Gen AI Practical No. 2

Name : **Priyanshu Wagh**

PRN No. **202201040135**

Div. : B

Batch : GAA 3

Objectives

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

Theory

- **Autoencoder (AE):** A neural network that compresses data into a lower-dimensional latent space (Encoder) and reconstructs it back (Decoder).
- Used for **dimensionality reduction** (like PCA but nonlinear) and **denoising** (removing unwanted noise from data).

Procedure

1. Load dataset (e.g., MNIST or CIFAR).
2. Build encoder-decoder network using dense or convolution layers.
3. Train the AE on clean data for dimensionality reduction.
4. Train with noisy data for denoising task.

5. Compare original, noisy, and reconstructed outputs.

Implementation

```
[ ] !pip install kagglehub --quiet
```

```python
# ----------------------------
# Autoencoder for Dimensionality Reduction & Denoising
# ----------------------------
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt

# Load MNIST dataset
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()

# Normalize and reshape
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
x_train = np.expand_dims(x_train, -1)  # shape: (60000, 28, 28, 1)
x_test = np.expand_dims(x_test, -1)

# ----------------------------
# Build Autoencoder Model
# ----------------------------
latent_dim = 32  # compressed representation

# Encoder
encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Flatten()(encoder_inputs)
x = layers.Dense(128, activation="relu")(x)
latent = layers.Dense(latent_dim, activation="relu")(x)

# Decoder
x = layers.Dense(128, activation="relu")(latent)
x = layers.Dense(28*28, activation="sigmoid")(x)
decoder_outputs = layers.Reshape((28, 28, 1))(x)

autoencoder = keras.Model(encoder_inputs, decoder_outputs, name="autoencoder")
```

```
autoencoder.coirpiie(optimizer-"adan", Toss-"bznary crossentropy")

# Tr'azn AuI 0e nC0der'

nistony = autoeccodeD.fit(
    x_train, x_train,
    epochs=10,
    batch_size=256,

encoder = keras.fiDdeL(encodes inputs, Patent, nane="encoder")
encoded ings = encoder.Predict(x test :10a

print("Latent representations (compressed vectors):")
print(encoded_imgs)

# Visualization - Reconstruction

decoded ings = autoencoder.predict(x test[:Ie])

olt.figure(fgie-28, 4)

    \ Original
    ax = p1t.subpLot(2, 10, i + 1)
    Det.insno (x test[i].reshaDe{28, 28,', cPaD-"gray")
    Det.axis          )

    4 hecons tru cted
    ax - p1t.subplot(2, 16, i +      18)
    Det.insnu(cecoed ings i .resnpe(B, 28;i, cna>"ga'")
    oTt.axis("off")
plt.suptitle("Original vs Reconstructed Images", fontsize=16)
```



```
lt.suptitle("Original vs Reconstructed Images", fontsize=16
```



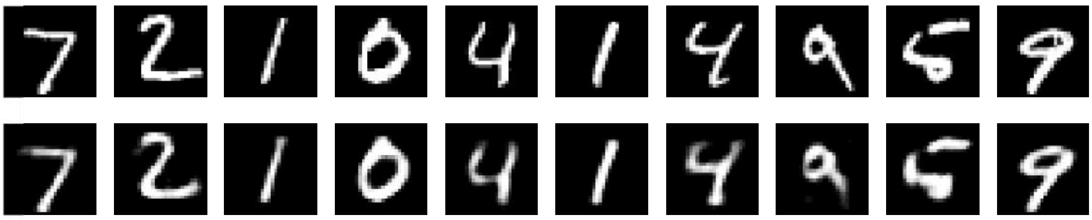
```
# Denois zng Autoencoder

# Add randorr nDzse I D test irrages

x_test_noisy = x_test + notse_lactor * nn.rannom.normal(loc=B.8, scale=1.B, size=x_test.sTape)
x tes t noisy - np.clip{x tes t noisy, B., I.)

# *razn autoencoder for denoising
autoencoder.fit(
    x_train + noise_factor no.random.normal(size-x_train.snape), x_train,

    oatch_size-256,
    vat1dat1on_data-(x_test_no1sy, x_test/i

demised ings - autoencoder.predict(x test noisy[:10")

pit.-£-igune(-figsize- 2e, 6i)

    ax = plt.subplot(3, 10, i + 1)
    rift.imshou(x_test_noisy[,i .reshapel 2B, 28I, cnap="gra5")

    # €lean (8round tr'utfi)
    ax = pit. subpion(3, ie, i +   + ia)
    rift.imshou(x test[i .rieshane(28, 28,', cnap="gca5'")

    ax - p1t.subplot(3, 10, i + I + 2B)
    alt .imshou(denoised_imgs[i] .res haoe 28, 28,', c'ap'"8'ay")

pit.suptitle("fJoisy '.'s Onzginal \"s Denoised", fontsize-16
```

```
Dovnloadzng data from h?tp :        ge„gagle ois     ?e     l      k                 p
lla u^3a/i1^9m 3^                      es bus/step
Model: "auCoeucoder"
```

