ECE 637 Deep Learning Lab Exercises

Name: Alexandre Olivé Pellicer

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

#1
A = [1,2,3]
B = ['Alex', 'Olive', 'Pellicer']

#2
C = A + B

#3
print(C[0])

#4
print(C[-2])

#5
rm = C.pop(1)

#6
print(C)
```

```
1
Olive
[1, 3, 'Alex', 'Olive', 'Pellicer']
```

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.

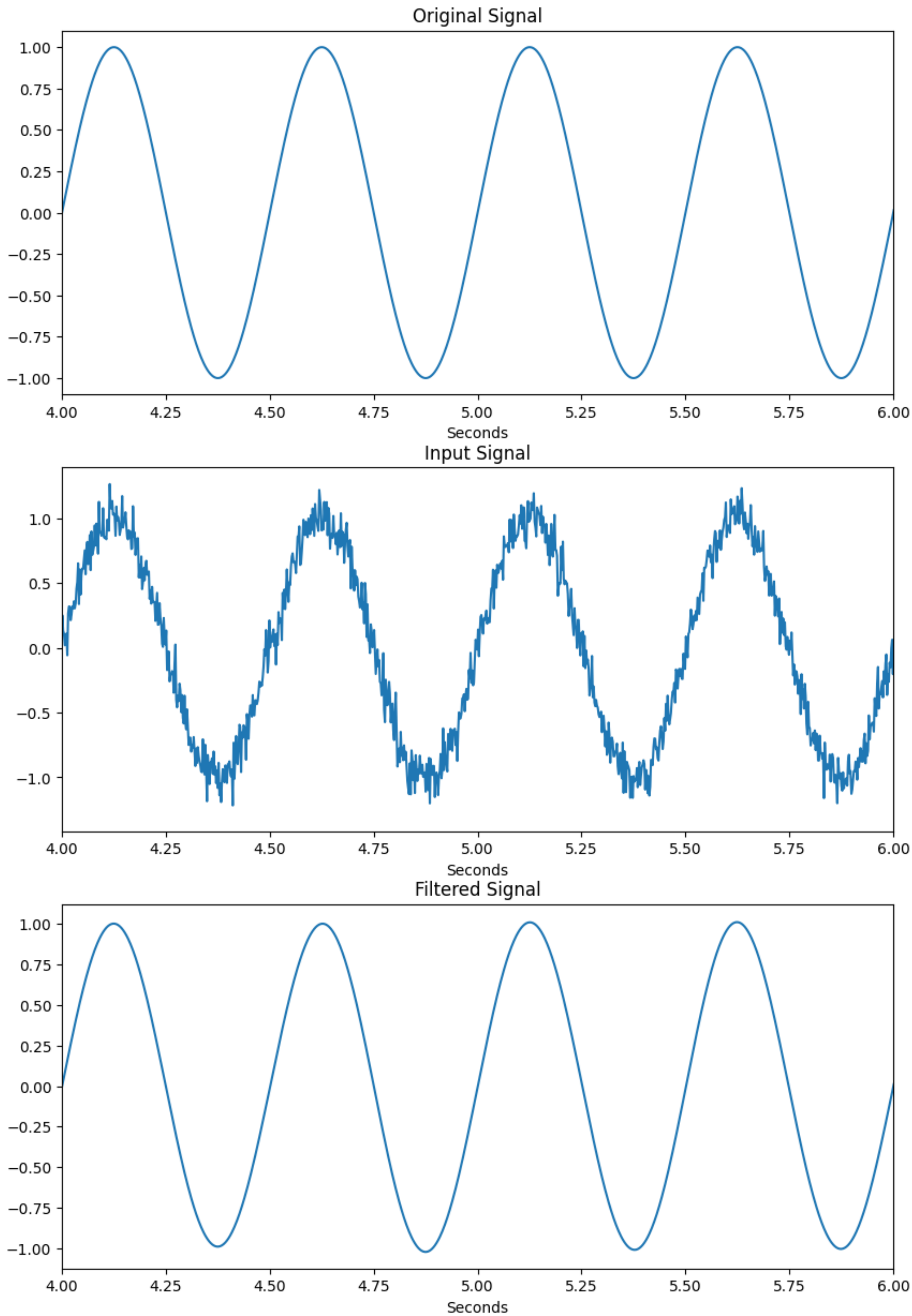
1. Generate a discrete-time signal, `x` , by sampling a 2Hz continuous time sine wave signal with peak amplitude 1 from time 0s to 10s and at a sampling frequency of 500 Hz. Display the signal, `x` , from time 4s to 6s in the first row of a 3x1 subplot with the title "original signal".

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

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

# Part 1
f = 2 # Hz
fs = 500 # sampling frequency
t_start = 0 # seconds
t_finish = 10 # seconds
num_samples = fs*(t_finish - t_start)
t = np.linspace(t_start, t_finish, num_samples)
x = np.sin(2*np.pi*f*t)
plt.subplot(3, 1, 1)
plt.plot(t, x)
plt.xlim([4, 6])
plt.title('Original Signal')
plt.xlabel('Seconds')

# Part 2
length = np.size(t);
n = np.random.randn(length)*0.1
x_n = x+n
plt.subplot(3, 1, 2)
plt.plot(t, x_n)
plt.xlim([4, 6])
plt.title('Input Signal')
plt.xlabel('Seconds')

# Part 3
cut_freq = 4
b, a = signal.butter(5, cut_freq, btype='low', fs=fs)
y = signal.filtfilt(b, a, x_n, padlen=3)
```

```
plt.subplot(3, 1, 3)
plt.plot(t, y)
plt.xlim([4, 6])
plt.title('Filtered Signal')
plt.xlabel('Seconds')

plt.show()
```


