سحر محمدی - یادگیری ماشین

تمرین اول، شبکه خودکدگذار و خوشه بندی

Dataset Description

First, I decided my dataset. It is a real dataset named Breast Cancer Dataset from scikit-learn.

It’s a binary classification dataset. Features describe characteristics of cell nuclei present in breast cancer biopsies.

For the Breast Cancer dataset, the **target** variable is binary:

0 typically represents benign tumors. (good)

1 typically represents malignant tumors. (bad)

Features

1. Mean Radius:

The mean of distances from the center to points on the perimeter. Represents the average size of the radius of the nuclei.

2. Mean Texture:

The mean gray-scale intensity values of the pixels in the image. Represents the average texture or smoothness of the cell nuclei.

3. Mean Perimeter:

The mean size of the nuclei's perimeter. Reflects the average length of the boundary of the cell nuclei.

4. Mean Area:

The mean size of the nuclei's area. Represents the average area occupied by the cell nuclei.

5. Mean Smoothness:

The mean of local variation in radius lengths. Describes the smoothness of the cell nuclei.

6. Mean Compactness:

The mean of perimeter^2 / area - 1.0. Measures how compact the shape of the cell nuclei is.

7. Mean Concavity:

The mean severity of concave portions of the contour. Indicates the degree of concavity in the boundary of the cell nuclei.

8. Mean Concave Points:

The mean number of concave portions of the contour. Measures the number of concave points in the boundary of the cell nuclei.

9. Mean Symmetry:

The mean symmetry of the cell nuclei. Reflects how symmetric the cell nuclei are.

10. Mean Fractal Dimension:

The mean fractal dimension of the cell nuclei. Describes the complexity of the cell nuclei shape.

11-20. Standard Error (se) Features:

- For each of the mean features mentioned above, there are corresponding standard error features, denoted by adding "se" as a prefix. For example, se radius, se texture, etc.

21-30. Worst Features:

- Similar to the mean and standard error features, there are corresponding "worst" features representing the worst or largest values among the measurements. For example, worst radius, worst texture, etc.

The dataset consists of 569 instances

Then, I printed the first five row of the dataset which is 30 feature and at the last column the label is printed. After that some visualizations are provided.

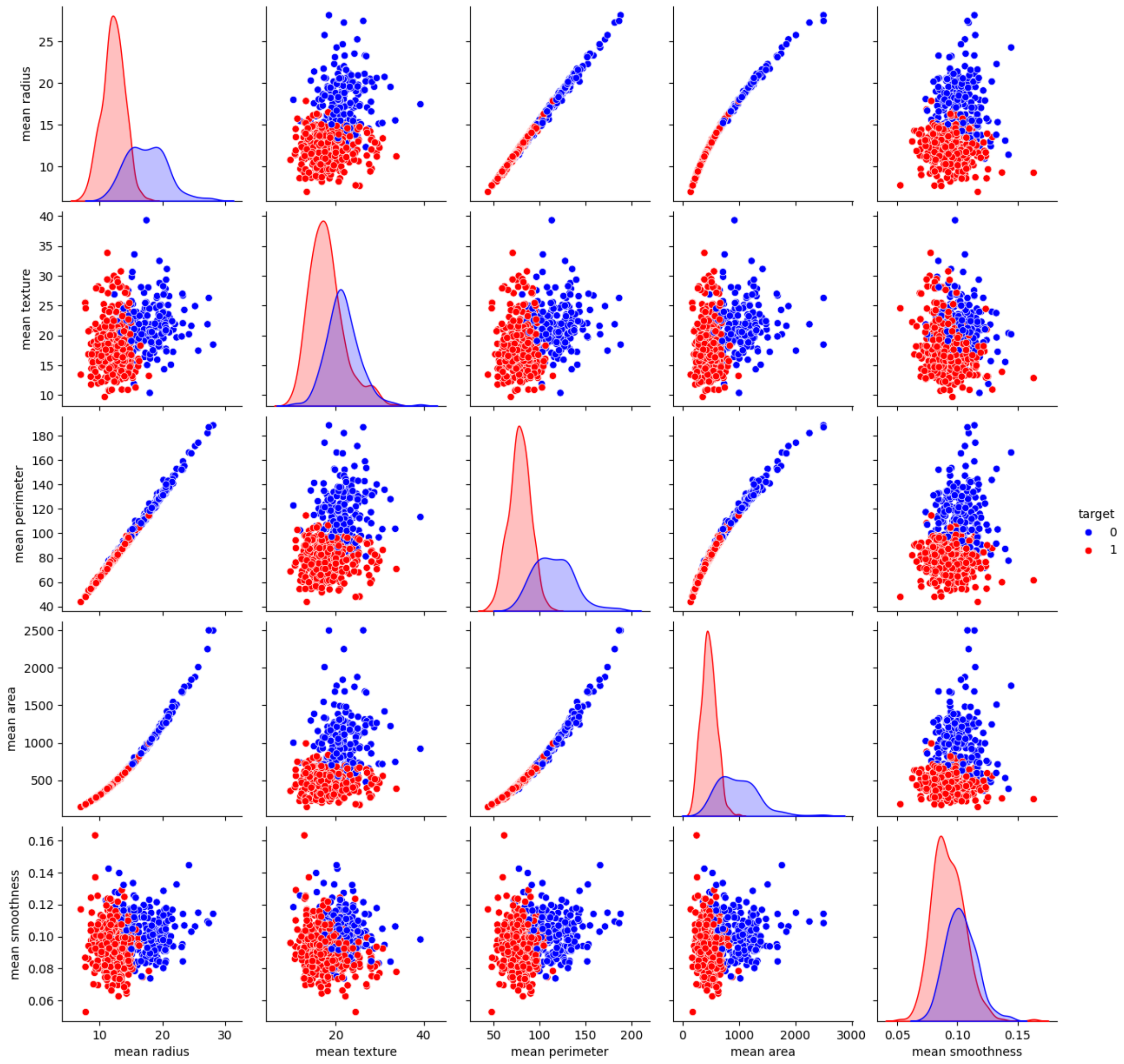
Then, I split the data to 80% for train and 20% for test, and standardized them.