Layer (type)	Output Shape	Param #
znput_l ayer (I inputL<'.'e")	(None, 28, 28, 1)	0
flatten (-la:ter.)	(None, 784)	0
dense (Cer.se)	(None, 128)	100,480
dense_1 (De'in-)	(None, 32)	4,128
dense_2 (De is <	(, 128)	4,224
dense_3 (De inE	(None, 784)	101,136

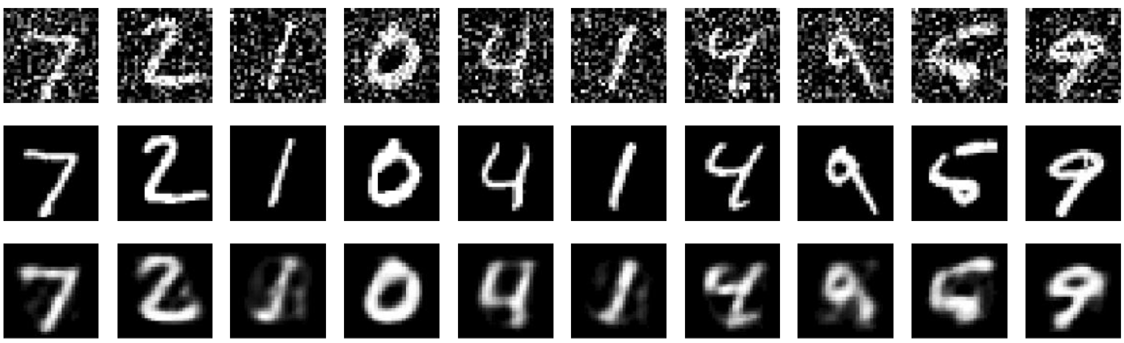
```
Total par'acis: 2 *, 9fi6 (820.19 KB)
7rainable pararis: 29" ?'6* (820.19 KB)
Non-l ralnable pararis: P (0.60 B}
Epoch 1/10
235/235                        6s l0ns/step - loss  8.3386 - yal_1ass: 0.1542
Epoch 2/18
235/235                        1s 3mS/Step - 10SS: 8.1AS4 - vol 10S : 8.1289
Epoch 3/1e
235/235                        1s 3ns/step   Pass: 6.1191-  vat Noss: 0.1115
233/235                        1s 3ns/step - Pass: 0.1104 - vat Noss: 6.1648
Epoch 5/10
Z35/235                        1S 3ñS/Step - 10SS: 8.18*6 - vol 10S : 8.1881
Epoch 6/10
235/235                        1s 3ns/step - Pass: 0.1009 - vat Noss: 6.B978
235/235                        1s 3ms/step - loss: 0.B982 - val loss: 8.0954
Z35/235                        1S 3nS/Step - 1OSS: 0.0965 - Va1 FOSS: 0. B943
Epoch 9/10
235/235                        2s 4ns/step - Pass: 0.6949 - vat Noss: 6. B933
```

Original vs Reconstructed Impages



```
Epoch
235/2
Epoch
235/2
Epoch
235/2
Epoch
235/2
Epoch
235/2
1/1 -
```

Noisy vs Original vs Denoised



Result

- Dimensionality of input data was successfully reduced.
- Autoencoder reconstructed clean images from noisy data.

