Ungraded Lab: MNIST Deep Autoencoder

Welcome back! In this lab, you will extend the shallow autoencoder you built in the previous exercise. The model here will have a deeper network so it can handle more complex images.

Imports

```
[1]  try:
       # %tensorflow_version only exists in Colab.
       %tensorflow_version 2.x
     except Exception:
       pass

     import tensorflow as tf
     import tensorflow_datasets as tfds

     import numpy as np
     import matplotlib.pyplot as plt
```

Prepare the Dataset

You will prepare the MNIST dataset just like in the previous lab.

```
[2]  def map_image(image, label):
       '''Normalizes and flattens the image. Returns image as input and label.'''
       image = tf.cast(image, dtype=tf.float32)
       image = image / 255.0
       image = tf.reshape(image, shape=(784,))

       return image, image
```

```
[3]  # Load the train and test sets from TFDS

     BATCH_SIZE = 128
     SHUFFLE_BUFFER_SIZE = 1024

     train_dataset = tfds.load('mnist', as_supervised=True, split="train")
     train_dataset = train_dataset.map(map_image)
     train_dataset = train_dataset.shuffle(SHUFFLE_BUFFER_SIZE).batch(BATCH_SIZE).repeat()

     test_dataset = tfds.load('mnist', as_supervised=True, split="test")
     test_dataset = test_dataset.map(map_image)
     test_dataset = test_dataset.batch(BATCH_SIZE).repeat()
```

```
Downloading and preparing dataset mnist/3.0.1 (download: 11.06 MiB, generated: 21.00 MiB, total: 32.06 MiB) to /root/tensorflow_datasets/mnist/3.0.1...
WARNING:absl:Dataset mnist is hosted on GCS. It will automatically be downloaded to your
local data directory. If you'd instead prefer to read directly from our public
GCS bucket (recommended if you're running on GCP), you can instead pass
`try_gcs=True` to `tfds.load` or set `data_dir=gs://tfds-data/datasets`.

Dl Completed...: 100% 4/4 [00:01<00:00, 3.78 file/s]

Dataset mnist downloaded and prepared to /root/tensorflow_datasets/mnist/3.0.1. Subsequent calls will reuse this data.
```

Build the Model

As mentioned, you will have a deeper network for the autoencoder. Compare the layers here with that of the shallow network you built in the previous lab.

```
[4]  def deep_autoencoder():
       '''Builds the encoder and decoder using Dense layers.'''
       encoder = tf.keras.layers.Dense(units=128, activation='relu')(inputs)
       encoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
       encoder = tf.keras.layers.Dense(units=32, activation='relu')(encoder)

       decoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
       decoder = tf.keras.layers.Dense(units=128, activation='relu')(decoder)
       decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(decoder)

       return encoder, decoder

     # set the input tensor
     inputs =  tf.keras.layers.Input(shape=(784,))

     # get the encoder and decoder output
     deep_encoder_output, deep_autoencoder_output = deep_autoencoder()

     # setup the encoder because you will visualize its output later
     deep_encoder_model = tf.keras.Model(inputs=inputs, outputs=deep_encoder_output)

     # setup the autoencoder
     deep_autoencoder_model = tf.keras.Model(inputs=inputs, outputs=deep_autoencoder_output)
```

Compile and Train the Model

```
[5]  train_steps = 60000 // BATCH_SIZE

     deep_autoencoder_model.compile(optimizer=tf.keras.optimizers.Adam(), loss='binary_crossentropy')
     deep_auto_history = deep_autoencoder_model.fit(train_dataset, steps_per_epoch=train_steps, epochs=50)
```

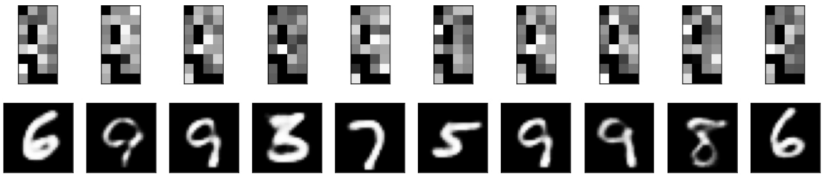
```
468/468 [==============================] - 3s 7ms/step - loss: 0.0896
Epoch 22/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0893
Epoch 23/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0890
Epoch 24/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0887
Epoch 25/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0885
Epoch 26/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0882
Epoch 27/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0880
Epoch 28/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0878
Epoch 29/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0877
Epoch 30/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0875
Epoch 31/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0874
Epoch 32/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0872
Epoch 33/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0871
Epoch 34/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0870
Epoch 35/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0868
Epoch 36/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0868
Epoch 37/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0866
Epoch 38/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0865
Epoch 39/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0864
Epoch 40/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0864
Epoch 41/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0863
Epoch 42/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0862
Epoch 43/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0861
Epoch 44/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0861
Epoch 45/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0860
Epoch 46/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0859
Epoch 47/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0858
Epoch 48/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0857
Epoch 49/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0857
Epoch 50/50
468/468 [==============================] - 3s 7ms/step - loss: 0.0856
```

Display sample results

See the results using the model you just trained.

```
[6]  def display_one_row(disp_images, offset, shape=(28, 28)):
       '''Display sample outputs in one row.'''
       for idx, test_image in enumerate(disp_images):
         plt.subplot(3, 10, offset + idx + 1)
         plt.xticks([])
         plt.yticks([])
         test_image = np.reshape(test_image, shape)
         plt.imshow(test_image, cmap='gray')


     def display_results(disp_input_images, disp_encoded, disp_predicted, enc_shape=(8,4)):
       '''Displays the input, encoded, and decoded output values.'''
       plt.figure(figsize=(15, 5))
       display_one_row(disp_input_images, 0, shape=(28,28,))
       display_one_row(disp_encoded, 10, shape=enc_shape)
       display_one_row(disp_predicted, 20, shape=(28,28,))
```

```
# take 1 batch of the dataset
test_dataset = test_dataset.take(1)

# take the input images and put them in a list
output_samples = []
for input_image, image in tfds.as_numpy(test_dataset):
      output_samples = input_image

# pick 10 random numbers to be used as indices to the list above
idxs = np.random.choice(BATCH_SIZE, size=10)

# get the encoder output
encoded_predicted = deep_encoder_model.predict(test_dataset)

# get a prediction for the test batch
deep_predicted = deep_autoencoder_model.predict(test_dataset)

# display the 10 samples, encodings and decoded values!
display_results(output_samples[idxs], encoded_predicted[idxs], deep_predicted[idxs])
```



✓ 1s completed at 5:04 PM