

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()

# Normalize and reshape
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

x_train = np.reshape(x_train, (-1, 28, 28, 1))
x_test = np.reshape(x_test, (-1, 28, 28, 1))
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 0s 0us/step

```
latent_dim = 2 # keep 2 for easy visualization
```

```
encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Flatten()(encoder_inputs)
x = layers.Dense(256, activation="relu")(x)

z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)

def sampling(args):
    z_mean, z_log_var = args
    epsilon = tf.random.normal(shape=tf.shape(z_mean))
    return z_mean + tf.exp(0.5 * z_log_var) * epsilon

z = layers.Lambda(sampling, name="z")([z_mean, z_log_var])

encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()
```

Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 28, 28, 1)	0	-
flatten (Flatten)	(None, 784)	0	input_layer[0][0]
dense (Dense)	(None, 256)	200,960	flatten[0][0]
z_mean (Dense)	(None, 2)	514	dense[0][0]
z_log_var (Dense)	(None, 2)	514	dense[0][0]
z (Lambda)	(None, 2)	0	z_mean[0][0], z_log_var[0][0]

Total params: 201,988 (789.02 KB)
Trainable params: 201,988 (789.02 KB)
Non-trainable params: 0 (0.00 B)

```
latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(256, activation="relu")(latent_inputs)
x = layers.Dense(28 * 28, activation="sigmoid")(x)
decoder_outputs = layers.Reshape((28, 28, 1))(x)

decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()
```

Model: "decoder"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 2)	0
dense_1 (Dense)	(None, 256)	768
dense_2 (Dense)	(None, 784)	201,488
reshape (Reshape)	(None, 28, 28, 1)	0

Total params: 202,256 (790.06 KB)
Trainable params: 202,256 (790.06 KB)
Non-trainable params: 0 (0.00 B)

```
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]

        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)

            reconstruction_loss = tf.reduce_mean(
                keras.losses.binary_crossentropy(data, reconstruction)
            )
            reconstruction_loss *= 28 * 28

            kl_loss = -0.5 * tf.reduce_mean(
                1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
            )

            total_loss = reconstruction_loss + kl_loss

        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))

        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction_loss,
            "kl_loss": kl_loss,
        }
```

```
vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam())

history = vae.fit(
    x_train,
    epochs=20,
    batch_size=128
)
```

```
Epoch 1/20
469/469 ━━━━━━━━━━ 10s 17ms/step - kl_loss: 3.0665 - loss: 164.5573 - reconstruction_loss: 161.4908
Epoch 2/20
469/469 ━━━━━━━━━━ 7s 14ms/step - kl_loss: 3.1123 - loss: 162.0864 - reconstruction_loss: 158.9741
Epoch 3/20
469/469 ━━━━━━━━━━ 8s 16ms/step - kl_loss: 3.1655 - loss: 160.3007 - reconstruction_loss: 157.1352
Epoch 4/20
469/469 ━━━━━━━━━━ 7s 14ms/step - kl_loss: 3.2229 - loss: 158.7709 - reconstruction_loss: 155.5480
Epoch 5/20
469/469 ━━━━━━━━━━ 8s 16ms/step - kl_loss: 3.2928 - loss: 157.4104 - reconstruction_loss: 154.1176
Epoch 6/20
469/469 ━━━━━━━━━━ 10s 16ms/step - kl_loss: 3.3396 - loss: 156.1215 - reconstruction_loss: 152.7819
Epoch 7/20
469/469 ━━━━━━━━━━ 6s 14ms/step - kl_loss: 3.3917 - loss: 155.1136 - reconstruction_loss: 151.7219
Epoch 8/20
469/469 ━━━━━━━━━━ 8s 16ms/step - kl_loss: 3.4232 - loss: 154.1968 - reconstruction_loss: 150.7737
Epoch 9/20
469/469 ━━━━━━━━━━ 7s 16ms/step - kl_loss: 3.4515 - loss: 153.4846 - reconstruction_loss: 150.0332
Epoch 10/20
469/469 ━━━━━━━━━━ 7s 14ms/step - kl_loss: 3.4676 - loss: 152.7955 - reconstruction_loss: 149.3279
Epoch 11/20
469/469 ━━━━━━━━━━ 8s 16ms/step - kl_loss: 3.4737 - loss: 152.1973 - reconstruction_loss: 148.7236
Epoch 12/20
469/469 ━━━━━━━━━━ 6s 14ms/step - kl_loss: 3.4965 - loss: 151.7166 - reconstruction_loss: 148.2201
```

```

Epoch 13/20
469/469 ━━━━━━━━━━ 8s 16ms/step - kl_loss: 3.5093 - loss: 151.3174 - reconstruction_loss: 147.8081
Epoch 14/20
469/469 ━━━━━━ 7s 15ms/step - kl_loss: 3.5195 - loss: 150.8797 - reconstruction_loss: 147.3602
Epoch 15/20
469/469 ━━━━ 9s 14ms/step - kl_loss: 3.5266 - loss: 150.5412 - reconstruction_loss: 147.0147
Epoch 16/20
469/469 ━━━━ 8s 16ms/step - kl_loss: 3.5350 - loss: 150.1596 - reconstruction_loss: 146.6245
Epoch 17/20
469/469 ━━━━ 7s 15ms/step - kl_loss: 3.5421 - loss: 149.7790 - reconstruction_loss: 146.2369
Epoch 18/20
469/469 ━━━━ 8s 17ms/step - kl_loss: 3.5504 - loss: 149.5797 - reconstruction_loss: 146.0293
Epoch 19/20
469/469 ━━━━ 8s 16ms/step - kl_loss: 3.5554 - loss: 149.2498 - reconstruction_loss: 145.6944
Epoch 20/20
469/469 ━━━━ 7s 15ms/step - kl_loss: 3.5603 - loss: 148.9770 - reconstruction_loss: 145.4167