Conclusion

The Autoencoder was effective for compressing high-dimensional data and denoising noisy inputs, demonstrating its practical applications in feature extraction and data cleaning.

Aim

To generaterealistic faces or interpolate between facial features for creative applications in entertainment and design using VAE.

Objectives

1. Understand the concept of Variational Autoencoders.
2. Generate new face samples using the latent space.
3. Perform interpolation between two latent representations of faces.
4. Explore applications in entertainment and design.

Theory

- **VAE:** An extension of Autoencoders that learns a probabilistic latent space.
- Useful for **generative tasks** such as creating new images and interpolating between features (e.g., smile, hair style).
- In creative industries, VAEs can help in entertainment, character design, and animation.

Procedure

1. Load a dataset of facial images (e.g., CelebA).

2. Build VAE with encoder (to latent distribution) and decoder (to reconstruct images).

3. Train the model on facial dataset.

4. Generate new face images from random latent vectors.

5. Interpolate between two latent codes to morph facial features.

Implementation

```
[ ] import kagglehub
    import os
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    import matplotlib.pyplot as plt
    import numpy as np
```

```
[ ] path = kagglehub.dataset_download("jessicali9530/celeba-dataset")
    print("Path to dataset files:", path)

    # Path to images
    image_dir = os.path.join(path, "img_align_celeba/img_align_celeba")

    # ----------------------------
    # 🔹 Dataset setup
    # ----------------------------
    IMG_SIZE = 64
    BATCH_SIZE = 128
    latent_dim = 256   # larger latent space for clarity
```

```
Path to dataset files: /kaggle/input/celeba-dataset
```

```
def load_and_preprocess(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, [IMG_SIZE, IMG_SIZE])
    img = img / 255.0
    return img

list_ds = tf.data.Dataset.list_files(os.path.join(image_dir, "*.jpg"), shuffle=True)
train_data = list_ds.map(load_and_preprocess, num_parallel_calls=tf.data.AUTOTUNE)
train_data = train_data.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
```

```
def build encoder():
    inpMts = keras.Input(shape=(I E_SIZE, I/'¥2_SIZE, 3))
    z = layers.Conv20(32, 4, strides=2. padding="saee" )(inputs)
    z = layers.BatchNorma1ization‹)(x)
    z = layers. LeakyReLU()(z)

    z = layers.Conv20(64, 4, strides=2. padding="saw")(x)
    z = layers.BatchNorma1ization() (x)
    z = layers. LeakyReLU( )(z)

    z = layers.Conv20(128. 4. str1des=2. padd1ng="s ") (x)
    z = layers.BatchNorma1ization‹) (x)
    z = layers. LeakyReLU( )(z)

    z = layers.Flatten() (x)
