```python
# corrected_edugen_vae_scienceqa.py
import os
import numpy as np
import matplotlib.pyplot as plt
from datasets import load_dataset
import tensorflow as tf
from tensorflow.keras import layers, Model, optimizers


# -----------------------
# Config
# -----------------------
IMG_SIZE = 128
CHANNELS = 3
BATCH_SIZE = 32
LATENT_DIM = 256
EPOCHS = 100
LEARNING_RATE = 1e-4
BETA = 0.5
MODEL_DIR = "./vae_model_fixed"
os.makedirs(MODEL_DIR, exist_ok=True)

# -----------------------
# Preprocess
# -----------------------
def preprocess_pil_image(pil_img):
    img = pil_img.convert("RGB").resize((IMG_SIZE, IMG_SIZE))
    arr = np.asarray(img, dtype=np.float32) / 255.0
    return arr

# -----------------------
# Load HuggingFace dataset
# -----------------------
print("Loading dataset...")
ds = load_dataset("derek-thomas/ScienceQA")  # assumes internet or
cached

# compute counts and steps
train_count = len(ds["train"])
val_count = len(ds["validation"])
test_count = len(ds["test"])
print(f"Train: {train_count}, Val: {val_count}, Test: {test_count}")

steps_per_epoch = max(1, train_count // BATCH_SIZE)
validation_steps = max(1, val_count // BATCH_SIZE)

# -----------------------
# Make tf.data datasets
# -----------------------
def make_tf_dataset(split="train", batch_size=BATCH_SIZE,
```

```
repeat=False, shuffle=True):
    # create a generator that yields items every time it's iterated
    def gen():
        while True:
            for ex in ds[split]:
                img = ex["image"]
                if img is None:
                    continue
                yield preprocess_pil_image(img)
            if not repeat:
                break

    output_signature = tf.TensorSpec(shape=(IMG_SIZE, IMG_SIZE,
CHANNELS), dtype=tf.float32)
    tfds = tf.data.Dataset.from_generator(gen,
output_signature=output_signature)
    if shuffle:
        tfds = tfds.shuffle(2048)
    tfds = tfds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    return tfds

# For training we want the generator to not stop across epochs -> set
repeat=True internally via the while True in gen above.
train_ds = make_tf_dataset("train", repeat=True, shuffle=True)
val_ds   = make_tf_dataset("validation", repeat=False, shuffle=False)
test_ds  = make_tf_dataset("test", repeat=False, shuffle=False)

print("Train batches per epoch (approx):", steps_per_epoch)
print("Validation steps (approx):", validation_steps)
```

```
Loading dataset...
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"3e277829b15249dc8734f5b9d422fcb2","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"b65470e0c7404ade9902fcfb917cf5f5","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"9aa72a1fc93e4006bd89bcb81e1fa993","version_major":2,"vers
ion_minor":0}
```

{"model_id":"e758d5a612c74bcc83d0df389168b7a9","version_major":2,"version_minor":0}

{"model_id":"8eda26ac52b542378d5fe94ee39fb017","version_major":2,"version_minor":0}

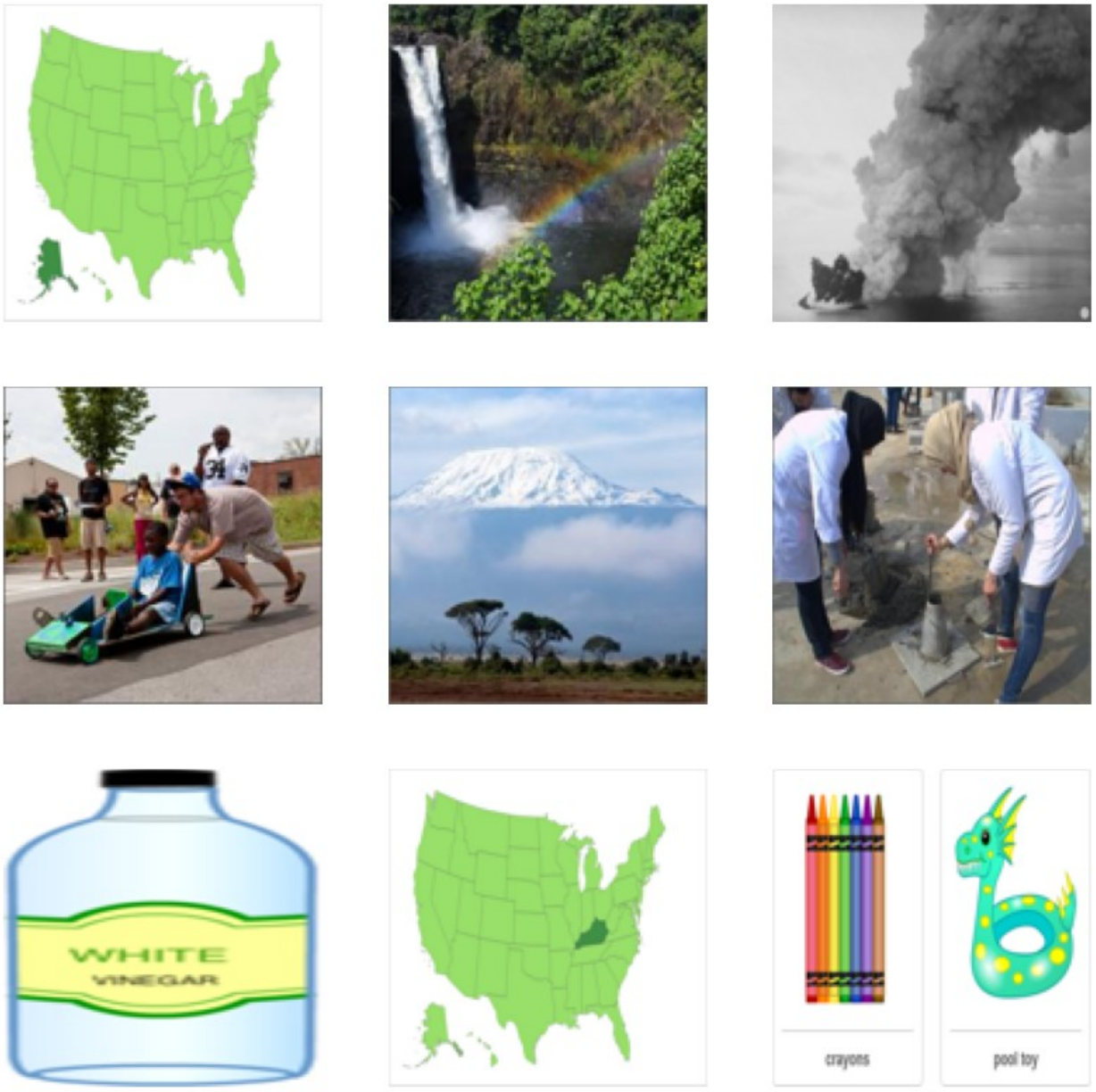
{"model_id":"18afc763a99e4468b80e4a765b07b3b9","version_major":2,"version_minor":0}

{"model_id":"46a5bee10e0c4a1eb2993bb16bc2ecfa","version_major":2,"version_minor":0}

```
Train: 12726, Val: 4241, Test: 4241
Train batches per epoch (approx): 397
Validation steps (approx): 132
```

```
# # -----------------------
# # Downsample datasets to 10% of original
# # -----------------------
# def downsample_dataset(split, fraction=0.1, batch_size=BATCH_SIZE, shuffle=True):
#     total_count = len(ds[split])
#     take_count = max(1, int(total_count * fraction))

#     def gen():
#         for idx, ex in enumerate(ds[split]):
#             if idx >= take_count:
#                 break
#             img = ex["image"]
#             if img is None:
#                 continue
#             yield preprocess_pil_image(img)

#     output_signature = tf.TensorSpec(shape=(IMG_SIZE, IMG_SIZE, CHANNELS), dtype=tf.float32)
#     tfds = tf.data.Dataset.from_generator(gen, output_signature=output_signature)
#     if shuffle:
#         tfds = tfds.shuffle(1024)
#     tfds = tfds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
#     return tfds

# # Apply downsampling
# train_ds = downsample_dataset("train", fraction=0.1, shuffle=False)
# val_ds   = downsample_dataset("validation", fraction=0.1, shuffle=False)
# test_ds  = downsample_dataset("test", fraction=0.1, shuffle=False)

# # Update steps_per_epoch and validation_steps
# steps_per_epoch = max(1, int(len(ds["train"]) * 0.1) // BATCH_SIZE)
# validation_steps = max(1, int(len(ds["validation"]) * 0.1) //
```

```
BATCH_SIZE)

# print("After downsampling:")
# print("Train batches per epoch (approx):", steps_per_epoch)
# print("Validation steps (approx):", validation_steps)

# print("Train (after downsampling):", int(len(ds["train"]) * 0.1))
# print("Validation (after downsampling):", int(len(ds["validation"])
* 0.1))
# print("Test (after downsampling):", int(len(ds["test"]) * 0.1))

import matplotlib.pyplot as plt

# Take a batch from the training dataset
for batch in train_ds.take(1):
    plt.figure(figsize=(10, 10))
    for i in range(min(9, batch.shape[0])):
        plt.subplot(3, 3, i + 1)
        plt.imshow(batch[i])
        plt.axis("off")
    plt.show()
```




```
# -----------------------
# VAE definition (with test_step)
# -----------------------
class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        eps = tf.random.normal(shape=tf.shape(z_mean))
        return z_mean + tf.exp(0.5 * z_log_var) * eps

def build_encoder(img_size=IMG_SIZE, channels=CHANNELS,
latent_dim=LATENT_DIM):
```

```
    inputs = layers.Input(shape=(img_size, img_size, channels))
    x = layers.Conv2D(32, 3, activation="relu", strides=2,
padding="same")(inputs)
    x = layers.Conv2D(64, 3, activation="relu", strides=2,
padding="same")(x)
    x = layers.Conv2D(128, 3, activation="relu", strides=2,
padding="same")(x)
    x = layers.Conv2D(256, 3, activation="relu", strides=2,
padding="same")(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)
    z = Sampling()([z_mean, z_log_var])
    return Model(inputs, [z_mean, z_log_var, z], name="encoder")

def build_decoder(img_size=IMG_SIZE, channels=CHANNELS,
latent_dim=LATENT_DIM):
    latent_inputs = layers.Input(shape=(latent_dim,))
    x = layers.Dense(8*8*256, activation="relu")(latent_inputs)
    x = layers.Reshape((8,8,256))(x)
    x = layers.Conv2DTranspose(128, 3, strides=2, padding="same",
activation="relu")(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)
    x = layers.Conv2DTranspose(16, 3, strides=2, padding="same",
activation="relu")(x)
