Experiment No.2 Autoencoders and Variational Autoencoders (VAE)

1. Build a simple AE model for Dimensionality Reduction and Denoising.
2. Generate realistic faces or interpolate between facial features for creative applications in entertainment and design using VAE.

Name:- Nabil Ansari

Batch :- GAA Lab 3

PRN/Roll no. :- 202302040004

1. Build a simple AE model for Dimensionality Reduction and Denoising.

---

1 What an Autoencoder Is

An **Autoencoder** is a neural network trained to copy its input to its output through a compressed intermediate representation.

It has two main parts:

Part	
Encoder	Compresses the input into a smaller**latent vector**(dimensionality reduction)
Decoder	Reconstructs the original input from the latent vector

---

Function

2 Dimensionality Reduction

- The encoder transforms an input image x into a latent vector z of much lower dimension than the original.
- This forces the model to learn an**efficient representation**of the input.
- You can then use this latent vector z for:
  - Feature extraction
  - Visualization (with PCA/t-SNE)
  - Feeding into other models

Example: An MNIST image of shape$28 \times 28 \times 1 = 784$ pixels → encoded into a 64-dimensional vector.

---

3 Denoising

During training you feed the **noisy version** of the input as x_{noisy} but ask the model to output the

- **clean version** x_{clean}.
- This encourages the latent representation to capture only the **signal**
- and ignore the noise. At inference time you can pass in noisy images

---

and the AE outputs denoised versions.

Setup

Install necessary libraries and import dependencies.

```
1 !pip install tensorflow tensorflow-datasets
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-
packages (2.19.0)
  Requirement already satisfied: tensorflow-datasets in
  /usr/local/lib/python3.12/dist-packages (4.9.9)
```

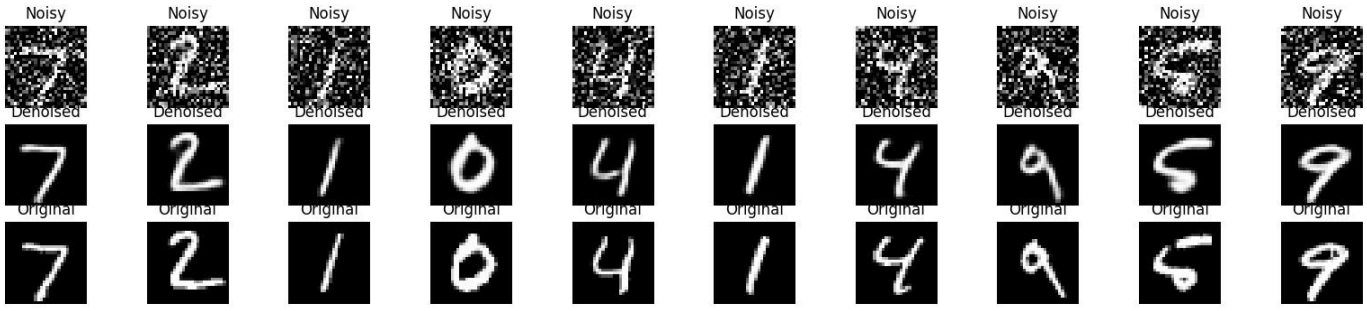
```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt

# Load MNIST
(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.expand_dims(x_train, -1)  # (batch, 28, 28, 1)
x_test  = np.expand_dims(x_test, -1)

# Add random noise for denoising AE
noise_factor = 0.5
x_train_noisy = np.clip(x_train + noise_factor * np.random.normal(size=x_train.shape),
0., 1.)
x_test_noisy  = np.clip(x_test  + noise_factor * np.random.normal(size=x_test.shape),
0., 1.)

# Build simple convolutional AE
latent_dim = 64

encoder_input = layers.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, (3,3), activation='relu', padding='same')(encoder_input)
x = layers.MaxPooling2D((2,2), padding='same')(x)
x = layers.Conv2D(64, (3,3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2,2), padding='same')(x)
x = layers.Flatten()(x)
latent = layers.Dense(latent_dim, name="latent")(x)

# Decoder
x = layers.Dense(7*7*64, activation='relu')(latent)
x = layers.Reshape((7,7,64))(x)
x = layers.Conv2DTranspose(64, (3,3), strides=2, activation='relu', padding='same')(x)
x = layers.Conv2DTranspose(32, (3,3), strides=2, activation='relu', padding='same')(x)
decoder_output = layers.Conv2DTranspose(1, (3,3), activation='sigmoid',
padding='same')(x)

autoencoder = models.Model(encoder_input, decoder_output)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.summary()

# Train
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=128,
```

```
                shuffle=True,
                validation_data=(x_test_noisy, x_test))

# Get latent representations (dimensionality reduction)
encoder = models.Model(encoder_input, latent)
latent_repr = encoder.predict(x_test_noisy)

print("Latent shape:", latent_repr.shape)  # (num_samples, latent_dim)

# Visualize denoising
decoded_imgs = autoencoder.predict(x_test_noisy)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Noisy input
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28,28), cmap='gray')
    plt.title("Noisy")
    plt.axis('off')

    # Denoised output
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28,28), cmap='gray')
    plt.title("Denoised")
    plt.axis('off')

    # Original
    ax = plt.subplot(3, n, i + 1 + 2*n)
    plt.imshow(x_test[i].reshape(28,28), cmap='gray')
    plt.title("Original")
    plt.axis('off')
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓ ┃ Layer
(type)                         ┃ Output Shape
┃        Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩ │
input_layer_4 (InputLayer)      │ (None, 28,
28, 1)       │             0 │
├─────────────────────────────────┼
────────────────────────┼───────────────┤ │ conv2d_10
(Conv2D)                   │ (None, 28, 28, 32)
│          320 │
├─────────────────────────────────┼
```

```
──────────────────┼──────────────────┼──────────┤
max_pooling2d (MaxPooling2D)     │ (None, 14, 14, 32)     │            0 │
├──────────────────────────────────┼──────────────────┼──────────┤
conv2d_11 (Conv2D)               │ (None, 14, 14, 64)     │       18,496 │
├──────────────────────────────────┼──────────────────┼──────────┤
max_pooling2d_1 (MaxPooling2D)   │ (None, 7, 7, 64)       │            0 │
├──────────────────────────────────┼──────────────────┼──────────┤
flatten_2 (Flatten)              │ (None, 3136)           │            0 │
├──────────────────────────────────┼──────────────────┼──────────┤
latent (Dense)                   │ (None, 64)             │      200,768 │
├──────────────────────────────────┼──────────────────┼──────────┤
dense_3 (Dense)                  │ (None, 3136)           │      203,840 │
├──────────────────────────────────┼──────────────────┼──────────┤
reshape_2 (Reshape)              │ (None, 7, 7, 64)       │            0 │
├──────────────────────────────────┼──────────────────┼──────────┤
conv2d_transpose_8               │ (None, 14, 14, 64)     │       36,928 │
(Conv2DTranspose)                │                  │              │
├──────────────────────────────────┼──────────────────┼──────────┤
conv2d_transpose_9               │ (None, 28, 28, 32)     │       18,464 │
(Conv2DTranspose)                │                  │              │
├──────────────────────────────────┼──────────────────┼──────────┤
conv2d_transpose_10              │ (None, 28, 28, 1)      │          289 │
(Conv2DTranspose)                │                  │              │
└──────────────────────────────────┴──────────────────┴──────────┘
 Total params: 479,105 (1.83 MB)
 Trainable params: 479,105 (1.83 MB)
 Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step -
loss: 0.2871 - val_loss: 0.1242 Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step -
loss: 0.1170 - val_loss: 0.1044 Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step -
loss: 0.1035 - val_loss: 0.1000 Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step -
loss: 0.0991 - val_loss: 0.0973
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step -
loss: 0.0966 - val_loss: 0.0968 Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step -
loss: 0.0950 - val_loss: 0.0952 Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step -
loss: 0.0936 - val_loss: 0.0947
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step -
loss: 0.0925 - val_loss: 0.0939 Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step -
loss: 0.0918 - val_loss: 0.0937 Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step -
loss: 0.0908 - val_loss: 0.0935
WARNING:tensorflow:5 out of the last 7 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distribute
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
Latent shape: (10000, 64)
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```



2. Generate realistic faces or interpolate between facial features for creative

applications in entertainment and design using VAE.

This demonstrates how to build and train convolutional autoencoders (AE) and variational autoencoders (VAE) using TensorFlow and Keras for image denoising and face generation.

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