Original Signal
Seconds
Input Signal
Seconds
Filtered Signal
Seconds

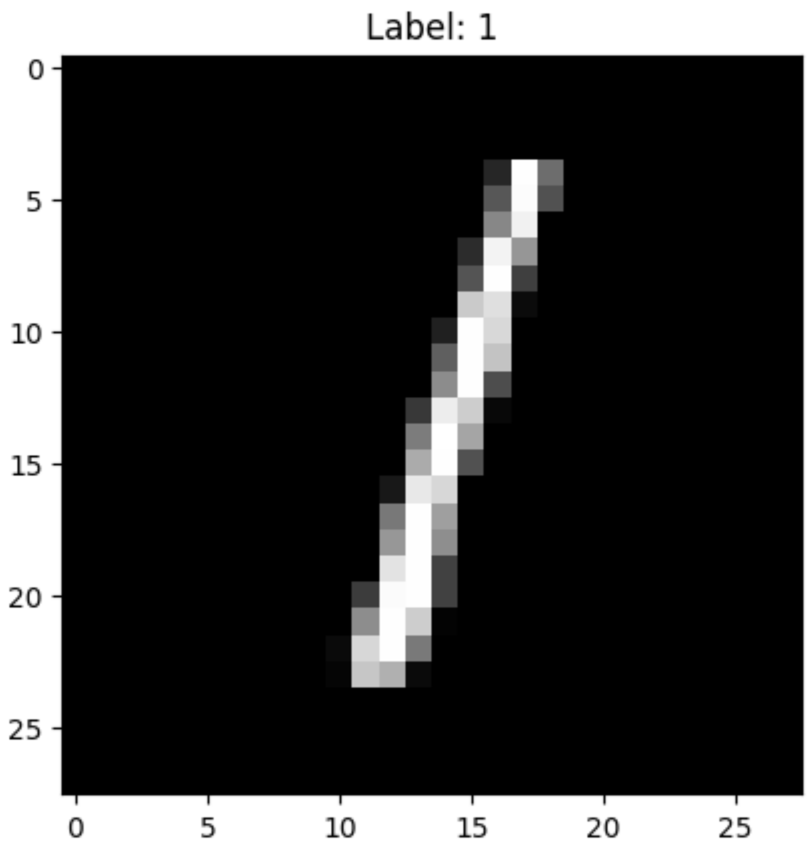
Section 2

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 -----------
third_image = test_images[2,:,:,0]
plt.imshow(third_image, cmap='gray')
plt.title("Label: " + str(test_labels[2]))
```

Out[ ]:
```
Text(0.5, 1.0, 'Label: 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.

