ECE 637 Deep Learning Lab Exercises

95/100

Name: *Praneet Singh*

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.

```
#  ----------- YOUR CODE -----------
import random
import string

A = [random.randint(1,9999) for i in range(3)]
B = [''.join(random.choice(string.ascii_lowercase+string.ascii_uppercase) for i in range(rand

C = A + B

print(C)
print(C[0])
print(C[-2])
C.remove(A[1])
print(C)
```

```
[6037, 439, 3511, 'OYc', 'ZMG', 'FlXd']
6037
ZMG
[6037, 3511, 'OYc', 'ZMG', 'FlXd']
```

Exercise 1.2

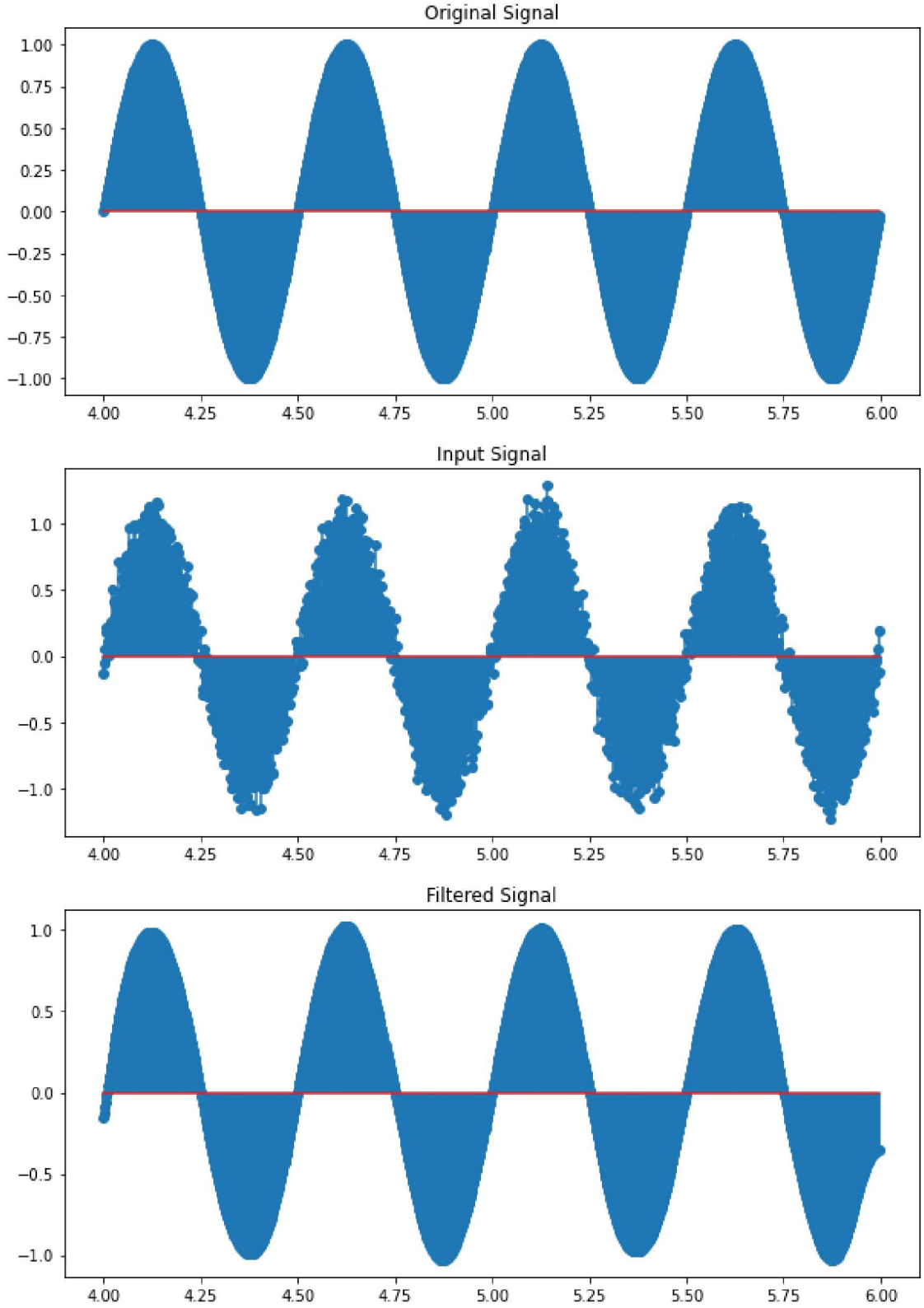
In this exercise, you will use a low-pass IIR filter to remove noise from a sine-wave signal.