    outputs = layers.Conv2D(channels, 3, activation="sigmoid",
padding="same")(x)
    return Model(latent_inputs, outputs, name="decoder")

class VAE(Model):
    def __init__(self, encoder, decoder, beta=1.0, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.beta = beta
        self.total_loss_tracker = tf.keras.metrics.Mean(name="loss")
        self.recon_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.recon_loss_tracker,
self.kl_loss_tracker]

    def train_step(self, data):
```

```
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data, training=True)
            reconstruction = self.decoder(z, training=True)
            recon_loss =
tf.reduce_mean(tf.reduce_sum(tf.math.squared_difference(data,
reconstruction), axis=[1,2,3]))
            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 + self.beta * kl_loss
        grads = tape.gradient(total_loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(grads,
self.trainable_variables))
        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(),
                "recon_loss": self.recon_loss_tracker.result(),
                "kl_loss": self.kl_loss_tracker.result()}

    # IMPORTANT: implement test_step so validation doesn't require a
compiled loss
    def test_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        z_mean, z_log_var, z = self.encoder(data, training=False)
        reconstruction = self.decoder(z, training=False)
        recon_loss =
tf.reduce_mean(tf.reduce_sum(tf.math.squared_difference(data,
reconstruction), axis=[1,2,3]))
        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 + self.beta * kl_loss
        # update metrics (these are the same trackers used in
training)
        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(),
                "recon_loss": self.recon_loss_tracker.result(),
                "kl_loss": self.kl_loss_tracker.result()}

    def call(self, inputs, training=False):
        _, _, z = self.encoder(inputs, training=training)
        return self.decoder(z, training=training)

# Build & compile
encoder = build_encoder()
decoder = build_decoder()
```

```
vae = VAE(encoder, decoder, beta=BETA)
vae.compile(optimizer=optimizers.Adam(LEARNING_RATE))


# ----------------------
# Sampling Layer
# ----------------------
class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        eps = tf.random.normal(shape=tf.shape(z_mean))
        return z_mean + tf.exp(0.5 * z_log_var) * eps

# ----------------------
# Encoder with LeakyReLU
# ----------------------
def build_encoder(img_size=IMG_SIZE, channels=CHANNELS,
latent_dim=LATENT_DIM):
    inputs = layers.Input(shape=(img_size, img_size, channels))
    x = layers.Conv2D(32, 3, strides=2, padding="same")(inputs)
    x = layers.LeakyReLU(alpha=0.2)(x)
    x = layers.Conv2D(64, 3, strides=2, padding="same")(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
    x = layers.Conv2D(128, 3, strides=2, padding="same")(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
    x = layers.Conv2D(256, 3, strides=2, padding="same")(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
    x = layers.Flatten()(x)
    x = layers.Dense(512)(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
    z_mean = layers.Dense(latent_dim, name="z_mean")(x)
    z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
    z = Sampling()([z_mean, z_log_var])
    return Model(inputs, [z_mean, z_log_var, z], name="encoder")

# ----------------------
# Decoder with LeakyReLU
# ----------------------
def build_decoder(img_size=IMG_SIZE, channels=CHANNELS,
latent_dim=LATENT_DIM):
    latent_inputs = layers.Input(shape=(latent_dim,))
    x = layers.Dense(8 * 8 * 256)(latent_inputs)
    x = layers.LeakyReLU(alpha=0.2)(x)
    x = layers.Reshape((8, 8, 256))(x)
    x = layers.Conv2DTranspose(128, 3, strides=2, padding="same")(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
    x = layers.Conv2DTranspose(64, 3, strides=2, padding="same")(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
    x = layers.Conv2DTranspose(32, 3, strides=2, padding="same")(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
```

```
    x = layers.Conv2DTranspose(16, 3, strides=2, padding="same")(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
    outputs = layers.Conv2D(channels, 3, activation="sigmoid",
padding="same")(x)
    return Model(latent_inputs, outputs, name="decoder")

# ----------------------
# VAE Model Definition
# ----------------------
class VAE(Model):
    def __init__(self, encoder, decoder, beta=1.0, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.beta = beta
        self.total_loss_tracker = tf.keras.metrics.Mean(name="loss")
        self.recon_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.recon_loss_tracker,
self.kl_loss_tracker]

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data, training=True)
            reconstruction = self.decoder(z, training=True)
            recon_loss = tf.reduce_mean(tf.reduce_sum(tf.square(data -
reconstruction), axis=[1,2,3]))
            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 + self.beta * kl_loss
        grads = tape.gradient(total_loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(grads,
self.trainable_variables))
        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(),
                "recon_loss": self.recon_loss_tracker.result(),
                "kl_loss": self.kl_loss_tracker.result()}

    def test_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        z_mean, z_log_var, z = self.encoder(data, training=False)
```

```
        reconstruction = self.decoder(z, training=False)
        recon_loss = tf.reduce_mean(tf.reduce_sum(tf.square(data -
reconstruction), axis=[1,2,3]))
        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 + self.beta * kl_loss
        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(),
                "recon_loss": self.recon_loss_tracker.result(),
                "kl_loss": self.kl_loss_tracker.result()}

    def call(self, inputs, training=False):
        _, _, z = self.encoder(inputs, training=training)
        return self.decoder(z, training=training)

# -----------------------
# Build & Compile Model
# -----------------------
encoder = build_encoder()
decoder = build_decoder()
vae = VAE(encoder, decoder, beta=BETA)
vae.compile(optimizer=optimizers.Adam(LEARNING_RATE))

# Print model summary
encoder.summary()
decoder.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/activations/
leaky_relu.py:41: UserWarning: Argument `alpha` is deprecated. Use
`negative_slope` instead.
