Build and evaluate a Variational Autoencoder (VAE) for image generation.

Aim:
Build and evaluate a Variational Autoencoder (VAE) for image generation.

Procedure:

Data Preparation:
Load and preprocess the image dataset.

Build the Variational Autoencoder (VAE):
Define the encoder and decoder architecture.
Implement the reparameterization trick to sample from the learned latent space.

Define Loss Function:
Use the reconstruction loss (e.g., binary cross-entropy or mean squared error) and the Kullback-Leibler (KL) divergence loss to regularize the learned latent space.

Compile the VAE Model:
Choose an optimizer and compile the model.

Train the VAE Model:
Fit the model to the training data.

Evaluate the VAE Model:
Visualize generated images.
Evaluate the quality of generated images quantitatively (e.g., using Frechet Inception Distance (FID)) if necessary.

Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, losses, metrics
import matplotlib.pyplot as plt

# Define the encoder architecture
def build_encoder(latent_dim):
    encoder_inputs = layers.Input(shape=(28, 28, 1))
    x = layers.Conv2D(32, 3, activation='relu', strides=2, padding='same')(encoder_inputs)
    x = layers.Conv2D(64, 3, activation='relu', strides=2, padding='same')(x)
    x = layers.Flatten()(x)
    z_mean = layers.Dense(latent_dim, name="z_mean")(x)
    z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
    encoder = models.Model(encoder_inputs, [z_mean, z_log_var], name="encoder")
    return encoder
```

```
# Define the decoder architecture
def build_decoder(latent_dim):
    latent_inputs = layers.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, activation='relu', strides=2, padding='same')(x)
    x = layers.Conv2DTranspose(32, 3, activation='relu', strides=2, padding='same')(x)
    decoder_outputs = layers.Conv2DTranspose(1, 3, activation='sigmoid', padding='same')(x)
    decoder = models.Model(latent_inputs, decoder_outputs, name="decoder")
    return decoder

# Define the VAE model
def build_vae(encoder, decoder):
    encoder_inputs = layers.Input(shape=(28, 28, 1))
    z_mean, z_log_var = encoder(encoder_inputs)
    z = layers.Lambda(sampling)([z_mean, z_log_var])
    decoder_outputs = decoder(z)
    vae = models.Model(encoder_inputs, decoder_outputs, name="vae")
    return vae

# Define the reparameterization trick for sampling
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

# Define the custom VAE loss function
def vae_loss(z_mean, z_log_var):
    def loss(y_true, y_pred):
        reconstruction_loss = losses.binary_crossentropy(tf.keras.backend.flatten(y_true), tf.keras.backend.flatten(y_pred))
        reconstruction_loss *= 28 * 28
        kl_loss = -0.5 * tf.reduce_sum(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var), axis=-1)
        return tf.reduce_mean(reconstruction_loss + kl_loss)
    return loss

# Load and preprocess MNIST dataset
(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

```
# Build and compile the VAE
latent_dim = 2
encoder = build_encoder(latent_dim)
decoder = build_decoder(latent_dim)
vae = build_vae(encoder, decoder)
vae.compile(optimizer='adam', loss=vae_loss(encoder.outputs[0], encoder.outputs[1]))

# Train the VAE
history = vae.fit(x_train, x_train, epochs=20, batch_size=128, validation_data=(x_test, x_test))

# Evaluate the VAE
# Visualize generated images
decoded_imgs = vae.predict(x_test)
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Explanation:

- The VAE architecture consists of an encoder and a decoder. The encoder maps input images to a latent space where mean and variance parameters are estimated.
- The decoder reconstructs images from samples drawn from the latent space.
- The reparameterization trick is employed to allow backpropagation through the sampling process.
- The VAE is trained using a custom loss function composed of reconstruction loss and KL divergence loss.
- MNIST dataset is loaded and preprocessed.
- The VAE is trained using the training data, and its performance is evaluated by reconstructing test images and visualizing them.
- The model's loss and metrics are tracked during training and can be analyzed for convergence and performance.