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

```
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
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

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.

```
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
```

```
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()
```

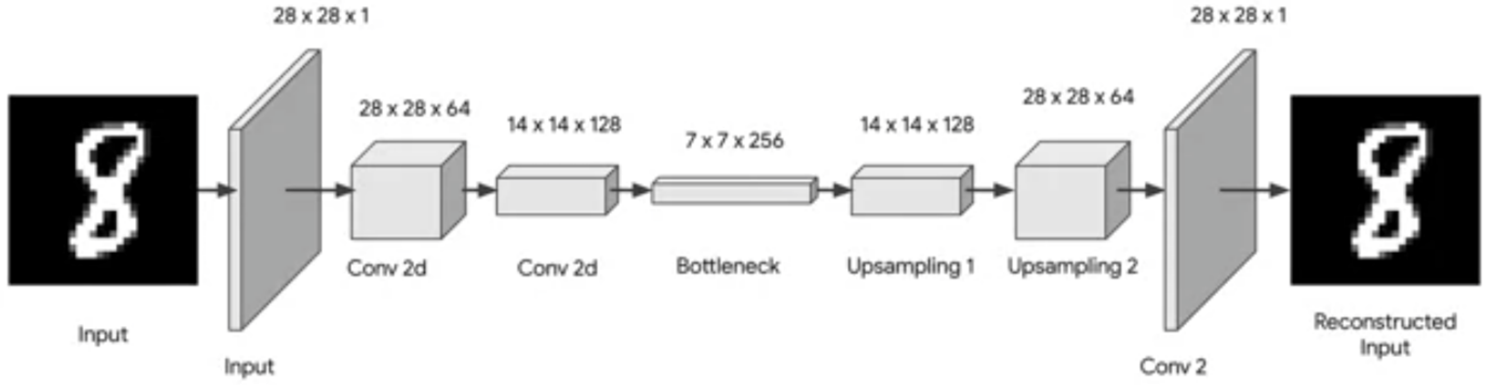
```
Downloading and preparing dataset 29.45 MiB (download: 29.45 MiB, generated: 36.42 MiB, total: 65.87 Mi
Dl Completed...: 100%     4/4 [00:04<00:00, 1.26s/ url]

Dl Size...: 100%     29/29 [00:04<00:00, 11.20 MiB/s]

Extraction completed...: 100%     4/4 [00:04<00:00, 1.58s/ file]
Dataset fashion_mnist downloaded and prepared to ~/tensorflow_datasets/fashion_mnist/3.0.1. Subsequent
```

Build the Model

You will use the same model from the previous lab.



```
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
```

```
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
```

```
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
```

```
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
```

```
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 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 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 (UpSampling  (None, 28, 28, 64)       0
 2D)

 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

```
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_s
```

```
Epoch 1/40
468/468 [==============================] - 28s 39ms/step - loss: 0.3237 - val_loss: 0.3011
Epoch 2/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2952 - val_loss: 0.2944
Epoch 3/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2906 - val_loss: 0.2914
Epoch 4/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2879 - val_loss: 0.2891
Epoch 5/40
468/468 [==============================] - 16s 35ms/step - loss: 0.2861 - val_loss: 0.2883
Epoch 6/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2851 - val_loss: 0.2866
Epoch 7/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2841 - val_loss: 0.2858
Epoch 8/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2834 - val_loss: 0.2852
Epoch 9/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2826 - val_loss: 0.2844
Epoch 10/40
468/468 [==============================] - 16s 35ms/step - loss: 0.2823 - val_loss: 0.2841
Epoch 11/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2818 - val_loss: 0.2838
Epoch 12/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2814 - val_loss: 0.2836
Epoch 13/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2811 - val_loss: 0.2830
Epoch 14/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2807 - val_loss: 0.2828
Epoch 15/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2806 - val_loss: 0.2824
Epoch 16/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2803 - val_loss: 0.2822
Epoch 17/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2802 - val_loss: 0.2823
Epoch 18/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2800 - val_loss: 0.2823
Epoch 19/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2796 - val_loss: 0.2818
Epoch 20/40
468/468 [==============================] - 17s 36ms/step - loss: 0.2797 - val_loss: 0.2817
Epoch 21/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2794 - val_loss: 0.2815
Epoch 22/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2793 - val_loss: 0.2816
Epoch 23/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2794 - val_loss: 0.2813
Epoch 24/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2792 - val_loss: 0.2812
Epoch 25/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2790 - val_loss: 0.2813
Epoch 26/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2790 - val_loss: 0.2811
Epoch 27/40
468/468 [==============================] - 16s 34ms/step - loss: 0.2789 - val_loss: 0.2809
Epoch 28/40
468/468 [==============================] - 16s 35ms/step - loss: 0.2787 - val_loss: 0.2815
Epoch 29/40
468/468 [==============================] - 16s 35ms/step - loss: 0.2787 - val_loss: 0.2808
```

Display sample results

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

```
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))
```

```
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 110ms/step
```