  warnings.warn(

Model: "encoder"
```

```
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to
┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━
┩
│ input_layer_2       │ (None, 128, 128,  │          0 │ -
│
│ (InputLayer)        │ 3)                │            │
│
├─────────────────────┼───────────────────┼────────────┼───────────────────
┤
│ conv2d_5 (Conv2D)   │ (None, 64, 64,    │        896 │
input_layer_2[0]… │
```

	32)		
leaky_re_lu (LeakyReLU)	(None, 64, 64, 32)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 64)	18,496	leaky_re_lu[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 32, 32, 64)	0	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 16, 16, 128)	73,856	leaky_re_lu_1[0]…
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 128)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 8, 8, 256)	295,168	leaky_re_lu_2[0]…
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 256)	0	conv2d_8[0][0]
flatten_1 (Flatten)	(None, 16384)	0	leaky_re_lu_3[0]…
dense_2 (Dense)	(None, 512)	8,389,120	flatten_1[0]

```
[0]     │
├───────────────────────────┼──────────────────────────┼─────────────────┼──────────────────
────────┤
│ leaky_re_lu_4             │ (None, 512)              │               0 │ dense_2[0][0]
│                                                                        │
│ (LeakyReLU)               │                          │                 │
│                                                                        │
├───────────────────────────┼──────────────────────────┼─────────────────┼──────────────────
────────┤
│ z_mean (Dense)            │ (None, 256)              │         131,328 │
leaky_re_lu_4[0]… │
├───────────────────────────┼──────────────────────────┼─────────────────┼──────────────────
────────┤
│ z_log_var (Dense)         │ (None, 256)              │         131,328 │
leaky_re_lu_4[0]… │
├───────────────────────────┼──────────────────────────┼─────────────────┼──────────────────
────────┤
│ sampling_1                │ (None, 256)              │               0 │ z_mean[0][0],
│                                                                        │
│ (Sampling)                │                          │                 │ z_log_var[0]
[0]     │
└───────────────────────────┴──────────────────────────┴─────────────────┴──────────────────
────────┘

 Total params: 9,040,192 (34.49 MB)

 Trainable params: 9,040,192 (34.49 MB)

 Non-trainable params: 0 (0.00 B)

Model: "decoder"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━
━━━━━━━━┓
┃ Layer (type)                            ┃ Output Shape                  ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━
━━━━━━━━┩
│ input_layer_3 (InputLayer)              │ (None, 256)                   │
0 │
├─────────────────────────────────────────┼───────────────────────────────┼────────────
────────┤
│ dense_3 (Dense)                         │ (None, 16384)                 │
4,210,688 │
├─────────────────────────────────────────┼───────────────────────────────┼────────────
────────┤
│ leaky_re_lu_5 (LeakyReLU)               │ (None, 16384)                 │
0 │
├─────────────────────────────────────────┼───────────────────────────────┼────────────
────────┤
```

```
| reshape_1 (Reshape)              | (None, 8, 8, 256)       |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| conv2d_transpose_4               | (None, 16, 16, 128)     |
295,040 |
| (Conv2DTranspose)                |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| leaky_re_lu_6 (LeakyReLU)        | (None, 16, 16, 128)     |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| conv2d_transpose_5               | (None, 32, 32, 64)      |
73,792 |
| (Conv2DTranspose)                |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| leaky_re_lu_7 (LeakyReLU)        | (None, 32, 32, 64)      |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| conv2d_transpose_6               | (None, 64, 64, 32)      |
18,464 |
| (Conv2DTranspose)                |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| leaky_re_lu_8 (LeakyReLU)        | (None, 64, 64, 32)      |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| conv2d_transpose_7               | (None, 128, 128, 16)    |
4,624 |
| (Conv2DTranspose)                |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| leaky_re_lu_9 (LeakyReLU)        | (None, 128, 128, 16)    |
0 |
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
| conv2d_9 (Conv2D)                | (None, 128, 128, 3)     |
435 |
└──────────────────────────────────┴─────────────────────────┴──────────
──────┘

 Total params: 4,603,043 (17.56 MB)
```

```
 Trainable params: 4,603,043 (17.56 MB)

 Non-trainable params: 0 (0.00 B)
```

```
# -----------------------
# Fit with explicit steps_per_epoch and validation_steps
# -----------------------
print("Starting training...")
history = vae.fit(
    train_ds,
    epochs=100,
    steps_per_epoch=steps_per_epoch,       # number of train batches
per epoch
    validation_data=val_ds,
    validation_steps=validation_steps     # number of val batches for
validation
)

# Save
vae.build(input_shape=(None, IMG_SIZE, IMG_SIZE, CHANNELS)) # Build
the model before saving weights
encoder.save(os.path.join(MODEL_DIR, "encoder.h5"))
decoder.save(os.path.join(MODEL_DIR, "decoder.h5"))
vae.save_weights(os.path.join(MODEL_DIR, "vae.weights.h5"))

# Save the entire VAE model (architecture + weights) for backend use
vae.save(os.path.join(MODEL_DIR, "vae_model.h5"))  # Save the whole
model (architecture + weights)

print("Model saved successfully.")
```

```
Starting training...
Epoch 1/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 113s 232ms/step - kl_loss: 126.7652 -
loss: 3853.5295 - recon_loss: 3790.1482 - val_kl_loss: 101.8955 -
val_loss: 1815.7919 - val_recon_loss: 1764.8440
Epoch 2/100
  1/397 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - kl_loss: 78.2839 - loss:
1947.0757 - recon_loss: 1907.9337

/usr/local/lib/python3.12/dist-packages/keras/src/trainers/
epoch_iterator.py:160: UserWarning: Your input ran out of data;
interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to
use the `.repeat()` function when building your dataset.