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 pe a sampling frequency of 500 Hz. Display the signal, `x`, from time 4s to 6s in the first row of a 3x
2. Add Gaussian white random noise with 0 mean and standard deviation 0.1 to `x` and call it `x_n` . 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 fi signal.butter function and note that the frequencies are relative to the Nyquist frequency. Apply `y`. Hint: Use signal.filtfilt function. Plot `y` from 4s to 6s on the third row of the subplot with the t

```
import numpy as np                      # import the numpy packages and use a shorter alising
import matplotlib.pyplot as plt         # again import the matplotlib's pyplot packages
from scipy import signal                # import a minor package signal from scipy
         # fix the plot size

fig, (ax1,ax2,ax3) = plt.subplots(3, 1, figsize= (10,15))

#  ----------- YOUR CODE -----------
A=1
tmin = 4
tmax = 6
f=2
Ts = 1/500
t = np.arange(tmin, tmax, Ts)
x = A* np.sin(2*np.pi*f*t); # signal sampling
ax1.stem(t, x, use_line_collection=True)
ax1.set_title("Original Signal")


x_noise = np.random.normal(0,0.1,len(x)) + x
ax2.stem(t, x_noise, use_line_collection=True)
ax2.set_title("Input Signal")


b,a = signal.butter(5, 8, 'low',fs = 500, analog=False)
filtered = signal.filtfilt(b, a,x_noise)
ax3.stem(t, filtered, use_line_collection=True)
ax3.set_title("Filtered Signal")
```

```
Text(0.5, 1.0, 'Filtered Signal')
```