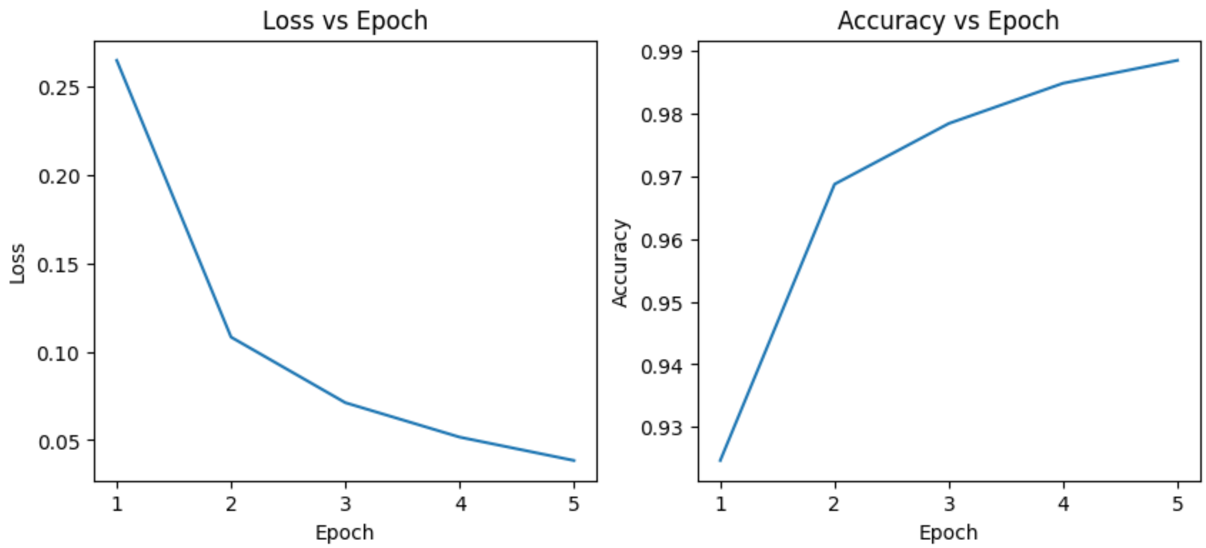
(Hint: The value of of loss and accuracy are stored in the `hist` variable. Try to print out `hist.history` and `his.history.keys()` .)

In [ ]:
```
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=['acc

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_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_13 (Flatten)        (None, 784)               0         
                                                                 
 dense_37 (Dense)            (None, 512)               401920    
                                                                 
 dense_38 (Dense)            (None, 10)                5130      
                                                                 
=================================================================
Total params: 407050 (1.55 MB)
Trainable params: 407050 (1.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/5
469/469 [==============================] - 2s 3ms/step - loss: 0.2648 - accuracy: 0.
9247
Epoch 2/5
469/469 [==============================] - 2s 3ms/step - loss: 0.1083 - accuracy: 0.
9687
Epoch 3/5
469/469 [==============================] - 2s 3ms/step - loss: 0.0712 - accuracy: 0.
9784
Epoch 4/5
469/469 [==============================] - 2s 3ms/step - loss: 0.0517 - accuracy: 0.
9848
Epoch 5/5
469/469 [==============================] - 2s 3ms/step - loss: 0.0385 - accuracy: 0.
9885
```

In [ ]:

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))

#  ----------- YOUR CODE -----------
epoch = [1, 2, 3, 4, 5]
plt.subplot(1, 2, 1)
plt.plot(epoch, hist.history['loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss vs Epoch')

plt.subplot(1, 2, 2)
plt.plot(epoch, hist.history['accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Epoch')

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-linearity

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

No. Not always that we make the neural network larger the accuracy will get better. There will be a moment where the neural network will be too large so that it will overfit the training data. We say that the capacity of the neural network will be too high with respect to the training data points. As a result, the model will not generalize so it will not perform well on the testing data.

