DL_Lab_Exercises

April 26, 2022

1 ECE 637 Deep Learning Lab Exercises

Name: *Zhankun Luo* luo333@purdue.edu

2 Section 1

2.1 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 -----------
A = [1, 2, 3]
B = ['a', 'b', 'c']
C = A + B
print(C)
print(C[0])
print(C[-2])
C.remove(A[1])
print(C)
```

```
[1, 2, 3, 'a', 'b', 'c']
1
b
[1, 3, 'a', 'b', 'c']
```

2.2 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.

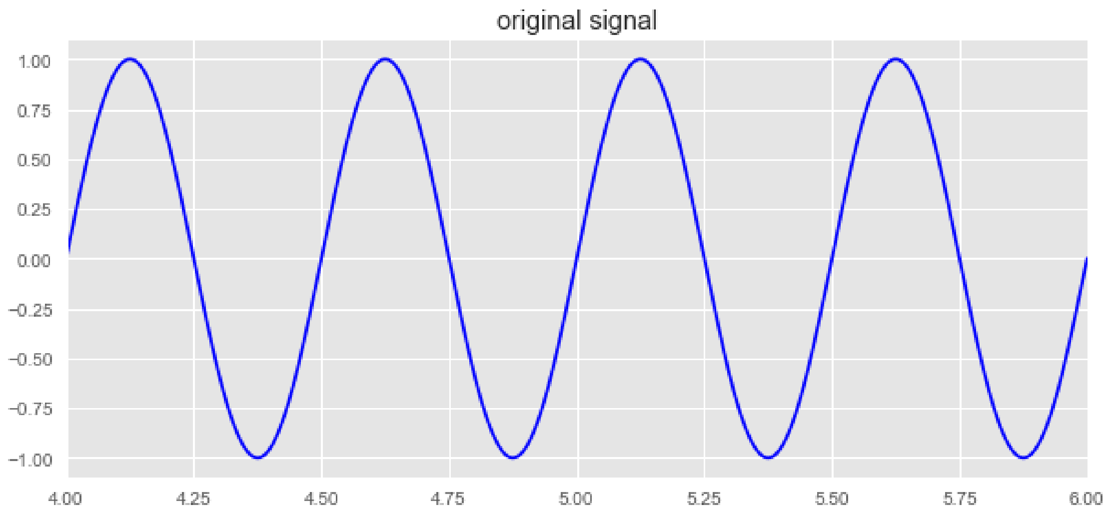
1. Generate a discrete-time signal, `x`, by sampling a 2Hz continuous time sine wave signal with peak amplitude 1 from time 0s to 10s and at a sampling frequency of 500 Hz. Display the signal, `x`, from time 4s to 6s in the first row of a 3x1 subplot with the title "original signal".

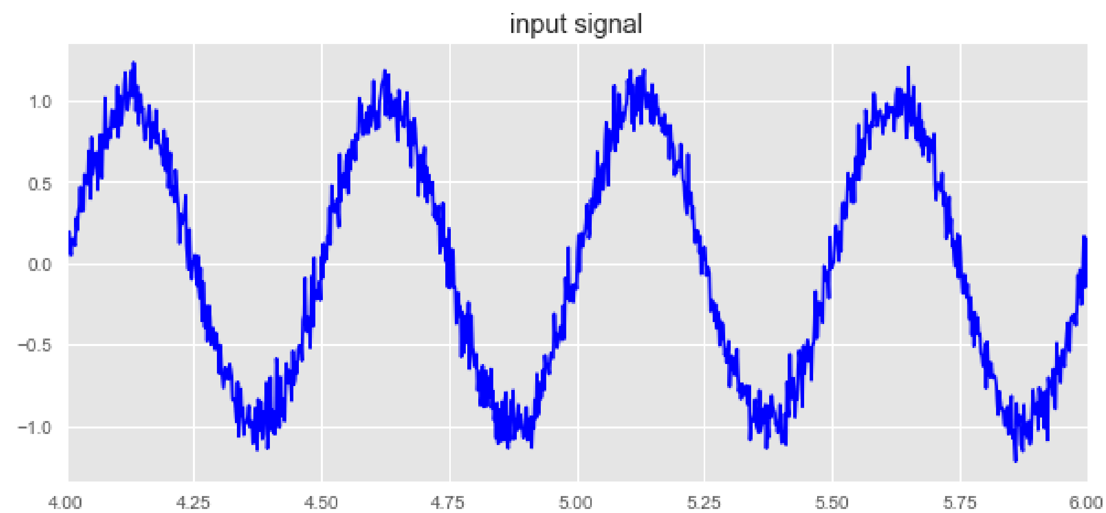
2. Add Gaussian white random noise with 0 mean and standard deviation 0.1 to `x` and call it `x_n` . Display `x_n` from 4s to 6s on the second row of the subplot with the title "input signal".

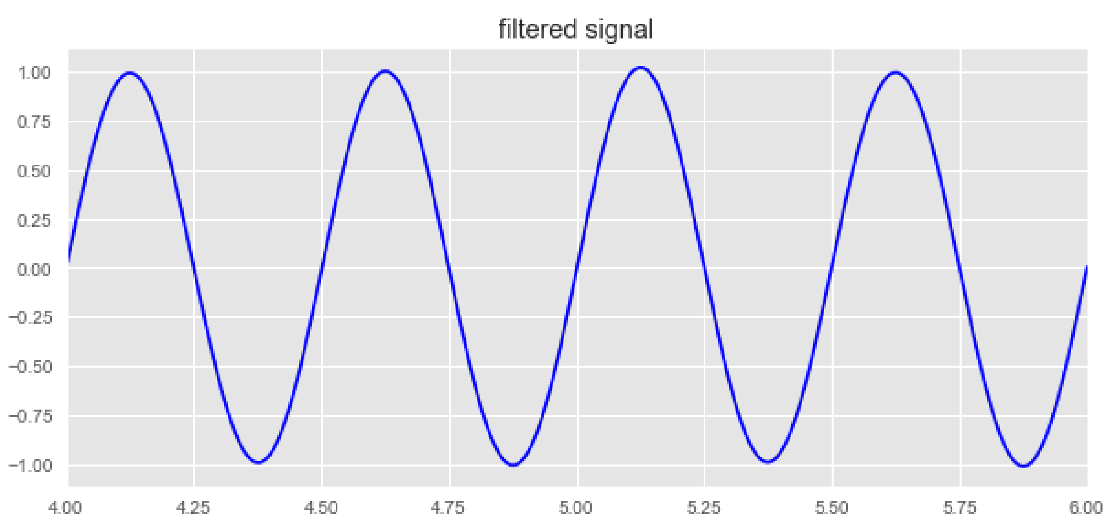