  self._interrupted_warning()

397/397 ━━━━━━━━━━━━━━━━━━━━ 91s 229ms/step - kl_loss: 108.9235 -
loss: 1732.7950 - recon_loss: 1678.3331 - val_kl_loss: 141.6399 -
val_loss: 1478.0538 - val_recon_loss: 1407.2339
Epoch 3/100
```

```
397/397 ━━━━━━━━━━━━━━━━━━━━ 90s 226ms/step - kl_loss: 143.6692 -
loss: 1411.5468 - recon_loss: 1339.7124 - val_kl_loss: 157.9417 -
val_loss: 1215.0144 - val_recon_loss: 1136.0433
Epoch 4/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 88s 223ms/step - kl_loss: 160.0324 -
loss: 1164.5609 - recon_loss: 1084.5447 - val_kl_loss: 165.7970 -
val_loss: 1069.7517 - val_recon_loss: 986.8535
Epoch 5/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 88s 223ms/step - kl_loss: 172.0611 -
loss: 1025.4067 - recon_loss: 939.3762 - val_kl_loss: 181.1554 -
val_loss: 968.1237 - val_recon_loss: 877.5459
Epoch 6/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 91s 230ms/step - kl_loss: 183.6887 -
loss: 931.8051 - recon_loss: 839.9609 - val_kl_loss: 185.9295 -
val_loss: 871.9246 - val_recon_loss: 778.9597
Epoch 7/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 210ms/step - kl_loss: 185.9911 -
loss: 828.3513 - recon_loss: 735.3556 - val_kl_loss: 188.8441 -
val_loss: 811.9066 - val_recon_loss: 717.4847
Epoch 8/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 190.7372 -
loss: 781.5989 - recon_loss: 686.2302 - val_kl_loss: 190.8081 -
val_loss: 774.2175 - val_recon_loss: 678.8132
Epoch 9/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 192.6033 -
loss: 720.2183 - recon_loss: 623.9167 - val_kl_loss: 189.1753 -
val_loss: 733.7488 - val_recon_loss: 639.1612
Epoch 10/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 194.8019 -
loss: 684.5867 - recon_loss: 587.1859 - val_kl_loss: 193.8644 -
val_loss: 700.3517 - val_recon_loss: 603.4195
Epoch 11/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 198.3167 -
loss: 650.8343 - recon_loss: 551.6760 - val_kl_loss: 199.8348 -
val_loss: 676.0852 - val_recon_loss: 576.1680
Epoch 12/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 215ms/step - kl_loss: 197.4458 -
loss: 614.3423 - recon_loss: 515.6194 - val_kl_loss: 202.3077 -
val_loss: 654.9505 - val_recon_loss: 553.7965
Epoch 13/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 199.4808 -
loss: 602.8712 - recon_loss: 503.1307 - val_kl_loss: 192.8605 -
val_loss: 642.5681 - val_recon_loss: 546.1378
Epoch 14/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 197.4272 -
loss: 573.3824 - recon_loss: 474.6688 - val_kl_loss: 201.4945 -
val_loss: 624.6237 - val_recon_loss: 523.8764
Epoch 15/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 197.4689 -
```

```
loss: 554.2175 - recon_loss: 455.4830 - val_kl_loss: 202.5294 -
val_loss: 611.4374 - val_recon_loss: 510.1727
Epoch 16/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 86s 216ms/step - kl_loss: 198.5797 -
loss: 540.7754 - recon_loss: 441.4854 - val_kl_loss: 194.0570 -
val_loss: 599.1907 - val_recon_loss: 502.1622
Epoch 17/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 197.1327 -
loss: 525.3777 - recon_loss: 426.8112 - val_kl_loss: 196.3353 -
val_loss: 591.3932 - val_recon_loss: 493.2256
Epoch 18/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 210ms/step - kl_loss: 195.9290 -
loss: 511.5315 - recon_loss: 413.5672 - val_kl_loss: 196.4717 -
val_loss: 579.7898 - val_recon_loss: 481.5538
Epoch 19/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 196.3365 -
loss: 495.2478 - recon_loss: 397.0797 - val_kl_loss: 195.5789 -
val_loss: 571.9053 - val_recon_loss: 474.1161
Epoch 20/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 194.9063 -
loss: 484.6638 - recon_loss: 387.2107 - val_kl_loss: 188.2451 -
val_loss: 566.9370 - val_recon_loss: 472.8143
Epoch 21/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 195.0635 -
loss: 478.6227 - recon_loss: 381.0909 - val_kl_loss: 192.0426 -
val_loss: 562.2097 - val_recon_loss: 466.1884
Epoch 22/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 196.3049 -
loss: 472.0302 - recon_loss: 373.8777 - val_kl_loss: 187.3457 -
val_loss: 556.5628 - val_recon_loss: 462.8900
Epoch 23/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 194.7290 -
loss: 464.4327 - recon_loss: 367.0681 - val_kl_loss: 193.5776 -
val_loss: 549.9200 - val_recon_loss: 453.1313
Epoch 24/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 194.2824 -
loss: 456.8365 - recon_loss: 359.6953 - val_kl_loss: 190.6814 -
val_loss: 543.7041 - val_recon_loss: 448.3633
Epoch 25/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 86s 217ms/step - kl_loss: 194.1212 -
loss: 447.6507 - recon_loss: 350.5901 - val_kl_loss: 191.3960 -
val_loss: 542.9481 - val_recon_loss: 447.2502
Epoch 26/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 215ms/step - kl_loss: 192.8582 -
loss: 436.9966 - recon_loss: 340.5675 - val_kl_loss: 191.7127 -
val_loss: 535.2379 - val_recon_loss: 439.3815
Epoch 27/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 192.9819 -
loss: 438.8006 - recon_loss: 342.3096 - val_kl_loss: 199.6924 -
```

```
val_loss: 532.8544 - val_recon_loss: 433.0080
Epoch 28/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 216ms/step - kl_loss: 192.9544 -
loss: 428.2887 - recon_loss: 331.8115 - val_kl_loss: 184.9339 -
val_loss: 534.0214 - val_recon_loss: 441.5544
Epoch 29/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 192.1455 -
loss: 427.1844 - recon_loss: 331.1115 - val_kl_loss: 186.5835 -
val_loss: 525.2082 - val_recon_loss: 431.9164
Epoch 30/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 191.7324 -
loss: 419.2663 - recon_loss: 323.4001 - val_kl_loss: 195.3132 -
val_loss: 522.9769 - val_recon_loss: 425.3205
Epoch 31/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 191.6064 -
loss: 412.3157 - recon_loss: 316.5124 - val_kl_loss: 189.9368 -
val_loss: 519.4412 - val_recon_loss: 424.4728
Epoch 32/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 190.3120 -
loss: 406.7287 - recon_loss: 311.5727 - val_kl_loss: 193.4462 -
val_loss: 514.6674 - val_recon_loss: 417.9442
Epoch 33/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 191.8482 -
loss: 404.8640 - recon_loss: 308.9399 - val_kl_loss: 190.5367 -
val_loss: 512.2364 - val_recon_loss: 416.9680
Epoch 34/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 190.7929 -
loss: 399.3550 - recon_loss: 303.9586 - val_kl_loss: 192.3810 -
val_loss: 512.2092 - val_recon_loss: 416.0187
Epoch 35/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 189.5810 -
loss: 392.4828 - recon_loss: 297.6922 - val_kl_loss: 186.5710 -
val_loss: 507.9233 - val_recon_loss: 414.6377
Epoch 36/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 188.2630 -
loss: 390.2875 - recon_loss: 296.1559 - val_kl_loss: 182.4393 -
val_loss: 506.6954 - val_recon_loss: 415.4757
Epoch 37/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 188.4711 -
loss: 387.9697 - recon_loss: 293.7341 - val_kl_loss: 179.3730 -
val_loss: 502.8741 - val_recon_loss: 413.1877
Epoch 38/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 98s 247ms/step - kl_loss: 187.4970 -
loss: 380.7285 - recon_loss: 286.9801 - val_kl_loss: 184.8230 -
val_loss: 501.1618 - val_recon_loss: 408.7504
