ECE 637 Deep Learning Lab Exercises

Name: Alim Karimi

Section 1

Exercise 1.1

1. Create two lists, `A` and `B`: `A` contains 3 arbitrary numbers and `B` contains 3 arbitrary strings.
2. Concatenate two lists into a bigger list and name that list `C`.
3. Print the first element in `C`.
4. Print the second last element in `C` via negative indexing.
5. Remove the second element of `A` from `C`.
6. Print `C` again.

In [19]:
```
#  ----------- YOUR CODE -----------
import numpy as np

#1
A = [1,2,3]
B = ['hi', 'hello', 'bye']

#2 - concatenate lists
C = A + B

#3 print first element in C
print(C[0])

#4 - print second last element in C via negative indexing
print(C[-2])

#5- remove the second element of A from C:
rm = C.pop(-2)

#6 - print C again
print(C) #hello is not in the list now
```

```
1
hello
[1, 2, 3, 'hi', 'bye']
```

Exercise 1.2

In this exercise, you will use a low-pass IIR filter to remove noise from a sine-wave signal.

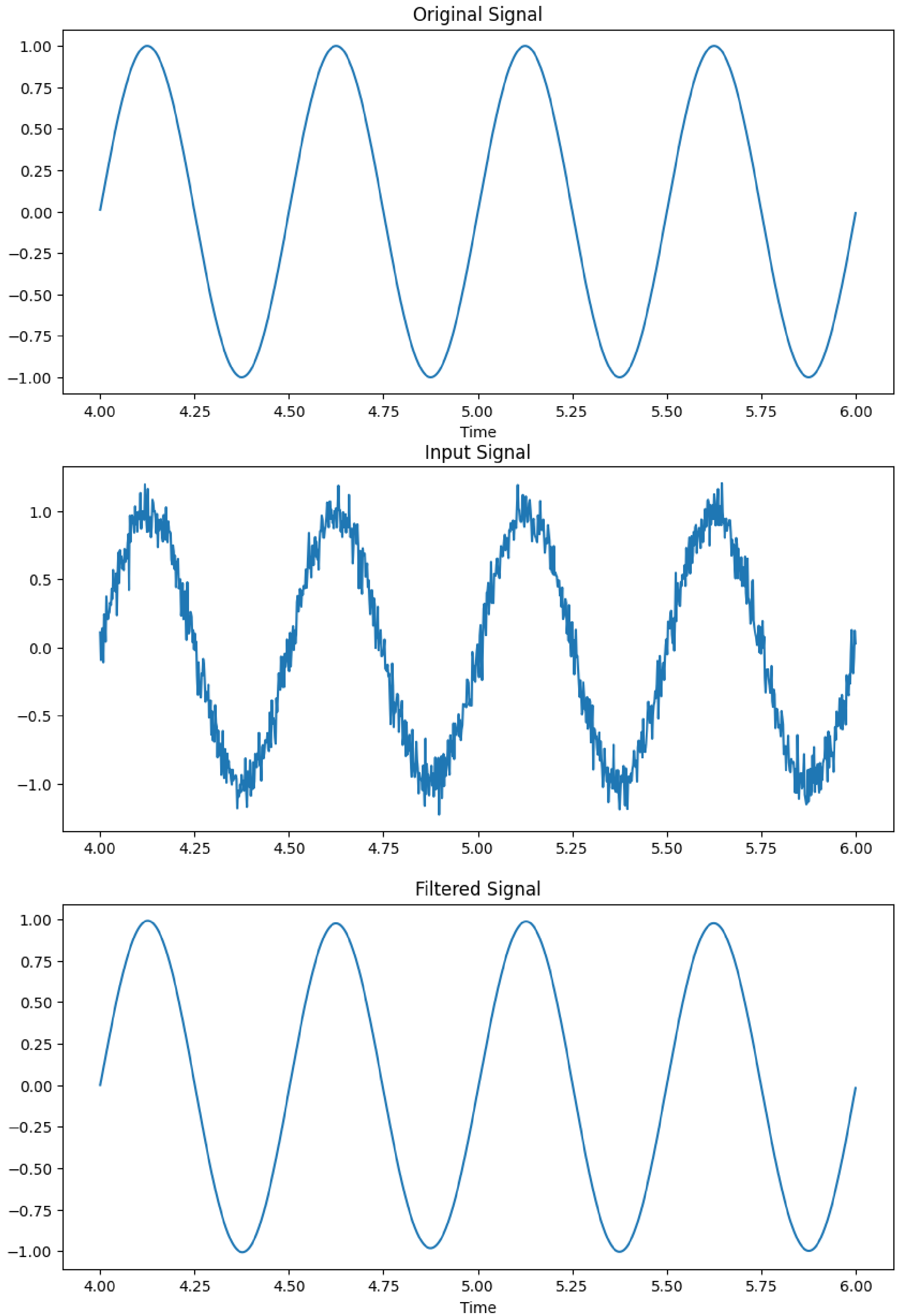
You should organize your plots in a 3x1 subplot format.