```

```

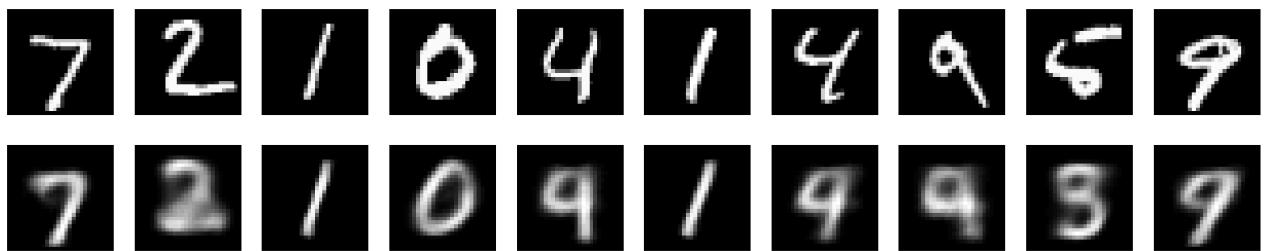
def plot_reconstructions(model, data, n=10):
    z_mean, z_log_var, z = model.encoder(data[:n])
    reconstructions = model.decoder(z)

    plt.figure(figsize=(20, 4))
    for i in range(n):
        # Original
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(data[i].reshape(28, 28), cmap="gray")
        plt.axis("off")

        # Reconstruction
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(reconstructions[i].numpy().reshape(28, 28), cmap="gray")
        plt.axis("off")
    plt.show()

plot_reconstructions(vae, x_test)

```



```

def generate_samples(decoder, n=16):
    random_latent_vectors = tf.random.normal(shape=(n, latent_dim))
    generated_images = decoder(random_latent_vectors)

    plt.figure(figsize=(4, 4))
    for i in range(n):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(generated_images[i].numpy().reshape(28, 28), cmap="gray")
        plt.axis("off")
    plt.show()

generate_samples(decoder)

```



```
z_mean, _, _ = encoder.predict(x_test)

plt.figure(figsize=(6, 6))
plt.scatter(z_mean[:, 0], z_mean[:, 1], s=2)
plt.xlabel("z[0]")
plt.ylabel("z[1]")
plt.title("Latent Space")
plt.show()
```