Epoch 39/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 73s 184ms/step - kl_loss: 188.6371 -
loss: 380.6442 - recon_loss: 286.3256 - val_kl_loss: 181.0891 -
val_loss: 498.1673 - val_recon_loss: 407.6227
```

```
Epoch 40/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 185.9325 -
loss: 375.3024 - recon_loss: 282.3361 - val_kl_loss: 180.2877 -
val_loss: 499.1284 - val_recon_loss: 408.9846
Epoch 41/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 186.5504 -
loss: 373.7096 - recon_loss: 280.4343 - val_kl_loss: 181.3772 -
val_loss: 494.5133 - val_recon_loss: 403.8247
Epoch 42/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 187.3870 -
loss: 371.9561 - recon_loss: 278.2628 - val_kl_loss: 180.6260 -
val_loss: 492.6389 - val_recon_loss: 402.3260
Epoch 43/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 215ms/step - kl_loss: 186.4799 -
loss: 367.1249 - recon_loss: 273.8850 - val_kl_loss: 187.6343 -
val_loss: 492.0283 - val_recon_loss: 398.2112
Epoch 44/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 211ms/step - kl_loss: 186.9799 -
loss: 364.9815 - recon_loss: 271.4916 - val_kl_loss: 187.9186 -
val_loss: 492.9370 - val_recon_loss: 398.9777
Epoch 45/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 186.6892 -
loss: 362.3646 - recon_loss: 269.0201 - val_kl_loss: 182.4887 -
val_loss: 489.6863 - val_recon_loss: 398.4420
Epoch 46/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 183.9746 -
loss: 355.0417 - recon_loss: 263.0544 - val_kl_loss: 185.3980 -
val_loss: 487.7436 - val_recon_loss: 395.0446
Epoch 47/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 185.3124 -
loss: 355.2714 - recon_loss: 262.6152 - val_kl_loss: 182.8681 -
val_loss: 485.0374 - val_recon_loss: 393.6034
Epoch 48/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 184.6990 -
loss: 353.1455 - recon_loss: 260.7961 - val_kl_loss: 182.5622 -
val_loss: 483.6089 - val_recon_loss: 392.3278
Epoch 49/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 182.5410 -
loss: 347.9254 - recon_loss: 256.6548 - val_kl_loss: 178.1419 -
val_loss: 482.7910 - val_recon_loss: 393.7198
Epoch 50/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 184.3720 -
loss: 349.9405 - recon_loss: 257.7545 - val_kl_loss: 181.0360 -
val_loss: 480.4009 - val_recon_loss: 389.8830
Epoch 51/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 185.3696 -
loss: 348.2293 - recon_loss: 255.5444 - val_kl_loss: 179.0532 -
val_loss: 479.3583 - val_recon_loss: 389.8316
Epoch 52/100
```

```
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 182.9436 -
loss: 342.7231 - recon_loss: 251.2512 - val_kl_loss: 177.1052 -
val_loss: 481.0240 - val_recon_loss: 392.4714
Epoch 53/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 183.0695 -
loss: 345.3685 - recon_loss: 253.8337 - val_kl_loss: 182.7268 -
val_loss: 476.3016 - val_recon_loss: 384.9384
Epoch 54/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 87s 219ms/step - kl_loss: 182.7924 -
loss: 339.8286 - recon_loss: 248.4323 - val_kl_loss: 174.8596 -
val_loss: 475.7113 - val_recon_loss: 388.2815
Epoch 55/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 210ms/step - kl_loss: 182.2934 -
loss: 336.0079 - recon_loss: 244.8611 - val_kl_loss: 180.0494 -
val_loss: 473.4184 - val_recon_loss: 383.3937
Epoch 56/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 182.7289 -
loss: 336.9541 - recon_loss: 245.5898 - val_kl_loss: 179.4378 -
val_loss: 472.4302 - val_recon_loss: 382.7114
Epoch 57/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 181.7954 -
loss: 334.2229 - recon_loss: 243.3252 - val_kl_loss: 175.6195 -
val_loss: 473.7302 - val_recon_loss: 385.9205
Epoch 58/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 182.6346 -
loss: 335.4322 - recon_loss: 244.1148 - val_kl_loss: 181.5050 -
val_loss: 471.1541 - val_recon_loss: 380.4016
Epoch 59/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 182.1035 -
loss: 332.2155 - recon_loss: 241.1638 - val_kl_loss: 177.1125 -
val_loss: 470.1824 - val_recon_loss: 381.6263
Epoch 60/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 181.2315 -
loss: 326.5168 - recon_loss: 235.9010 - val_kl_loss: 175.2708 -
val_loss: 469.6216 - val_recon_loss: 381.9862
Epoch 61/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 180.9902 -
loss: 326.7655 - recon_loss: 236.2703 - val_kl_loss: 183.6490 -
val_loss: 467.4348 - val_recon_loss: 375.6102
Epoch 62/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 210ms/step - kl_loss: 182.1210 -
loss: 325.3265 - recon_loss: 234.2661 - val_kl_loss: 179.0830 -
val_loss: 466.1223 - val_recon_loss: 376.5807
Epoch 63/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 210ms/step - kl_loss: 180.9976 -
loss: 323.8239 - recon_loss: 233.3251 - val_kl_loss: 185.4650 -
val_loss: 467.5808 - val_recon_loss: 374.8484
Epoch 64/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 215ms/step - kl_loss: 180.7664 -
```

```
loss: 322.3535 - recon_loss: 231.9703 - val_kl_loss: 175.4102 -
val_loss: 464.6532 - val_recon_loss: 376.9482
Epoch 65/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 181.6115 -
loss: 324.8668 - recon_loss: 234.0611 - val_kl_loss: 175.8858 -
val_loss: 464.4009 - val_recon_loss: 376.4579
Epoch 66/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 178.8120 -
loss: 316.3154 - recon_loss: 226.9094 - val_kl_loss: 175.0534 -
val_loss: 464.4220 - val_recon_loss: 376.8954
Epoch 67/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 180.2836 -
loss: 319.3364 - recon_loss: 229.1946 - val_kl_loss: 175.1710 -
val_loss: 462.2750 - val_recon_loss: 374.6895
Epoch 68/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 214ms/step - kl_loss: 179.2089 -
loss: 316.2050 - recon_loss: 226.6006 - val_kl_loss: 175.1222 -
val_loss: 462.5642 - val_recon_loss: 375.0031
Epoch 69/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 86s 218ms/step - kl_loss: 178.9884 -
loss: 314.1700 - recon_loss: 224.6757 - val_kl_loss: 175.5686 -
val_loss: 460.0536 - val_recon_loss: 372.2692
Epoch 70/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 86s 216ms/step - kl_loss: 179.0684 -
loss: 312.4656 - recon_loss: 222.9314 - val_kl_loss: 174.7529 -
val_loss: 459.3632 - val_recon_loss: 371.9867
Epoch 71/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 215ms/step - kl_loss: 179.8289 -
loss: 313.6096 - recon_loss: 223.6951 - val_kl_loss: 175.8995 -
val_loss: 461.0616 - val_recon_loss: 373.1119
Epoch 72/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 215ms/step - kl_loss: 179.2427 -
loss: 310.9734 - recon_loss: 221.3521 - val_kl_loss: 178.0737 -
val_loss: 458.8851 - val_recon_loss: 369.8484
Epoch 73/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 85s 213ms/step - kl_loss: 179.8041 -
loss: 310.7659 - recon_loss: 220.8638 - val_kl_loss: 176.4449 -
val_loss: 458.2409 - val_recon_loss: 370.0184
Epoch 74/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 178.0807 -
loss: 308.6753 - recon_loss: 219.6349 - val_kl_loss: 179.0429 -
val_loss: 457.4668 - val_recon_loss: 367.9454
Epoch 75/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 178.5547 -
loss: 305.9591 - recon_loss: 216.6818 - val_kl_loss: 173.2050 -
val_loss: 457.2966 - val_recon_loss: 370.6941
