
s: 0.0233 - val_loss: 0.3450 - val_decode4_loss: 0.0440 - val_classifier3_loss: 0.6000
Epoch 92/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0506 - decode4_loss: 0.0408 - classifier3_los
s: 0.0175 - val_loss: 0.3815 - val_decode4_loss: 0.0437 - val_classifier3_loss: 0.6734
Epoch 93/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0468 - decode4_loss: 0.0406 - classifier3_los
s: 0.0102 - val_loss: 0.3551 - val_decode4_loss: 0.0443 - val_classifier3_loss: 0.6195
Epoch 94/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0462 - decode4_loss: 0.0406 - classifier3_los
s: 0.0091 - val_loss: 0.3988 - val_decode4_loss: 0.0437 - val_classifier3_loss: 0.7080
Epoch 95/100
313/313 [==============================] - 2s 8ms/step - loss: 0.0532 - decode4_loss: 0.0409 - classifier3_los
s: 0.0224 - val_loss: 0.3730 - val_decode4_loss: 0.0440 - val_classifier3_loss: 0.6561
Epoch 96/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0491 - decode4_loss: 0.0407 - classifier3_los
s: 0.0149 - val_loss: 0.3608 - val_decode4_loss: 0.0433 - val_classifier3_loss: 0.6328
Epoch 97/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0478 - decode4_loss: 0.0406 - classifier3_los
s: 0.0124 - val_loss: 0.3687 - val_decode4_loss: 0.0432 - val_classifier3_loss: 0.6491
Epoch 98/100
313/313 [==============================] - 2s 6ms/step - loss: 0.0556 - decode4_loss: 0.0406 - classifier3_los
s: 0.0279 - val_loss: 0.3783 - val_decode4_loss: 0.0441 - val_classifier3_loss: 0.6659
Epoch 99/100
313/313 [==============================] - 2s 7ms/step - loss: 0.0467 - decode4_loss: 0.0404 - classifier3_los
s: 0.0103 - val_loss: 0.4118 - val_decode4_loss: 0.0437 - val_classifier3_loss: 0.7344
Epoch 100/100
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.0500 - decode4_loss: 0.0403 - classifier3_los
s: 0.0174 - val_loss: 0.4003 - val_decode4_loss: 0.0445 - val_classifier3_loss: 0.7098
```

In [21]:

```
import matplotlib.pyplot as plt
%matplotlib inline

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Question (10 points)

Do you think overfitting is happening? If yes, what can you do? Please make necessary changes to the supervised autoencoder network structure.

You can use the new model without overfitting for the following sections.