1. Generate a discrete-time signal, `x`, by sampling a 2Hz continuous time sine wave signal with peak amplitude 1 from time 0s to 10s and at a sampling frequency of 500 Hz. Display the signal, `x`, from time 4s to 6s in the first row of a 3x1 subplot with the title "original signal".

2. Add Gaussian white random noise with 0 mean and standard deviation 0.1 to `x` and call it `x_n`. Display `x_n` from 4s to 6s on the second row of the subplot with the title "input signal".

3. Design a low-pass butterworth IIR filter of order 5 with a cut-off frequency of 4Hz, designed to filter out the noise. Hint: Use the signal.butter function and note that the frequencies are relative to the Nyquist frequency. Apply the IIR filter to `x_n`, and name the output `y`. Hint: Use signal.filtfilt function. Plot `y` from 4s to 6s on the third row of the subplot with the title "filtered signal".

In [34]:
```
import numpy as np                       # import the numpy packages and use a sh
import matplotlib.pyplot as plt          # again import the matplotlib's pyplot p
from scipy import signal                 # import a minor package signal from sci
plt.figure(figsize=(10, 15))             # fix the plot size

#  ----------- YOUR CODE -----------
#part 1
t = np.linspace(0, 10, 500 * 10) #sampling freq of 500 Hz for 10 seconds means
x = np.sin(2 * np.pi * 2 *t)
plt.subplot(3,1,1)
plt.plot(t[2000:3000], x[2000:3000])
plt.xlabel('Time')

plt.title("Original Signal")


#part 2
mu = 0 #mean
sigma = 0.1 #std dev
noise = np.random.randn(500 * 10)*sigma + mu
x_n = x + noise
plt.subplot(3,1,2)
plt.plot(t[2000:3000], x_n[2000:3000])
plt.title("Input Signal")


#part 3
#Nyquist is half of the sampling rate (i.e, 0.5 * 500 = 250 Hz)
b, a = signal.butter(N = 5, Wn = 4, btype='low', analog=False, output='ba', fs=
#note: can also do b, a = signal.butter(N = 5, Wn = (4 / Nyquist_freq), btype=
#but in this case note that Wn is between 0 and 1 and relative to Nyquist
plt.subplot(3,1,3)
y = signal.filtfilt(b, a, x_n)
plt.plot(t[2000:3000], y[2000:3000])
plt.title("Filtered Signal")
plt.xlabel('Time')
plt.show()
```


Original Signal
Time
Input Signal
Filtered Signal
Time

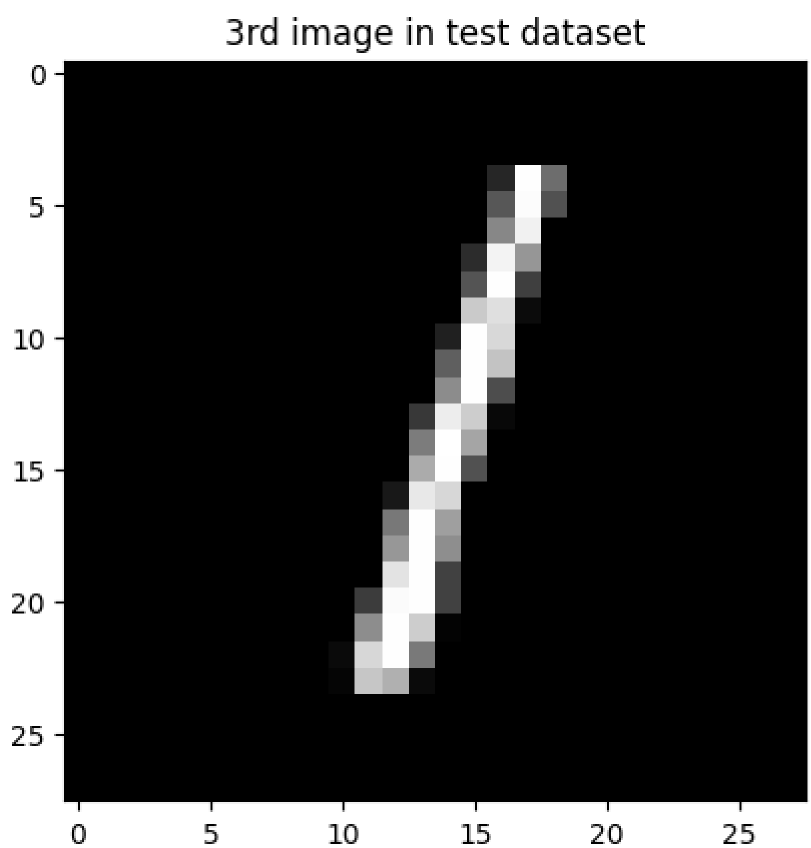
Section 2

Exercise 2.1

- Plot the third image in the test data set
- Find the correspoding label for the this image and make it the title of the figure

In [ ]:
```
import keras
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

#  ----------- YOUR CODE -----------

#Plot the 3rd image in the Test dataset
#print(test_images.shape)
plt.imshow(test_images[2], cmap='gray')
plt.title("3rd image in test dataset")

print("label for 3rd image in test dataset is:", test_labels[2])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
ets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
label for 3rd image in test dataset is: 1
```



Exercise 2.2

It is usually helpful to have an accuracy plot as well as a loss value plot to get an intuitive sense of how effectively the model is being trained.