[1]

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.metrics import accuracy_score
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.cluster import DBSCAN
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
import pandas as pd

[2]
# Load Breast Cancer dataset
breast_cancer = load_breast_cancer()

# Create a DataFrame with features and target
df = pd.DataFrame(data=breast_cancer.data,
columns=breast_cancer.feature_names)
df['target'] = breast_cancer.target

# Display the first few rows of the dataset
print(df.head())

# Select a subset of features for visualization (you can customize this)
selected_features = ['mean radius', 'mean texture', 'mean perimeter',
'mean area', 'mean smoothness']

# Add the target variable to the selected features
selected_features_with_target = selected_features + ['target']

# Subset the DataFrame with selected features
df_subset = df[selected_features_with_target]
```

```python
# Plot pair plots colored by target variable
sns.pairplot(df_subset, hue='target', palette={0: 'blue', 1: 'red'})
plt.show()
```

```
  mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0       17.99         10.38          122.80     1001.0          0.11840   
1       20.57         17.77          132.90     1326.0          0.08474   
2       19.69         21.25          130.00     1203.0          0.10960   
3       11.42         20.38           77.58      386.1          0.14250   
4       20.29         14.34          135.10     1297.0          0.10030   

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419   
1           0.07864          0.0869              0.07017         0.1812   
2           0.15990          0.1974              0.12790         0.2069   
3           0.28390          0.2414              0.10520         0.2597   
4           0.13280          0.1980              0.10430         0.1809   

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0   
1                 0.05667  ...          23.41           158.80      1956.0   
2                 0.05999  ...          25.53           152.50      1709.0   
3                 0.09744  ...          26.50            98.87       567.7   
4                 0.05883  ...          16.67           152.20      1575.0   

   worst smoothness  worst compactness  worst concavity  worst concave points  
\
0            0.1622             0.6656           0.7119                0.2654   
1            0.1238             0.1866           0.2416                0.1860   
2            0.1444             0.4245           0.4504                0.2430   
3            0.2098             0.8663           0.6869                0.2575   
4            0.1374             0.2050           0.4000                0.1625   

   worst symmetry  worst fractal dimension  target  
0          0.4601                  0.11890       0  
1          0.2750                  0.08902       0  
2          0.3613                  0.08758       0  
3          0.6638                  0.17300       0  
4          0.2364                  0.07678       0  

[5 rows x 31 columns]
```



[3]

```
data = breast_cancer.data
labels = breast_cancer.target

# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(data, labels,
test_size=0.2, random_state=42)

