1a) Training a Regression Neural Network

As always start with importing what we need and finding our data locations.

```
import tensorflow as tf
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

import os

os.chdir("C:/Users/rique/Downloads/datasets")
```

Load the data.

To normalize an 8Bit image, just divide by max amount of pixel value. (255)

```
def load_dataset():
    img = Image.open("horse025b.png")

    img_array = np.array(img)

    height, width = img_array.shape[0], img_array.shape[1]

    X = img_array/255.0 #normalize

    return X, height, width, img
```

Reduce the learning rate by half every 100 epochs

```
def LR_reduce(epoch, lr):
    if(epoch % 100 == 0) and (epoch > 0):
        return lr / 2
    else:
        return lr
```

Initalize our Neural Network, with different number of layers, depending on the problem.

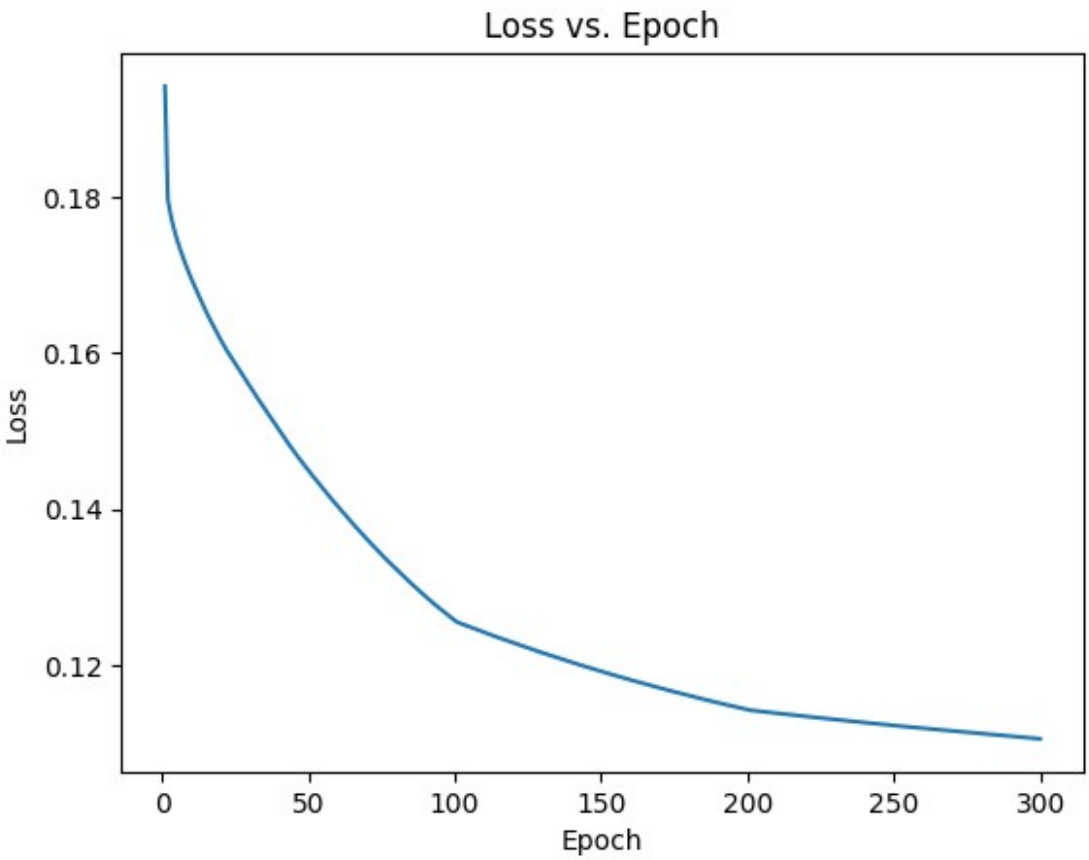
```
def init_NN(probChar):

    #2D input, 1D output each
    if(probChar == 'a'): #1 hidden layer
```