Epoch 76/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 211ms/step - kl_loss: 176.3226 -
loss: 300.9209 - recon_loss: 212.7596 - val_kl_loss: 172.5565 -
```

```
val_loss: 455.6700 - val_recon_loss: 369.3917
Epoch 77/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 77s 195ms/step - kl_loss: 177.5479 -
loss: 304.7189 - recon_loss: 215.9449 - val_kl_loss: 169.4243 -
val_loss: 455.2054 - val_recon_loss: 370.4932
Epoch 78/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 128s 324ms/step - kl_loss: 177.9488 -
loss: 305.9331 - recon_loss: 216.9587 - val_kl_loss: 173.0288 -
val_loss: 454.2426 - val_recon_loss: 367.7282
Epoch 79/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 210ms/step - kl_loss: 177.6028 -
loss: 302.5898 - recon_loss: 213.7884 - val_kl_loss: 172.9750 -
val_loss: 455.3479 - val_recon_loss: 368.8604
Epoch 80/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 177.0632 -
loss: 299.8981 - recon_loss: 211.3665 - val_kl_loss: 180.4977 -
val_loss: 452.2907 - val_recon_loss: 362.0418
Epoch 81/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 177.4750 -
loss: 301.9592 - recon_loss: 213.2218 - val_kl_loss: 178.7307 -
val_loss: 452.4159 - val_recon_loss: 363.0505
Epoch 82/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 176.7240 -
loss: 299.6747 - recon_loss: 211.3127 - val_kl_loss: 174.9691 -
val_loss: 451.6470 - val_recon_loss: 364.1624
Epoch 83/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 176.5528 -
loss: 298.0345 - recon_loss: 209.7581 - val_kl_loss: 172.9971 -
val_loss: 452.3188 - val_recon_loss: 365.8202
Epoch 84/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 177.6260 -
loss: 299.0969 - recon_loss: 210.2839 - val_kl_loss: 177.4576 -
val_loss: 451.1262 - val_recon_loss: 362.3973
Epoch 85/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 174.0850 -
loss: 292.6502 - recon_loss: 205.6078 - val_kl_loss: 173.6883 -
val_loss: 450.2018 - val_recon_loss: 363.3576
Epoch 86/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 209ms/step - kl_loss: 175.9132 -
loss: 294.9383 - recon_loss: 206.9816 - val_kl_loss: 173.9330 -
val_loss: 449.9289 - val_recon_loss: 362.9623
Epoch 87/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 77s 195ms/step - kl_loss: 176.2081 -
loss: 295.0035 - recon_loss: 206.8994 - val_kl_loss: 170.2385 -
val_loss: 448.9372 - val_recon_loss: 363.8179
Epoch 88/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 142s 242ms/step - kl_loss: 174.9968 -
loss: 290.8130 - recon_loss: 203.3147 - val_kl_loss: 169.7401 -
val_loss: 446.9751 - val_recon_loss: 362.1050
```

```
Epoch 89/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 210ms/step - kl_loss: 173.9716 -
loss: 290.1295 - recon_loss: 203.1438 - val_kl_loss: 173.4629 -
val_loss: 447.2426 - val_recon_loss: 360.5111
Epoch 90/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 175.5027 -
loss: 291.0460 - recon_loss: 203.2946 - val_kl_loss: 174.3886 -
val_loss: 447.5047 - val_recon_loss: 360.3104
Epoch 91/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 174.4395 -
loss: 289.3593 - recon_loss: 202.1396 - val_kl_loss: 175.0871 -
val_loss: 446.2183 - val_recon_loss: 358.6747
Epoch 92/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 174.3379 -
loss: 288.2228 - recon_loss: 201.0539 - val_kl_loss: 169.7458 -
val_loss: 448.4787 - val_recon_loss: 363.6058
Epoch 93/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 175.1653 -
loss: 288.7104 - recon_loss: 201.1279 - val_kl_loss: 169.8662 -
val_loss: 444.9477 - val_recon_loss: 360.0146
Epoch 94/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 174.2249 -
loss: 287.1753 - recon_loss: 200.0629 - val_kl_loss: 171.1138 -
val_loss: 445.6196 - val_recon_loss: 360.0627
Epoch 95/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 174.5315 -
loss: 286.3491 - recon_loss: 199.0834 - val_kl_loss: 168.1643 -
val_loss: 443.6255 - val_recon_loss: 359.5434
Epoch 96/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 83s 210ms/step - kl_loss: 173.9683 -
loss: 282.9483 - recon_loss: 195.9642 - val_kl_loss: 170.7166 -
val_loss: 444.4608 - val_recon_loss: 359.1025
Epoch 97/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 173.3535 -
loss: 282.3488 - recon_loss: 195.6721 - val_kl_loss: 171.3916 -
val_loss: 444.1351 - val_recon_loss: 358.4394
Epoch 98/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 211ms/step - kl_loss: 173.9797 -
loss: 283.3767 - recon_loss: 196.3869 - val_kl_loss: 168.3330 -
val_loss: 442.3321 - val_recon_loss: 358.1656
Epoch 99/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 213ms/step - kl_loss: 173.0468 -
loss: 282.6350 - recon_loss: 196.1116 - val_kl_loss: 166.8408 -
val_loss: 442.0617 - val_recon_loss: 358.6414
Epoch 100/100
397/397 ━━━━━━━━━━━━━━━━━━━━ 84s 212ms/step - kl_loss: 171.7968 -
loss: 280.0370 - recon_loss: 194.1386 - val_kl_loss: 170.1760 -
val_loss: 441.0966 - val_recon_loss: 356.0086
```

```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

Model saved successfully.
```

```python
# -----------------------
# Evaluation (MSE & SSIM on some val batches)
# -----------------------
def evaluate_on_dataset(tf_dataset, max_batches=100):
    mse_vals = []
    ssim_vals = []
    cnt = 0
    for batch in tf_dataset:
        recon = vae(batch, training=False)
        batch_np = batch.numpy()
        recon_np = recon.numpy()
        mse_batch = np.mean((batch_np - recon_np)**2, axis=(1,2,3))
        mse_vals.extend(mse_batch.tolist())
        for i in range(batch_np.shape[0]):
            s = tf.image.ssim(batch_np[i], recon_np[i],
max_val=1.0).numpy()
            ssim_vals.append(float(s))
        cnt += 1
        if cnt >= max_batches:
            break
    return np.mean(mse_vals), np.mean(ssim_vals)

print("Evaluating on validation (first 100 batches)...")
val_mse, val_ssim = evaluate_on_dataset(val_ds, max_batches=100)
print(f"Validation MSE: {val_mse:.6f}, SSIM: {val_ssim:.4f}")
```

```
Evaluating on validation (first 100 batches)...
Validation MSE: 0.007225, SSIM: 0.7106
```

```python
# -----------------------
# Evaluation (MSE & SSIM on some train batches)
```

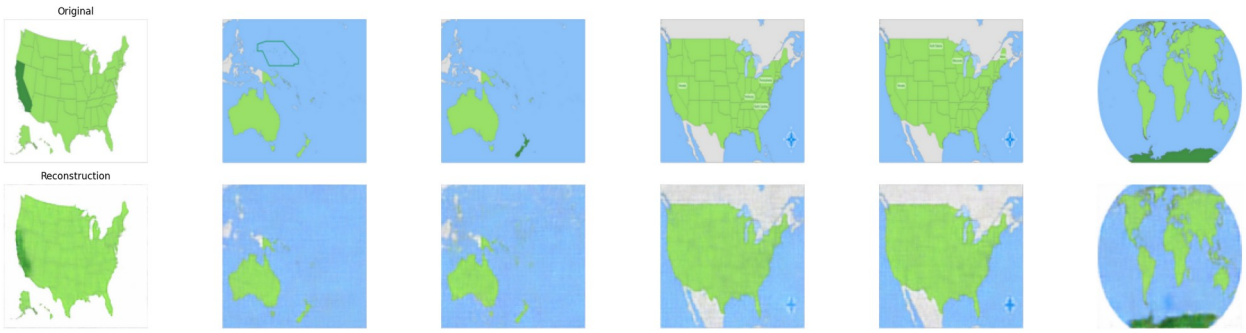
```python
# ----------------------
def evaluate_on_dataset(tf_dataset, max_batches=50):  # smaller number
for train set
    mse_vals = []
    ssim_vals = []
    cnt = 0
    for batch in tf_dataset:
        recon = vae(batch, training=False)
        batch_np = batch.numpy()
        recon_np = recon.numpy()
        # MSE per image
        mse_batch = np.mean((batch_np - recon_np)**2, axis=(1,2,3))
        mse_vals.extend(mse_batch.tolist())
        # SSIM per image
        for i in range(batch_np.shape[0]):
            s = tf.image.ssim(batch_np[i], recon_np[i],
max_val=1.0).numpy()
            ssim_vals.append(float(s))
        cnt += 1
        if cnt >= max_batches:
            break
    return np.mean(mse_vals), np.mean(ssim_vals)

print("Evaluating on training set (first 50 batches)...")
train_mse, train_ssim = evaluate_on_dataset(train_ds, max_batches=50)
print(f"Training MSE: {train_mse:.6f}, SSIM: {train_ssim:.4f}")
```

```
Evaluating on training set (first 50 batches)...