In [ ]:
```
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.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_28"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_14 (Flatten)        (None, 784)               0

 dense_39 (Dense)            (None, 512)               401920

 dense_40 (Dense)            (None, 256)               131328

 dense_41 (Dense)            (None, 128)               32896

 dense_42 (Dense)            (None, 64)                8256

 dense_43 (Dense)            (None, 32)                2080

 dense_44 (Dense)            (None, 10)                330

=================================================================
Total params: 576810 (2.20 MB)
Trainable params: 576810 (2.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:

```
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=['acc

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
469/469 [==============================] - 3s 4ms/step - loss: 0.3013 - accuracy: 0.
9072
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 0.1016 - accuracy: 0.
9686
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0683 - accuracy: 0.
9789
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0499 - accuracy: 0.
9845
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0387 - accuracy: 0.
9882
313/313 [==============================] - 1s 3ms/step - loss: 0.0683 - accuracy: 0.
9805
test_accuracy: 0.9804999828338623
```

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

network = models.Sequential()

#  ----------- YOUR CODE -----------
network.add(layers.Conv2D(16, (3, 3), activation='relu', padding = 'same', input_sh
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(32, (3, 3), activation='relu', padding = 'same'))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu', padding = 'same'))
network.add(layers.MaxPooling2D((2, 2)))

network.add(layers.Flatten())
network.add(layers.Dense(128, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))

network.summary()
```

```
Model: "sequential_29"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_43 (Conv2D)          (None, 28, 28, 16)        160       
                                                                 
 max_pooling2d_19 (MaxPooli  (None, 14, 14, 16)        0         
 ng2D)                                                           
                                                                 
 conv2d_44 (Conv2D)          (None, 14, 14, 32)        4640      
                                                                 
 max_pooling2d_20 (MaxPooli  (None, 7, 7, 32)          0         
 ng2D)                                                           
                                                                 
 conv2d_45 (Conv2D)          (None, 7, 7, 64)          18496     
                                                                 
 max_pooling2d_21 (MaxPooli  (None, 3, 3, 64)          0         
 ng2D)                                                           
                                                                 
 flatten_15 (Flatten)        (None, 576)               0         
                                                                 
 dense_45 (Dense)            (None, 128)               73856     
                                                                 
 dense_46 (Dense)            (None, 10)                1290      
                                                                 
=================================================================
Total params: 98442 (384.54 KB)
Trainable params: 98442 (384.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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=['acc
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
469/469 [==============================] - 3s 5ms/step - loss: 0.2746 - accuracy: 0.
9137
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0626 - accuracy: 0.
9803
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0410 - accuracy: 0.
9867
Epoch 4/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0304 - accuracy: 0.
9902
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0239 - accuracy: 0.
9924
313/313 [==============================] - 1s 4ms/step - loss: 0.0263 - accuracy: 0.
9902
test_accuracy: 0.9901999831199646
```