3. Design a low-pass butterworth IIR filter of order 5 with a cut-off frequency of 4Hz, designed to filter out the noise. Hint: Use the signal.butter function and note that the frequencies are relative to the Nyquist frequency. Apply the IIR filter to `x_n`, and name the output `y`. Hint: Use signal.filtfilt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

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

#  ----------- YOUR CODE -----------
import matplotlib
matplotlib.rcParams['mathtext.fontset'] = 'cm'
plt.style.use('ggplot')
fs, T = 500, 10
t  = np.linspace(0, T, T*fs+1, endpoint=True)
f, A = 2, 1
func = lambda t: A*np.sin(2*np.pi*f*t)
x  = func(t)
n = np.random.randn(len(x))*0.1
x_n = x + n
order, fc = 5, 4
b, a = signal.butter(order, fc/(fs/2), 'low', analog=False)
y = signal.filtfilt(b, a, x_n, padtype=None)
t_start, t_end = 4, 6
ind_start, ind_end = int(t_start*fs), int(t_end*fs)+1
_t = t[ind_start:ind_end]
_x = x[ind_start:ind_end]
_x_n = x_n[ind_start:ind_end]
_y = y[ind_start:ind_end]
plt.subplot(3, 1, 1)
plt.plot(_t, _x, 'b')
plt.xlim([t_start, t_end])
plt.title('original signal')
plt.subplot(3, 1, 2)
plt.plot(_t, _x_n, 'b')
plt.xlim([t_start, t_end])
plt.title('input signal')
plt.subplot(3, 1, 3)
plt.plot(_t, _y, 'b')
plt.xlim([t_start, t_end])
```

```
plt.title('filtered signal')
plt.show()
```