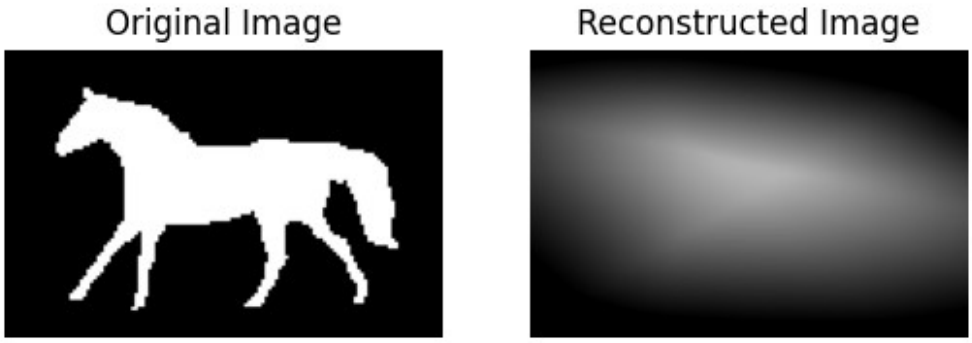
```
        rNN = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(2,)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(1)])
    if(probChar == 'b'): #2 hidden layer
        rNN = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(2,)),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(1)])
    if(probChar == 'c'): #3 hidden layer
        rNN = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(2,)),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(1)])
    if(probChar == 'd'): #4 hidden layer
        rNN = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(2,)),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(1)])

    return rNN

def HW7(probChar):

    X, height, width, img = load_dataset()

    rNN = init_NN(probChar)

    rNN.compile(optimizer=tf.keras.optimizers.SGD(learning_rate =
0.003), loss='mean_squared_error')

    lr_schedule = tf.keras.callbacks.LearningRateScheduler(LR_reduce)

    pixel_coords = []
    for y in range(height):
        for x in range(width):
            pixel_coords.append((x/width, y/height))

    pixel_coords = np.array(pixel_coords)

    rNN_trained = rNN.fit(pixel_coords, X.reshape(-1), epochs = 300,
batch_size = 64, callbacks=[lr_schedule], verbose=0)

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

```
    epochs = range(1, len(loss) + 1)

    plt.plot(epochs, loss)
    plt.title('Loss vs. Epoch')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()

    reconstructed_image =
rNN.predict(pixel_coords).reshape(height,width)

    # Display the original and reconstructed images
    plt.figure()
    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap='gray')  # Reshape for display
    plt.axis('off')
    plt.title('Original Image')

    plt.subplot(1, 2, 2)
    plt.imshow(reconstructed_image, cmap='gray', vmin=0, vmax=1)  #
Reshape for display
    plt.axis('off')
    plt.title('Reconstructed Image')