- Add code to this example for plotting two graphs with the following requirements:
  - Use a 1x2 subplot with the left subplot showing the loss function and right subplot showing the accuracy.
  - For each graph, plot the value with respect to epochs. Clearly label the x-axis, y-axis and the title.

(Hint: The value of of loss and accuracy are stored in the `hist` variable. Try to print out `hist.history` and `his.history.keys()`.)

In [35]:

```
import keras
from keras.datasets import mnist
from keras import models
from keras import layers
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

network = models.Sequential()
network.add(layers.Flatten(input_shape=(28, 28, 1))) #network takes in a 28*28
network.add(layers.Dense(512, activation='relu')) #takes the 784 --> 512, appl
network.add(layers.Dense(10, activation='softmax')) #takes the 512 --> 10, appl

network.summary()

network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=[

train_images_nor = train_images.astype('float32') / 255 #rescale data to 0-1
test_images_nor = test_images.astype('float32') / 255 #rescale data to 0-1 (fro

train_labels_cat = to_categorical(train_labels) #one hot encoded label (i.e 3 i
test_labels_cat = to_categorical(test_labels)

hist = network.fit(train_images_nor, train_labels_cat, epochs=20, batch_size=12

test_loss, test_acc = network.evaluate(test_images_nor, test_labels_cat)
print('test_accuracy:', test_acc)
```

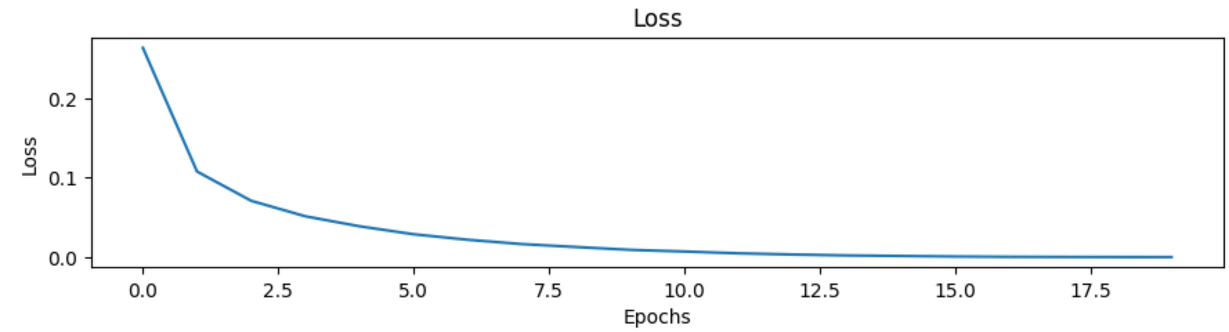
```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 784)               0

 dense_12 (Dense)            (None, 512)               401920

 dense_13 (Dense)            (None, 10)                5130

=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 2s 4ms/step - loss: 0.2627 - accura
cy: 0.9236
Epoch 2/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1077 - accura
cy: 0.9680
Epoch 3/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0709 - accura
cy: 0.9787
Epoch 4/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0515 - accura
cy: 0.9843
Epoch 5/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0390 - accura
cy: 0.9881
Epoch 6/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0290 - accura
cy: 0.9915
Epoch 7/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0221 - accura
cy: 0.9935
Epoch 8/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0166 - accura
cy: 0.9955
Epoch 9/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0130 - accura
cy: 0.9966
Epoch 10/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0094 - accura
cy: 0.9979
Epoch 11/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0073 - accura
cy: 0.9984
Epoch 12/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0050 - accura
cy: 0.9990
Epoch 13/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0036 - accura
cy: 0.9993
Epoch 14/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0024 - accura
cy: 0.9997
Epoch 15/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0017 - accura
cy: 0.9998
```