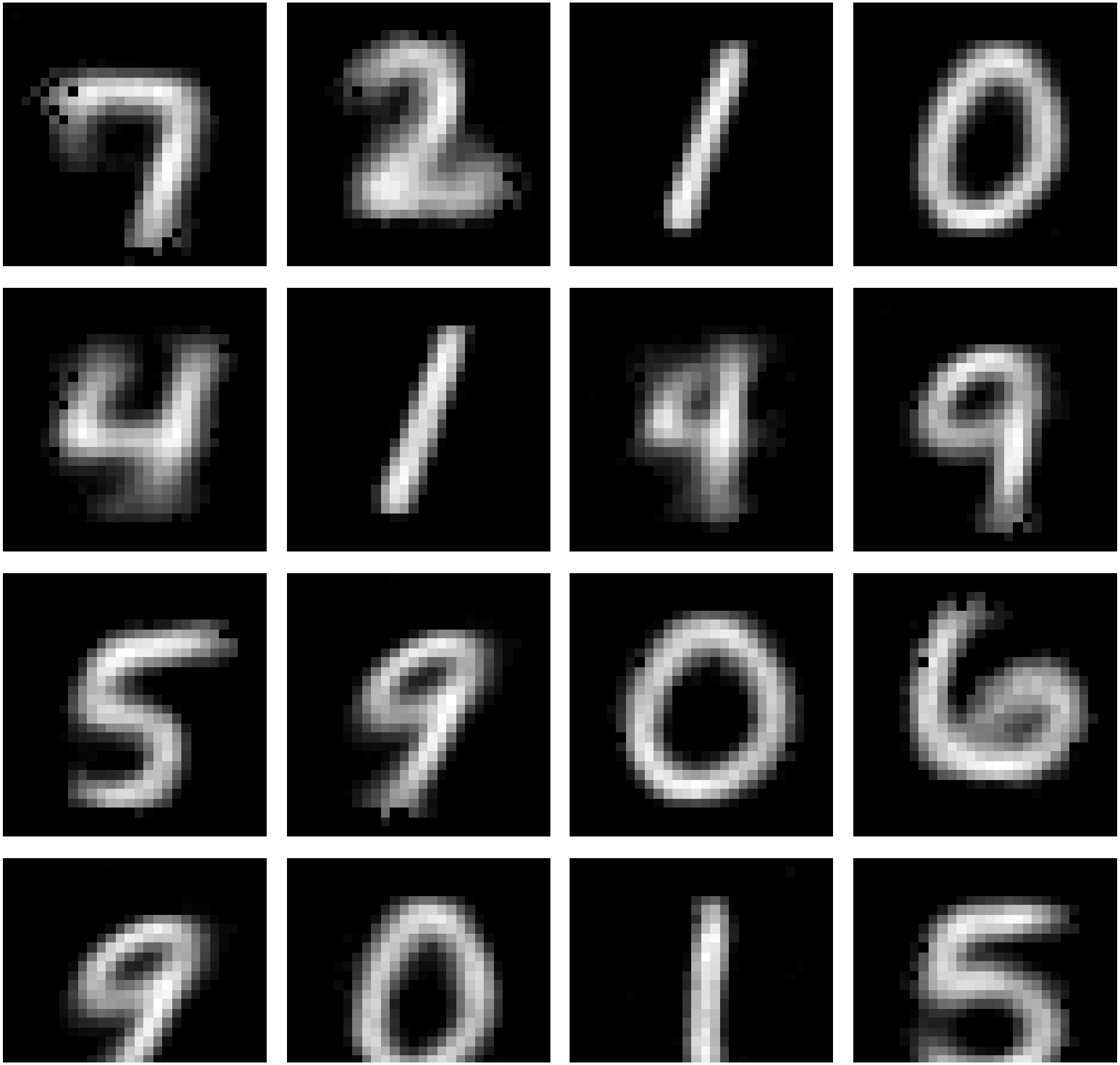
Yes model overfitted initially. With regularization on dense layers, overfitting got reduced and now model seems better.

4.3. Visualize the reconstructed test images

In [23]:
```
sae_output = sae.predict(x_test)[0].reshape((10000, 28, 28))

ROW = 5
COLUMN = 4
```

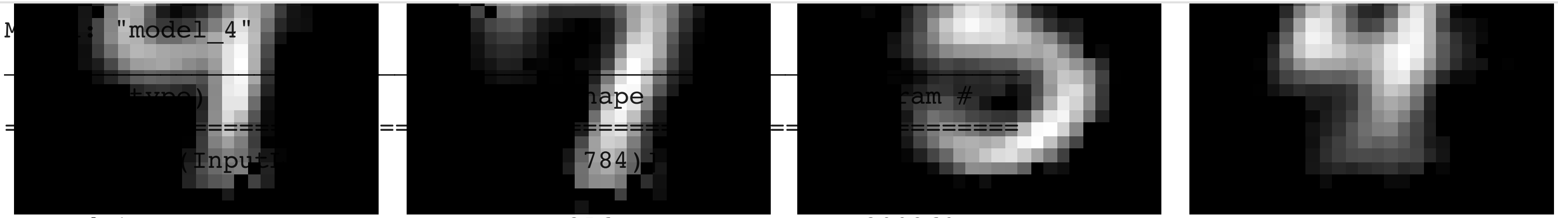
```
x = sae_output
fname = 'reconstruct_sae.pdf'

fig, axes = plt.subplots(nrows=ROW, ncols=COLUMN, figsize=(8, 10))
for ax, i in zip(axes.flat, np.arange(ROW*COLUMN)):
    image = x[i].reshape(28, 28)
    ax.imshow(image, cmap='gray')
    ax.axis('off')

plt.tight_layout()
plt.savefig(fname)
plt.show()
```

```
313/313 [==============================] - 1s 2ms/step
```



