Ungraded Lab: Denoising with a CNN Autoencoder

In the final lab for this week, you will introduce noise to the Fashion MNIST dataset and train an autoencoder to reconstruct the original input 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 train and test sets a little differently this time. Instead of just normalizing the images, you will also introduce random noise and the generated images will be used as input to your model. The target or label will still be the clean images.

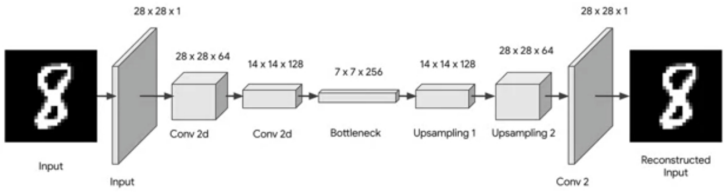
```
[2] def map_image_with_noise(image, label):
      '''Normalizes the images and generates noisy inputs.'''
      image = tf.cast(image, dtype=tf.float32)
      image = image / 255.0

      noise_factor = 0.5
      factor = noise_factor * tf.random.normal(shape=image.shape)
      image_noisy = image + factor
      image_noisy = tf.clip_by_value(image_noisy, 0.0, 1.0)

      return image_noisy, image
```

```
[3] BATCH_SIZE = 128
    SHUFFLE_BUFFER_SIZE = 1024

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

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

```
Downloading and preparing dataset fashion_mnist/3.0.1 (download: 29.45 MiB, generated: 36.42 MiB, total: 65.87 MiB) to /root/tensorflow_datasets/fashion_mnist/3.0.1...
Dl Completed...: 100% 4/4 [00:03<00:00, 1.09 url/s]
Dl Size...: 100% 29/29 [00:03<00:00, 15.41 MiB/s]
Extraction completed...: 100% 4/4 [00:03<00:00, 1.07s/ file]

Shuffling and writing examples to /root/tensorflow_datasets/fashion_mnist/3.0.1.incompleteOJDF1B/fashion_mnist-train.tfrecord
100% 59999/60000 [00:00<00:00, 192582.15 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/fashion_mnist/3.0.1.incompleteOJDF1B/fashion_mnist-test.tfrecord
100% 9999/10000 [00:00<00:00, 128073.19 examples/s]
Dataset fashion_mnist downloaded and prepared to /root/tensorflow_datasets/fashion_mnist/3.0.1. Subsequent calls will reuse this data.
```

Build the Model

You will use the same model from the previous lab.




```
[4] def encoder(inputs):
      '''Defines the encoder with two Conv2D and max pooling layers.'''
      conv_1 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same')(inputs)
      max_pool_1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_1)

      conv_2 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same')(max_pool_1)
      max_pool_2 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_2)
```

```
    return max_pool_2
```

```
[5] def bottle_neck(inputs):
      '''Defines the bottleneck.'''
      bottle_neck = tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3), activation='relu', padding='same')(inputs)
      encoder_visualization = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3), activation='sigmoid', padding='same')(bottle_neck)

      return bottle_neck, encoder_visualization
```

```
[6] def decoder(inputs):
      '''Defines the decoder path to upsample back to the original image size.'''
      conv_1 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same')(inputs)
      up_sample_1 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_1)

      conv_2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same')(up_sample_1)
      up_sample_2 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_2)

      conv_3 = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3), activation='sigmoid', padding='same')(up_sample_2)

      return conv_3
```

```
[7] def convolutional_auto_encoder():
      '''Builds the entire autoencoder model.'''
      inputs = tf.keras.layers.Input(shape=(28, 28, 1,))
      encoder_output = encoder(inputs)
      bottleneck_output, encoder_visualization = bottle_neck(encoder_output)
      decoder_output = decoder(bottleneck_output)

      model = tf.keras.Model(inputs =inputs, outputs=decoder_output)
      encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_visualization)
      return model, encoder_model
```

```
[8] convolutional_model, convolutional_encoder_model = convolutional_auto_encoder()
    convolutional_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 256)         295168
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 128)         295040
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        73792
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         577
=================================================================
Total params: 739,073
Trainable params: 739,073
Non-trainable params: 0
_________________________________________________________________
```

Compile and Train the Model