Section 4

Exercise 4.1

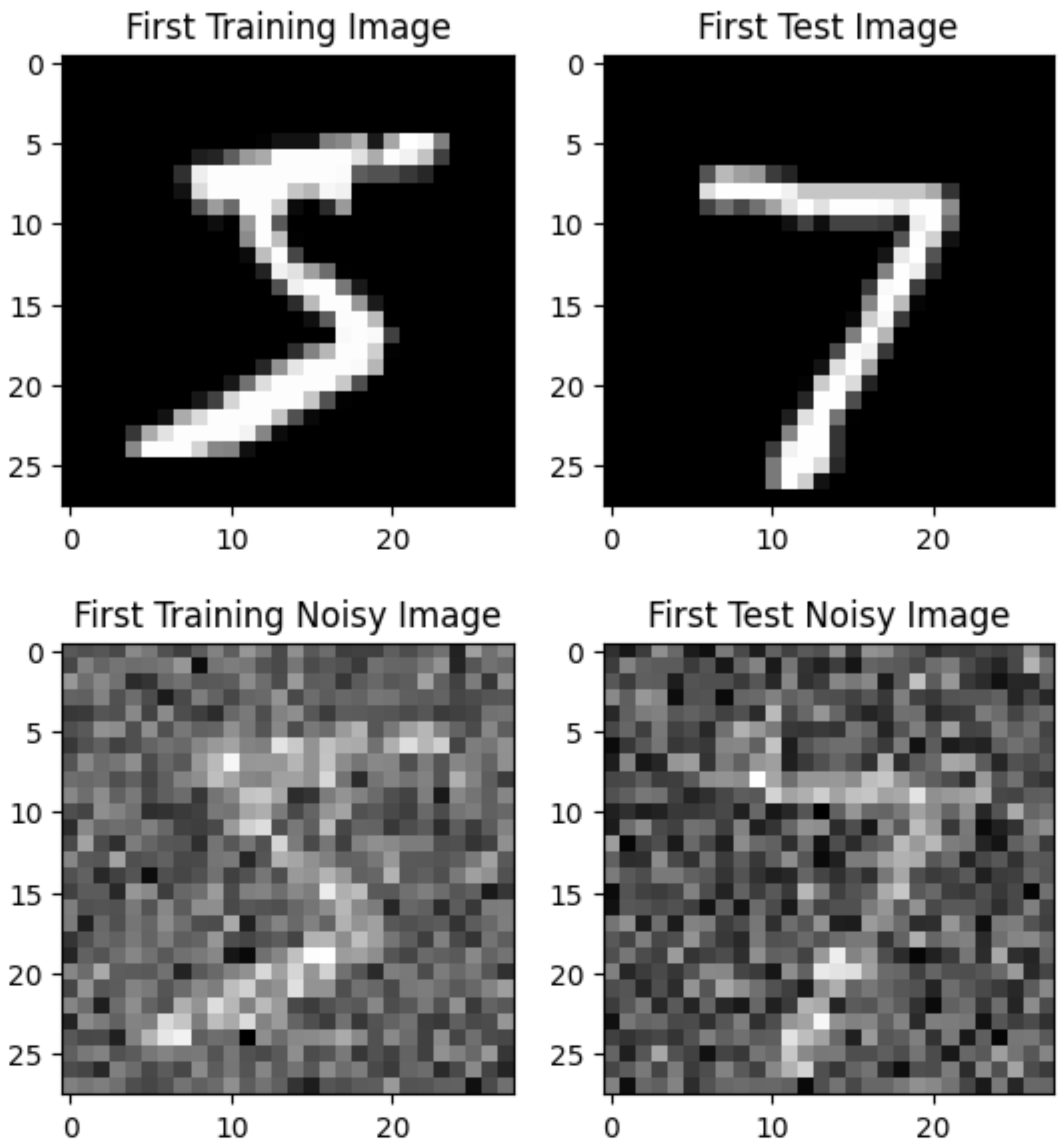
In this exercise you will need to create the entire neural network that does image denoising tasks. Try to mimic the code provided above and follow the structure as provided in the instructions below.

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 -----------
import keras
from keras.datasets import mnist
from keras import models
from keras import layers
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np

(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_noise = np.random.normal(0, 0.4, train_images.shape)
test_noise = np.random.normal(0, 0.4, test_images.shape)

train_images_noisy = train_images_nor + train_noise
test_images_noisy = test_images_nor + test_noise

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

first_train = train_images[0,:,:,0]
first_test = test_images[0,:,:,0]

plt.subplot(1, 2, 1)
plt.imshow(first_train, cmap='gray')
plt.title('First Training Image')

plt.subplot(1, 2, 2)
plt.imshow(first_test, cmap='gray')
plt.title('First Test Image')

plt.show()

first_noisy_train = train_images_noisy[0,:,:,0]
first_noisy_test = test_images_noisy[0,:,:,0]

plt.subplot(1, 2, 1)
plt.imshow(first_noisy_train, cmap='gray')
plt.title('First Training Noisy Image')

plt.subplot(1, 2, 2)
plt.imshow(first_noisy_test, cmap='gray')
plt.title('First Test Noisy Image')