In [24]:

```python
# build the encoder model
sae_encoder = models.Model(input_img, bottleneck)
sae_encoder.summary()
```



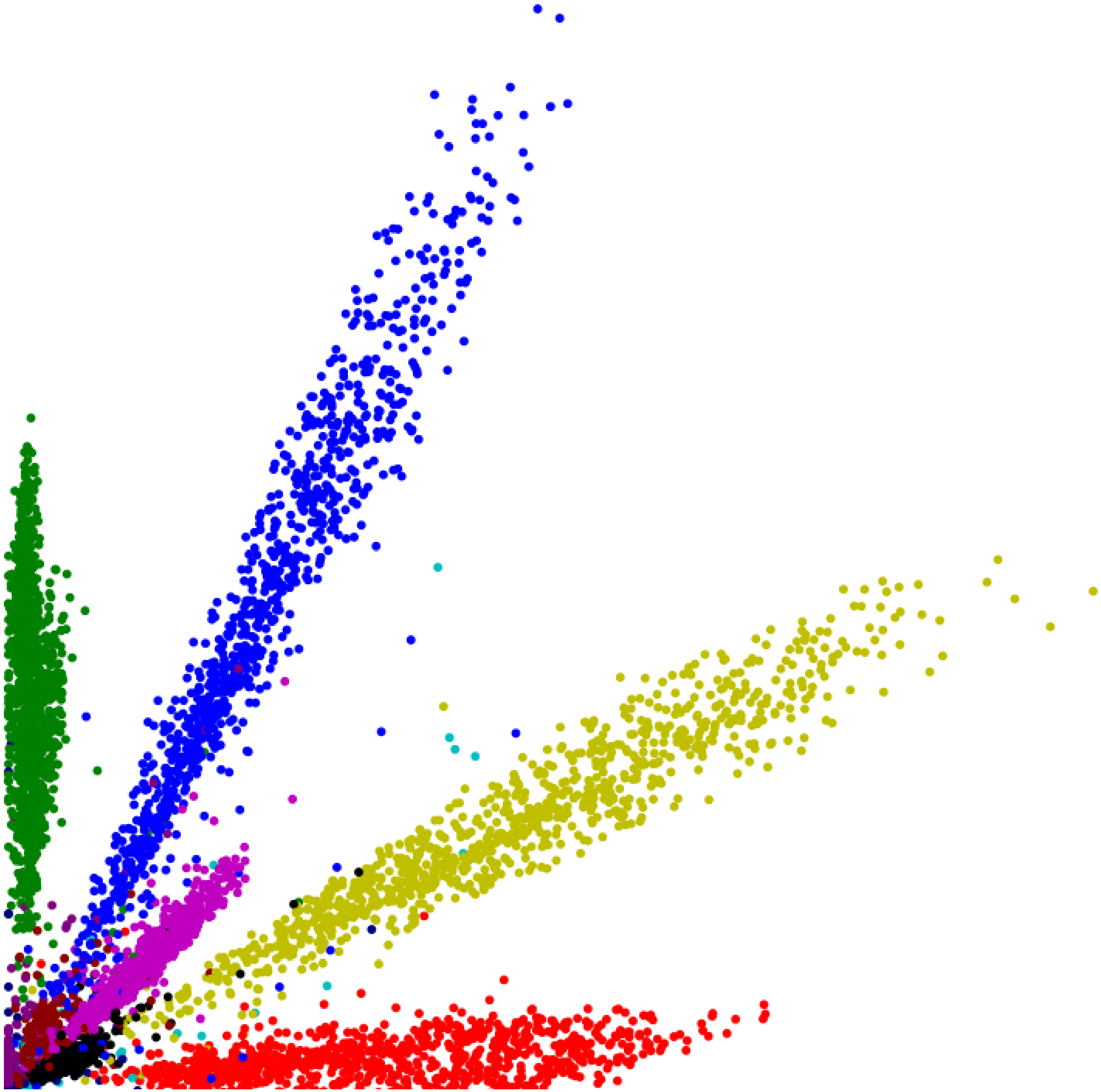
```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 (InputLayer)              [(None, 784)]
 encode1 (Dense)             (None, 256)               200960

 encode2 (Dense)             (None, 128)               32896

 encode3 (Dense)             (None, 64)                8256

 bottleneck (Dense)          (None, 2)                 130

=================================================================
Total params: 242,242
Trainable params: 242,242
Non-trainable params: 0
_________________________________________________________________
```