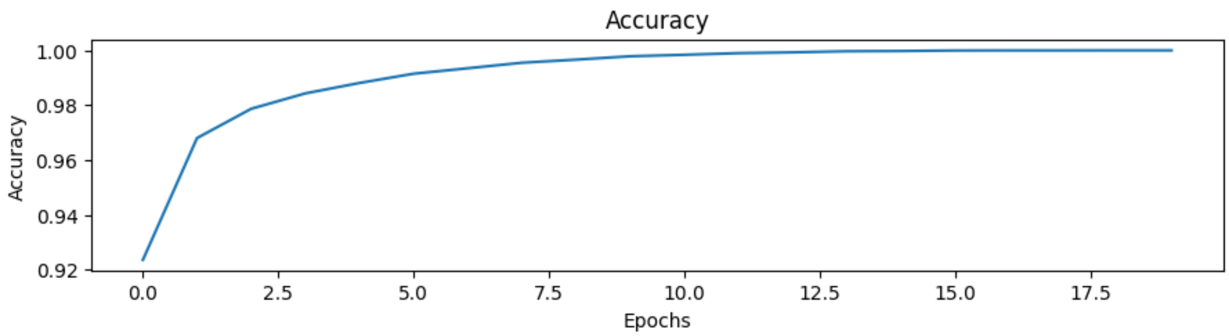
```
Epoch 16/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0011 - accura
cy: 0.9999
Epoch 17/20
469/469 [==============================] - 2s 5ms/step - loss: 6.5604e-04 - ac
curacy: 1.0000
Epoch 18/20
469/469 [==============================] - 3s 6ms/step - loss: 4.9241e-04 - ac
curacy: 1.0000
Epoch 19/20
469/469 [==============================] - 2s 4ms/step - loss: 3.8513e-04 - ac
curacy: 1.0000
Epoch 20/20
469/469 [==============================] - 1s 3ms/step - loss: 3.2693e-04 - ac
curacy: 1.0000
313/313 [==============================] - 1s 2ms/step - loss: 0.0672 - accura
cy: 0.9829
test_accuracy: 0.9829000234603882
```

In [36]:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(2, figsize=(10,6))
fig.tight_layout(pad=5.0)


#  ----------- YOUR CODE -----------
print(hist.history)
print(hist.history.keys())
ax[0].plot(hist.history['loss'])
ax[0].set_title("Loss")
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Loss")



ax[1].plot(hist.history['accuracy'])
ax[1].set_title("Accuracy")
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Accuracy")

plt.show()
```

```
{'loss': [0.26269108057022095, 0.10765278339385986, 0.07094603031873703, 0.051
503609865903854, 0.038977112621068954, 0.02895783632993698, 0.0220700260251760
5, 0.016626844182610512, 0.012956601567566395, 0.009391698986291885, 0.0073214
382864534855, 0.005036741495132446, 0.003593217581510544, 0.00237691565416753
3, 0.0017094099894165993, 0.0010693982476368546, 0.0006560396286658943, 0.0004
92409395519644, 0.00038513014442287385, 0.00032693208777345717], 'accuracy':
[0.9235666394233704, 0.9679999947547913, 0.978683352470398, 0.984300017356872
6, 0.988099992275238, 0.9914833307266235, 0.9934666752815247, 0.99548333883285
52, 0.9966166615486145, 0.9978500008583069, 0.9983833432197571, 0.998983323574
0662, 0.9992833137512207, 0.9996833205223083, 0.9997666478157043, 0.9999499917
030334, 0.9999833106994629, 0.9999833106994629, 1.0, 1.0]}
dict_keys(['loss', 'accuracy'])
```





Exercise 2.3

Use the dense network from Section 2 as the basis to construct of a deeper network with

- 5 dense hidden layers with dimensions [512, 256, 128, 64, 32] each of which uses a ReLU non-linearity

Question: Will the accuracy on the testing data always get better if we keep making the neural network larger?

No, it doesn't necessarily get better - it is very close to the accuracy of the smaller network, but actually a little bit less accurate in this case.

In [37]:
```
import keras
from keras import models
from keras import layers

#  ----------- YOUR CODE -----------
network = models.Sequential()
network.add(layers.Flatten(input_shape=(28, 28, 1))) #network takes in a 28*28 
network.add(layers.Dense(512, activation='relu')) #takes the 784 --> 512, appl
network.add(layers.Dense(256, activation='relu'))
network.add(layers.Dense(128, activation='relu'))
network.add(layers.Dense(64, activation='relu'))
network.add(layers.Dense(32, activation='relu'))
network.add(layers.Dense(10, activation='softmax')) #takes the 512 --> 10, appl

network.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_5 (Flatten)         (None, 784)               0         
                                                                 
