ICP6 REPORT

1. EarlyStopping

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import EarlyStopping

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

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Define EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=5,  # Number of epochs with no improvement after which training will be stopped
                               restore_best_weights=True)  # Restores model to best weights with the lowest validation loss

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=100,  # Set a high number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[early_stopping])  # Add the early stopping callback
```

Output:-

```
Epoch 95/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.1335 - val_loss: 0.1318
Epoch 96/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - loss: 0.1329 - val_loss: 0.1318
Epoch 97/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1334 - val_loss: 0.1319
Epoch 98/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1330 - val_loss: 0.1318
Epoch 99/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.1332 - val_loss: 0.1317
Epoch 100/100
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - loss: 0.1333 - val_loss: 0.1318
<keras.src.callbacks.history.History at 0x7da48f9cc880>
```

2. TerminateOnNaN

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import TerminateOnNaN

# Define the TerminateOnNaN callback
terminate_on_nan = TerminateOnNaN()

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

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)
```

```
# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
# Assuming x_train and x_test are your training and validation datasets
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=30,  # Set the number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[terminate_on_nan])  # Add the TerminateOnNaN callback
```

Output:

```
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1527 - val_loss: 0.1512
Epoch 24/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.1525 - val_loss: 0.1511
Epoch 25/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1526 - val_loss: 0.1510
Epoch 26/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1522 - val_loss: 0.1508
Epoch 27/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1522 - val_loss: 0.1506
Epoch 28/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1518 - val_loss: 0.1505
Epoch 29/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.1519 - val_loss: 0.1506
Epoch 30/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1519 - val_loss: 0.1503
<keras.src.callbacks.history.History at 0x7da497d9cc10>
```

3. ModelCheckpoint

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import ModelCheckpoint

# Define the ModelCheckpoint callback
checkpoint = ModelCheckpoint(filepath='autoencoder_best.keras',  # File path to save the model
                             monitor='val_loss',  # Metric to monitor
                             save_best_only=True,  # Save only the best model (based on the monitored metric)
                             mode='min',  # Minimize the monitored metric (e.g., validation loss)
                             save_weights_only=False,  # Save the entire model (set to True to save only weights)
                             verbose=1)  # Print a message when saving the model


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

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features
```

```python
# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
# Assuming x_train and x_test are your training and validation datasets
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=30,  # Number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),  # Validation data
                callbacks=[checkpoint])  # Add the ModelCheckpoint callback
```

Output:

```
Epoch 25: val_loss improved from 0.14227 to 0.14186, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1437 - val_loss: 0.1419
Epoch 26/30
232/235 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.1434
Epoch 26: val_loss improved from 0.14186 to 0.14147, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - loss: 0.1434 - val_loss: 0.1415
Epoch 27/30
230/235 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.1433
Epoch 27: val_loss did not improve from 0.14147
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - loss: 0.1433 - val_loss: 0.1415
Epoch 28/30
234/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1430
Epoch 28: val_loss improved from 0.14147 to 0.14122, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - loss: 0.1430 - val_loss: 0.1412
Epoch 29/30
231/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1428
Epoch 29: val_loss improved from 0.14122 to 0.14084, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1428 - val_loss: 0.1408
Epoch 30/30
232/235 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1425
Epoch 30: val_loss did not improve from 0.14084
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.1425 - val_loss: 0.1408
<keras.src.callbacks.history.History at 0x7da497a1cdf0>
```

4. ReduceLROnPlateau

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import ReduceLROnPlateau

# Define the ReduceLROnPlateau callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss',  # Metric to monitor
                              factor=0.5,  # Factor by which the learning rate will be reduced (new_lr = lr * factor)
                              patience=3,  # Number of epochs with no improvement after which learning rate will be reduced
                              min_lr=1e-6,  # Lower bound for the learning rate
                              verbose=1)  # Print message when the learning rate is reduced


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

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))
```

```python
# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
# Assuming x_train and x_test are your training and validation datasets
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=30,  # Number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),  # Validation data
                callbacks=[reduce_lr])  # Add the ReduceLROnPlateau callback
```