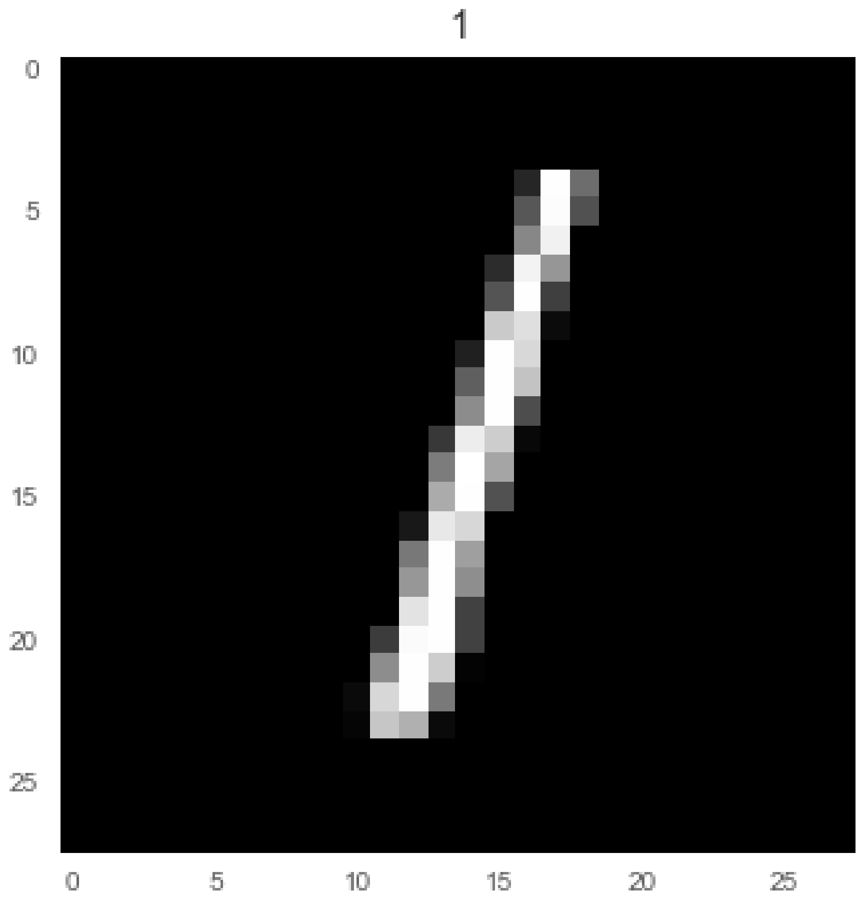


3 Section 2

3.1 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

[ ]:
```
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 -----------
ind_image = 3
image = test_images[ind_image-1,:,:,0]      # Select 3rd image
label = test_labels[ind_image-1]
plt.imshow(image, cmap='gray')  # Display as a gray scale image
plt.title(label)
ax = plt.gca()                  # Get handle to image
ax.grid(b=None)                 # Turn off grid
plt.show()                      # Show image
```



3.2 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()`.)

[ ]:
```
import keras
from keras.datasets import mnist
from keras import models
from keras import layers
from keras.utils import to_categorical
```

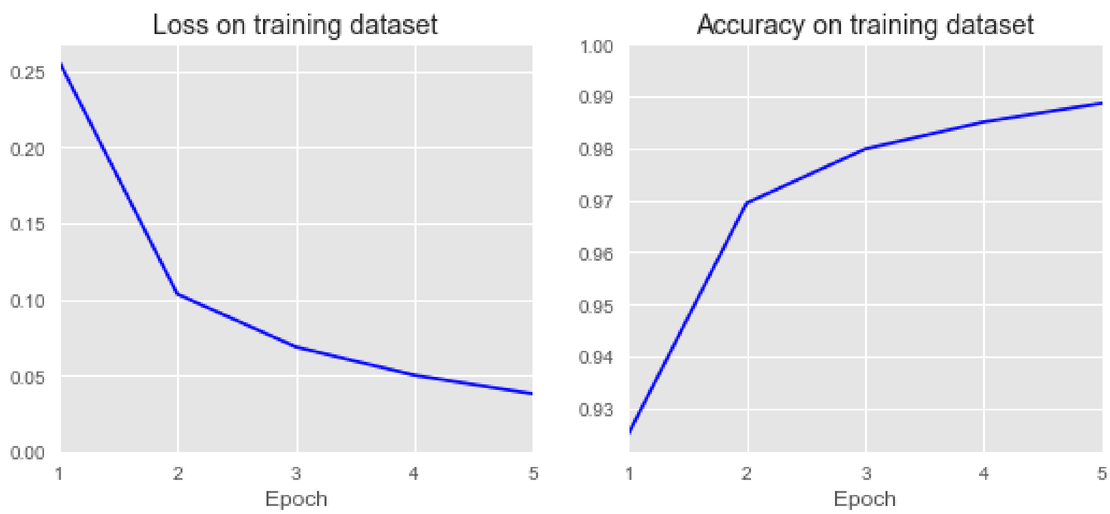
```python
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

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"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0

_________________________________________________________________
dense (Dense)                (None, 512)               401920

_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0

_________________________________________________________________
Epoch 1/5
469/469 [==============================] - 23s 49ms/step - loss: 0.2570 -
accuracy: 0.9249
Epoch 2/5
469/469 [==============================] - 23s 49ms/step - loss: 0.1035 -
accuracy: 0.9694
Epoch 3/5
469/469 [==============================] - 23s 48ms/step - loss: 0.0687 -
accuracy: 0.9799
Epoch 4/5
```

```
469/469 [==============================] - 21s 45ms/step - loss: 0.0501 -
accuracy: 0.9851
Epoch 5/5
469/469 [==============================] - 20s 44ms/step - loss: 0.0379 -
accuracy: 0.9887
```