 dense_14 (Dense)            (None, 512)               401920    
                                                                 
 dense_15 (Dense)            (None, 256)               131328    
                                                                 
 dense_16 (Dense)            (None, 128)               32896     
                                                                 
 dense_17 (Dense)            (None, 64)                8256      
                                                                 
 dense_18 (Dense)            (None, 32)                2080      
                                                                 
 dense_19 (Dense)            (None, 10)                330       
                                                                 
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

In [38]:
```python
import keras
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))


network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=[

train_images_nor = train_images.astype('float32') / 255
test_images_nor = test_images.astype('float32') / 255

train_labels_cat = to_categorical(train_labels)
test_labels_cat = to_categorical(test_labels)

hist = network.fit(train_images_nor, train_labels_cat, epochs=20, batch_size=12

test_loss, test_acc = network.evaluate(test_images_nor, test_labels_cat)
print('test_accuracy:', test_acc)
```

```
Epoch 1/20
469/469 [==============================] - 4s 4ms/step - loss: 0.2973 - accura
cy: 0.9097
Epoch 2/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1035 - accura
cy: 0.9686
Epoch 3/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0675 - accura
cy: 0.9797
Epoch 4/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0517 - accura
cy: 0.9841
Epoch 5/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0393 - accura
cy: 0.9879
Epoch 6/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0289 - accura
cy: 0.9913
Epoch 7/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0260 - accura
cy: 0.9924
Epoch 8/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0217 - accura
cy: 0.9936
Epoch 9/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0175 - accura
cy: 0.9948
Epoch 10/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0158 - accura
cy: 0.9949
Epoch 11/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0144 - accura
cy: 0.9958
Epoch 12/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0112 - accura
cy: 0.9966
Epoch 13/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0104 - accura
cy: 0.9968
Epoch 14/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0084 - accura
cy: 0.9975
Epoch 15/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0111 - accura
cy: 0.9974
Epoch 16/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0075 - accura
cy: 0.9981
Epoch 17/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0067 - accura
cy: 0.9980
Epoch 18/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0073 - accura
cy: 0.9978
Epoch 19/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0057 - accura
cy: 0.9985
Epoch 20/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0063 - accura
cy: 0.9984
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1180 - accura
cy: 0.9841
test_accuracy: 0.9840999841690063
```

Section 3

Exercise 3.1

In this exercise, you will access the relationship between the feature extraction layer and classification layer. The example above uses two sets of convolutional layers and pooling layers in the feature extraction layer and two dense layers in the classification layers. The overall performance is around 98% for both training and test dataset. In this exercise, try to create a similar CNN network with the following requirements:

- Achieve the overall accuracy higher than 99% for training and testing dataset.
- Keep the total number of parameters used in the network lower than 100,000.

In [39]:

```
import keras
from keras import models
from keras import layers

network = models.Sequential()

#  ----------- YOUR CODE -----------

# ---- Feature extraction section
# First Layer
network.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28, 1
network.add(layers.MaxPooling2D((2, 2)))
# Second Layer
network.add(layers.Conv2D(32, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
#add a third layer:
# network.add(layers.Conv2D(64, (3,3), activation='relu'))
# network.add(layers.MaxPooling2D((2,2)))


# ---- Classification section
# Rearrange the data
network.add(layers.Flatten())
# Third Layer
#network.add(layers.Dense(512, activation='relu'))
#third layer
network.add(layers.Dense(64, activation = 'relu'))
# fifth Layer
network.add(layers.Dense(10, activation='softmax'))

network.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_8 (Conv2D)           (None, 26, 26, 16)        160       
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 13, 13, 16)       0         
 2D)                                                             
                                                                 
 conv2d_9 (Conv2D)           (None, 11, 11, 32)        4640      
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 5, 5, 32)         0         
 2D)                                                             
                                                                 
 flatten_6 (Flatten)         (None, 800)               0         
                                                                 
 dense_20 (Dense)            (None, 64)                51264     
                                                                 
 dense_21 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 56,714
Trainable params: 56,714
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images_nor = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images_nor = test_images.astype('float32') / 255

train_labels_cat = to_categorical(train_labels)
test_labels_cat = to_categorical(test_labels)

network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=[
network.fit(train_images_nor, train_labels_cat, epochs=6, batch_size=64)

test_loss, test_acc = network.evaluate(test_images_nor, test_labels_cat)
print('test_accuracy:', test_acc)
```

```
Epoch 1/6
938/938 [==============================] - 10s 4ms/step - loss: 0.1955 - accur
acy: 0.9421
Epoch 2/6
938/938 [==============================] - 4s 4ms/step - loss: 0.0585 - accura
cy: 0.9817
Epoch 3/6
938/938 [==============================] - 4s 4ms/step - loss: 0.0418 - accura
cy: 0.9875
Epoch 4/6
938/938 [==============================] - 4s 4ms/step - loss: 0.0323 - accura
cy: 0.9904
Epoch 5/6
938/938 [==============================] - 4s 4ms/step - loss: 0.0266 - accura
cy: 0.9918
Epoch 6/6
938/938 [==============================] - 4s 5ms/step - loss: 0.0228 - accura
cy: 0.9933
313/313 [==============================] - 1s 3ms/step - loss: 0.0363 - accura
cy: 0.9875
test_accuracy: 0.987500011920929
```

Section 4

Exercise 4.1

In this exercise you will need to create the entire neural network that does image denoising tasks. Try to mimic the code provided above and follow the structure as provided in the instructions below.

