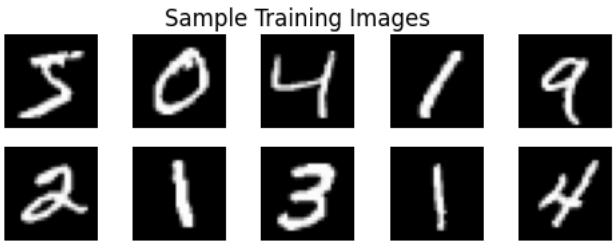
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

print("TensorFlow Version:", tf.__version__)
```

```
TensorFlow Version: 2.19.0
```

```
DATASET = "mnist"   # change to "fashion" for Fashion-MNIST

if DATASET == "mnist":
    (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
else:
    (x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# Normalize: 0-255 -> 0-1
x_train = x_train.astype("float32") / 255.0
x_test  = x_test.astype("float32") / 255.0

# Add channel dimension: (28,28) -> (28,28,1)
x_train = np.expand_dims(x_train, axis=-1)
x_test  = np.expand_dims(x_test, axis=-1)

print("Train shape:", x_train.shape)
print("Test shape :", x_test.shape)

# show sample images
plt.figure(figsize=(6,2))
for i in range(10):
    plt.subplot(2,5,i+1)
    plt.imshow(x_train[i].squeeze(), cmap="gray")
    plt.axis("off")
plt.suptitle("Sample Training Images")
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Train shape: (60000, 28, 28, 1)
Test shape : (10000, 28, 28, 1)
```




```
LATENT_DIM = 16   # change to 2 for latent visualization Task 6

# Encoder
encoder_inputs = keras.Input(shape=(28, 28, 1))

x = layers.Conv2D(32, 3, strides=2, padding="same", activation="relu")(encoder_inputs)
x = layers.Conv2D(64, 3, strides=2, padding="same", activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dense(128, 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)

# Reparameterization trick
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()


# Decoder
latent_inputs = keras.Input(shape=(LATENT_DIM,))
```

```
x = layers.Dense(7 * 7 * 64, activation="relu")(latent_inputs)
x = layers.Reshape((7, 7, 64))(x)
x = layers.Conv2DTranspose(64, 3, strides=2, padding="same", activation="relu")(x)
x = layers.Conv2DTranspose(32, 3, strides=2, padding="same", activation="relu")(x)
decoder_outputs = layers.Conv2DTranspose(1, 3, padding="same", activation="sigmoid")(x)

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

Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 28, 28, 1)	0	–
conv2d (Conv2D)	(None, 14, 14, 32)	320	input_layer[0][0]
conv2d_1 (Conv2D)	(None, 7, 7, 64)	18,496	conv2d[0][0]
flatten (Flatten)	(None, 3136)	0	conv2d_1[0][0]
dense (Dense)	(None, 128)	401,536	flatten[0][0]
z_mean (Dense)	(None, 16)	2,064	dense[0][0]
z_log_var (Dense)	(None, 16)	2,064	dense[0][0]
z (Lambda)	(None, 16)	0	z_mean[0][0], z_log_var[0][0]

Total params: 424,480 (1.62 MB)
Trainable params: 424,480 (1.62 MB)
Non-trainable params: 0 (0.00 B)

Model: "decoder"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 16)	0
dense_1 (Dense)	(None, 3136)	53,312
reshape (Reshape)	(None, 7, 7, 64)	0
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 64)	36,928
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 32)	18,464
conv2d_transpose_2 (Conv2DTranspose)	(None, 28, 28, 1)	289

