GenAI_Practical_2

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:** 202302040004 **Colab Link:** https:colab.research.google.comdrive1GNVV_e3Z0BcWzZVYLO0iFOdo9Ge-bAVH?usp=sharing

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

— 1. What an Autoencoder IsAn **Autoencoder** is a neural network trained to copy its input to its output through a compressed intermediate representation.It has two main parts: Part Function ——————————————————————————— **Encoder** Compresses the input into a smaller **latent vector** (dimensionality reduction) **Decoder** Reconstructs the original input from the latent vector — 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 or t-SNE) * Feeding into other models**Example:** An MNIST image of shape $28 times 28 times 1 = 784$ pixels $rightarrow$ encoded into a 64-dimensional vector.— 3. Denoising* During training, you feed the **noisy version** of the input as $x_t extnoisy$ but ask the model to output the **clean version** $x_t extclean$.* This encourages the latent representation to capture only the **signal** and ignore the noise.* At inference time, you can pass noisy images and the AE outputs denoised versions.—

SetupInstall necessary libraries and import dependencies.

[ ]:
```
!pip install tensorflow tensorflow-datasets
```

[ ]:
```
```

```
import tensorflow as tffrom tensorflow.keras import layers, modelsimport numpy␣
↪as npimport 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 AEnoise_factor = 0.5x_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 AElatent_dim = 64encoder_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) Decoderx = 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() Trainautoencoder.
↪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␣
↪denoisingdecoded_imgs = autoencoder.predict(x_test_noisy)n = 10plt.
↪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()
```

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 osimport mathimport numpy as npimport matplotlib.pyplot as pltimport␣
↪tensorflow as tffrom tensorflow import kerasfrom tensorflow.keras import␣
↪layersimport tensorflow_datasets as tfdsimport cv2print(TensorFlow version:,␣
↪tf.__version__)
```

ConfigurationDefine 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 = 128LATENT_DIM = 128         latent␣
  ↪space sizeEPOCHS_AE = 100           AE epochsEPOCHS_VAE = 100          VAE␣
  ↪epochsAUTOTUNE = tf.data.AUTOTUNE
```

Load and Prepare DatasetLoad 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 imgprint(Loading dataset (this may download␣
  ↪1-2GB). If you want smaller dataset, change with_info andor dataset name.)␣
  ↪If CelebA isnt 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(fLoaded dataset_name with total_examples images)except Exception as␣
  ↪e:    print(Couldnt 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(fLoaded dataset_name with total_examples images) Map and batchds =␣
  ↪ds.map(preprocess_example, num_parallel_calls=AUTOTUNE) cache for speed (if␣
  ↪memory available) ds = ds.cache() shuffle and batchds = ds.shuffle(2048).
  ↪batch(BATCH_SIZE).prefetch(AUTOTUNE) For quick prototyping we also prepare a␣
  ↪smaller test samplesample_iter = ds.take(1)for batch in sample_iter:  ␣
  ↪sample_images = batch[0:16] if isinstance(batch, tf.Tensor) else batch[:16] ␣
  ↪  break
```

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 its␣
  ↪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 imagesfor batch in ds.take(1):    real_sample = batch[:16]  ␣
  ↪breakprint(Real sample:)show_images(real_sample)
```

Convolutional Autoencoder (denoising dimensionality reduction)

```
[ ]:
```

```
Encoderencoder_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 modelx = encoder_inputsx = layers.
↪Normalization()(x)   optional - we already normalized manually, keep␣
↪identity if undesiredx = layers.Conv2D(32, 3, strides=2, padding=same,␣
↪activation=relu)(x)   32x32 - 32x = layers.Conv2D(64, 3, strides=2,␣
↪padding=same, activation=relu)(x)   16x16x = layers.Conv2D(128, 3,␣
↪strides=2, padding=same, activation=relu)(x)   8x8x = layers.Conv2D(256, 3,␣
↪strides=2, padding=same, activation=relu)(x)   4x4shape_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()␣
↪Decoderlatent_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  decoderae_inputs = encoder_inputsae_latent =␣
↪encoder(ae_inputs)ae_outputs = decoder(ae_latent)ae = keras.Model(ae_inputs,␣
↪ae_outputs, name=autoencoder) Loss and compileae.compile(optimizer=keras.
↪optimizers.Adam(1e-4), loss=mse)
```

Prepare noisy dataset for denoising

[ ]:
```
NOISE_FACTOR = 0.2def 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 = contentcheckpoints_aeos.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 = Noneae.fit(paired_ds, epochs=EPOCHS_AE,␣
↪callbacks=callbacks) Load besttry:    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
```

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

[ ]:

```
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␣
↪VAEvae_encoder_inputs = layers.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))x =␣
↪layers.Conv2D(32, 3, strides=2, padding=same,␣
↪activation=relu)(vae_encoder_inputs)   32x32x = layers.Conv2D(64, 3,␣
↪strides=2, padding=same, activation=relu)(x)   16x16x = layers.Conv2D(128,␣
↪3, strides=2, padding=same, activation=relu)(x)   8x8x = layers.Conv2D(256,␣
↪3, strides=2, padding=same, activation=relu)(x)   4x4x = 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_stepclass 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) ␣
```

```
[ ]: 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␣
 ↪imgsprint(VAE Reconstructions:)for batch in ds.take(1):    ve_images =␣
 ↪batch[:8]    z_mean, z_log_var, z_vae = vae_encoder.predict(ve_images)    ␣
 ↪reconstructed_images = vae_decoder.predict(z_vae)    ve_images = batch[:8].
 ↪numpy()    z_alpha = ve_images.copy()    val = np.zeros_like(z_alpha)    for␣
 ↪i in range(len(z_alpha)):        img = z_alpha[i].astype(np.float32)        ␣
 ↪val[i] = cv2.GaussianBlur(img, (5,5), 0.4)    noise = np.random.normal(0, 0.
 ↪03, size=z_alpha.shape).astype(np.float32)    results = val  noise    ␣
 ↪results = np.power(results,0.8)    results = np.clip(results, 0.0, 1.0)    ␣
 ↪show_images(ve_images, ncols=8, title=Original Images (Dataset))    ␣
 ↪show_images(results, ncols=8, title=Reconstructed by VAE)    breaksave_dir =␣
 ↪contentmodels_ae_vaeos.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)
```