    plt.show()
```

part a. 1 hidden layer

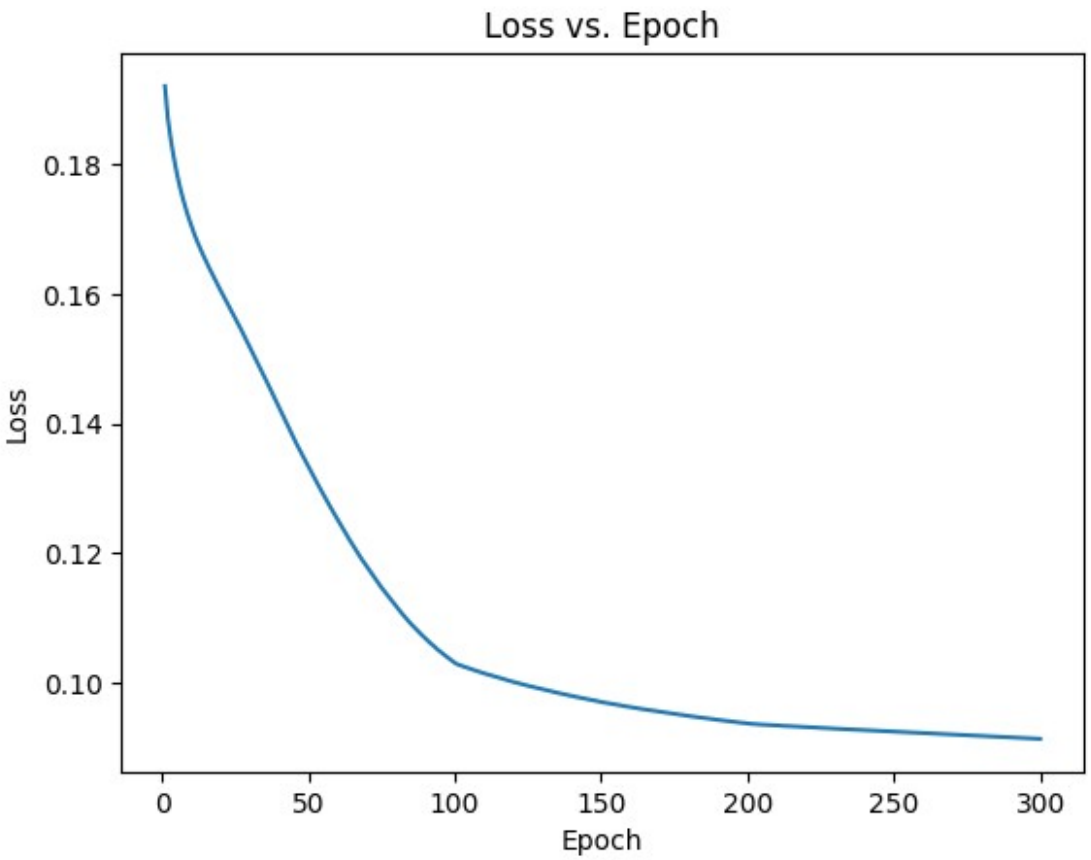
```
HW7('a')
```

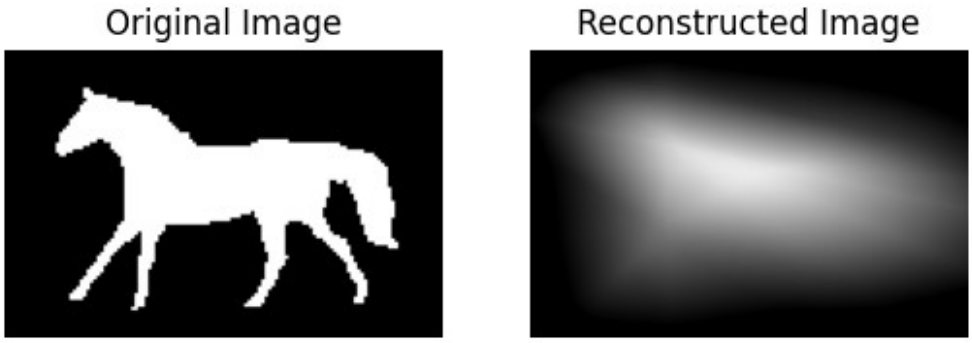




part b. 2 hidden layers