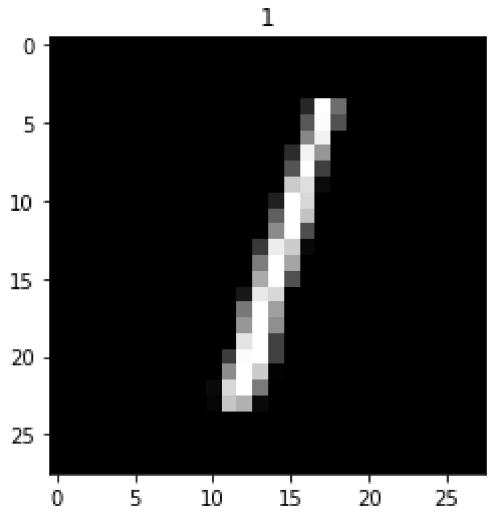
Section 2

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

```
%pylab inline
import matplotlib.pyplot as plt
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 -----------
plt.imshow(test_images[2].squeeze(),cmap="gray")
plt.title(test_labels[2])
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
/usr/local/lib/python3.6/dist-packages/IPython/core/magics/pylab.py:161: UserWarning: py
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```



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 ho

- 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 showin

- For each graph, plot the value with respect to epochs. Clearly label the x-axis, y-axis and th

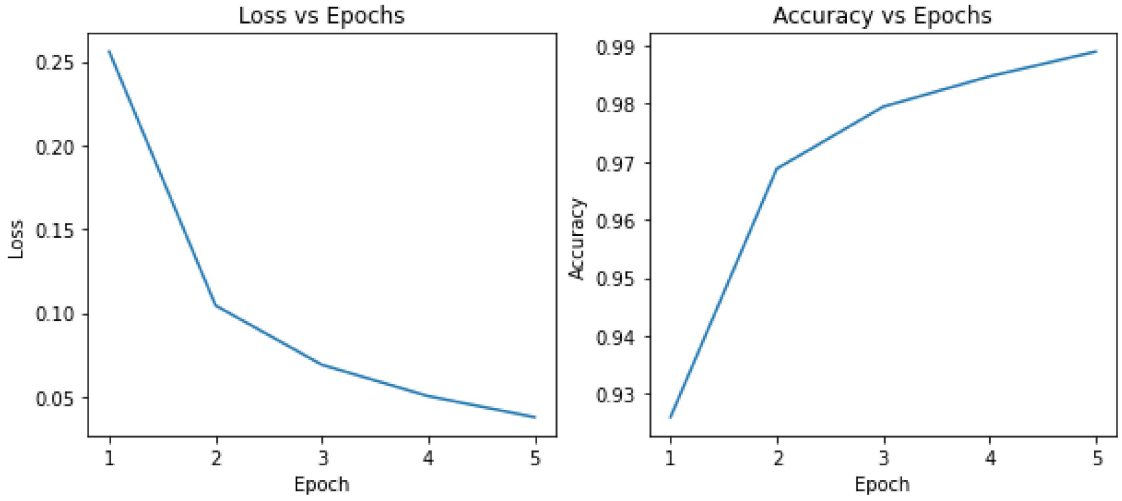
(Hint: The value of of loss and accuracy are stored in the `hist` variable. Try to print out `hist.history`

```
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.add(layers.Dense(512, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))

network.summary()

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

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=5, batch_size=128)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
60000/60000 [==============================] - 3s 51us/step - loss: 0.2558 - accuracy: 0
Epoch 2/5
60000/60000 [==============================] - 1s 22us/step - loss: 0.1045 - accuracy: 0
Epoch 3/5
60000/60000 [==============================] - 1s 21us/step - loss: 0.0691 - accuracy: 0
Epoch 4/5
60000/60000 [==============================] - 1s 21us/step - loss: 0.0504 - accuracy: 0
Epoch 5/5
60000/60000 [==============================] - 1s 21us/step - loss: 0.0379 - accuracy: 0
```

-5
Why is training accuracy zero?

```
import matplotlib.pyplot as plt

loss = hist.history['loss']
accuracy = hist.history['accuracy']
epochs = [i for i in range(1,6)]

fig, (ax1,ax2) = plt.subplots(1, 2, figsize= (10,4))

#  ----------- YOUR CODE -----------
ax1.plot(epochs,loss)
ax1.set_xlabel("Epoch")
ax1.set_ylabel("Loss")
ax1.set_title("Loss vs Epochs")

ax2.plot(epochs,accuracy)
ax2.set_xlabel("Epoch")
ax2.set_ylabel("Accuracy")
ax2.set_title("Accuracy vs Epochs")

plt.show()
```



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-lir

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

No, the model's accuracy on testing data might increase till the point we obtain the Goldilocks model or estimate the function). If we continue to increase the neural net size after this, the model will tend to ov testing data

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

#  ----------- YOUR CODE -----------
# network = ...
network = models.Sequential()
network.add(layers.Flatten(input_shape=(28, 28, 1)))
network.add(layers.Dense(512, activation='relu'))
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'))

network.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_2 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_4 (Dense)              (None, 256)               131328    
_________________________________________________________________
dense_5 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_6 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_7 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_8 (Dense)              (None, 10)                330       
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

```
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=['accuracy'])

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=5, batch_size=128)