[ ]:
```python
import matplotlib.pyplot as plt

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

#  ----------- YOUR CODE -----------
print(hist.history)
print(hist.history.keys())
loss, accuracy = hist.history['loss'], hist.history['accuracy']
epoch = list(range(1, len(loss)+1))
plt.subplot(1, 2, 1)
plt.plot(epoch, loss, 'b')
plt.xlim([epoch[0], epoch[-1]])
plt.ylim([0, None])
plt.xticks(epoch)
plt.xlabel('Epoch')
plt.title('Loss on training dataset')
plt.subplot(1, 2, 2)
plt.plot(epoch, accuracy, 'b')
plt.xlim([epoch[0], epoch[-1]])
plt.ylim([None, 1])
plt.xticks(epoch)
plt.xlabel('Epoch')
plt.title('Accuracy on training dataset')
plt.show()
```

```
{'loss': [0.2569579482078552, 0.10349418222904205, 0.06872203201055527,
0.050066981464624405, 0.037850093096494675], 'accuracy': [0.9248999953269958,
0.9694499969482422, 0.9798666834831238, 0.9850500226020813, 0.9886666536331177]}
dict_keys(['loss', 'accuracy'])
```



3.3 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?

Your answer

[ ]:
```
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_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_7 (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
469/469 [==============================] - 28s 60ms/step - loss: 0.0197 -
accuracy: 0.9942
Epoch 2/5
469/469 [==============================] - 29s 62ms/step - loss: 0.0162 -
```

```
accuracy: 0.9955
Epoch 3/5
469/469 [==============================] - 25s 54ms/step - loss: 0.0137 -
accuracy: 0.9964
Epoch 4/5
469/469 [==============================] - 24s 52ms/step - loss: 0.0135 -
accuracy: 0.9967
Epoch 5/5
469/469 [==============================] - 23s 49ms/step - loss: 0.0127 -
accuracy: 0.9969
313/313 [==============================] - 6s 18ms/step - loss: 0.1529 -
accuracy: 0.9794
test_accuracy: 0.9793999791145325
```

4 Section 3

4.1 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.

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

network = models.Sequential()

#  ----------- YOUR CODE -----------
# ---- Feature extraction section
# First Layer
network.add(layers.Conv2D(90, (3, 3), activation='relu', input_shape=(28, 28,␣
↪1)))
network.add(layers.MaxPooling2D((2, 2)))
# Second Layer
network.add(layers.Conv2D(40, (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(64, activation='relu'))
```

```python
# Fourth Layer
network.add(layers.Dense(10, activation='softmax'))
network.summary()
```

```
Model: "sequential_53"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_102 (Conv2D)          (None, 26, 26, 90)        900
_________________________________________________________________
max_pooling2d_102 (MaxPoolin (None, 13, 13, 90)        0
_________________________________________________________________
conv2d_103 (Conv2D)          (None, 11, 11, 40)        32440
_________________________________________________________________
max_pooling2d_103 (MaxPoolin (None, 5, 5, 40)          0
_________________________________________________________________
flatten_53 (Flatten)         (None, 1000)              0
_________________________________________________________________
dense_110 (Dense)            (None, 64)                64064
_________________________________________________________________
dense_111 (Dense)            (None, 10)                650
=================================================================
Total params: 98,054
Trainable params: 98,054
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```python
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=6, batch_size=128)