Output:

```
Epoch 21/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1394 - val_loss: 0.1376 - learning_rate: 0.0010
Epoch 22/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1389 - val_loss: 0.1374 - learning_rate: 0.0010
Epoch 23/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1386 - val_loss: 0.1372 - learning_rate: 0.0010
Epoch 24/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - loss: 0.1387 - val_loss: 0.1371 - learning_rate: 0.0010
Epoch 25/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.1385 - val_loss: 0.1369 - learning_rate: 0.0010
Epoch 26/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1384 - val_loss: 0.1367 - learning_rate: 0.0010
Epoch 27/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1383 - val_loss: 0.1366 - learning_rate: 0.0010
Epoch 28/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.1383 - val_loss: 0.1365 - learning_rate: 0.0010
Epoch 29/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.1381 - val_loss: 0.1364 - learning_rate: 0.0010
Epoch 30/30
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1378 - val_loss: 0.1363 - learning_rate: 0.0010
<keras.src.callbacks.history.History at 0x7da4978cb730>
```

5. All combined callbacks

```python
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TerminateOnNaN, ReduceLROnPlateau

# EarlyStopping callback to stop training if validation loss stops improving
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# ModelCheckpoint callback to save the best model based on validation loss
checkpoint = ModelCheckpoint(filepath='autoencoder_best.keras', monitor='val_loss', save_best_only=True, verbose=1)

# TerminateOnNaN callback to stop training if the loss becomes NaN
terminate_on_nan = TerminateOnNaN()

# Define the ReduceLROnPlateau callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,   patience=3, min_lr=1e-6, verbose=1)

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

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten the images for the autoencoder
x_train = x_train.reshape((len(x_train), -1))  # -1 infers the remaining dimension
x_test = x_test.reshape((len(x_test), -1))  # -1 infers the remain

# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Adding a layer
encoded1 = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer
decoded1 = Dense(encoding_dim, activation='relu')(encoded1)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded1)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Training with multiple callbacks
autoencoder.fit(x_train, x_train,
                epochs=30,  # You can set a high number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[reduce_lr, early_stopping, checkpoint, terminate_on_nan])  # Using multiple callbacks
```

Output:

```
Epoch 27: val_loss improved from 0.14445 to 0.14428, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1458 - val_loss: 0.1443 - learning_rate: 0.0010
Epoch 28/30
234/235 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1458
Epoch 28: val_loss improved from 0.14428 to 0.14414, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1458 - val_loss: 0.1441 - learning_rate: 0.0010
Epoch 29/30
234/235 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.1454
Epoch 29: val_loss did not improve from 0.14414
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1454 - val_loss: 0.1442 - learning_rate: 0.0010
Epoch 30/30
234/235 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.1455
Epoch 30: val_loss improved from 0.14414 to 0.14409, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.1455 - val_loss: 0.1441 - learning_rate: 0.0010
<keras.src.callbacks.history.History at 0x7da48f936a40>
```

6. Loading the model and prediction

```python
from tensorflow.keras.models import load_model

# Load the entire model
best_autoencoder = load_model('autoencoder_best.keras')

# Let's look at the encoded representations
encoded_data = best_autoencoder.predict(x_test)
print(encoded_data)
print(encoded_data.shape)
```

Output:

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
[[1.64483538e-08 1.10975877e-08 3.07585424e-08 ... 8.94513619e-09
  3.49731941e-08 1.74193104e-08]
 [2.68281708e-12 2.92231031e-11 1.14608084e-12 ... 1.87413453e-12
  1.56083826e-11 2.35215276e-12]
 [1.24451766e-19 2.81275679e-21 8.90675610e-21 ... 1.63003541e-22
  1.10593965e-17 6.16832367e-23]
 ...
 [7.96392195e-13 3.71499619e-12 3.52240958e-12 ... 1.91721747e-12
  7.92532652e-13 2.35633323e-12]
 [7.35418410e-12 4.97987138e-11 7.45816811e-11 ... 1.90720616e-11
  4.65448999e-11 2.55047784e-12]
 [3.65559650e-18 1.12815326e-16 6.62991837e-18 ... 9.29480934e-18
  1.10364274e-17 1.59720098e-18]]
(10000, 784)
```

Github Link:- https://github.com/Ksahitha/BDA.git

YouTube Link:- https://youtu.be/nuyCM5Rw6tE