Assignment-2

In this assignment, we need to perform dimensionality reduction using various methods on the MNIST dataset. For feature extraction, we are using PCA, t-SNE and autoencoder. The following steps and research summarize what I have done to complete the assignment:

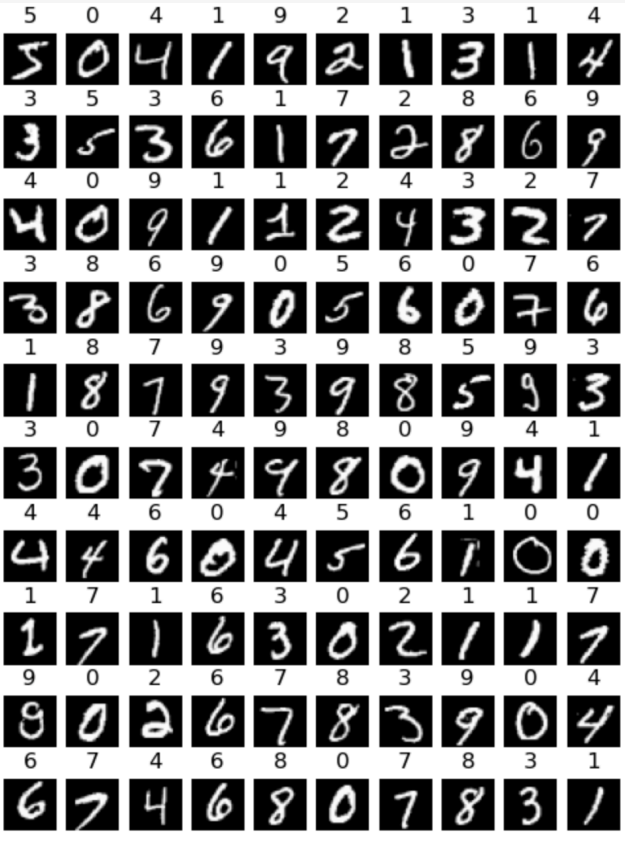
- To start, the MNIST dataset needs to be loaded. I found two possible methods for this. The first involves downloading the dataset file, placing it in the IDE's working directory, and, if it's in CSV format, reading it using pandas. The alternative approach is to use TensorFlow's built-in method to directly download the dataset. I chose the second method, as it avoids any additional setup and simplifies running the program.

```
# First of all MNIST dataset needs to be loaded.
# There are many ways like pandas can read the MNIST dataset with CSV file extension
# For reading dataset by pandas it needs to be downloaded manually and file should be added into IDE.
# Due to that reason, I am loading MNIST dataset using tensorflow which has build in medthod for reading the MNIST dataset

# Import the tensorflow
import tensorflow as tf
# Loading the dataset: already separated into training and testing with feature and target varibales
(feature_train, target_train),(feature_test, target_test) =  tf.keras.datasets.mnist.load_data()
```

- The dataset is displayed in two formats: visually through plotting and in a numerical format.

```
# Importing matplotlib for ploting the dataset
import matplotlib.pyplot as plt
# ploting the dataset
plt.figure(figsize = (6,8))
for i in range(100):
    plt.subplot(10,10, i+1)
    plt.imshow(feature_train[i], cmap = "gray") # Displaying the feature values
    plt.title(f"{target_train[i]}") # Displaying the target value
    plt.axis("off")  # Hide the axes for clarity
plt.show()
```



```
:  # Displaying the dataset 28 X 28
   print(f'{feature_train[0]} is {target_train[0]}')
```

```
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
  25   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
 150  27   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
 253 187   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
 253 249  64   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
 250 182   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
  78   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]] is 5
```

- To normalize the data between 0 and 1, each pixel value is divided by 255, as pixel values in the dataset range from 0 to 255. Dividing by the maximum pixel value scales all values to fall within the range of 0 to 1.

```
# Normalizing the dataset by dividing 255 inorder to convert the values between 0 and 1: pixel range is 0 to 255
feature_train_normalize = feature_train / 255
feature_test_normalize = feature_test / 255
# Note: Target values are not normalized
```