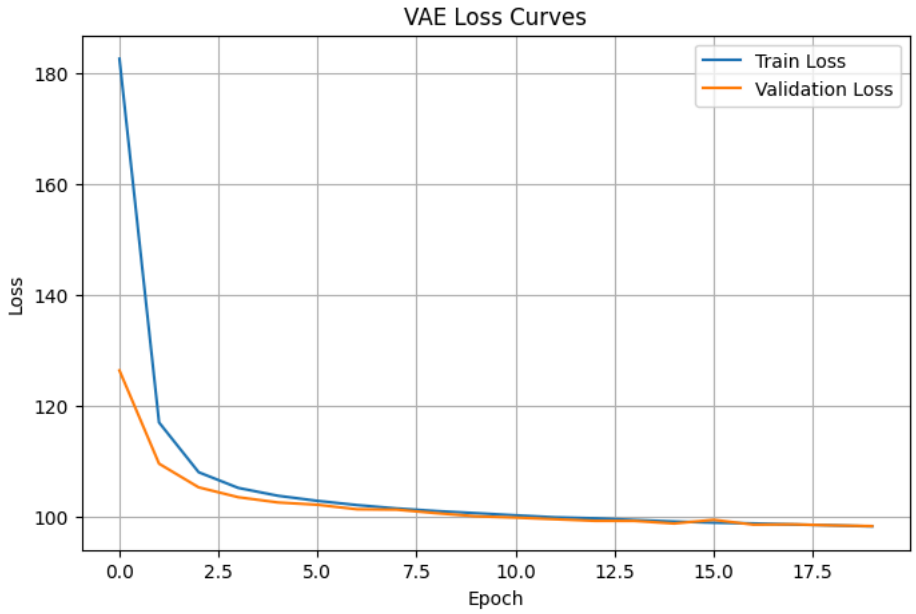
Total params: 108,993 (425.75 KB)
Trainable params: 108,993 (425.75 KB)
Non-trainable params: 0 (0.00 B)

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

        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.recon_loss_tracker = keras.metrics.Mean(name="reconstruction_loss")
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [self.total_loss_tracker, self.recon_loss_tracker, self.kl_loss_tracker]

    # ✅ This fixes your NotImplementedError
    def call(self, inputs):
        z_mean, z_log_var, z = self.encoder(inputs)
        reconstruction = self.decoder(z)
        return reconstruction

    def compute_losses(self, data):
        z_mean, z_log_var, z = self.encoder(data)
        reconstruction = self.decoder(z)

        # Reconstruction Loss (BCE)
        recon_loss = tf.reduce_mean(
            tf.reduce_sum(
                keras.losses.binary_crossentropy(data, reconstruction),
                axis=(1, 2)
```

```
            )
        )

        # KL Divergence Loss
        kl_loss = -0.5 * tf.reduce_mean(
            tf.reduce_sum(
                1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var),
                axis=1
            )
        )

        total_loss = recon_loss + kl_loss
        return total_loss, recon_loss, kl_loss

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

        with tf.GradientTape() as tape:
            total_loss, recon_loss, kl_loss = self.compute_losses(data)

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

        self.total_loss_tracker.update_state(total_loss)
        self.recon_loss_tracker.update_state(recon_loss)
        self.kl_loss_tracker.update_state(kl_loss)

        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.recon_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }

    # ✅ for validation loss
    def test_step(self, data):
        if isinstance(data, tuple):
            data = data[0]

        total_loss, recon_loss, kl_loss = self.compute_losses(data)

        self.total_loss_tracker.update_state(total_loss)
        self.recon_loss_tracker.update_state(recon_loss)
        self.kl_loss_tracker.update_state(kl_loss)