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

```
Epoch 1/5
60000/60000 [==============================] - 2s 31us/step - loss: 0.2986 - accuracy: 0
Epoch 2/5
60000/60000 [==============================] - 2s 30us/step - loss: 0.1033 - accuracy: 0
Epoch 3/5
60000/60000 [==============================] - 2s 30us/step - loss: 0.0692 - accuracy: 0
Epoch 4/5
60000/60000 [==============================] - 2s 30us/step - loss: 0.0528 - accuracy: 0
Epoch 5/5
60000/60000 [==============================] - 2s 29us/step - loss: 0.0410 - accuracy: 0
10000/10000 [==============================] - 1s 59us/step
test_accuracy: 0.9789000153541565
```

Section 3

Exercise 3.1

In this exercise, you will access the relationship between the feature extraction layer and classificatio of convolutional layers and pooling layers in the feature extraction layer and two dense layers in the c performance is around 98% for both training and test dataset. In this exercise, try to create a similar C 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.

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

network = models.Sequential()

#  ----------- YOUR CODE -----------
network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1), padding ="s
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu', padding ="same"))
network.add(layers.MaxPooling2D((2, 2)))# Second Layer
network.add(layers.Conv2D(128, (3, 3), activation='relu', padding ="same"))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(256, (3, 3), activation='relu', padding ="same"))
network.add(layers.MaxPooling2D((2, 2)))


network.add(layers.Flatten())
network.add(layers.Dense(512, activation='relu'))
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(10, activation='softmax'))


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=['accuracy'])
network.fit(train_images_nor, train_labels_cat, epochs=5, batch_size=128)

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

```
Epoch 1/5
60000/60000 [==============================] - 4s 66us/step - loss: 0.3684 - accuracy: 0
Epoch 2/5
60000/60000 [==============================] - 4s 62us/step - loss: 0.0640 - accuracy: 0
Epoch 3/5
60000/60000 [==============================] - 4s 62us/step - loss: 0.0420 - accuracy: 0
Epoch 4/5
60000/60000 [==============================] - 4s 62us/step - loss: 0.0340 - accuracy: 0
Epoch 5/5
60000/60000 [==============================] - 4s 62us/step - loss: 0.0293 - accuracy: 0
10000/10000 [==============================] - 1s 73us/step
test_accuracy: 0.9908999800682068
```