plt.show()
```



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 -----------
encoding_dim = 32

# 1
encoder = models.Sequential()
encoder.add(layers.Conv2D(32, (3, 3), activation='relu', padding = 'same', input_sh
encoder.add(layers.MaxPooling2D((2, 2)))
encoder.add(layers.Conv2D(16, (3, 3), activation='relu', padding = 'same'))
encoder.add(layers.MaxPooling2D((2, 2)))
encoder.add(layers.Conv2D(8, (3, 3), activation='relu', padding = 'same', name = 'c
encoder.add(layers.Flatten())
encoder.add(layers.Dense(encoding_dim, activation='relu'))

# 2
conv_output_shape = encoder.get_layer('convOutput').output_shape[1:]
dense_output_shape = conv_output_shape[0]*conv_output_shape[1]*conv_output_shape[2]

decoder = models.Sequential()
decoder.add(layers.Dense(dense_output_shape, input_shape=(encoding_dim,)))
decoder.add(layers.Reshape(conv_output_shape))
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'))

# 3
autoencoder = models.Sequential()
autoencoder.add(encoder)
autoencoder.add(decoder)
```

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

```
Model: "sequential_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_46 (Conv2D)          (None, 28, 28, 32)        320

 max_pooling2d_22 (MaxPooli  (None, 14, 14, 32)        0
 ng2D)

 conv2d_47 (Conv2D)          (None, 14, 14, 16)        4624

 max_pooling2d_23 (MaxPooli  (None, 7, 7, 16)          0
 ng2D)

 convOutput (Conv2D)         (None, 7, 7, 8)           1160

 flatten_16 (Flatten)        (None, 392)               0

 dense_47 (Dense)            (None, 32)                12576

=================================================================
Total params: 18680 (72.97 KB)
Trainable params: 18680 (72.97 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "sequential_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_48 (Dense)            (None, 392)               12936

 reshape_6 (Reshape)         (None, 7, 7, 8)           0

 conv2d_48 (Conv2D)          (None, 7, 7, 8)           584

 up_sampling2d_12 (UpSampli  (None, 14, 14, 8)         0
 ng2D)

 conv2d_49 (Conv2D)          (None, 14, 14, 16)        1168

 up_sampling2d_13 (UpSampli  (None, 28, 28, 16)        0
 ng2D)

 conv2d_50 (Conv2D)          (None, 28, 28, 32)        4640

 conv2d_51 (Conv2D)          (None, 28, 28, 1)         289

=================================================================
Total params: 19617 (76.63 KB)
Trainable params: 19617 (76.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "sequential_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 sequential_30 (Sequential)  (None, 32)                18680

 sequential_31 (Sequential)  (None, 28, 28, 1)         19617

=================================================================
Total params: 38297 (149.60 KB)
Trainable params: 38297 (149.60 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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 [==============================] - 5s 11ms/step - loss: 0.2400
Epoch 2/20
235/235 [==============================] - 3s 12ms/step - loss: 0.1413
Epoch 3/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1253
Epoch 4/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1179
Epoch 5/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1137
Epoch 6/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1106
Epoch 7/20
235/235 [==============================] - 3s 12ms/step - loss: 0.1083
Epoch 8/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1065
Epoch 9/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1051
Epoch 10/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1038
Epoch 11/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1029
Epoch 12/20
235/235 [==============================] - 3s 12ms/step - loss: 0.1021
Epoch 13/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1014
Epoch 14/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1008
Epoch 15/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1001
Epoch 16/20
235/235 [==============================] - 3s 11ms/step - loss: 0.0998
Epoch 17/20
235/235 [==============================] - 3s 11ms/step - loss: 0.0992
Epoch 18/20
235/235 [==============================] - 2s 11ms/step - loss: 0.0989
Epoch 19/20
235/235 [==============================] - 3s 11ms/step - loss: 0.0985
Epoch 20/20
235/235 [==============================] - 2s 11ms/step - loss: 0.0981
```