```
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_datasets as tfds

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

```
TensorFlow version: 2.19.0
```

Configuration

Define parameters for image size, batch size, latent dimension, and training epochs.

```
IMAGE_SIZE = 64        # resize images to 64x64 (balanced speed-quality). Use 128 if
you have time & RAM.
BATCH_SIZE = 128
LATENT_DIM = 128       # latent space size
EPOCHS_AE = 25         # AE epochs
EPOCHS_VAE = 40        # VAE epochs
AUTOTUNE = tf.data.AUTOTUNE
```

Load and Prepare Dataset

Load the Labeled Faces in the Wild (LFW) dataset using `tensorflow_datasets`. Preprocess the images by resizing and normalizing them.

```
def preprocess_example(example):
    # example is a dict from tfds for celeb_a that has 'image' key
    img = example['image']
    img = tf.image.resize(img, [IMAGE_SIZE, IMAGE_SIZE])
    img = tf.cast(img, tf.float32) / 127.5 - 1.0  # normalize to [-1, +1]
    return img

print("Loading dataset (this may download ~1-2GB). If you want smaller dataset, change
`with_info` and/or dataset name.")

# If CelebA isn't desirable, you can switch to 'lfw_people' by changing dataset_name
below.
try:
    dataset_name = 'lfw'
    ds, ds_info = tfds.load(dataset_name, split='train', shuffle_files=True,
with_info=True, as_supervised=False)
    total_examples = ds_info.splits['train'].num_examples
    print(f"Loaded {dataset_name} with {total_examples} images")
except Exception as e:
    print("Couldn't load CelebA via tfds. Falling back to lfw_people (smaller).
Error:", e)
    dataset_name = 'lfw_people'
    ds, ds_info = tfds.load(dataset_name, split='train', shuffle_files=True,
with_info=True, as_supervised=False)
    total_examples = ds_info.splits['train'].num_examples
```

```
    print(f"Loaded {dataset_name} with {total_examples} images")

# Map and batch
ds = ds.map(preprocess_example, num_parallel_calls=AUTOTUNE)
# cache for speed (if memory available)
# ds = ds.cache()
# shuffle and batch
ds = ds.shuffle(2048).batch(BATCH_SIZE).prefetch(AUTOTUNE)

# For quick prototyping we also prepare a smaller test sample
sample_iter = ds.take(1)
for batch in sample_iter:
    sample_images = batch[0:16] if isinstance(batch, tf.Tensor) else batch[:16]
    break
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/lfw/0.1.1
has no dataset_info.json Loading dataset (this may download ~1-
2GB). If you want smaller dataset, change `with_info` and/or
dataset name.
Downloading and preparing dataset Unknown size (download: Unknown size,
generated: Unknown size, total: Unknown size) to /root/tensorflo
Dl Completed...: 100% 1/1 [00:40<00:00, 12.57s/ url]
Dl Size...: 100% 172/172 [00:40<00:00, 17.49 MiB/s]
Extraction completed...: 100% 13233/13233 [00:39<00:00, 969.34 file/s]
Dataset lfw downloaded and prepared to /root/tensorflow_datasets/lfw/0.1.1.
Subsequent calls will reuse this data.
Loaded lfw with 13233 images
```