Section 4

Exercise 4.1

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

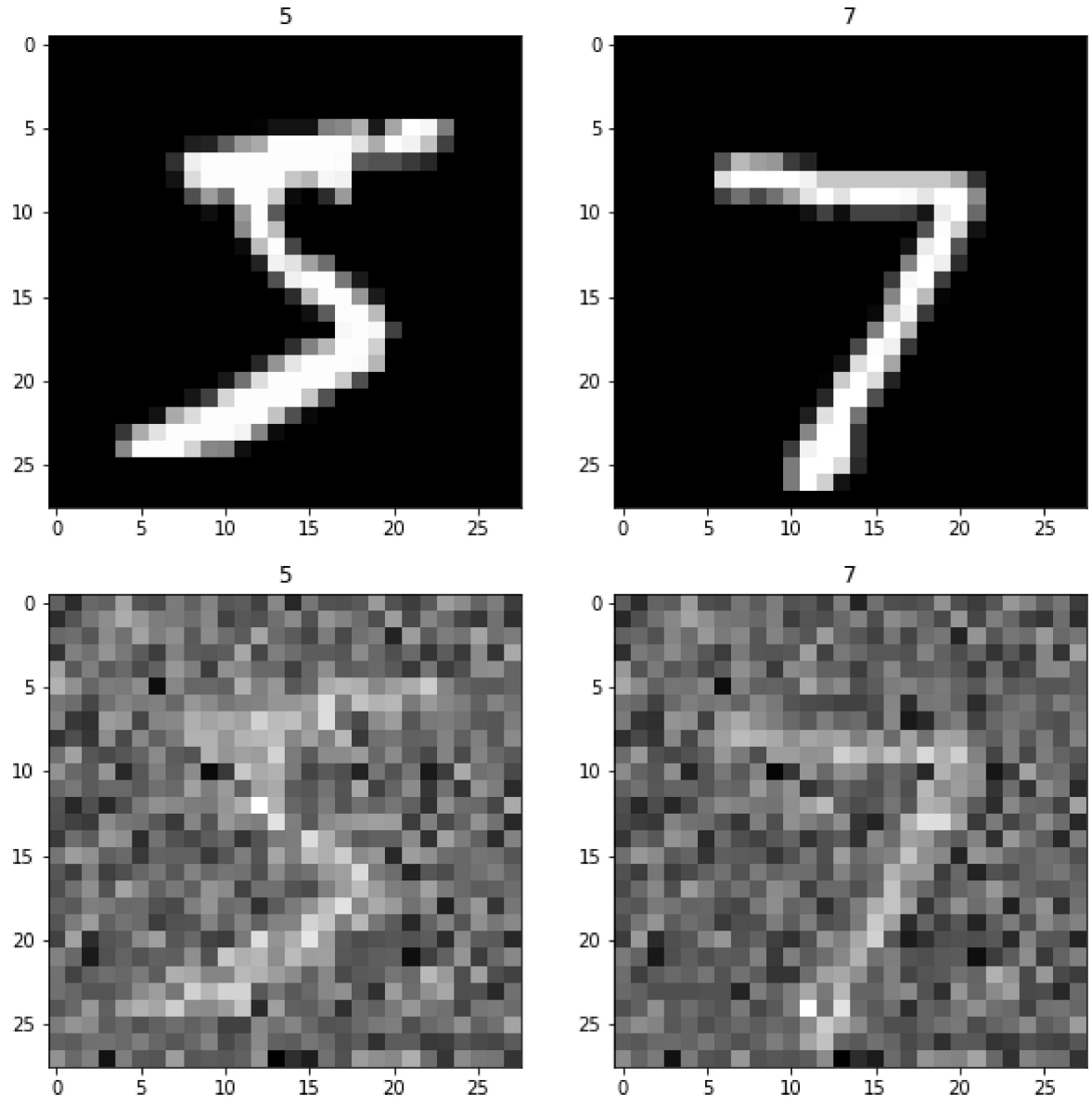
Task 1: Create the datasets

1. Import necessary packages
2. Load the MNIST data from Keras, and save the training dataset images as `train_images`, save t

3. Add additive white gaussian noise to the train images as well as the test images and save the n
   `test_images_noisy` respectivly. The noise should have mean value 0, and standard deviation 0.
4. Show the first image in the training dataset as well as the test dataset (plot the images in 1 x 2 s

```
#  ----------- YOUR CODE -----------
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))
train_images_nor = train_images.astype('float32') / 255
test_images_nor = test_images.astype('float32') / 255

noise = np.random.normal(0,0.4,(28,28,1))
noise = noise.reshape(28,28,1)

train_images_noisy = [i+noise for i in train_images_nor]
train_images_noisy = np.asarray(train_images_noisy)
test_images_noisy = [i+noise for i in test_images_nor]
test_images_noisy = np.asarray(test_images_noisy)

fig , (ax1,ax2) = plt.subplots(1,2,figsize=(10,5))
ax1.imshow(train_images[0].squeeze(),cmap="gray")
ax1.set_title(train_labels[0])
ax2.imshow(test_images[0].squeeze(),cmap="gray")
ax2.set_title(test_labels[0])

fig , (ax1,ax2) = plt.subplots(1,2,figsize=(10,5))
ax1.imshow(train_images_noisy[0].squeeze(),cmap="gray")
ax1.set_title(train_labels[0])
ax2.imshow(test_images_noisy[0].squeeze(),cmap="gray")
ax2.set_title(test_labels[0])
```

```
Text(0.5, 1.0, '7')
```