[9]:
```
# Displaying the normalize dataset 28 X 28
print(f'{feature_train_normalize[0]} is {target_train[0]}')
```

```
  0.         0.         0.         0.         0.         0.
  0.01176471 0.07058824 0.07058824 0.07058824 0.49411765 0.53333333
  0.68627451 0.10196078 0.65098039 1.         0.96862745 0.49803922
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
  0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
  0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.19215686 0.93333333 0.99215686 0.99215686 0.99215686
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.98431373
  0.36470588 0.32156863 0.32156863 0.21960784 0.15294118 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.07058824 0.85882353 0.99215686 0.99215686 0.99215686
  0.99215686 0.99215686 0.77647059 0.71372549 0.96862745 0.94509804
  0.         0.         0.         0.         0.         0.
```

- Verifying the dataset's shapes and reshaping it as needed. Since each MNIST image is 28x28 pixels, this results in 784 features per image.

[52]:
```
# Checking the shape of the dataset
print(f'feature train shape : {feature_train_normalize.shape}')
print(f'feature test shape : {feature_test_normalize.shape}')
# Reshaping the datasize for PCA, t-SNE and autoencoder
feature_train_reshape = feature_train_normalize.reshape(-1,784)
feature_test_reshape = feature_test_normalize.reshape(-1,784)
# Displaying the shape after reshaping
print(f'feature train shape after reshaping : {feature_train_reshape.shape}')
print(f'feature test shape after reshaping: {feature_test_reshape.shape}')
```

```
feature train shape : (60000, 28, 28)
feature test shape : (10000, 28, 28)
feature train shape after reshaping : (60000, 784)
feature test shape after reshaping: (10000, 784)
```

- Applied PCA for feature extraction.

[54]:
```
# Applying PCA for feature extraction
# Using the scikit-learning Api's for doing PCA
# Import the PCA from scikit-learn
from sklearn.decomposition import PCA
# Defing the scaler for PCA
pca = PCA(n_components = 100)
# Use the scaler in dataset to extract the feature
feature_train_pca = pca.fit_transform(feature_train_reshape)
feature_test_pca = pca.transform(feature_test_reshape)
```

[12]:
```
# Checking the shape of the dataset after applying PCA
print(f'feature train shape after pca: {feature_train_pca.shape}')
print(f'feature test shape after pca : {feature_test_pca.shape}')
```

```
feature train shape after pca: (60000, 100)
feature test shape after pca : (10000, 100)
```

- I applied t-SNE for feature extraction but encountered computational challenges. Processing all 60,000 images and reducing features to a 3D representation requires significant computing power. I attempted to increase the output dimensions, but with the scikit-learn API, t-SNE only supports up to 4D; beyond this, it produces errors indicating limitations with the algorithm. Consequently, I performed the t-SNE on a smaller subset of the data.

[56]:
```
# Applying t-SNE for feature extraction
# Using the scikit-learning Api's for doing t-SNE
# Import the TSNE from scikit-learn
from sklearn.manifold import TSNE
# Defining the sacler for TSNE
tSNE = TSNE(n_components = 3, random_state = 0)
feature_train_tSNE = tSNE.fit_transform(feature_train_reshape[:100]) # Due to computational time consumtion : cap the input data
feature_test_tSNE = tSNE.fit_transform(feature_test_reshape[:50])
```

[64]:
```
# Checking the shape of the dataset after applying t-SNE
print(f'feature train shape after t-SNE: {feature_train_tSNE.shape}')
print(f'feature test shape after t-SNE: {feature_test_tSNE.shape}')
```

```
feature train shape after t-SNE: (100, 3)
feature test shape after t-SNE: (50, 3)
```

