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


# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
# Normalize data
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0


# Use only digit '1' for training (normal class)
x_train = x_train[y_train == 1]
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)


# Latent dimension
latent_dim = 2


# Build the Encoder
encoder_inputs = tf.keras.Input(shape=(28, 28, 1))
x = tf.keras.layers.Flatten()(encoder_inputs)
x = tf.keras.layers.Dense(128, activation="relu")(x)
z_mean = tf.keras.layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = tf.keras.layers.Dense(latent_dim, name="z_log_var")(x)

def sampling(args):
    z_mean, z_log_var = args
    epsilon = tf.keras.backend.random_normal(shape=(tf.keras.backend.shape(z_mean)[0], latent_dim))
    return z_mean + tf.exp(0.5 * z_log_var) * epsilon

z = tf.keras.layers.Lambda(sampling, output_shape=(latent_dim,), name="z")([z_mean, z_log_var])
encoder = tf.keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")


# Build the Decoder
latent_inputs = tf.keras.Input(shape=(latent_dim,))
x = tf.keras.layers.Dense(128, activation="relu")(latent_inputs)
x = tf.keras.layers.Dense(28 * 28, activation="sigmoid")(x)
decoder_outputs = tf.keras.layers.Reshape((28, 28, 1))(x)

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


# Build the VAE
outputs = decoder(encoder(encoder_inputs)[2])
vae = tf.keras.Model(encoder_inputs, outputs, name="vae")



# Build the VAE
outputs = decoder(encoder(encoder_inputs)[2])
vae = tf.keras.Model(encoder_inputs, outputs, name="vae")

# VAE loss = Reconstruction loss + KL divergence
# Define a custom loss function that returns the combined loss
def vae_loss_fn(inputs, outputs):
    reconstruction_loss = tf.keras.layers.Flatten()(inputs)
    reconstruction_loss = tf.keras.losses.binary_crossentropy(
        reconstruction_loss, tf.keras.layers.Flatten()(outputs)
    )
    reconstruction_loss = reconstruction_loss * 28 * 28  # Multiply outside the Lambda layer

    # Get z_mean and z_log_var from the encoder output
    z_mean, z_log_var, _ = encoder(inputs)

    # Calculate KL loss using Keras backend functions
    kl_loss = 1 + z_log_var - tf.keras.backend.square(z_mean) - tf.keras.backend.exp(z_log_var)
    kl_loss = tf.reduce_sum(kl_loss, axis=-1)
    kl_loss *= -0.5  # Multiply outside the Lambda layer

    # Calculate total loss
    total_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
    return total_loss
```

```
# Compile the model using the custom loss function
vae.compile(optimizer="adam", loss=vae_loss_fn)


# Train the VAE
vae.fit(x_train, x_train, epochs=10, batch_size=128, verbose=1)
```

```
Epoch 1/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 416.9309
Epoch 2/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 122.2469
Epoch 3/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 92.4153
Epoch 4/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 80.3654
Epoch 5/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 74.9991
Epoch 6/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 71.7749
Epoch 7/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 69.6242
Epoch 8/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 67.7558
Epoch 9/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 66.4822
Epoch 10/10
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 65.7188
<keras.src.callbacks.history.History at 0x7dd8b1507950>
```

```
# Calculate reconstruction errors
reconstructions = vae.predict(x_test)
mse = np.mean(np.power(x_test - reconstructions, 2), axis=(1, 2, 3))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

```
# Set anomaly threshold (95th percentile)
threshold = np.percentile(mse, 95)


# Identify anomalies
anomalies = mse > threshold


# Display a few anomaly samples
plt.figure(figsize=(10, 5))
for i, idx in enumerate(np.where(anomalies)[0][:10]):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test[idx].squeeze(), cmap='gray')
    plt.title("Anomaly")
    plt.axis("off")
plt.show()
```