Helper to show images

```
def show_images(imgs, ncols=8, title=None):
    imgs = (imgs + 1.0) * 127.5  # [-1,1] -> [0,255]
    # Check if imgs is a TensorFlow tensor before calling .numpy()
    if isinstance(imgs, tf.Tensor):
        imgs = imgs.numpy().astype(np.uint8)
    else:
        imgs = imgs.astype(np.uint8) # Assume it's already a NumPy array
    n = imgs.shape[0]
    nrows = math.ceil(n / ncols)
    plt.figure(figsize=(ncols * 1.6, nrows * 1.6))
    for i in range(n):
        plt.subplot(nrows, ncols, i + 1)
        plt.imshow(imgs[i])
        plt.axis('off')
    if title:
        plt.suptitle(title)
```

```
    plt.show()

# show a few real images
for batch in ds.take(1):
    real_sample = batch[:16]
    break

print('Real sample:')
show_images(real_sample)
```

26

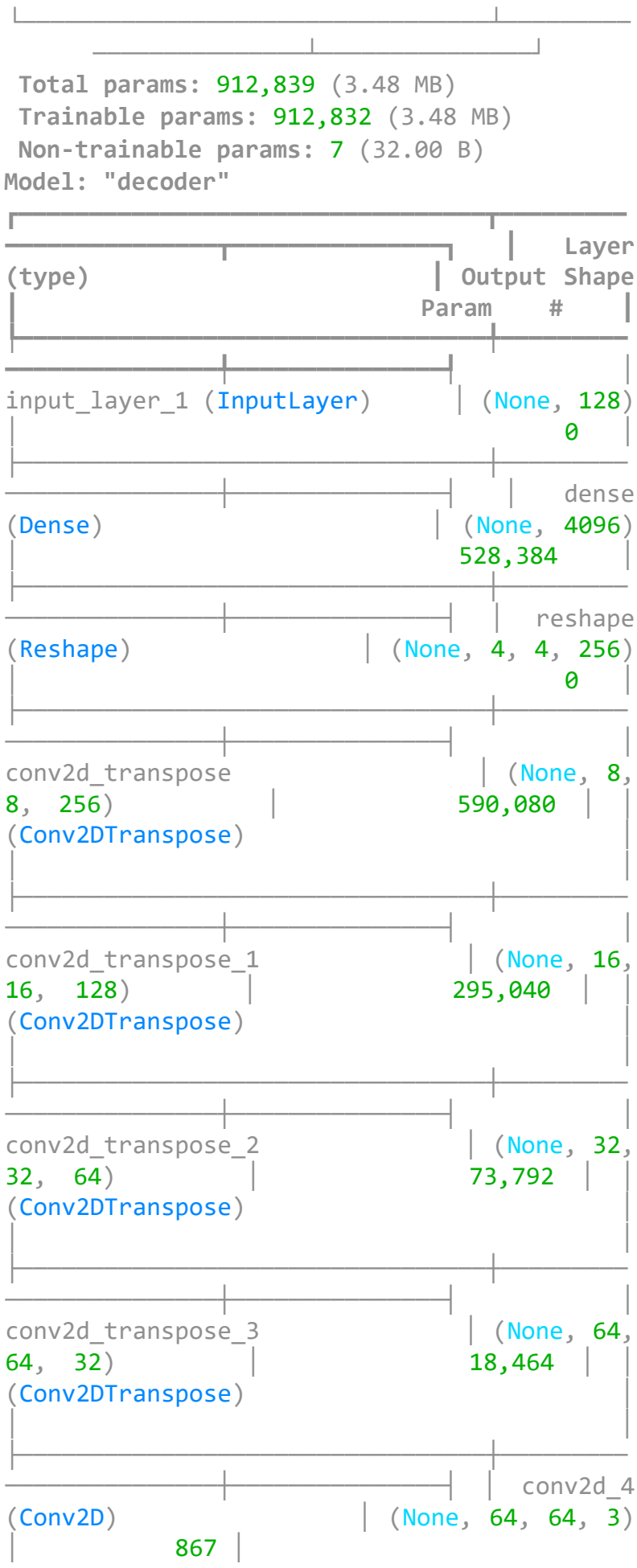
Real sample:

Convolutional Autoencoder (denoising + dimensionality reduction)

```
# Encoder
encoder_inputs = layers.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
# Add Gaussian noise for denoising ability at train-time via a separate input or using
Noise layer inside model
x = encoder_inputs
x = layers.Normalization()(x)  # optional - we already normalized manually, keep
identity if undesired

x = layers.Conv2D(32, 3, strides=2, padding='same', activation='relu')(x)  # 32x32 ->
32
x = layers.Conv2D(64, 3, strides=2, padding='same', activation='relu')(x)  # 16x16
x = layers.Conv2D(128, 3, strides=2, padding='same', activation='relu')(x)  # 8x8
x = layers.Conv2D(256, 3, strides=2, padding='same', activation='relu')(x)  # 4x4
shape_before_flatten = tf.keras.backend.int_shape(x)[1:]
x = layers.Flatten()(x)
latent = layers.Dense(LATENT_DIM, name='latent_vector')(x)
encoder = keras.Model(encoder_inputs, latent, name='encoder')
encoder.summary()

# Decoder
latent_inputs = layers.Input(shape=(LATENT_DIM,))
x = layers.Dense(np.prod(shape_before_flatten), activation='relu')(latent_inputs)
x = layers.Reshape(shape_before_flatten)(x)
x = layers.Conv2DTranspose(256, 3, strides=2, padding='same', activation='relu')(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)
# final layer, tanh to match normalized [-1,1]
```

```
decoder_outputs = layers.Conv2D(3, 3, activation='tanh', padding='same')(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name='decoder')
decoder.summary()

# Autoencoder = encoder + decoder
ae_inputs = encoder_inputs
ae_latent = encoder(ae_inputs)
ae_outputs = decoder(ae_latent)
ae = keras.Model(ae_inputs, ae_outputs, name='autoencoder')

# Loss and compile
ae.compile(optimizer=keras.optimizers.Adam(1e-4), loss='mse')
```

```
Model: "encoder"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)             │ (None, 64, 64, 3)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ normalization (Normalization)        │ (None, 64, 64, 3)           │               7 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d (Conv2D)                      │ (None, 32, 32, 32)          │             896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 16, 16, 64)          │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 8, 8, 128)           │          73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_3 (Conv2D)                    │ (None, 4, 4, 256)           │         295,168 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 4096)                │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ latent_vector (Dense)                │ (None, 128)                 │         524,416 │
```


Total params: 912,839 (3.48 MB)
Trainable params: 912,832 (3.48 MB)
Non-trainable params: 7 (32.00 B)
Model: "decoder"
Layer (type) Output Shape Param #
input_layer_1 (InputLayer) (None, 128) 0
dense (Dense) (None, 4096) 528,384
reshape (Reshape) (None, 4, 4, 256) 0
conv2d_transpose (Conv2DTranspose) (None, 8, 8, 256) 590,080
conv2d_transpose_1 (Conv2DTranspose) (None, 16, 16, 128) 295,040
conv2d_transpose_2 (Conv2DTranspose) (None, 32, 32, 64) 73,792
conv2d_transpose_3 (Conv2DTranspose) (None, 64, 64, 64) 18,464
conv2d_4 (Conv2D) (None, 64, 64, 3) 867

└──────────────────────────────────────┘└──────────
──────────┘└──────────┘

Prepare noisy dataset for denoising

```
NOISE_FACTOR = 0.2

def add_noise(images):
    noise = tf.random.normal(shape=tf.shape(images), mean=0.0, stddev=NOISE_FACTOR)
    noisy = images + noise
    noisy = tf.clip_by_value(noisy, -1.0, 1.0)
    return noisy

# Create dataset pairs: (noisy_image, clean_image)
paired_ds = ds.map(lambda x: (add_noise(x), x), num_parallel_calls=AUTOTUNE)
paired_ds = paired_ds.prefetch(AUTOTUNE)
```

Train Autoencoder (Denoising) & Visualize AE denoising results

```


checkpoint_dir = '/content/checkpoints_ae'
os.makedirs(checkpoint_dir, exist_ok=True)

callbacks = [
    keras.callbacks.ModelCheckpoint(os.path.join(checkpoint_dir, 'ae_best.h5'),
save_best_only=True, monitor='loss'),
    keras.callbacks.ReduceLROnPlateau(monitor='loss', factor=0.5, patience=3,
verbose=1)
]

print('Training AE (denoising) ...')
# For faster runs on limited resources, set steps_per_epoch to a smaller number (e.g.,
100)
steps_per_epoch = None

ae.fit(paired_ds, epochs=EPOCHS_AE, callbacks=callbacks)

# Load best
try:
    ae.load_weights(os.path.join(checkpoint_dir, 'ae_best.h5'))
    print('Loaded best AE weights')
except Exception as e:
    print('Could not load weights, continuing with current model. Error:', e)


for batch in ds.take(1):
    clean = batch[:8]
    noisy = add_noise(clean)
    denoised = ae.predict(noisy)
```