    z = layers.Dense‹512, activat1on="relu")(x)

    z_nean = 1ayers .Dense(latent din›(x)
    z_Io var = layers.0ense(1atent_din)‹z›
    reLMrn keras .Model(inputs, [z nean. z_1o var], nane="enco4er")
```

```
[ ) class Sanpling‹1ayers. Layer):
    def call(self, inputs):
        z mean, z_1 var = inputs
        eps = If. random. nomal(shape=tf.shape(z_nean) )
        return z nean + If.exp(0.5 * z_Io var) * eps
```

```
def build decoder():
    inpMts = keras.Input(shape=(Iatent_din, ))
    X = layers. Dense{8•8*2Sfi. activation="relu")(in puts)
    X = layers. Reshape( (8. 8. 2Sfi)) (x)

    z = layers.Conv2D7ranspose(128, 4, strides=2. padding="saae")(x)
    z = layers.BatchNorma1ization‹)(x)
    z = layers. LeakyReLU()(z)


    z - layers.Conv20\ranspose(12B, 4, strides-2. padding-"saee- ) (x)
    z = layervBatchNormalizatiou()(*)
    z = layers.LeakyReLu()(z)

    z = layers.Couv2DTnauspose(64, 4. strides=2, paddiug='same'}(*)
    z = layervBatchNormaIizatiou()(*)
    * =layers.LeAymLu)(*

    x - layers.Conv20Jranspose(3z, 4. strides-2, padding--saw'")(x)
    z - layers.BatchNormal1zation()(x}
    z = layers . LeakyReLu() (z}

    outputs = layers.Conv2DTranspose‹3. 3, acLivation="sig•oid", padding="s "){x)
    netuwn keras.Model(iuputs. outputs. uane='decoden')
```

```
t ) class VA£(keras.Odel ):
    def init (self, encoder, decoder):
        super(). init ()
        self.encOder - encoder
        self.decOder = decoder
        self-.sanpling = Sanpling( )

    def ccmpzIe(sell, optimizer):
        super().conpi1e‹)
        self.optimizer = optimizer
        self-.total_loss_tracker = teras .retries.Mean(nane="lnss')
        self-. recon_loss_tracker - teras .rx tries.nean(nane-"reconstruction_loss'} I
        self-. kl_los s_tracker - keras. netrics.nean(nane-'kl_loss')


            z_nean. z_loyvar = se lf. encoder(data)
            z = sell. sanpling( [ z riean, z I var] )
            reconstruction = self. decoder(z)
```

```
reconst iuct on \oss = tf-.re duce re an(
    t*. redr‹e sun t-f. sguare data - reconstru‹t von;', axls - 1, 2, 3:

    t'. i"edMCe stJIII 1 + z l ag \'ar - t'. sou at"el z ne ar " - t'. exp z log \'at" , axi s=

tota1 lass = reconstru‹tTon lass + kl fass

grads - tape. gradsent total loss, sell-.trainable 'ueights!
self. optiuizer. appl\' gradients (z.u grads, self- .traznahle height s '!

set-f. total loss I Mackev. update state Stat at loss /i
set-f. recon loss trackev. update state reconsI ruct on loss
set-f. k\ loss tramker. update st ate(kJ Noss/'

" econst" ‹-:c \ \o s" : seal.recon los s tracker. result ,
```

```
vae = VAE(encoder, decoder)

vae.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-4))
vae.fit(train data. epochs=10)
```