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
   - max pooling layer with `2x2` kernel size
   - convolutional layer with `16` output channels, `3x3` kernel size, and the padding convention
   - max pooling layer with `2x2` kernel size
   - convolutional layer with `8` output channels, `3x3` kernel size, and the padding convention ' 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 pr `'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 '
- upsampling layer with `2x2` kernel size
- convolutional layer with `16` output channels, `3x3` kernel size, and the padding convention
- upsampling layer with `2x2` kernel size
- convolutional layer with `32` output channels, `3x3` kernel size, and the padding convention
- convolutional layer with `1` output channels, `3x3` kernel size, and the padding convention '

3. Create a sequential model called `autoencoder` with the following layers sequentially:
   - `encoder` model
   - `decoder` model

```
#  ----------- YOUR CODE -----------
encoding_dim = 32

encoder = models.Sequential()
encoder.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=train_im
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='convOutput'))
encoder.add(layers.Flatten())
encoder.add(layers.Dense(encoding_dim, activation='relu'))
# shape considerations
convShape = encoder.get_layer('convOutput').output_shape[1:]
denseShape = convShape[0]*convShape[1]*convShape[2]


# 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'))

# concatenate the encoder and decoder
autoencoder = models.Sequential()
autoencoder.add(encoder)
autoencoder.add(decoder)
```

```
encoder.summary()
decoder.summary()
autoencoder.summary()
```

```
Model: "sequential_39"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_133 (Conv2D)          (None, 28, 28, 32)        320       
_________________________________________________________________
max_pooling2d_84 (MaxPooling (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_134 (Conv2D)          (None, 14, 14, 16)        4624      
_________________________________________________________________
max_pooling2d_85 (MaxPooling (None, 7, 7, 16)          0         
_________________________________________________________________
convOutput (Conv2D)          (None, 7, 7, 8)           1160      
_________________________________________________________________
flatten_27 (Flatten)         (None, 392)               0         
_________________________________________________________________
dense_95 (Dense)             (None, 32)                12576     
=================================================================
Total params: 18,680
Trainable params: 18,680
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_40"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_96 (Dense)             (None, 392)               12936     
_________________________________________________________________
reshape_7 (Reshape)          (None, 7, 7, 8)           0         
_________________________________________________________________
conv2d_135 (Conv2D)          (None, 7, 7, 8)           584       
_________________________________________________________________
up_sampling2d_13 (UpSampling (None, 14, 14, 8)         0         
_________________________________________________________________
conv2d_136 (Conv2D)          (None, 14, 14, 16)        1168      
_________________________________________________________________
up_sampling2d_14 (UpSampling (None, 28, 28, 16)        0         
_________________________________________________________________
conv2d_137 (Conv2D)          (None, 28, 28, 32)        4640      
_________________________________________________________________
conv2d_138 (Conv2D)          (None, 28, 28, 1)         289       
=================================================================
Total params: 19,617
Trainable params: 19,617
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_41"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
sequential_39 (Sequential)   (None, 32)                18680     
_________________________________________________________________
sequential_40 (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 ...