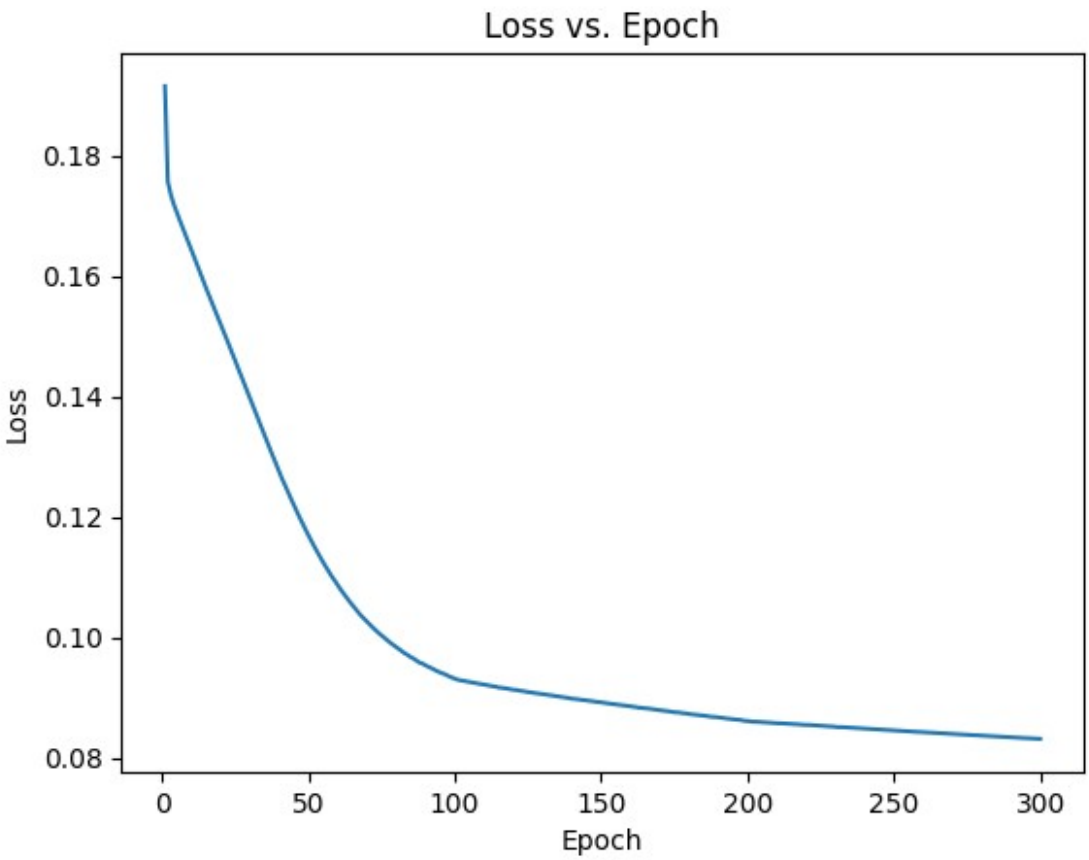
```
HW7('b')
```

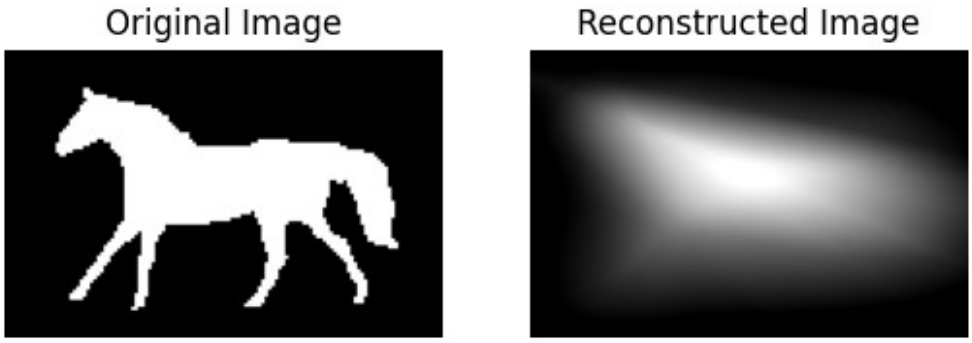




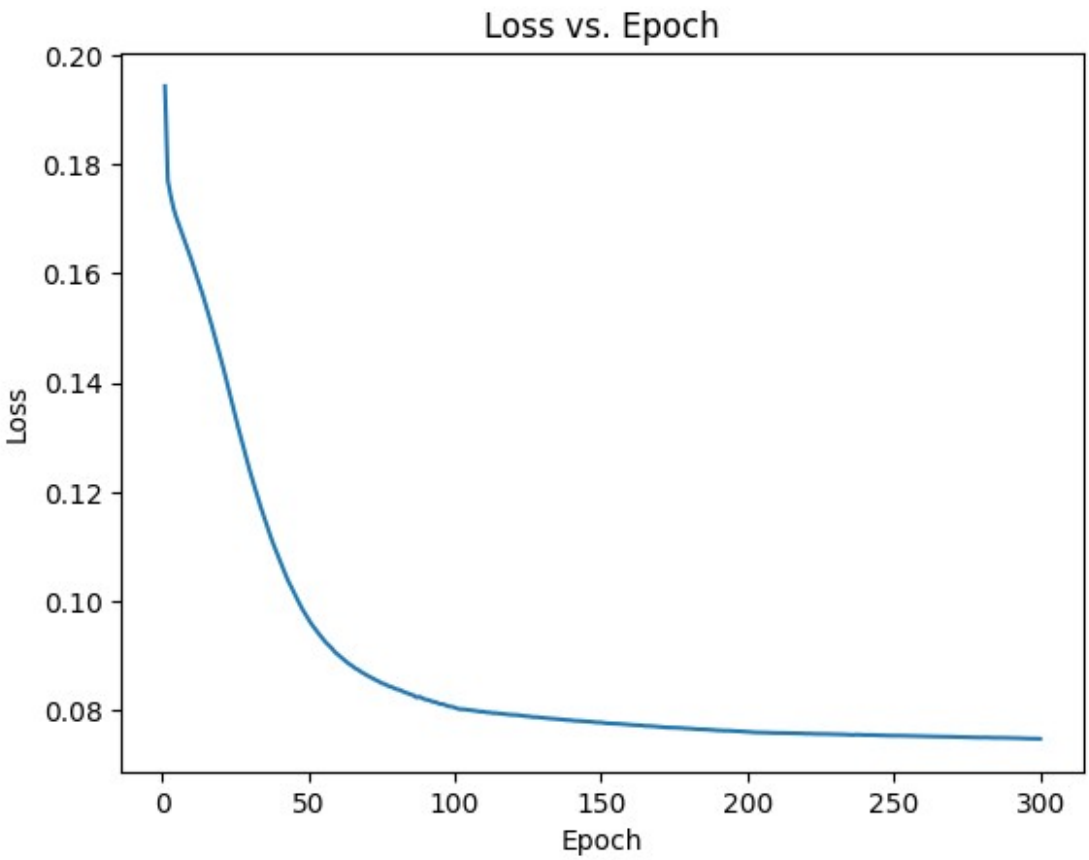
part c. 3 hidden layers

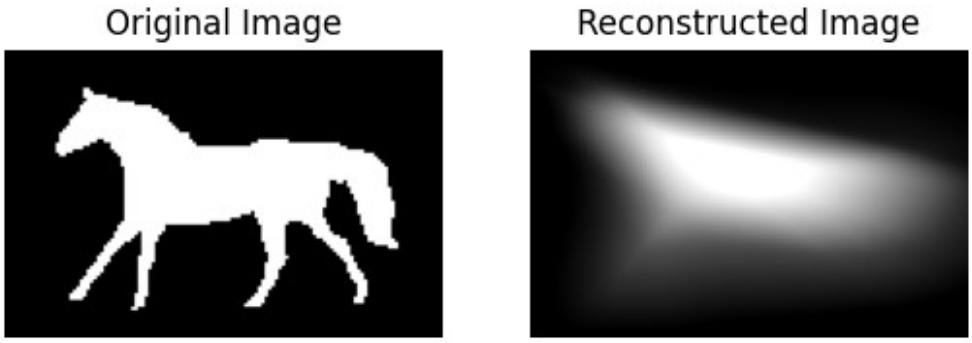
```
HW7('c')
```






```
# part d. 4 hidden layers

HW7('d')
```


Loss vs. Epoch
0.20
0.18
0.16
0.14
0.12
0.10
0.08
Loss
0
50
100
150
200
250
300
Epoch


Original Image
Reconstructed Image