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=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
```

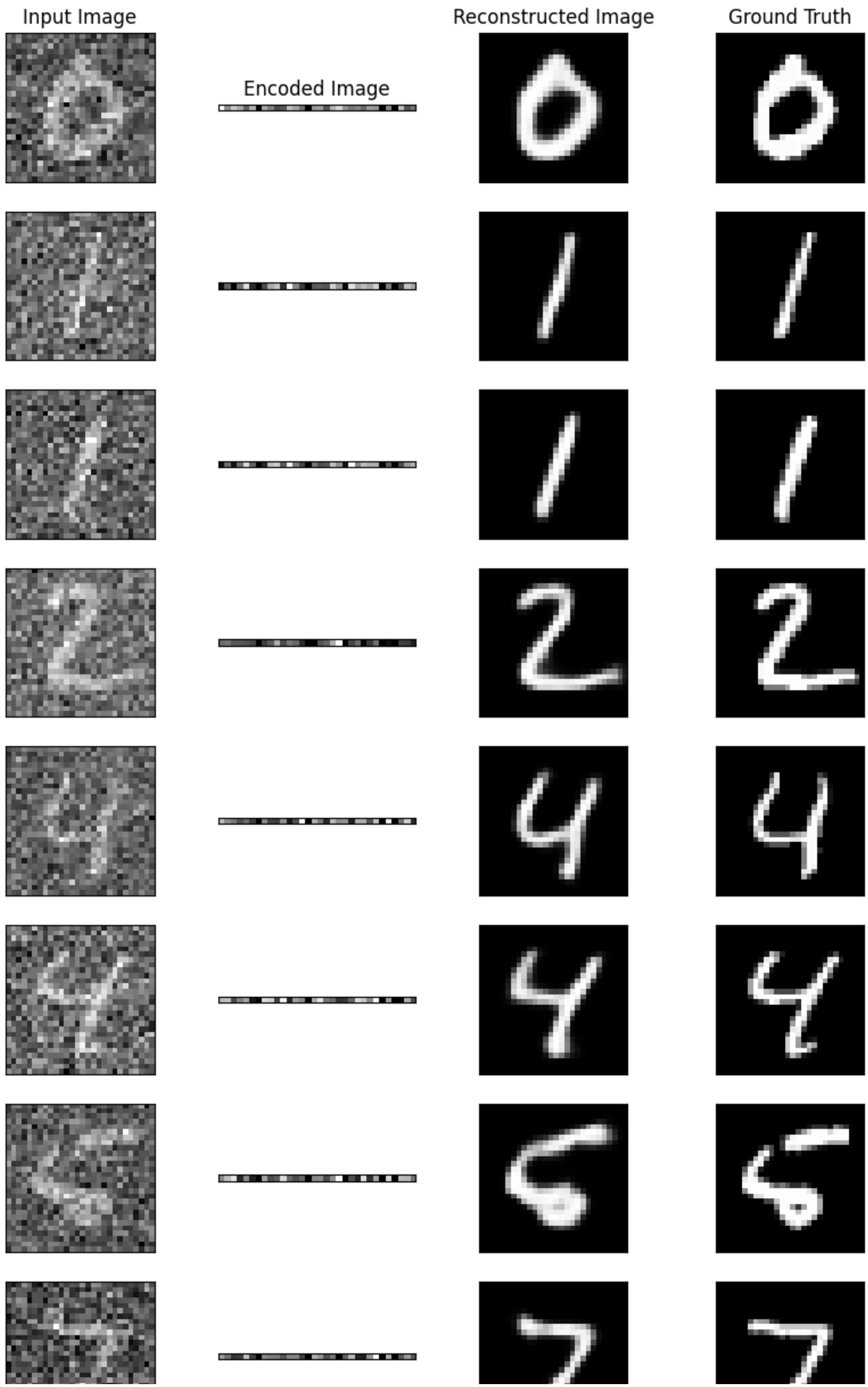
```
  pltIdx = 0
  col = 0
  for i in range(0, num_images):

    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=2, groundTruth=test_images_n
```

```
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 82ms/step
```


Input Image
Encoded Image
Reconstructed Image
Ground Truth