Training MSE: 0.003881, SSIM: 0.7625
```

```python
import random
import math
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

def sample_indices_by_topic(split, topic_keyword="geography",
max_samples=1000):
    """Return list of dataset indices from ds[split] whose topic
matches topic_keyword."""
    topic_keyword = topic_keyword.lower().strip()
    indices = []
    for i, ex in enumerate(ds[split]):
        t = ex.get("topic", None)
        # handle None, str, list
        if t is None:
            continue
        if isinstance(t, list):
```

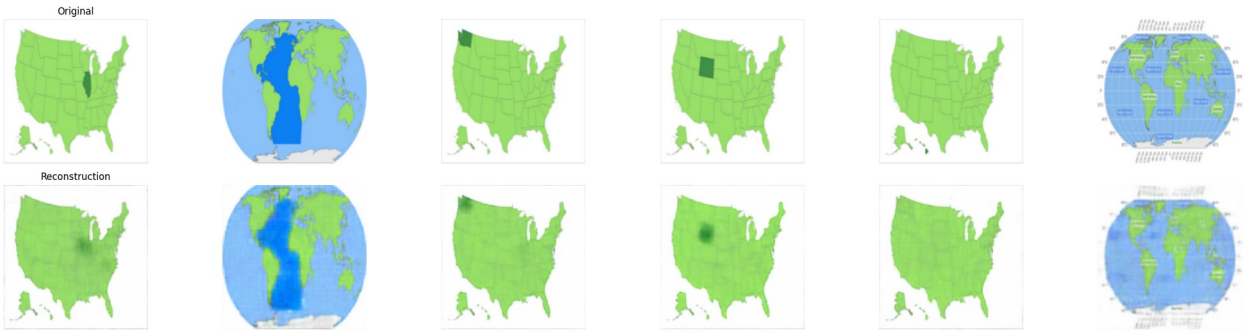
```
            t_vals = [str(x).lower() for x in t]
        else:
            t_vals = [str(t).lower()]
        # check substring match (so 'World Geography' also matches)
        if any(topic_keyword in tv for tv in t_vals):
            # ensure image exists
            if ex.get("image", None) is not None:
                indices.append(i)
    return indices

def show_reconstructions_for_topic(split="train", topic="geography",
n=6, random_seed=42):
    # find candidate indices
    indices = sample_indices_by_topic(split, topic_keyword=topic)
    if len(indices) == 0:
        print(f"No images found for topic '{topic}' in split
'{split}'.")
        return
    # choose n random indices
    random.seed(random_seed)
    n_show = min(n, len(indices))
    chosen = random.sample(indices, n_show)

    originals = []
    reconstructions = []

    for idx in chosen:
        ex = ds[split][idx]
        pil_img = ex.get("image", None)
        if pil_img is None:
            continue
        arr = preprocess_pil_image(pil_img)                    # your
preprocessing function
        arr_tf = tf.expand_dims(arr, axis=0)                   # shape
(1,H,W,C)
        recon_tf = vae(arr_tf, training=False)                 # runs
through your VAE
        recon_np = recon_tf.numpy()[0]
        originals.append(arr)
        reconstructions.append(recon_np)

    # plotting
    m = len(originals)
    cols = min(n, m)
    plt.figure(figsize=(4*cols, 6))
    for i in range(m):
        ax = plt.subplot(2, cols, i+1)
        plt.imshow(np.clip(originals[i], 0, 1))
        ax.axis("off")
        if i == 0:
```

```
            ax.set_title("Original")
        ax = plt.subplot(2, cols, cols + i + 1)
        plt.imshow(np.clip(reconstructions[i], 0, 1))
        ax.axis("off")
        if i == 0:
            ax.set_title("Reconstruction")
    plt.tight_layout()
    plt.show()

# Example usage:
show_reconstructions_for_topic(split="train", topic="geography", n=6,
random_seed=7)
```

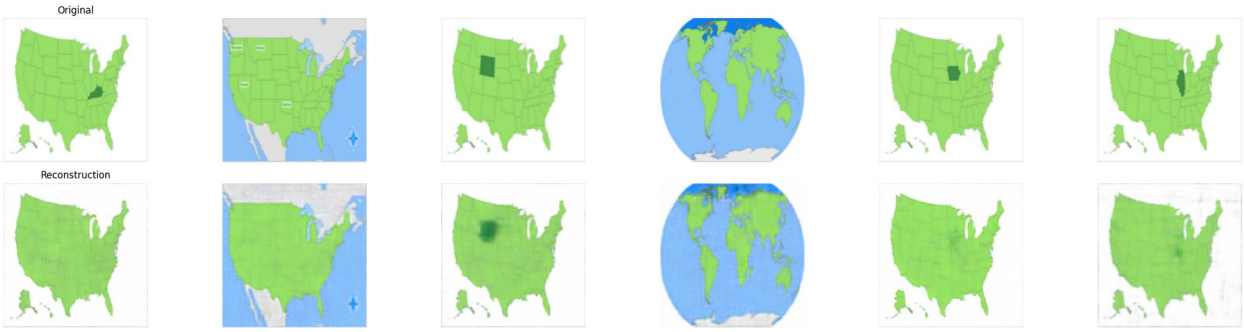



```
show_reconstructions_for_topic(split="train", topic="geography", n=6,
random_seed=2)
```


```
show_reconstructions_for_topic(split="train", topic="geography", n=6,
random_seed=5)
```




```
show_reconstructions_for_topic(split="train", topic="geography", n=6,
random_seed=60)
```




```
# ------------------------
# Metrics computation snippet (append after training/eval)
# ------------------------
import math
import numpy as np
import tensorflow as tf
from tqdm import tqdm
from scipy import linalg
from sklearn.metrics.pairwise import cosine_similarity
from tensorflow.keras.applications.inception_v3 import InceptionV3,
preprocess_input as inception_preprocess

# ---- Config ----
NUM_SAMPLES = 500           # how many images to sample for metrics
(reduce if memory/time constrained)
BATCH_METRIC = 64
INCEPTION_SIZE = 299       # InceptionV3 input size
RANDOM_SEED = 42

# ---- Utility: gather N originals + reconstructions from a
tf.data.Dataset or HF ds split ----
def gather_originals_and_recons_from_tfds(tf_dataset, vae_model,
num_samples=NUM_SAMPLES):
    originals = []
    recons = []
```

```
    cnt = 0
    for batch in tf_dataset:
        # batch: shape (B,H,W,C) with values in [0,1]
        recon_batch = vae_model(batch, training=False).numpy()
        orig_batch = batch.numpy()
        batch_size = orig_batch.shape[0]
        for i in range(batch_size):
            originals.append(orig_batch[i])
            recons.append(recon_batch[i])
            cnt += 1
            if cnt >= num_samples:
                break
        if cnt >= num_samples:
            break
    originals = np.asarray(originals)[:num_samples]
    recons = np.asarray(recons)[:num_samples]
    return originals, recons

# If you want to sample from HF dataset 'ds' directly (useful if tf
datasets were downsampled), use this:
def gather_from_hf_split(split_name="validation",
num_samples=NUM_SAMPLES):
    originals = []
    recons = []
    idxs = []
    for i, ex in enumerate(ds[split_name]):
        if ex.get("image", None) is None:
            continue
        arr = preprocess_pil_image(ex["image"])
        originals.append(arr)
        idxs.append(i)
        if len(originals) >= num_samples:
            break
    originals = np.array(originals[:num_samples], dtype=np.float32)
    # compute recons using vae
    recon_batches = []
    for i in range(0, originals.shape[0], BATCH_METRIC):
        batch = originals[i:i+BATCH_METRIC]
        batch_tf = tf.convert_to_tensor(batch)
        recon_tf = vae(batch_tf, training=False)
        recon_batches.append(recon_tf.numpy())
    recons = np.vstack(recon_batches)[:num_samples]
    return originals, recons

# ---- choose source and gather ----
# Option A: from your downsampled tf.data 'val_ds' or 'train_ds'
# originals, recons = gather_originals_and_recons_from_tfds(val_ds,
vae, num_samples=NUM_SAMPLES)

# Option B: from HF split directly (works reliably)
```

```
originals, recons = gather_from_hf_split(split_name="validation",
num_samples=NUM_SAMPLES)
print(f"Collected {originals.shape[0]} original images and
{recons.shape[0]} reconstructions")

# ---- Basic per-image metrics: MSE, MAE, SSIM ----
def compute_mse_mae_ssim_batch(orig, recon):
    # orig, recon shape (N,H,W,C), values in [0,1]
    mse = np.mean((orig - recon)**2, axis=(1,2,3))
    mae = np.mean(np.abs(orig - recon), axis=(1,2,3))
    # SSIM compute per image using tf.image.ssim
    ssim_vals = []
    for i in range(orig.shape[0]):
        s = tf.image.ssim(orig[i], recon[i], max_val=1.0).numpy()
