ICP 6 REPORT

```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
encoded_layer = Dense(encoding_dim, activation='relu')(encoded)
```

```python
# Define the decoder, use encoded_layer as input instead of encoded1
# Adding a layer
decoded_layer = Dense(encoding_dim, activation='relu')(encoded_layer) # Use encoded_layer here
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(decoded_layer) # Use decoded_layer here


# 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
```

```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,))
```

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

# Adding a layer, use encoded_layer as input instead of encoded1
decoded_layer = Dense(encoding_dim, activation='relu')(encoded_layer) # Use encoded_layer here
# Define the decoder, use decoded_layer as input instead of decoded1
decoded = Dense(input_dim, activation='sigmoid')(decoded_layer) # Use decoded_layer here

# 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 TerminateOn
```

Untitled10.ipynb

```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]
```

```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
encoded_layer = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer, use encoded_layer as input instead of encoded1
decoded_layer = Dense(encoding_dim, activation='relu')(encoded_layer)  # Use encoded_layer here
# Define the decoder, use decoded_layer as input instead of decoded1
decoded = Dense(input_dim, activation='sigmoid')(decoded_layer)  # Use decoded_layer here

# 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
```

```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
```

```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
encoded_layer = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer, use encoded_layer as input instead of encoded1
decoded_layer = Dense(encoding_dim, activation='relu')(encoded_layer) # Use encoded_layer here
# Define the decoder, use decoded_layer as input instead of decoded1
decoded = Dense(input_dim, activation='sigmoid')(decoded_layer) # Use decoded_layer here


# 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
```

```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
```

```python
# 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
encoded_layer = Dense(encoding_dim, activation='relu')(encoded)

# Adding a layer, use encoded_layer as input instead of encoded1
decoded_layer= Dense(encoding_dim, activation='relu')(encoded_layer)  # Fixed: Use encoded_layer here
# Define the decoder, use decoded_layer as input instead of decoded1
decoded = Dense(input_dim, activation='sigmoid')(decoded_layer)  # Fixed: Use decoded_layer here

# 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,
```

min_lr: Any

```python
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
```

```
Epoch 17/30
228/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.1469
Epoch 17: val_loss improved from 0.14541 to 0.14494, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1469 - val_loss: 0.1449 - learning_rate: 0.0010
Epoch 18/30
229/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.1465
Epoch 18: val_loss improved from 0.14494 to 0.14440, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1465 - val_loss: 0.1444 - learning_rate: 0.0010
Epoch 19/30
231/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.1461
Epoch 19: val_loss improved from 0.14440 to 0.14390, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1461 - val_loss: 0.1439 - learning_rate: 0.0010
Epoch 20/30
225/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.1457
Epoch 20: val_loss improved from 0.14390 to 0.14365, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1457 - val_loss: 0.1437 - learning_rate: 0.0010
Epoch 21/30
228/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.1446
Epoch 21: val_loss improved from 0.14365 to 0.14300, saving model to autoencoder_best.keras
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1446 - val_loss: 0.1430 - learning_rate: 0.0010
Epoch 22/30
226/235 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.1443
Epoch 22: val_loss improved from 0.14300 to 0.14253, saving model to autoencoder_best.keras
```

[14]
```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)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
[[1.75119294e-11 5.90835920e-13 3.34731982e-12 ... 9.35920265e-12
  1.14782457e-12 5.60907223e-13]
 [3.73498033e-09 4.47984192e-08 2.67801195e-08 ... 1.48025379e-08
  7.60447794e-09 1.10647704e-07]
 [9.43129949e-11 2.13933468e-10 9.09995745e-10 ... 3.27405131e-10
  3.95407387e-11 2.15216733e-08]
 ...
 [1.53404189e-15 1.70396599e-16 3.91804577e-15 ... 2.86246039e-15
  3.23801558e-16 2.59001584e-15]
 [1.27121022e-11 9.90096980e-12 8.65826369e-11 ... 4.37883167e-11
  6.05044834e-12 6.50171861e-11]
 [3.22090504e-15 1.58808479e-14 6.83981053e-15 ... 2.27953473e-14
  5.10732788e-15 2.07234864e-13]]
(10000, 784)
```

My Github Link :

https://github.com/Nitish300903/bda.git