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

```
Epoch 1/6
469/469 [==============================] - 87s 185ms/step - loss: 0.0295 -
accuracy: 0.9907
Epoch 2/6
469/469 [==============================] - 84s 180ms/step - loss: 0.0239 -
accuracy: 0.9927
Epoch 3/6
469/469 [==============================] - 85s 182ms/step - loss: 0.0208 -
accuracy: 0.9936
Epoch 4/6
469/469 [==============================] - 83s 176ms/step - loss: 0.0184 -
accuracy: 0.9944
Epoch 5/6
469/469 [==============================] - 87s 185ms/step - loss: 0.0157 -
accuracy: 0.9949
Epoch 6/6
469/469 [==============================] - 84s 179ms/step - loss: 0.0140 -
accuracy: 0.9956
313/313 [==============================] - 9s 29ms/step - loss: 0.0314 -
accuracy: 0.9909
test_accuracy: 0.9908999800682068
```

5 Section 4

5.1 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)

[ ]:
```
#  ----------- YOUR CODE -----------
(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)
mu, sigma = 0, 0.4 # mean and standard deviation
train_images_noisy = keras.layers.Add(dtype='float64')([train_images_nor, np.
 ↪random.normal(mu, sigma, train_images.shape)])
```

```
test_images_noisy  = keras.layers.Add(dtype='float64')([test_images_nor,  np.
 ↪random.normal(mu, sigma, test_images.shape)])
```

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

[ ]:
```
#  ----------- YOUR CODE -----------
encoding_dim = 32
input_dim = train_images_nor.shape[1:]
# Build Encoder
encoder = models.Sequential()
encoder.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same',␣
 ↪input_shape=input_dim))
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_57"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_110 (Conv2D)          (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_106 (MaxPoolin (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_111 (Conv2D)          (None, 14, 14, 16)        4624
_________________________________________________________________
max_pooling2d_107 (MaxPoolin (None, 7, 7, 16)          0
_________________________________________________________________
convOutput (Conv2D)          (None, 7, 7, 8)           1160
_________________________________________________________________
flatten_55 (Flatten)         (None, 392)               0
_________________________________________________________________
dense_114 (Dense)            (None, 32)                12576
=================================================================
Total params: 18,680
Trainable params: 18,680
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_58"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_115 (Dense)            (None, 392)               12936
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_112 (Conv2D)          (None, 7, 7, 8)           584
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_113 (Conv2D)          (None, 14, 14, 16)        1168
```

```
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_114 (Conv2D)          (None, 28, 28, 32)        4640
_________________________________________________________________
conv2d_115 (Conv2D)          (None, 28, 28, 1)         289
=================================================================
Total params: 19,617
Trainable params: 19,617
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_59"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_57 (Sequential)   (None, 32)                18680
_________________________________________________________________
sequential_58 (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 …

[ ]:
```python
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 [==============================] - 101s 428ms/step - loss: 0.2749
Epoch 2/20
235/235 [==============================] - 99s 421ms/step - loss: 0.1647
Epoch 3/20
235/235 [==============================] - 97s 412ms/step - loss: 0.1396
Epoch 4/20
235/235 [==============================] - 98s 419ms/step - loss: 0.1308
Epoch 5/20
235/235 [==============================] - 98s 419ms/step - loss: 0.1262
Epoch 6/20
235/235 [==============================] - 96s 409ms/step - loss: 0.1230
```