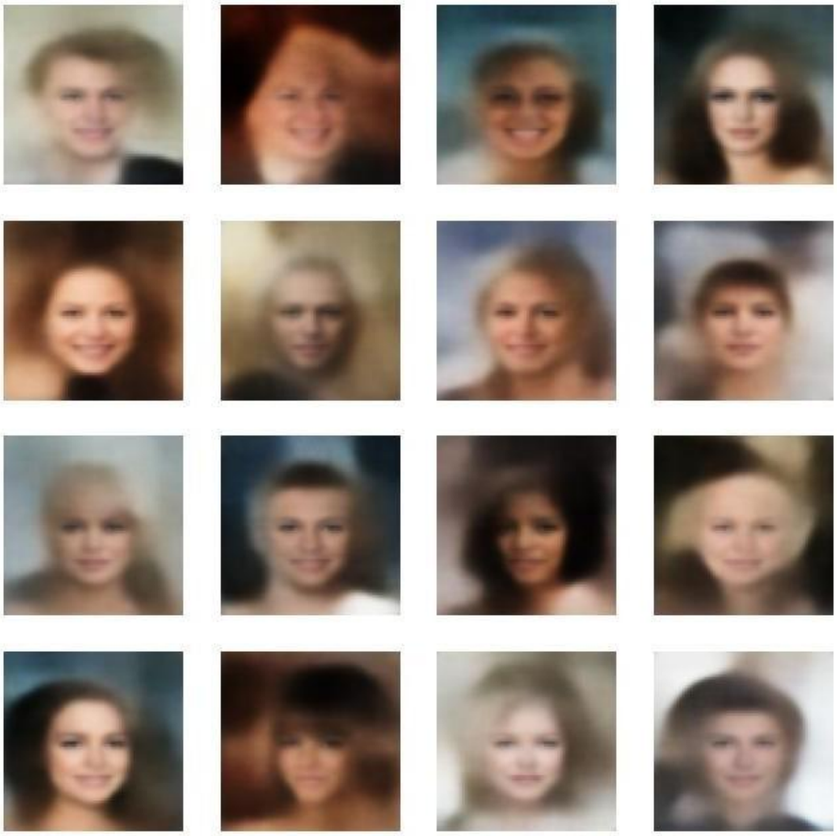
```
1583/1583 ━━━━━━━━━━━━━━━━━━━━ SSO 3?8 /t # - t1 o : 8.8?°8 - 10 : ?97 138f - ti0, Io ??8.2798
Epoch 2/10
```

```
vae.fit(train_data, epochs=10)
```

```
Epoch 1/10
1583/1583 ━━━━━━━━━━━━━━━━━━━━ 550s 338ms/step - kl_loss: 38.8590 - loss: 597.1385 - reconstruction_loss: 558.2798

1583/1583 ━━━━━━━━━━━━━━━━━━━━ 125s 68ms/step - kl_loss: 58.8094 - loss: 294.3011 - reconstruction_loss: 235.4918

1583/1583 ━━━━━━━━━━━━━━━━━━━━ 113s 71ms/step - kl_loss: 57.2343 - loss: 262.2190 - reconstruction_loss: 204.9847

1583/1583 ━━━━━━━━━━━━━━━━━━━━ 137s 68ms/step - kl_loss: 57.7106 - loss: 250.7358 - reconstruction_loss: 193.0251

1583/1583 ━━━━━━━━━━━━━━━━━━━━ 142s 68ms/step - kl_loss: 58.2287 - loss: 244.1204 - reconstruction_loss: 185.8917

1583/1583 ━━━━━━━━━━━━━━━━━━━━ 142s 69ms/step - kl_loss: 58.7620 - loss: 239.3699 - reconstruction_loss: 180.6080

1583/1583 ━━━━━━━━━━━━━━━━━━━━ 140s 68ms/step - kl_loss: 59.3566 - loss: 236.2734 - reconstruction_loss: 176.9169

1583/1583 ━━━━━━━━━━━━━━━━━━━━ 109s 69ms/step - kl_loss: 60.0661 - loss: 233.1404 - reconstruction_loss: 173.0743

1583/1583 ━━━━━━━━━━━━━━━━━━━━ 142s 69ms/step - kl_loss: 60.8176 - loss: 230.5452 - reconstruction_loss: 169.7274
Epoch 10/10
1583/1583 141s 68ms/step - kl_loss: 61.2333 - loss: 227.8262 - reconstruction_loss: 166.5929
<keras.src.callbacks.history.History at 0x7ff7d586ce60>
```

```
z_samples = np.random.normal(size=(16, latent_dim))
generated = decoder.predict(z_samples)

  .figure(figsize=(8,8))
   i in range(16):
   plt.subplot(4,4,i+1)
   plt.imshow(generated[i])
   plt.axis("off")
```

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 895ms/step



```python
def interpolate_points(z1, z2, n_steps=10):
    ratios = np.linspace(0, 1, num=n_steps)
    return np.array([(1-r)*z1 + r*z2 for r in ratios])

z1 = np.random.normal(size=(latent_dim,))
z2 = np.random.normal(size=(latent_dim,))
interpolated = interpolate_points(z1, z2, n_steps=10)
decoded_faces = decoder.predict(interpolated)

plt.figure(figsize=(20, 4))
for i in range(10):
    ax = plt.subplot(1, 10, i + 1)
    plt.imshow(decoded_faces[i])
    plt.axis("off")
plt.suptitle("Latent Space Interpolation Between Two Faces", fontsize=16)
plt.show()
```

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 790ms/step

Latent Space Interpolation Between Two Faces

Result

- Generated realistic facial images using trained VAE.
- Successfully interpolated between two faces to create smooth transitions of features.

Conclusion

The VAE successfully generated realistic faces and performed smooth interpolation between features, showing its potential in creative design and entertainment applications.