        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.recon_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }
```

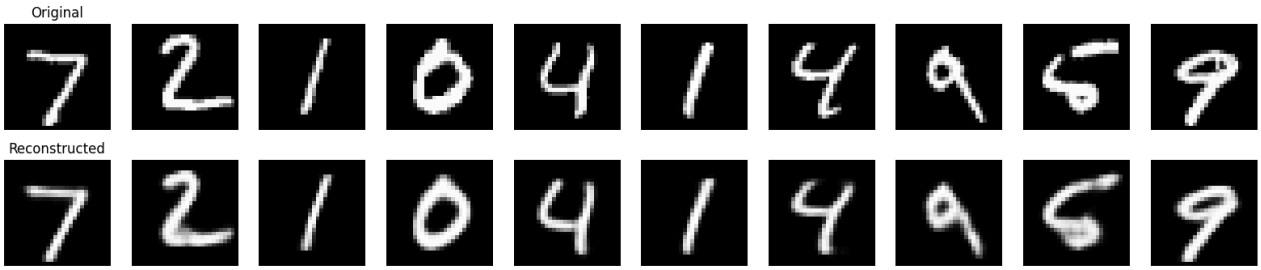
```
vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3))

EPOCHS = 20
BATCH_SIZE = 128

history = vae.fit(
    x_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_data=(x_test, None)
)

# Plot Loss Curves
plt.figure(figsize=(8,5))
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("VAE Loss Curves")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```

```
Epoch 1/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 16s 18ms/step - kl_loss: 7.6075 - loss: 236.7377 - reconstruction_loss: 229.1302 -
Epoch 2/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - kl_loss: 19.9432 - loss: 121.5951 - reconstruction_loss: 101.6520 - v
Epoch 3/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 22.3245 - loss: 109.0963 - reconstruction_loss: 86.7718 - va
Epoch 4/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 22.9679 - loss: 105.5541 - reconstruction_loss: 82.5862 - va
Epoch 5/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.2350 - loss: 104.0516 - reconstruction_loss: 80.8166 - va
Epoch 6/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.3451 - loss: 102.9474 - reconstruction_loss: 79.6023 - va
Epoch 7/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - kl_loss: 23.4757 - loss: 102.2943 - reconstruction_loss: 78.8187 - va
Epoch 8/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.5266 - loss: 101.7614 - reconstruction_loss: 78.2349 - va
Epoch 9/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.5404 - loss: 101.1756 - reconstruction_loss: 77.6353 - va
Epoch 10/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.5452 - loss: 100.7406 - reconstruction_loss: 77.1954 - va
Epoch 11/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.6231 - loss: 100.2778 - reconstruction_loss: 76.6547 - va
Epoch 12/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - kl_loss: 23.5443 - loss: 99.7062 - reconstruction_loss: 76.1619 - val
Epoch 13/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - kl_loss: 23.6488 - loss: 99.6870 - reconstruction_loss: 76.0382 - val
Epoch 14/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - kl_loss: 23.6273 - loss: 99.6437 - reconstruction_loss: 76.0164 - val
Epoch 15/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - kl_loss: 23.6015 - loss: 99.1918 - reconstruction_loss: 75.5902 - val
Epoch 16/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.6448 - loss: 99.0013 - reconstruction_loss: 75.3565 - val
Epoch 17/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.6229 - loss: 98.8380 - reconstruction_loss: 75.2150 - val
Epoch 18/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - kl_loss: 23.5993 - loss: 98.7172 - reconstruction_loss: 75.1179 - val
Epoch 19/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - kl_loss: 23.5988 - loss: 98.5089 - reconstruction_loss: 74.9100 - val
Epoch 20/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - kl_loss: 23.5943 - loss: 98.3420 - reconstruction_loss: 74.7477 - val
```