- Applied autoencoder for feature extraction.

```
# Applying Autoencoder for feature extraction
# Using the tensorflow Api's for doing t-SNE
# Import the  layers and models from tensorflow
from tensorflow.keras import layers, models

# Designing the model of autoencoder
# Defining the input
dataset_input = layers.Input(shape = (784,))
# Defining the scaler for encoding
encode = layers.Dense(200, activation = 'relu')(dataset_input)
encode = layers.Dense(100, activation = 'relu')(encode)
# Defing the sacler for decoding
decode = layers.Dense(200, activation = 'relu')(encode)
decode = layers.Dense(784, activation = 'sigmoid')(decode)

# Computing the autoencoder
autoencoder = models.Model(dataset_input, decode)
autoencoder.compile(optimizer = 'adam', loss = 'binary_crossentropy')

# Training the autoencoder
autoencoder.fit(feature_train_reshape, feature_train_reshape, epochs = 20, batch_size = 300,
                shuffle = True, validation_data = (feature_test_reshape, feature_test_reshape))

# Extracting the feature
encoder = models.Model(dataset_input, encode)
feature_train_autoencoder = encoder.predict(feature_train_reshape)
feature_test_autoencoder = encoder.predict(feature_test_reshape)
```

```
Epoch 13/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0760 - val_loss: 0.0748
Epoch 14/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0752 - val_loss: 0.0743
Epoch 15/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0748 - val_loss: 0.0744
Epoch 16/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0745 - val_loss: 0.0736
Epoch 17/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0739 - val_loss: 0.0730
Epoch 18/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0735 - val_loss: 0.0727
Epoch 19/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0733 - val_loss: 0.0723
Epoch 20/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0727 - val_loss: 0.0726
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 250us/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 265us/step
```

[62]:
```
# Checking the shape of the dataset after applying t-SNE
print(f'feature train shape after autoencoder: {feature_train_autoencoder.shape}')
print(f'feature test shape after autoencoder : {feature_test_autoencoder.shape}')
```

```
feature train shape after autoencoder: (60000, 100)
feature test shape after autoencoder : (10000, 100)
```

- After reducing the dimensions using all three methods, I used k-NN to build a model for each of the reduced-dimension datasets. I then calculated the accuracy for each model to evaluate performance.

[48]:
```
# After reducing the dimension by PCA, t-SNE and autoencoder
# Lets evaluate them by using k-NN
# Importing the library for k-NN form sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score # for checking the accuracy of the model

# Predicting by using Dataset that are reduce using PCA
# Defining the scaler for k-nn
KNN_PCA = KNeighborsClassifier(n_neighbors = 5)
# Training the model
KNN_PCA.fit(feature_train_pca, target_train)
# Predicting
y_hat_pca = KNN_PCA.predict(feature_test_pca)
accuracy_pca = accuracy_score(target_test, y_hat_pca)

# Predicting by using Dataset that are reduce using t-SNE
# Defining the scaler for k-nn
KNN_tSNE = KNeighborsClassifier(n_neighbors = 5)
# Training the model
KNN_tSNE.fit(feature_train_tSNE, target_train[:100])
# Predicting
y_hat_tSNE = KNN_tSNE.predict(feature_test_tSNE)
accuracy_tSNE = accuracy_score(target_test[:50], y_hat_tSNE)


# Predicting by using Dataset that are reduce using autoencoder
# Defining the scaler for k-nn
KNN_autoencoder = KNeighborsClassifier(n_neighbors = 5)
# Training the model
KNN_autoencoder.fit(feature_train_autoencoder, target_train)
# Predicting
y_hat_autoencoder = KNN_autoencoder.predict(feature_test_autoencoder)
accuracy_autoencoder = accuracy_score(target_test, y_hat_autoencoder)
```

50]:
```
# Checking the accuracy of all the model with different feature extraction technique
print(f'Accuracy of model with pca feature extraction: {accuracy_pca }')
print(f'Accuracy of model with t-SNE feature extraction: {accuracy_tSNE}')
print(f'Accuracy of model with autoencoder feature extraction: {accuracy_autoencoder}')
```

```
Accuracy of model with pca feature extraction: 0.9726
Accuracy of model with t-SNE feature extraction: 0.14
Accuracy of model with autoencoder feature extraction: 0.9641
```

Conclusion:

After training the model on all three reduced-dimension datasets and comparing their accuracies, it was found that PCA achieved the highest accuracy at 97.26%, followed by the autoencoder with 96.41%, while t-SNE had the lowest accuracy at only 14%. Both PCA and autoencoders are effective for reducing very high-dimensional data to lower dimensions, while t-SNE is more suited for reducing smaller datasets to 2D or 3D. Using t-SNE to reduce higher-dimensional data requires substantial computational power.