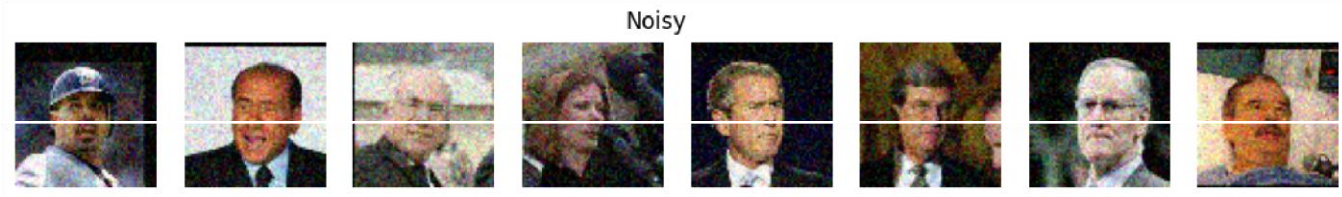
```
    print('Noisy:')
    show_images(noisy[:8], ncols=8, title='Noisy')
    print('Denoised by AE:')
    show_images(denoised[:8], ncols=8, title='Denoised')
    print('Original:')
    show_images(clean[:8], ncols=8, title='Original')
    break
```

```
36
```

```
Training AE (denoising) ...
Epoch 1/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.0567WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 75ms/step - loss: 0.0567 - learning_rate:
1.0000e-04
Epoch 2/25
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.0560WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 74ms/step - loss: 0.0560 - learning_rate:
1.0000e-04
Epoch 3/25
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0551WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 82ms/step - loss: 0.0551 - learning_rate:
1.0000e-04
Epoch 4/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - loss: 0.0548WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 84ms/step - loss: 0.0548 - learning_rate:
1.0000e-04
Epoch 5/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0548WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 8s 73ms/step - loss: 0.0547 - learning_rate:
1.0000e-04
Epoch 6/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.0538WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 74ms/step - loss: 0.0538 - learning_rate:
1.0000e-04
Epoch 7/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - loss: 0.0533WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 83ms/step - loss: 0.0533 - learning_rate:
1.0000e-04
Epoch 8/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - loss: 0.0525WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
```

```
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 81ms/step - loss: 0.0525 - learning_rate:
1.0000e-04
Epoch 9/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0522WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 72ms/step - loss: 0.0522 - learning_rate:
1.0000e-04
Epoch 10/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.0519WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 74ms/step - loss: 0.0519 - learning_rate:
1.0000e-04
Epoch 11/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0514WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 82ms/step - loss: 0.0514 - learning_rate:
1.0000e-04
Epoch 12/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0510WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 82ms/step - loss: 0.0510 - learning_rate:
1.0000e-04
Epoch 13/25
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - loss: 0.0509WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 76ms/step - loss: 0.0509 - learning_rate:
1.0000e-04
Epoch 14/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - loss: 0.0503WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 71ms/step - loss: 0.0503 - learning_rate:
1.0000e-04
Epoch 15/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0501WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 71ms/step - loss: 0.0501 - learning_rate:
1.0000e-04
Epoch 16/25
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - loss: 0.0495WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 11s 79ms/step - loss: 0.0495 - learning_rate:
1.0000e-04
Epoch 17/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0491WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 82ms/step - loss: 0.0491 - learning_rate:
1.0000e-04
Epoch 18/25
```

```
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0491WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 82ms/step - loss: 0.0491 - learning_rate:
1.0000e-04
Epoch 19/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.0485WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 8s 72ms/step - loss: 0.0485 - learning_rate:
1.0000e-04
Epoch 20/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step - loss: 0.0482WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 80ms/step - loss: 0.0482 - learning_rate:
1.0000e-04
Epoch 21/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0480WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 82ms/step - loss: 0.0480 - learning_rate:
1.0000e-04
Epoch 22/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0478WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 81ms/step - loss: 0.0478 - learning_rate:
1.0000e-04
Epoch 23/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0473WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 71ms/step - loss: 0.0473 - learning_rate:
1.0000e-04
Epoch 24/25
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 0.0472WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 10s 71ms/step - loss: 0.0472 - learning_rate:
1.0000e-04
Epoch 25/25
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - loss: 0.0470WARNING:absl:You are
saving your model as an HDF5 file via `model.save()` or `ke
104/104 ━━━━━━━━━━━━━━━━━━━━ 9s 82ms/step - loss: 0.0470 - learning_rate:
1.0000e-04
Loaded best AE weights
1/1 ━━━━━━━━━━━━━━━━━━━━
0s 42ms/step Noisy:
```



Denoised by AE:

Original:

Build a Convolutional Variational Autoencoder (VAE) for face generation & Train the VAE

```
   1 ----------------------


class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.random.normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon

# Encoder for VAE
vae_encoder_inputs = layers.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))

x = layers.Conv2D(32, 3, strides=2, padding='same',
activation='relu')(vae_encoder_inputs)  # 32x32
x = layers.Conv2D(64, 3, strides=2, padding='same', activation='relu')(x)  # 16x16
x = layers.Conv2D(128, 3, strides=2, padding='same', activation='relu')(x)  # 8x8
x = layers.Conv2D(256, 3, strides=2, padding='same', activation='relu')(x)  # 4x4
x = layers.Flatten()(x)
vae_x = layers.Dense(512, activation='relu')(x)
z_mean = layers.Dense(LATENT_DIM, name='z_mean')(vae_x)
z_log_var = layers.Dense(LATENT_DIM, name='z_log_var')(vae_x)
z = Sampling()([z_mean, z_log_var])
vae_encoder = keras.Model(vae_encoder_inputs, [z_mean, z_log_var, z],
name='vae_encoder')
vae_encoder.summary()

# Decoder for VAE (re-using architecture pattern from AE)
latent_inputs = layers.Input(shape=(LATENT_DIM,))
x = layers.Dense(np.prod(shape_before_flatten), activation='relu')(latent_inputs)
x = layers.Reshape(shape_before_flatten)(x)
x = layers.Conv2DTranspose(256, 3, strides=2, padding='same', activation='relu')(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)
```

```
vae_outputs = layers.Conv2D(3, 3, activation='tanh', padding='same')(x)
vae_decoder = keras.Model(latent_inputs, vae_outputs, name='vae_decoder')
vae_decoder.summary()

# VAE model with custom train_step
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_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.reconstruction_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)
            reconstruction = self.decoder(z)
            reconstruction_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 = reconstruction_loss + kl_loss * 1.0
        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.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
        }

vae = VAE(vae_encoder, vae_decoder)
vae.compile(optimizer=keras.optimizers.Adam(1e-4))


checkpoint_dir_vae = '/content/checkpoints_vae'
os.makedirs(checkpoint_dir_vae, exist_ok=True)

callbacks_vae = [
    keras.callbacks.ModelCheckpoint(os.path.join(checkpoint_dir_vae, 'vae_best.h5'),
save_best_only=True, monitor='loss'),
```

```
    keras.callbacks.ReduceLROnPlateau(monitor='loss', factor=0.5, patience=4,
verbose=1)
]

print('Training VAE ...')
vae.fit(ds, epochs=EPOCHS_VAE, callbacks=callbacks_vae)

try:
    vae.load_weights(os.path.join(checkpoint_dir_vae, 'vae_best.h5'))
    print('Loaded best VAE weights')
except Exception as e:
    print('Could not load VAE weights, continuing. Error:', e)
```

```
Model: "vae_encoder"
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer_2       │ (None, 64, 64, 3) │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_5 (Conv2D)   │ (None, 32, 32,    │        896 │ input_layer_2[0]… │
│                     │ 32)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_6 (Conv2D)   │ (None, 16, 16,    │     18,496 │ conv2d_5[0][0]    │
│                     │ 64)               │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_7 (Conv2D)   │ (None, 8, 8, 128) │     73,856 │ conv2d_6[0][0]    │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ conv2d_8 (Conv2D)   │ (None, 4, 4, 256) │    295,168 │ conv2d_7[0][0]    │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ flatten_1           │ (None, 4096)      │          0 │ conv2d_8[0][0]    │
│ (Flatten)           │                   │            │                   │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘
```

```
──────────────┼────────────────────────┤ │ dense_1
(Dense)       │ (None, 512)           │  2,097,664
│ flatten_1[0][0]    │
├──────────────────────┼────────────────────┼─
──────────────┼────────────────────────┤ │ z_mean
(Dense)       │ (None, 128)           │
65,664 │ dense_1[0][0]       │
├──────────────────────┼────────────────────┼─
──────────────┼────────────────────────┤ │ z_log_var
(Dense)    │ (None, 128)          │     65,664 │
dense_1[0][0]      │
├──────────────────────┼────────────────────┼─
──────────────┼────────────────────────┤ │ sampling
(Sampling) │ (None, 128)          │          0
│ z_mean[0][0],      │ │
│                    │            │
z_log_var[0][0]    │
└──────────────────────┴────────────────────┴─
──────────────┴────────────────────────┘
 Total params: 2,617,408
(9.98 MB)
 Trainable params: 2,617,408
(9.98 MB)
 Non-trainable params: 0
(0.00 B)
Model: "vae_decoder"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┓ ┃ Layer
(type)                          ┃ Output Shape
┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━┩ │
input_layer_3 (InputLayer)        │ (None,
128)            │            0 │
├───────────────────────────────────┼──────────
───────────────┼────────────────┤ │ dense_2
(Dense)                         │ (None, 4096)
│       528,384 │
├───────────────────────────────────┼──────────
───────────────┼────────────────┤ │ reshape_1
(Reshape)                       │ (None, 4, 4, 256)
│             0 │
├───────────────────────────────────┼──────────
───────────────┼────────────────┤ │
conv2d_transpose_4                │ (None, 8,
8, 256)      │        590,080 │ │
(Conv2DTranspose)                 │
│                │
├───────────────────────────────────┼──────────
───────────────┼────────────────┤ │
```

```
conv2d_transpose_5               | (None, 16,
16, 128)    |       295,040 | |
(Conv2DTranspose)                |
|                |
├──────────────────────────────────┼──────────
──────────────┼──────────────────┤ |
conv2d_transpose_6               | (None, 32,
32, 64)     |        73,792 | |
(Conv2DTranspose)                |
|                |
├──────────────────────────────────┼──────────
──────────────┼──────────────────┤ |
conv2d_transpose_7               | (None, 64,
64, 32)     |        18,464 | |
(Conv2DTranspose)                |
|                |
├──────────────────────────────────┼──────────
──────────────┼──────────────────┤ | conv2d_9
(Conv2D)                | (None, 64, 64, 3)
|           867 |
└──────────────────────────────────┴──────────
──────────────┴──────────────────┘
 Total params: 1,506,627
(5.75 MB)
 Trainable params: 1,506,627
(5.75 MB)  Non-trainable
params: 0 (0.00 B)
Training VAE ...
Epoch 1/40
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - kl_loss: 19.8784 - loss: 4098.8823
- reconstruction_loss: 4079.0034WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 17s 92ms/step - kl_loss: 20.1048 - loss: 4094.4055 -
reconstruction_loss: 4074.3005 - learning_rate: 1.0000
Epoch 2/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - kl_loss: 63.5873 - loss: 2671.8186
- reconstruction_loss: 2608.2312WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - kl_loss: 63.6239 - loss: 2669.0698 -
reconstruction_loss: 2605.4460 - learning_rate: 1.0000e
Epoch 3/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - kl_loss: 88.2948 - loss: 2081.1458
- reconstruction_loss: 1992.8510WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 67ms/step - kl_loss: 88.5771 - loss: 2077.9160 -
reconstruction_loss: 1989.3390 - learning_rate: 1.0000e
Epoch 4/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - kl_loss: 121.2925 - loss: 1595.5642
- reconstruction_loss: 1474.2719WARNING:absl:You are sav 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 56ms/step - kl_loss: 121.3507 - loss: 1594.8188 -
reconstruction_loss: 1473.4684 - learning_rate: 1.0000
Epoch 5/40
```

```
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - kl_loss: 129.0582 - loss: 1457.0836
- reconstruction_loss: 1328.0253WARNING:absl:You are sav 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 59ms/step - kl_loss: 129.0856 - loss: 1456.8007 -
reconstruction_loss: 1327.7150 - learning_rate: 1.0000
Epoch 6/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - kl_loss: 134.6435 - loss: 1380.6661
- reconstruction_loss: 1246.0227WARNING:absl:You are sav 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 64ms/step - kl_loss: 134.6626 - loss: 1380.4089 -
reconstruction_loss: 1245.7463 - learning_rate: 1.0000
Epoch 7/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - kl_loss: 139.1489 - loss: 1316.0416
- reconstruction_loss: 1176.8927WARNING:absl:You are sav 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - kl_loss: 139.1698 - loss: 1315.8423 -
reconstruction_loss: 1176.6725 - learning_rate: 1.0000
Epoch 8/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 144.4747 - loss: 1252.5205
- reconstruction_loss: 1108.0455WARNING:absl:You are sav 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 66ms/step - kl_loss: 144.4922 - loss: 1252.4009 -
reconstruction_loss: 1107.9084 - learning_rate: 1.0000
Epoch 9/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - kl_loss: 147.3841 - loss: 1206.9362
- reconstruction_loss: 1059.5522WARNING:absl:You are sav 104/104
━━━━━━━━━━━━━━━━━━━━ 9s 58ms/step - kl loss: 147 4032 - loss: 1206 7762 -