Task 1: Create the datasets

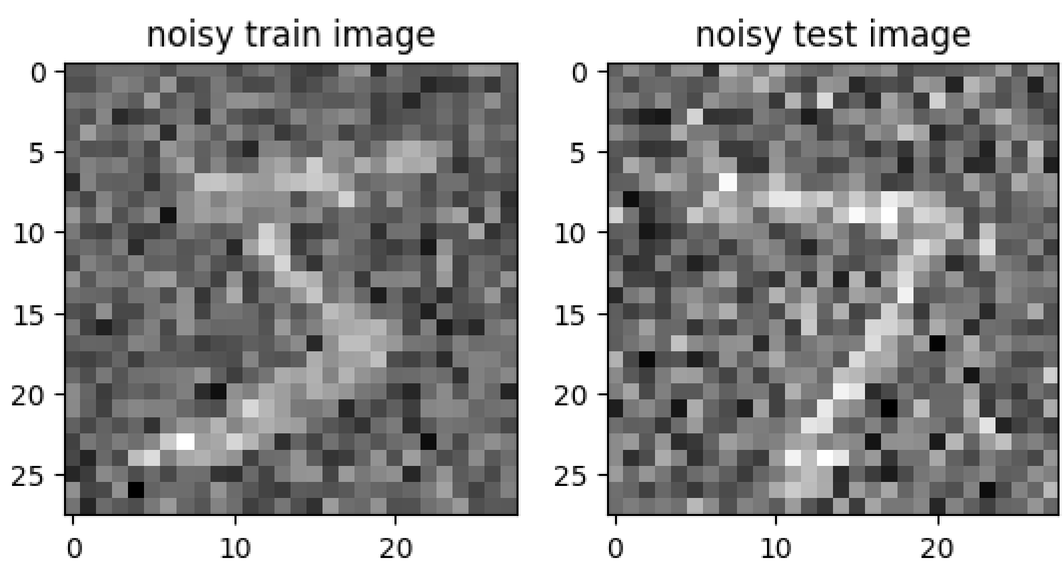
1. Import necessary packages
2. Load the MNIST data from Keras, and save the training dataset images as `train_images`, save the test dataset images as `test_images`
3. Add additive white gaussian noise to the train images as well as the test images and save the noisy images to `train_images_noisy` and `test_images_noisy` respectivly. The noise should have mean value 0, and standard deviation 0.4. (Hint: Use np.random.normal)
4. Show the first image in the training dataset as well as the test dataset (plot the images in 1 x 2 subplot form)

In [ ]:

```
#  ----------- YOUR CODE -----------
### Part 1, load packages:
from keras.datasets import mnist
from keras.utils import to_categorical


#part 2, load MNIST data from Keras, save the training dataset as train_images,
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
train_images = train_images.reshape((60000, 28, 28, 1))

train_images_nor = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images_nor = test_images.astype('float32') / 255

#Part 3, add additive white gaussian noise to the train images and test images.
noise_train = np.random.normal(loc=0, scale=0.4, size = (60000,28,28,1))
noise_test = np.random.normal(loc = 0, scale = 0.4, size = (10000, 28, 28, 1))
train_images_noisy = train_images_nor + noise_train
test_images_noisy = test_images_nor + noise_test

#part 4, show the first image in training and first image in test dataset

plt.subplot(1,2,1)
plt.imshow(train_images_noisy[0], cmap='gray')
plt.title("noisy train image")

plt.subplot(1,2,2)
plt.imshow(test_images_noisy[0], cmap='gray')
plt.title("noisy test image")
```

Out[ ]:
```
Text(0.5, 1.0, 'noisy test image')
```



Task 2: Create the neural network model

1. Create a sequential model called `encoder` with the following layers sequentially:
   - convolutional layer with `32` output channels, `3x3` kernel size, and the padding convention `'same'` with `'relu'` activition function.
   - max pooling layer with `2x2` kernel size
   - convolutional layer with `16` output channels, `3x3` kernel size, and the padding convention `'same'` with `'relu'` activition function.
   - max pooling layer with `2x2` kernel size
   - convolutional layer with `8` output channels, `3x3` kernel size, and the padding convention `'same'` with `'relu'` activition function and name the layer as `'convOutput'`.
   - flatten layer

- dense layer with output dimension as `encoding_dim` with `'relu'` activition function.

2. Create a sequential model called `decoder` with the following layers sequentially:
   - dense layer with the input dimension as `encoding_dim` and the output dimension as the product of the output dimenstions of the `'convOutput'` layer.
   - reshape layer that convert the tensor into the same shape as `'convOutput'`
   - convolutional layer with `8` output channels, `3x3` kernel size, and the padding convention `'same'` with `'relu'` activition function.
   - upsampling layer with `2x2` kernel size
   - convolutional layer with `16` output channels, `3x3` kernel size, and the padding convention `'same'` with `'relu'` activition function.
   - upsampling layer with `2x2` kernel size
   - convolutional layer with `32` output channels, `3x3` kernel size, and the padding convention `'same'` with `'relu'` activition function
   - convolutional layer with `1` output channels, `3x3` kernel size, and the padding convention `'same'` with `'sigmoid'` activition function
3. Create a sequential model called `autoencoder` with the following layers sequentially:
   - `encoder` model
   - `decoder` model

