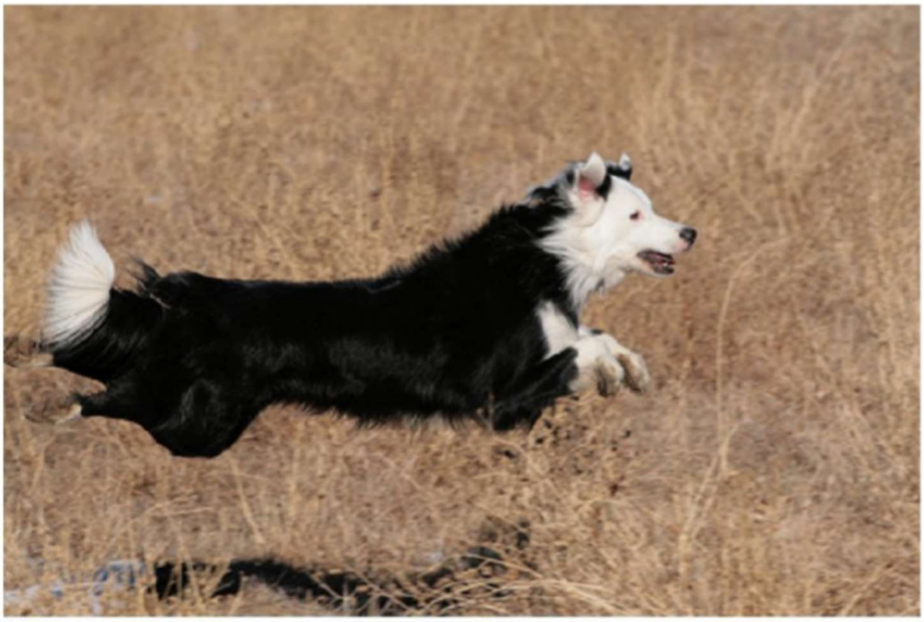
```
X1, X2, y = np.array(X1), np.array(X2), np.array(y)

# Define CNN+RNN model
img_in = Input(shape=(2048,))
cap_in = Input(shape=(maxlen,))
emb = Embedding(vocab, 256, mask_zero=True)(cap_in)
lstm = LSTM(256)(emb)
x = Add()([Dense(256, activation='relu')(img_in), lstm])
out = Dense(vocab, activation='softmax', dtype='float32')(x)
model = Model([img_in, cap_in], out)
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
model.summary()

# Train model
model.fit([X1, X2], y, batch_size=128, epochs=5, verbose=1)

# Caption generator
def generate_caption(img_path):
    f = extract_feat(img_path).reshape(1,-1)
    cap = ['startseq']
    for _ in range(maxlen):
        seq = pad_sequences([tok.texts_to_sequences([' '.join(cap)])[0]], maxlen=maxlen)
        pred = np.argmax(model.predict([f, seq], verbose=0))
        word = tok.index_word.get(pred, '')
        cap.append(word)
        if word == 'endseq': break
    return ' '.join(cap[1:-1])
```

```
# Test
test_img = os.path.join(IMG_DIR, df.iloc[0]['image'])
caption = generate_caption(test_img)
print("Generated caption:", caption)

# Display image with caption
img = Image.open(test_img)
plt.figure(figsize=(10, 6))
plt.imshow(img)
plt.axis('off')
plt.title(f"Generated Caption:\n{caption}", fontsize=14, weight='bold', pad=20)
plt.tight_layout()
plt.show()
```

OUTPUT:

Generated Caption:
a black and white dog is jumping in the air



RESULT: The CNN+RNN model successfully generated meaningful captions for images, demonstrating the effectiveness of combining CNN for feature extraction and RNN for sequence generation.

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")
```