In [25]:

```python
# extract test features
encoded_test = sae_encoder.predict(x_test)
print('Shape of encoded_test: ' + str(encoded_test.shape))

colors = np.array(['r', 'g', 'b', 'm', 'c', 'k', 'y', 'purple', 'darkred', 'navy'])
colors_test = colors[y_test]


import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(8, 8))
plt.scatter(encoded_test[:, 0], encoded_test[:, 1], s=10, c=colors_test, edgecolors=colors_test)
plt.axis('off')
plt.tight_layout()
```

```
fname = 'sae_code.pdf'
plt.savefig(fname)
```

```
313/313 [==============================] - 0s 1ms/step
Shape of encoded_test: (10000, 2)
```



4.5. Are the learned low-dim features discriminative? (10 points)

To find the answer, lets train a classifier on the training set (the extracted 2-dim features) and evaluation on the validation and test set.

In [26]:

```
# extract 2D features from the training, validation, and test samples
f_tr = sae_encoder.predict(x_tr)
f_val = sae_encoder.predict(x_val)
f_te = sae_encoder.predict(x_test)
```

```
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
```

In [27]:

```
# build a classifier which takes the 2D features as input
from keras.layers import *
from keras import models

input_feat = Input(shape=(2,))

hidden1 = Dense(128, activation='relu')(input_feat)
do1 = Dropout(0.4)(hidden1)
hidden2 = Dense(128, activation='relu')(do1)
output = Dense(10, activation='softmax')(hidden2)

classifier = models.Model(input_feat, output)

classifier.summary()
```

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 2)]               0

 dense_3 (Dense)             (None, 128)               384

 dropout (Dropout)           (None, 128)               0

 dense_4 (Dense)             (None, 128)               16512
```

```
 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 18,186
Trainable params: 18,186
Non-trainable params: 0
_________________________________________________________________
```

In [28]:

```
classifier.compile(loss='categorical_crossentropy',
                   optimizer=optimizers.RMSprop(learning_rate=1E-4),
                   metrics=['acc'])

history = classifier.fit(f_tr, y_tr,
                         batch_size=32,
                         epochs=30,
                         validation_data=(f_val, y_val))
```