In [ ]:
```
#  ----------- YOUR CODE -----------
# Build Encoder
encoder = models.Sequential()
encoder.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_
encoder.add(layers.MaxPooling2D((2, 2),                  padding='same'))
encoder.add(layers.Conv2D(16, (3, 3),  activation='relu', padding='same'))
encoder.add(layers.MaxPooling2D((2, 2),                  padding='same'))
encoder.add(layers.Conv2D(8, (3, 3),  activation='relu', padding='same', name='
encoder.add(layers.Flatten())
encoding_dim = 32
encoder.add(layers.Dense(encoding_dim, activation='relu'))

# shape considerations
convShape = encoder.get_layer('convOutput').output_shape[1:]
print(convShape)
denseShape = convShape[0]*convShape[1]*convShape[2]
print(denseShape)
#Build Decoder
decoder = models.Sequential()
decoder.add(layers.Dense(denseShape, input_shape=(encoding_dim,)))
decoder.add(layers.Reshape(convShape))

decoder.add(layers.Conv2D(8, (3, 3),  activation='relu',    padding='same'))
decoder.add(layers.UpSampling2D((2, 2)))
decoder.add(layers.Conv2D(16, (3, 3),  activation='relu',    padding='same'))
decoder.add(layers.UpSampling2D((2, 2)))
decoder.add(layers.Conv2D(32, (3, 3), activation='relu',    padding='same'))
decoder.add(layers.Conv2D(1, (3, 3),  activation='sigmoid', padding='same'))
```

```
(7, 7, 8)
392
```

In [ ]:
```
encoder.summary()
decoder.summary()

autoencoder = models.Sequential()
autoencoder.add(encoder)
autoencoder.add(decoder)
autoencoder.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 28, 28, 32)        320       
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 14, 14, 32)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 14, 14, 16)        4624      
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 7, 7, 16)         0         
 2D)                                                             
                                                                 
 convOutput (Conv2D)         (None, 7, 7, 8)           1160      
                                                                 
 flatten_3 (Flatten)         (None, 392)               0         
                                                                 
 dense_10 (Dense)            (None, 32)                12576     
                                                                 
=================================================================
Total params: 18,680
Trainable params: 18,680
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_11 (Dense)            (None, 392)               12936     
                                                                 
 reshape (Reshape)           (None, 7, 7, 8)           0         
                                                                 
 conv2d_4 (Conv2D)           (None, 7, 7, 8)           584       
                                                                 
 up_sampling2d (UpSampling2D  (None, 14, 14, 8)        0         
 )                                                               
                                                                 
 conv2d_5 (Conv2D)           (None, 14, 14, 16)        1168      
                                                                 
 up_sampling2d_1 (UpSampling  (None, 28, 28, 16)       0         
 2D)                                                             
                                                                 
 conv2d_6 (Conv2D)           (None, 28, 28, 32)        4640      
                                                                 
 conv2d_7 (Conv2D)           (None, 28, 28, 1)         289       
                                                                 
=================================================================
Total params: 19,617
Trainable params: 19,617
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential_3 (Sequential)   (None, 32)                18680     
                                                                 
 sequential_4 (Sequential)   (None, 28, 28, 1)         19617     
                                                                 
```

```
=================================================================
Total params: 38,297
Trainable params: 38,297
Non-trainable params: 0
_________________________________________________________________
```

Task 3: Create the neural network model

Fit the model to the training data using the following hyper-parameters:

- `adam` optimizer
- `binary_crossentropy` loss function
- `20` training epochs
- batch size as `256`
- set `shuffle` as `True`

Compile the model and fit ...

In [ ]:
```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
history = autoencoder.fit(train_images_noisy, train_images_nor,
                epochs=20,
                batch_size=256,
                shuffle=True)
```

```
Epoch 1/20
235/235 [==============================] - 8s 12ms/step - loss: 0.1682
Epoch 2/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1385
Epoch 3/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1291
Epoch 4/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1236
Epoch 5/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1202
Epoch 6/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1177
Epoch 7/20
235/235 [==============================] - 3s 12ms/step - loss: 0.1159
Epoch 8/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1143
Epoch 9/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1132
Epoch 10/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1120
Epoch 11/20
235/235 [==============================] - 3s 14ms/step - loss: 0.1111
Epoch 12/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1102
Epoch 13/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1095
Epoch 14/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1088
Epoch 15/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1083
Epoch 16/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1076
Epoch 17/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1071
Epoch 18/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1067
Epoch 19/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1063
Epoch 20/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1057
```

Task 4: Create the neural network model (No need to write code, just run the following commands)

In [ ]:

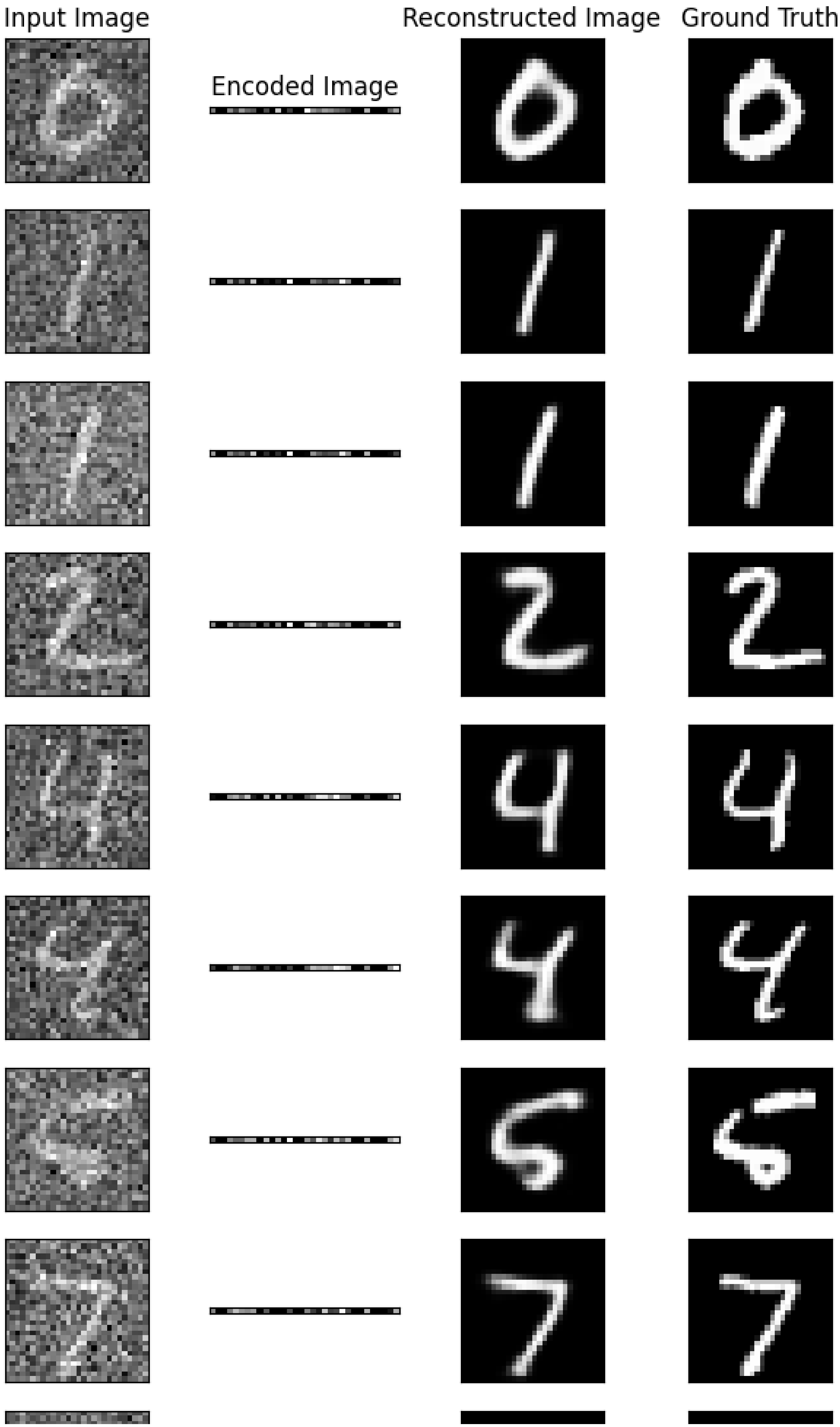
```
def showImages(input_imgs, encoded_imgs, output_imgs, size=1.5, groundTruth=Nor

    numCols = 3 if groundTruth is None else 4

    num_images = input_imgs.shape[0]

    encoded_imgs = encoded_imgs.reshape((num_images, 1, -1))


    plt.figure(figsize=((numCols+encoded_imgs.shape[2]/input_imgs.shape[2])*siz

    pltIdx = 0
    col = 0
    for i in range(0, num_images):
```

```
        col += 1
    # plot input image
        pltIdx += 1
        ax = plt.subplot(num_images, numCols, pltIdx)
        plt.imshow(input_imgs[i].reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        if col == 1:
            plt.title('Input Image')

    # plot encoding
        pltIdx += 1
        ax = plt.subplot(num_images, numCols, pltIdx)
        plt.imshow(encoded_imgs[i])
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        if col == 1:
            plt.title('Encoded Image')

    # plot reconstructed image
        pltIdx += 1
        ax = plt.subplot(num_images, numCols, pltIdx)
        plt.imshow(output_imgs[i].reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        if col == 1:
            plt.title('Reconstructed Image')

        if numCols == 4:
          # plot ground truth image
            pltIdx += 1
            ax = plt.subplot(num_images, numCols, pltIdx)
            plt.imshow(groundTruth[i].reshape(28, 28))
            plt.gray()
            ax.get_xaxis().set_visible(False)
            ax.get_yaxis().set_visible(False)

            if col == 1:
                plt.title('Ground Truth')

    plt.show()
```

In [ ]:

```
num_images = 10

input_labels = test_labels[0:num_images]
I = np.argsort(input_labels)

input_imgs = test_images_noisy[I]

encoded_imgs = encoder.predict(test_images_noisy[I])
output_imgs = decoder.predict(encoded_imgs)

showImages(input_imgs, encoded_imgs, output_imgs, size=1.5, groundTruth=test_in
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
```


Input Image
Encoded Image
Reconstructed Image
Ground Truth