reconstruction loss: 1059 3732 - learning rate: 1 0000 104/104
━━━━━━━━━━━━━━━━━━━━ 9s 58ms/step - kl_loss: 147.4032 - loss: 1206.7762 -
reconstruction_loss: 1059.3732 - learning_rate: 1.0000
Epoch 10/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - kl_loss: 150.4894 - loss: 1162.9700
- reconstruction_loss: 1012.4805WARNING:absl:You are sav 104/104
━━━━━━━━━━━━━━━━━━━━ 10s 55ms/step - kl_loss: 150.5131 - loss: 1162.7937 -
reconstruction_loss: 1012.2806 - learning_rate: 1.000
Epoch 11/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - kl_loss: 152.8593 - loss: 1125.4773
- reconstruction_loss: 972.6182WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 64ms/step - kl_loss: 152.8784 - loss: 1125.4172 -
reconstruction_loss: 972.5391 - learning_rate: 1.0000e
Epoch 12/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 154.9528 - loss: 1099.9663
- reconstruction_loss: 945.0135WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 10s 66ms/step - kl_loss: 154.9662 - loss: 1099.8890 -
reconstruction_loss: 944.9229 - learning_rate: 1.0000
Epoch 13/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 156.5574 - loss: 1076.7629
- reconstruction_loss: 920.2054WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 57ms/step - kl_loss: 156.5628 - loss: 1076.6506 -
reconstruction_loss: 920.0878 - learning_rate: 1.0000e
Epoch 14/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 157.1162 - loss: 1059.2067
- reconstruction_loss: 902.0903WARNING:absl:You are savi 104/104
```

```
━━━━━━━━━━━━━━━━━━━━ 8s 66ms/step - kl_loss: 157.1295 - loss: 1059.1086 -
reconstruction_loss: 901.9791 - learning_rate: 1.0000e
Epoch 15/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 157.5963 - loss: 1041.4174
- reconstruction_loss: 883.8210WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 57ms/step - kl_loss: 157.6131 - loss: 1041.3361 -
reconstruction_loss: 883.7230 - learning_rate: 1.0000e
Epoch 16/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - kl_loss: 158.5779 - loss: 1023.4907
- reconstruction_loss: 864.9129WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 10s 55ms/step - kl_loss: 158.5849 - loss: 1023.4551 -
reconstruction_loss: 864.8703 - learning_rate: 1.0000
Epoch 17/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 158.8062 - loss: 1016.5831
- reconstruction_loss: 857.7769WARNING:absl:You are savi 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 67ms/step - kl_loss: 158.8239 - loss: 1016.4711 -
reconstruction_loss: 857.6472 - learning_rate: 1.0000e
Epoch 18/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - kl_loss: 160.0612 - loss: 997.6310
- reconstruction_loss: 837.5698WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 59ms/step - kl_loss: 160.0675 - loss: 997.5922 -
reconstruction_loss: 837.5245 - learning_rate: 1.0000e-
Epoch 19/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 160.5851 - loss: 985.6735
- reconstruction_loss: 825.0883WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 56ms/step - kl_loss: 160.5918 - loss: 985.6576 -
reconstruction_loss: 825.0657 - learning_rate: 1.0000e-
Epoch 20/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 160.7849 - loss: 977.3094
- reconstruction_loss: 816.5245WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 66ms/step - kl_loss: 160.7927 - loss: 977.2952 -
reconstruction_loss: 816.5025 - learning_rate: 1.0000e-
Epoch 21/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 161.5073 - loss: 968.7128
- reconstruction_loss: 807.2054WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 9s 57ms/step - kl_loss: 161.5113 - loss: 968.6285 -
reconstruction_loss: 807.1169 - learning_rate: 1.0000e-
Epoch 22/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - kl_loss: 161.4843 - loss: 951.8209
- reconstruction_loss: 790.3367WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 10s 57ms/step - kl_loss: 161.4926 - loss: 951.8999 -
reconstruction_loss: 790.4073 - learning_rate: 1.0000e
Epoch 23/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - kl_loss: 161.6411 - loss: 947.3717
- reconstruction_loss: 785.7305WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 65ms/step - kl_loss: 161.6570 - loss: 947.3860 -
reconstruction_loss: 785.7289 - learning_rate: 1.0000e-
Epoch 24/40
```

```
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - kl_loss: 162.2359 - loss: 945.4188
- reconstruction_loss: 783.1829WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 60ms/step - kl_loss: 162.2409 - loss: 945.3525 -
reconstruction_loss: 783.1117 - learning_rate: 1.0000e-
Epoch 25/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 163.5051 - loss: 933.4561
- reconstruction_loss: 769.9509WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 10s 57ms/step - kl_loss: 163.5008 - loss: 933.4390 -
reconstruction_loss: 769.9381 - learning_rate: 1.0000e
Epoch 26/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 162.1755 - loss: 926.2592
- reconstruction_loss: 764.0837WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 66ms/step - kl_loss: 162.1920 - loss: 926.2355 -
reconstruction_loss: 764.0436 - learning_rate: 1.0000e-
Epoch 27/40
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - kl_loss: 163.1324 - loss: 925.7191
- reconstruction_loss: 762.5865WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 62ms/step - kl_loss: 163.1343 - loss: 925.6879 -
reconstruction_loss: 762.5535 - learning_rate: 1.0000e-
Epoch 28/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - kl_loss: 162.9794 - loss: 912.4647
- reconstruction_loss: 749.4853WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 57ms/step - kl_loss: 162.9894 - loss: 912.4447 -
reconstruction_loss: 749.4553 - learning_rate: 1.0000e-
Epoch 29/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - kl_loss: 163.8819 - loss: 905.6840
- reconstruction_loss: 741.8021WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 11s 61ms/step - kl_loss: 163.8795 - loss: 905.6917 -
reconstruction_loss: 741.8122 - learning_rate: 1.0000e
Epoch 30/40
104/104 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - kl_loss: 162.9729 - loss: 905.7121
- reconstruction_loss: 742.7393WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 65ms/step - kl_loss: 162.9803 - loss: 905.6872 -
reconstruction_loss: 742.7070 - learning_rate: 1.0000e-
Epoch 31/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 163.5599 - loss: 897.8604
- reconstruction_loss: 734.3005WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 57ms/step - kl_loss: 163.5750 - loss: 897.8298 -
reconstruction_loss: 734.2549 - learning_rate: 1.0000e-
Epoch 32/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 164.0580 - loss: 892.0839
- reconstruction_loss: 728.0260WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 66ms/step - kl_loss: 164.0666 - loss: 892.0456 -
reconstruction_loss: 727.9791 - learning_rate: 1.0000e-
Epoch 33/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - kl_loss: 163.8987 - loss: 884.7950
- reconstruction_loss: 720.8962WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 10s 59ms/step - kl_loss: 163.9088 - loss: 884.8052 -
reconstruction_loss: 720.8962 - learning_rate: 1.0000e
```

```
Epoch 34/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 164.9460 - loss: 880.7004
- reconstruction_loss: 715.7545WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 7s 57ms/step - kl_loss: 164.9467 - loss: 880.6918 -
reconstruction_loss: 715.7451 - learning_rate: 1.0000e-
Epoch 35/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - kl_loss: 164.5918 - loss: 877.4930
- reconstruction_loss: 712.9013WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 8s 68ms/step - kl_loss: 164.6018 - loss: 877.4391 -
reconstruction_loss: 712.8373 - learning_rate: 1.0000e-
Epoch 36/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 164.4513 - loss: 871.6179
- reconstruction_loss: 707.1666WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 9s 57ms/step - kl_loss: 164.4637 - loss: 871.6053 -
reconstruction_loss: 707.1415 - learning_rate: 1.0000e-
Epoch 37/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - kl_loss: 164.6115 - loss: 868.3679
- reconstruction_loss: 703.7566WARNING:absl:You are savin 104/104
━━━━━━━━━━━━━━━━━━━━ 10s 56ms/step - kl_loss: 164.6213 - loss: 868.3298 -
reconstruction_loss: 703.7087 - learning_rate: 1.0000e
Eh 38/40
Epoch 38/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 165.3562 - loss: 862.2353 -
        reconstruction_loss: 696.8791WARNING:absl:You are savin 104/104
      ━━━━━━━━━━━━━━━━━━━━ 8s 66ms/step - kl_loss: 165.3538 - loss: 862.2403 -
             reconstruction_loss: 696.8865 - learning_rate: 1.0000e-
Epoch 39/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - kl_loss: 165.2515 - loss: 854.7203 -
        reconstruction_loss: 689.4685WARNING:absl:You are savin 104/104
      ━━━━━━━━━━━━━━━━━━━━ 10s 66ms/step - kl_loss: 165.2572 - loss: 854.7574 -
             reconstruction_loss: 689.5000 - learning_rate: 1.0000e
Epoch 40/40
103/104 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - kl loss: 165.6622 - loss: 853.1840
- reconstruction loss: 687.5219WARNING:absl:You are savin
```