```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
history = autoencoder.fit(train_images_noisy, train_images_nor,
                epochs=20,
                batch_size=256,
                shuffle=True)
```

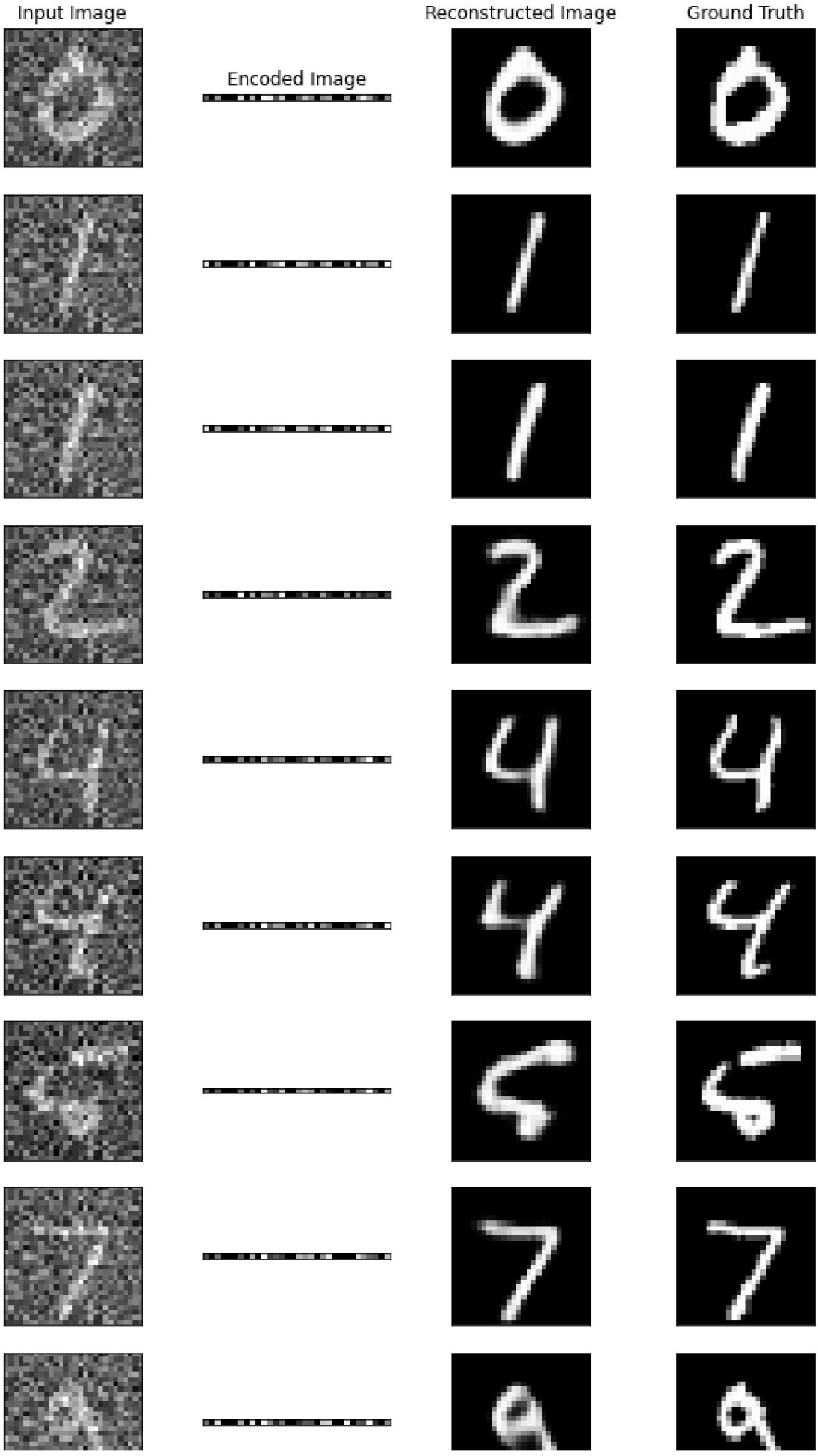
```
Epoch 1/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.2566
Epoch 2/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1409
Epoch 3/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1233
Epoch 4/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1151
Epoch 5/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1108
Epoch 6/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1079
Epoch 7/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1058
Epoch 8/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1040
Epoch 9/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1021
Epoch 10/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1003
Epoch 11/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0993
Epoch 12/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0984
Epoch 13/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0975
Epoch 14/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0968
Epoch 15/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0960
Epoch 16/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0956
Epoch 17/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0950
Epoch 18/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0946
Epoch 19/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0941
Epoch 20/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0936
```

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

```
def showImages(input_imgs, encoded_imgs, output_imgs, size=1.5, groundTruth=None):

  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])*size, num_images*si

  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()
```

```
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=2, groundTruth=test_images_nor[I])
```


Input Image
Encoded Image
Reconstructed Image
Ground Truth