        ssim_vals.append(float(s))
    ssim_vals = np.array(ssim_vals)
    return mse, mae, ssim_vals

mse_vals, mae_vals, ssim_vals = compute_mse_mae_ssim_batch(originals,
recons)
print(f"MSE (mean ± std) : {mse_vals.mean():.6f} ±
{mse_vals.std():.6f}")
print(f"MAE (mean ± std) : {mae_vals.mean():.6f} ±
{mae_vals.std():.6f}")
print(f"SSIM (mean ± std): {ssim_vals.mean():.4f} ±
{ssim_vals.std():.4f}")
```

```
Collected 500 original images and 500 reconstructions
MSE (mean ± std) : 0.007712 ± 0.009950
MAE (mean ± std) : 0.045278 ± 0.034165
SSIM (mean ± std): 0.6955 ± 0.2137
```

```
# ---- Prepare InceptionV3 model for embeddings (for FID and cosine
similarity) ----
inception_model = InceptionV3(include_top=False, pooling="avg",
input_shape=(INCEPTION_SIZE, INCEPTION_SIZE, 3))

def get_inception_activations(images):
    """
    images: numpy array shape (N,H,W,C) values in [0,1]
    returns: activations shape (N, feat_dim)
    """
    # Resize to INCEPTION_SIZE and scale to 0-255 for preprocess_input
    imgs_resized = tf.image.resize(images, (INCEPTION_SIZE,
INCEPTION_SIZE)).numpy() * 255.0
    imgs_proc = inception_preprocess(imgs_resized)  # scales to model
expected range
    activations = inception_model.predict(imgs_proc,
batch_size=BATCH_METRIC, verbose=0)
```

```
    return activations

print("Computing Inception activations (may take a while)...")
orig_acts = get_inception_activations(originals)
recon_acts = get_inception_activations(recons)
print("Activations shapes:", orig_acts.shape, recon_acts.shape)

# ---- FID computation ----
def calculate_fid(mu1, sigma1, mu2, sigma2, eps=1e-6):
    """
    mu1, sigma1: mean and covariance of activations for set 1
    mu2, sigma2: mean and covariance for set 2
    """
    diff = mu1 - mu2
    # sqrt of product of covariances
    covmean, _ = linalg.sqrtm(sigma1.dot(sigma2), disp=False)
    # numerical error might give slight imaginary component
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    fid = diff.dot(diff) + np.trace(sigma1) + np.trace(sigma2) - 2 *
np.trace(covmean)
    return float(np.real(fid))

mu_orig = np.mean(orig_acts, axis=0)
sigma_orig = np.cov(orig_acts, rowvar=False)
mu_recon = np.mean(recon_acts, axis=0)
sigma_recon = np.cov(recon_acts, rowvar=False)

fid_value = calculate_fid(mu_orig, sigma_orig, mu_recon, sigma_recon)
print(f"FID between originals and reconstructions (lower better):
{fid_value:.4f}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/inception_v3/
inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
Computing Inception activations (may take a while)...
Activations shapes: (500, 2048) (500, 2048)

/tmp/ipython-input-628984618.py:28: DeprecationWarning: The `disp`
argument is deprecated and will be removed in SciPy 1.18.0.
  covmean, _ = linalg.sqrtm(sigma1.dot(sigma2), disp=False)

FID between originals and reconstructions (lower better): 184.7342
```

```
# ---- Cosine similarity between original & reconstruction embeddings
(per-image) ----
# compute cosine between corresponding pairs using Inception
activations
cos_sims = []
```

```
for i in range(orig_acts.shape[0]):
    a = orig_acts[i].reshape(1, -1)
    b = recon_acts[i].reshape(1, -1)
    cos = cosine_similarity(a, b)[0][0]
    cos_sims.append(cos)
cos_sims = np.array(cos_sims)
print(f"Cosine similarity (embedding) mean ± std:
{cos_sims.mean():.6f} ± {cos_sims.std():.6f}")
```

```
Cosine similarity (embedding) mean ± std: 0.644214 ± 0.123782
```

```
# ---- Diversity metrics ----
# 1) Entropy of grayscale histogram across reconstructions (average
over images)
def image_entropy(image, num_bins=256):
    # image in [0,1], compute grayscale histogram entropy (Shannon)
    gray = np.mean(image, axis=2)  # HxW
    hist, _ = np.histogram((gray * 255).astype(np.uint8).ravel(),
bins=num_bins, range=(0,255), density=True)
    # avoid zeros
    hist = hist + 1e-12
    ent = -np.sum(hist * np.log(hist))
    return ent

entropies = [image_entropy(im) for im in recons]
print(f"Average entropy of reconstructions (Grayscale histogram):
{np.mean(entropies):.4f} ± {np.std(entropies):.4f}")

# 2) Dist-N: average pairwise Euclidean distance between embeddings
(Inception activations)
# We'll compute average pairwise distance for a subset (to limit
computation)
def avg_pairwise_distance(acts, sample_limit=500):
    n = acts.shape[0]
    idxs = np.arange(n)
    if n > sample_limit:
        rng = np.random.default_rng(RANDOM_SEED)
        idxs = rng.choice(n, size=sample_limit, replace=False)
    acts_s = acts[idxs]
    # pairwise distances
    # compute squared norms trick
    sq = np.sum(acts_s**2, axis=1, keepdims=True)
    dists = np.sqrt(np.maximum(sq + sq.T - 2 * acts_s.dot(acts_s.T),
0.0))
    # take upper triangle mean
    triu_idx = np.triu_indices_from(dists, k=1)
    avg = np.mean(dists[triu_idx])
    return avg
```

```
avg_dist_recon = avg_pairwise_distance(recon_acts, sample_limit=200)
avg_dist_orig = avg_pairwise_distance(orig_acts, sample_limit=200)
print(f"Avg pairwise embedding distance (orig): {avg_dist_orig:.4f}")
print(f"Avg pairwise embedding distance (recon):
{avg_dist_recon:.4f}")
```

```
Average entropy of reconstructions (Grayscale histogram): 3.7397 ±
1.0219
Avg pairwise embedding distance (orig): 18.7593
Avg pairwise embedding distance (recon): 18.6714
```

```
# ---- Summary printout ----
print("\n--- Summary Metrics ---")
print(f"Samples used: {originals.shape[0]}")
print(f"MSE (mean): {mse_vals.mean():.6f}")
print(f"MAE (mean): {mae_vals.mean():.6f}")
print(f"SSIM (mean): {ssim_vals.mean():.4f}")
print(f"FID: {fid_value:.4f}")
print(f"Cosine similarity (mean): {cos_sims.mean():.6f}")
print(f"Reconstruction entropy (mean): {np.mean(entropies):.4f}")
print(f"Avg pairwise embedding distance - originals:
{avg_dist_orig:.4f}, reconstructions: {avg_dist_recon:.4f}")
```

```
--- Summary Metrics ---
Samples used: 500
MSE (mean): 0.007712
MAE (mean): 0.045278
SSIM (mean): 0.6955
FID: 184.7342
Cosine similarity (mean): 0.644214
Reconstruction entropy (mean): 3.7397
Avg pairwise embedding distance - originals: 18.7593, reconstructions:
18.6714
```