Output : Generate faces from random latent vectors

```
def generate_random_faces(model_decoder, n=8):
    z = np.random.normal(size=(n, LATENT_DIM)).astype(np.float32)
    imgs = model_decoder.predict(z)
    return imgs

rand_faces = generate_random_faces(vae_decoder, n=8)
show_images(rand_faces, ncols=8, title='Random VAE samples')



# pick two images from dataset and get their latent encodings
for batch in ds.take(1):
```

```
    imgs = batch[:16]
    break

img_a = imgs[2:3]
img_b = imgs[7:8]

z_mean_a, z_log_var_a, z_a = vae_encoder.predict(img_a)
z_mean_b, z_log_var_b, z_b = vae_encoder.predict(img_b)

# interpolate between means for deterministic interpolation
def interpolate(z1, z2, steps=10):
    ratios = np.linspace(0, 1, steps)
    interps = [(1 - r) * z1 + r * z2 for r in ratios]
    return np.vstack(interps)

z_inter = interpolate(z_mean_a, z_mean_b, steps=12)
imgs_inter = vae_decoder.predict(z_inter)
show_images(imgs_inter, ncols=6, title='Latent interpolation (means)')

# ------------------------------
# Cell 13: Save model artifacts to /content (or Drive if mounted)
# ------------------------------

save_dir = '/content/models_ae_vae'
os.makedirs(save_dir, exist_ok=True)
ae.save(os.path.join(save_dir, 'autoencoder.h5'))
vae_encoder.save(os.path.join(save_dir, 'vae_encoder.h5'))
vae_decoder.save(os.path.join(save_dir, 'vae_decoder.h5'))
print('Saved models to', save_dir)
```

Latent interpolation (means)