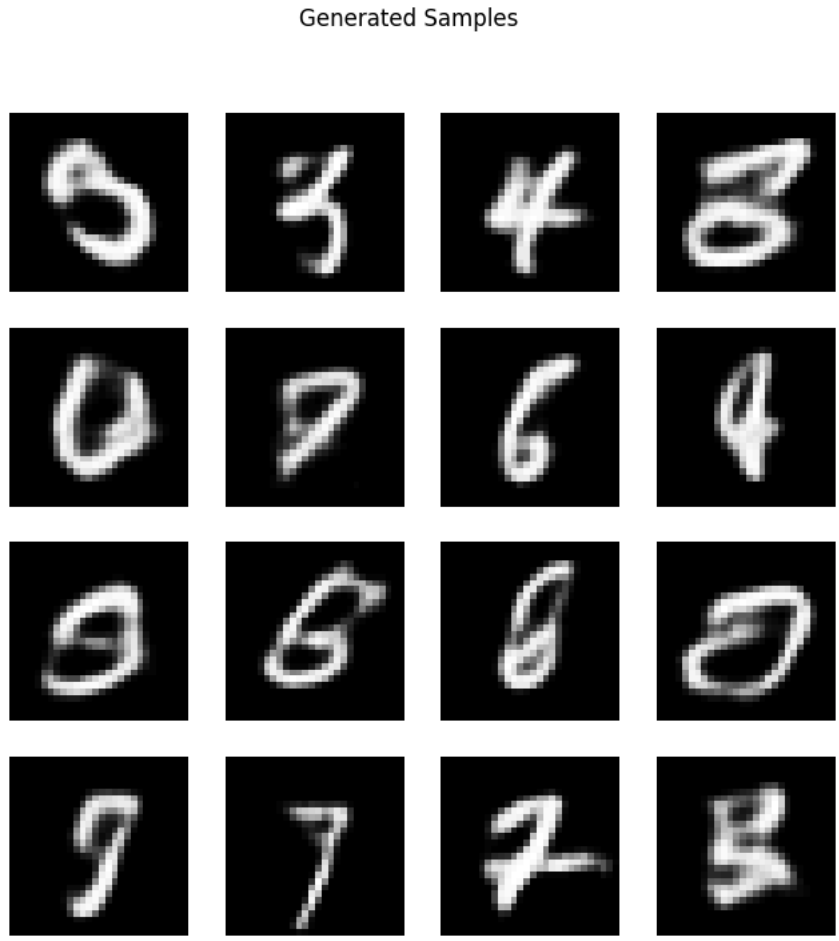



```
def show_reconstructions(model, x_data, n=10):
    z_mean, z_log_var, z = model.encoder(x_data[:n])
    recon = model.decoder(z)

    plt.figure(figsize=(2*n, 4))
    for i in range(n):
        # original
        plt.subplot(2, n, i+1)
        plt.imshow(x_data[i].squeeze(), cmap="gray")
        plt.axis("off")
        if i == 0:
            plt.title("Original")

        # reconstructed
        plt.subplot(2, n, i+1+n)
        plt.imshow(recon[i].numpy().squeeze(), cmap="gray")
        plt.axis("off")
```

```
            if i == 0:
                plt.title("Reconstructed")
        plt.show()

show_reconstructions(vae, x_test, 10)
```




```
def generate_samples(model, n=16):
    random_latent_vectors = tf.random.normal(shape=(n, LATENT_DIM))
    generated = model.decoder(random_latent_vectors)

    side = int(np.sqrt(n))
    plt.figure(figsize=(side*2, side*2))
    for i in range(n):
        plt.subplot(side, side, i+1)
        plt.imshow(generated[i].numpy().squeeze(), cmap="gray")
        plt.axis("off")
    plt.suptitle("Generated Samples")
    plt.show()

generate_samples(vae, n=16)
```




```
if LATENT_DIM == 2:
    z_mean, z_log_var, z = encoder.predict(x_test, batch_size=256)

    plt.figure(figsize=(7,6))
    plt.scatter(z_mean[:, 0], z_mean[:, 1], c=y_test, s=2, cmap="tab10")
```

```
        plt.colorbar()
        plt.title("Latent Space Visualization (z_mean)")
        plt.xlabel("Latent Dimension 1")
        plt.ylabel("Latent Dimension 2")
        plt.grid(True)
        plt.show()
    else:
        print("Set LATENT_DIM = 2 and retrain to visualize latent space.")
```

```
Set LATENT_DIM = 2 and retrain to visualize latent space.
```