```
Epoch 7/20
235/235 [==============================] - 98s 416ms/step - loss: 0.1206
Epoch 8/20
235/235 [==============================] - 99s 423ms/step - loss: 0.1188
Epoch 9/20
235/235 [==============================] - 98s 418ms/step - loss: 0.1172
Epoch 10/20
235/235 [==============================] - 98s 417ms/step - loss: 0.1157
Epoch 11/20
235/235 [==============================] - 98s 418ms/step - loss: 0.1144
Epoch 12/20
235/235 [==============================] - 98s 417ms/step - loss: 0.1136
Epoch 13/20
235/235 [==============================] - 97s 414ms/step - loss: 0.1124
Epoch 14/20
235/235 [==============================] - 96s 410ms/step - loss: 0.1118
Epoch 15/20
235/235 [==============================] - 96s 410ms/step - loss: 0.1112
Epoch 16/20
235/235 [==============================] - 98s 416ms/step - loss: 0.1105
Epoch 17/20
235/235 [==============================] - 97s 414ms/step - loss: 0.1099
Epoch 18/20
235/235 [==============================] - 98s 417ms/step - loss: 0.1094
Epoch 19/20
235/235 [==============================] - 97s 415ms/step - loss: 0.1091
Epoch 20/20
235/235 [==============================] - 99s 422ms/step - loss: 0.1086
```

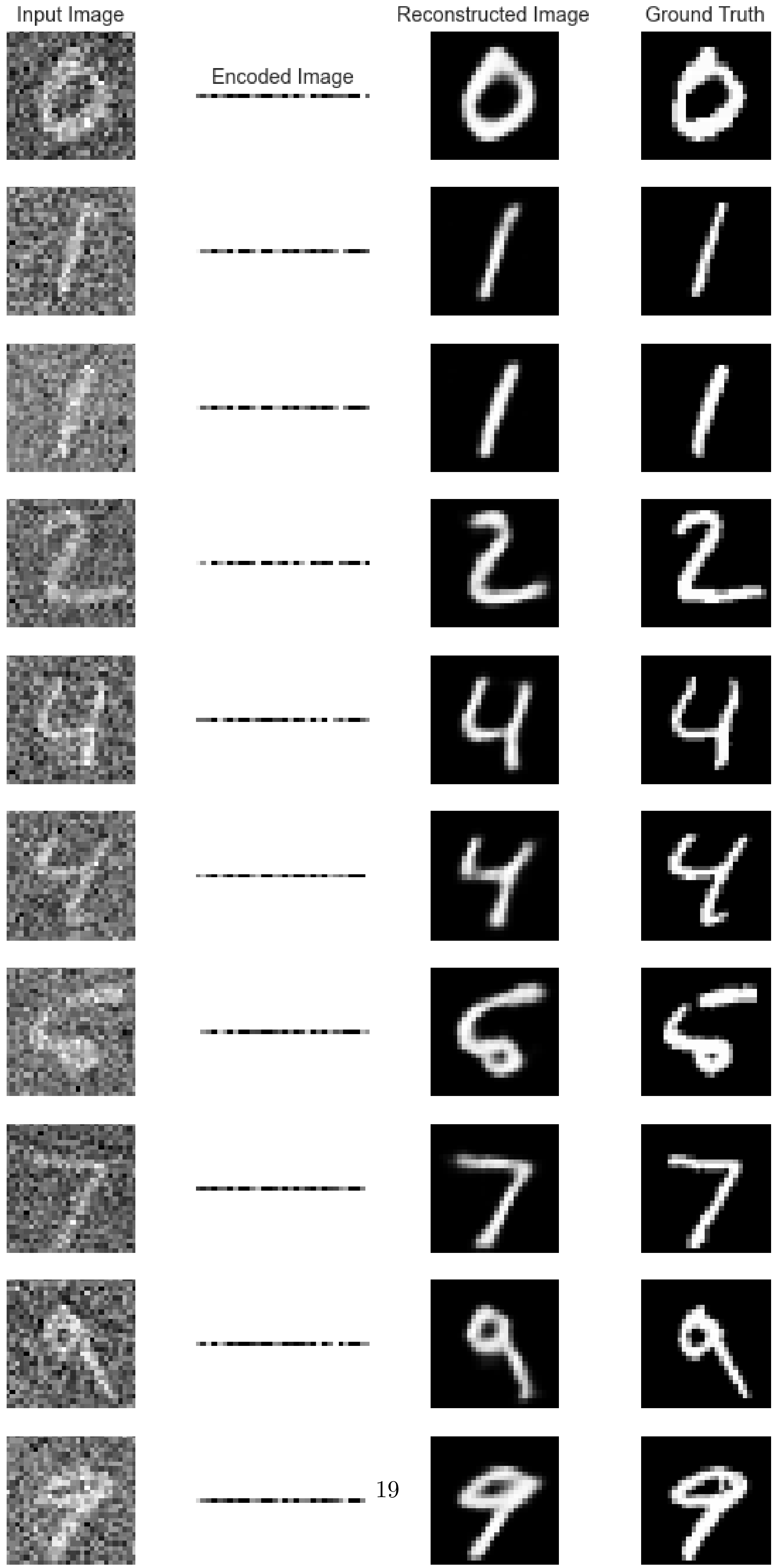
Task 4: Create the neural network model (No need to write code, just run the following commands)

[ ]:
```
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*size))

  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]
input_imgs = keras.backend.gather(test_images_noisy, I)
# gt_imgs = test_images_nor[I]
gt_imgs = keras.backend.gather(test_images_nor, I)
encoded_imgs = encoder.predict(input_imgs)
output_imgs = decoder.predict(encoded_imgs)
showImages(input_imgs.numpy(), encoded_imgs, output_imgs, size=2, 
 ↪groundTruth=gt_imgs.numpy())
```


Input Image
Encoded Image
Reconstructed Image
Ground Truth