```
[9] train_steps = 60000 // BATCH_SIZE
    valid_steps = 60000 // BATCH_SIZE

    convolutional_model.compile(optimizer=tf.keras.optimizers.Adam(), loss='binary_crossentropy')
    conv_model_history = convolutional_model.fit(train_dataset, steps_per_epoch=train_steps, validation_data=test_dataset, validation_steps=valid_steps, epochs=40)
```

```
468/468 [==============================] - 17s 36ms/step - loss: 0.2817 - val_loss: 0.2837
Epoch 12/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2813 - val_loss: 0.2839
Epoch 13/40
468/468 [==============================] - 16s 33ms/step - loss: 0.2810 - val_loss: 0.2837
Epoch 14/40
468/468 [==============================] - 16s 35ms/step - loss: 0.2808 - val_loss: 0.2829
Epoch 15/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2805 - val_loss: 0.2828
Epoch 16/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2803 - val_loss: 0.2823
Epoch 17/40
468/468 [==============================] - 16s 33ms/step - loss: 0.2802 - val_loss: 0.2821
Epoch 18/40
468/468 [==============================] - 16s 35ms/step - loss: 0.2798 - val_loss: 0.2821
Epoch 19/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2797 - val_loss: 0.2819
Epoch 20/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2796 - val_loss: 0.2815
Epoch 21/40
468/468 [==============================] - 15s 33ms/step - loss: 0.2794 - val_loss: 0.2815
Epoch 22/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2793 - val_loss: 0.2819
Epoch 23/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2793 - val_loss: 0.2813
Epoch 24/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2791 - val_loss: 0.2813
Epoch 25/40
468/468 [==============================] - 15s 33ms/step - loss: 0.2789 - val_loss: 0.2815
Epoch 26/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2790 - val_loss: 0.2813
Epoch 27/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2787 - val_loss: 0.2810
Epoch 28/40
468/468 [==============================] - 16s 35ms/step - loss: 0.2788 - val_loss: 0.2811
Epoch 29/40
```

```
468/468 [==============================] - 17s 36ms/step - loss: 0.2786 - val_loss: 0.2811
Epoch 30/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2786 - val_loss: 0.2810
Epoch 31/40
468/468 [==============================] - 15s 33ms/step - loss: 0.2786 - val_loss: 0.2809
Epoch 32/40
468/468 [==============================] - 15s 33ms/step - loss: 0.2785 - val_loss: 0.2806
Epoch 33/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2785 - val_loss: 0.2807
Epoch 34/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2784 - val_loss: 0.2807
Epoch 35/40
468/468 [==============================] - 16s 33ms/step - loss: 0.2782 - val_loss: 0.2805
Epoch 36/40
468/468 [==============================] - 16s 33ms/step - loss: 0.2784 - val_loss: 0.2806
Epoch 37/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2782 - val_loss: 0.2808
Epoch 38/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2782 - val_loss: 0.2808
Epoch 39/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2781 - val_loss: 0.2803
Epoch 40/40
468/468 [==============================] - 16s 33ms/step - loss: 0.2780 - val_loss: 0.2804
```

Display sample results

Let's see if the model can generate the clean image from noisy inputs.

```
[10] def display_one_row(disp_images, offset, shape=(28, 28)):
       '''Display sample outputs in one row.'''
       for idx, noisy_image in enumerate(disp_images):
         plt.subplot(3, 10, offset + idx + 1)
         plt.xticks([])
         plt.yticks([])
         noisy_image = np.reshape(noisy_image, shape)
         plt.imshow(noisy_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 indices
idxs = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# prepare test samples as a batch of 10 images
conv_output_samples = np.array(output_samples[idxs])
conv_output_samples = np.reshape(conv_output_samples, (10, 28, 28, 1))

# get the encoder ouput
encoded = convolutional_encoder_model.predict(conv_output_samples)

# get a prediction for some values in the dataset
predicted = convolutional_model.predict(conv_output_samples)

# display the samples, encodings and decoded values!
display_results(conv_output_samples, encoded, predicted, enc_shape=(7,7))
```