EXP NO: 08 DATE: 27/09/2025	**IMAGE GENERATION USING VAE**

AIM: Train a Variational Autoencoder (VAE) to learn a compact latent representation of face images (CelebA) and generate new realistic images by sampling from the learned latent distribution.

ALGORITHM:

- Load and preprocess image dataset (CelebA subset; fallback to CIFAR-10 for demo) and normalize pixel values to [0,1].
- Build an encoder that maps images to two vectors: latent mean and log-variance, and use the reparameterization trick to sample latent vectors.
- Build a decoder that maps latent vectors back to image space using transposed convolutions.
- Implement a custom VAE model that computes reconstruction loss (binary cross-entropy) and KL divergence, and optimizes their sum.
- Compile the VAE and train it on the image dataset with mini-batch gradient descent.
- After training, sample random latent vectors from the prior (standard normal) and decode them to generate new images.
- Evaluate qualitatively by visualizing generated images and quantitatively via reconstruction/latent metrics if required.

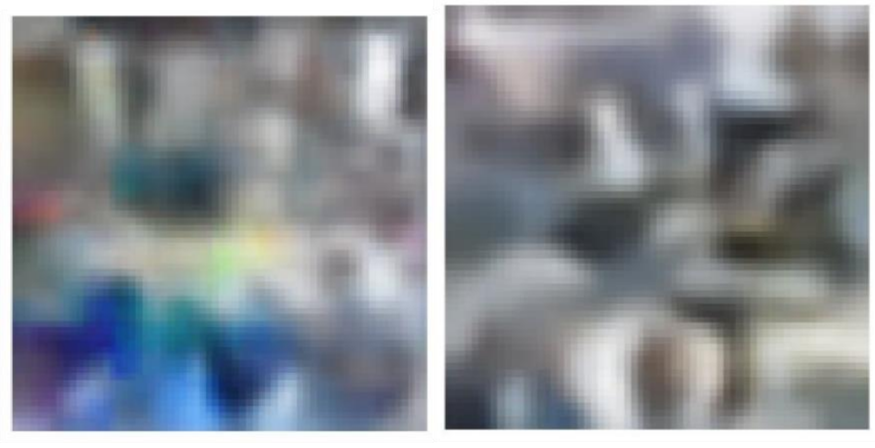
CODE:

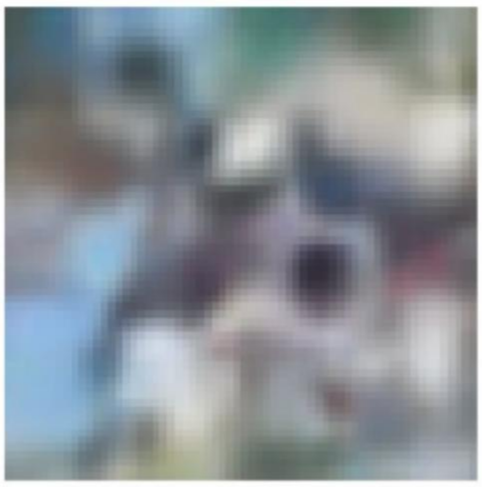
```
import tensorflow as tf

from tensorflow.keras import layers, Model

import tensorflow_datasets as tfds

import numpy as np

import matplotlib.pyplot as plt


SEED = 42

tf.random.set_seed(SEED)

np.random.seed(SEED)
```

```
# 1) Load CelebA subset (fallback to CIFAR-10 for quick demo)
try:
    ds = tfds.load('celeb_a', split='train[:10%]', as_supervised=True)
    def preprocess(img, _):
        img = tf.image.resize(img, (64,64))
        img = tf.cast(img, tf.float32) / 255.0
        return img
    x_train = np.array([preprocess(img, lbl).numpy() for img, lbl in ds])
except Exception:
    (x_train, _), _ = tf.keras.datasets.cifar10.load_data()
    x_train = x_train.astype('float32') / 255.0
    x_train = tf.image.resize(x_train, (64,64)).numpy()


img_shape = x_train.shape[1:]
latent_dim = 256


# 2) Encoder
def build_encoder(img_shape, latent_dim):
    inp = layers.Input(shape=img_shape)
    x = layers.Conv2D(32,3,2,'same', activation='relu')(inp)
    x = layers.Conv2D(64,3,2,'same', activation='relu')(x)
    x = layers.Conv2D(128,3,2,'same', activation='relu')(x)
    x = layers.Conv2D(256,3,2,'same', activation='relu')(x)
    x = layers.Flatten()(x)
    x = layers.Dense(512, 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):
        mean, log_var = args
        eps = tf.random.normal(shape=tf.shape(mean))
        return mean + tf.exp(0.5 * log_var) * eps

    z = layers.Lambda(sampling, name='z')([z_mean, z_log_var])
    return Model(inp, [z_mean, z_log_var, z], name='encoder')

# 3) Decoder
def build_decoder(latent_dim, img_shape):
    inp = layers.Input(shape=(latent_dim,))
    x = layers.Dense(4*4*256, activation='relu')(inp)
    x = layers.Reshape((4,4,256))(x)
    x = layers.Conv2DTranspose(256,3,2,'same', activation='relu')(x)
    x = layers.Conv2DTranspose(128,3,2,'same', activation='relu')(x)
    x = layers.Conv2DTranspose(64,3,2,'same', activation='relu')(x)
    x = layers.Conv2DTranspose(32,3,2,'same', activation='relu')(x)
    out = layers.Conv2DTranspose(img_shape[2], 3, padding='same', activation='sigmoid')(x)
    return Model(inp, out, name='decoder')

# 4) VAE model with custom train_step
class VAE(Model):
    def __init__(self, encoder, decoder, img_shape, **kwargs):
        super().__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.img_shape = img_shape
        self.total_loss_tracker = tf.keras.metrics.Mean(name="total_loss")
        self.rec_loss_tracker = tf.keras.metrics.Mean(name="recon_loss")
        self.kl_loss_tracker = tf.keras.metrics.Mean(name="kl_loss")

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

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            rec_loss = tf.reduce_mean(tf.keras.losses.binary_crossentropy(data, reconstruction))
            rec_loss *= self.img_shape[0] * self.img_shape[1] * self.img_shape[2]
            kl_loss = -0.5 * tf.reduce_mean(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var))
            total_loss = rec_loss + kl_loss
        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.rec_loss_tracker.update_state(rec_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {"loss": self.total_loss_tracker.result(),
                "recon_loss": self.rec_loss_tracker.result(),
                "kl_loss": self.kl_loss_tracker.result()}

# Instantiate
encoder = build_encoder(img_shape, latent_dim)
decoder = build_decoder(latent_dim, img_shape)
vae = VAE(encoder, decoder, img_shape)
```

```
# 5) Compile & 6) Train

vae.compile(optimizer=tf.keras.optimizers.Adam())

vae.fit(x_train, epochs=50, batch_size=64)


# 7) Generate images

z_sample = tf.random.normal(shape=(5, latent_dim))

generated = vae.decoder.predict(z_sample)

for i in range(len(generated)):

    plt.imshow(np.clip(generated[i], 0, 1))

    plt.axis('off')

    plt.show()
```

OUTPUT:





RESULT: The trained VAE learned meaningful latent structure and generated realistic-looking images when sampling from the latent prior.