```
Epoch 1/30
313/313 [==============================] - 3s 5ms/step - loss: 4.3757 - acc: 0.3147 - val_loss: 1.4797 - val_ac
c: 0.4488
Epoch 2/30
313/313 [==============================] - 2s 5ms/step - loss: 1.7604 - acc: 0.4696 - val_loss: 1.1149 - val_ac
c: 0.5475
Epoch 3/30
313/313 [==============================] - 1s 4ms/step - loss: 1.1946 - acc: 0.5598 - val_loss: 0.9630 - val_ac
c: 0.5964
Epoch 4/30
313/313 [==============================] - 1s 4ms/step - loss: 0.8905 - acc: 0.6548 - val_loss: 0.8613 - val_ac
c: 0.6784
Epoch 5/30
313/313 [==============================] - 1s 4ms/step - loss: 0.7254 - acc: 0.7270 - val_loss: 0.7667 - val_ac
c: 0.7354
Epoch 6/30
313/313 [==============================] - 1s 4ms/step - loss: 0.5868 - acc: 0.8029 - val_loss: 0.7022 - val_ac
c: 0.7977
Epoch 7/30
313/313 [==============================] - 1s 4ms/step - loss: 0.4648 - acc: 0.8590 - val_loss: 0.6545 - val_ac
c: 0.8393
Epoch 8/30
313/313 [==============================] - 1s 4ms/step - loss: 0.3751 - acc: 0.8967 - val_loss: 0.5973 - val_ac
c: 0.9031
Epoch 9/30
313/313 [==============================] - 1s 4ms/step - loss: 0.2937 - acc: 0.9354 - val_loss: 0.5787 - val_ac
c: 0.9313
Epoch 10/30
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.2412 - acc: 0.9543 - val_loss: 0.5598 - val_ac
c: 0.9369
Epoch 11/30
313/313 [==============================] - 1s 5ms/step - loss: 0.1931 - acc: 0.9693 - val_loss: 0.5523 - val_ac
c: 0.9415
Epoch 12/30
313/313 [==============================] - 1s 4ms/step - loss: 0.1592 - acc: 0.9780 - val_loss: 0.5465 - val_ac
c: 0.9486
Epoch 13/30
313/313 [==============================] - 1s 4ms/step - loss: 0.1298 - acc: 0.9834 - val_loss: 0.5458 - val_ac
c: 0.9490
Epoch 14/30
313/313 [==============================] - 1s 4ms/step - loss: 0.1021 - acc: 0.9878 - val_loss: 0.5515 - val_ac
c: 0.9493
Epoch 15/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0831 - acc: 0.9914 - val_loss: 0.5730 - val_ac
c: 0.9494
Epoch 16/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0706 - acc: 0.9904 - val_loss: 0.5796 - val_ac
c: 0.9495
Epoch 17/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0605 - acc: 0.9915 - val_loss: 0.6037 - val_ac
c: 0.9497
Epoch 18/30
313/313 [==============================] - 2s 5ms/step - loss: 0.0501 - acc: 0.9935 - val_loss: 0.6136 - val_ac
c: 0.9500
Epoch 19/30
313/313 [==============================] - 1s 5ms/step - loss: 0.0443 - acc: 0.9938 - val_loss: 0.6299 - val_ac
c: 0.9498
Epoch 20/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0364 - acc: 0.9953 - val_loss: 0.6428 - val_ac
c: 0.9506
Epoch 21/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0330 - acc: 0.9950 - val_loss: 0.6690 - val_ac
c: 0.9500
Epoch 22/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0242 - acc: 0.9963 - val_loss: 0.6850 - val_ac
c: 0.9500
Epoch 23/30
313/313 [==============================] - 1s 5ms/step - loss: 0.0294 - acc: 0.9950 - val_loss: 0.7002 - val_ac
c: 0.9505
Epoch 24/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0230 - acc: 0.9970 - val_loss: 0.7192 - val_ac
c: 0.9506
Epoch 25/30
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0226 - acc: 0.9964 - val_loss: 0.7345 - val_ac
c: 0.9507
Epoch 26/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0182 - acc: 0.9970 - val_loss: 0.7491 - val_ac
c: 0.9512
Epoch 27/30
313/313 [==============================] - 2s 5ms/step - loss: 0.0196 - acc: 0.9963 - val_loss: 0.7715 - val_ac
c: 0.9507
Epoch 28/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0165 - acc: 0.9967 - val_loss: 0.7746 - val_ac
c: 0.9503
Epoch 29/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0174 - acc: 0.9970 - val_loss: 0.7965 - val_ac
c: 0.9506
Epoch 30/30
313/313 [==============================] - 1s 4ms/step - loss: 0.0178 - acc: 0.9969 - val_loss: 0.8025 - val_ac
c: 0.9511
```

Remark: (10 points)

The validation accuracy must be above 90%. It means the low-dim features learned by the supervised autoencoder are very effective.

In [29]:

```
# evaluate your model on the never-seen-before test data
# write your code here:

# will use f_te data which we never used for training
loss_and_accuracy = classifier.evaluate(f_te, y_test_vec)

print('loss = ' + str(loss_and_accuracy[0]))
print('accuracy = ' + str(loss_and_accuracy[1]))
```

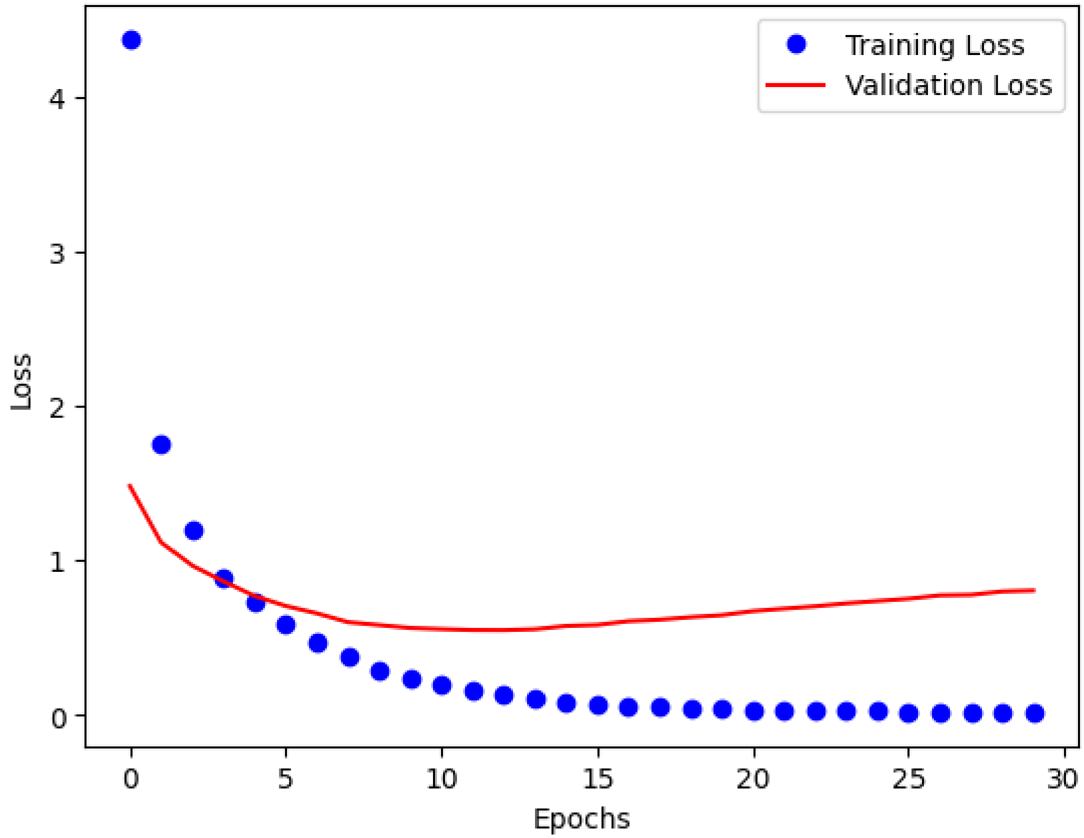
```
313/313 [==============================] - 1s 2ms/step - loss: 0.8722 - acc: 0.9553
loss = 0.8722113966941833
accuracy = 0.9552999973487854
```

Accuracy is 95% which is a good classifcation aaccuracy.

In [30]:

```
import matplotlib.pyplot as plt
%matplotlib inline

loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
epochs = range(len(loss))

plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Now the model is not overfitting and we got 95% accuracy.