# Standardize the data
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
data_scaled = scaler.fit_transform(data)
```

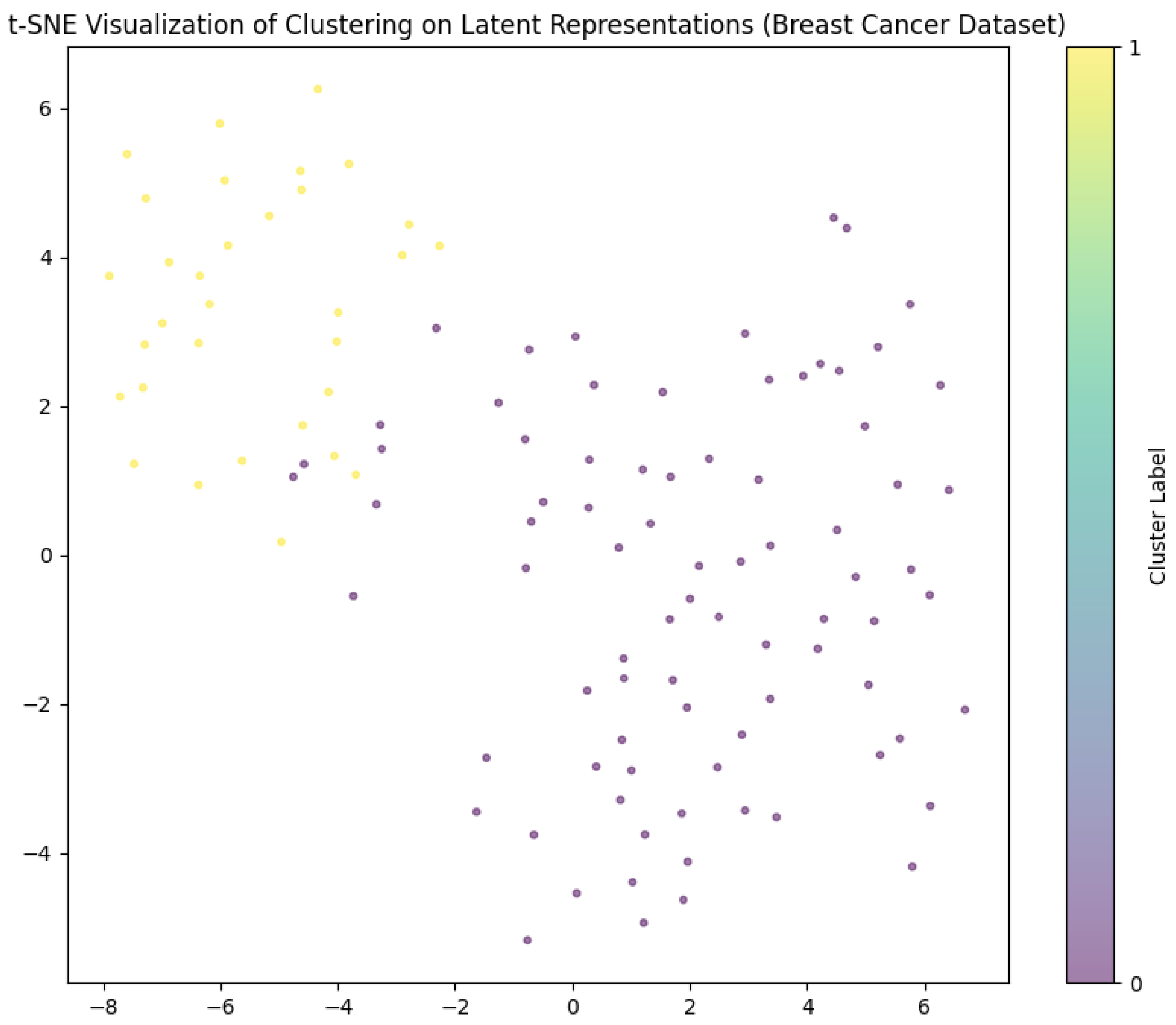
Clustering first attempt

At this stage I used a k-means clustering or the raw test data. The accuracy is 9%.

[4]

```
# Use K-Means clustering on the raw test data
kmeans = KMeans(n_clusters=2, random_state=42)
clustered_labels = kmeans.fit_predict(x_test_scaled)

# Apply t-SNE to reduce dimensionality for visualization
tsne = TSNE(n_components=2, random_state=42)
embedded = tsne.fit_transform(x_test_scaled)

# Evaluate clustering accuracy using true labels
train_accuracy = accuracy_score(y_test, clustered_labels)

print("Train Clustering Accuracy:", train_accuracy)

# Plot the clustered points
plt.figure(figsize=(10, 8))
scatter = plt.scatter(embedded[:, 0], embedded[:, 1], c=clustered_labels,
cmap='viridis', s=10, alpha=0.5)
plt.title('t-SNE Visualization of Clustering on Latent Representations
(Breast Cancer Dataset)')
plt.colorbar(scatter, ticks=range(2), label='Cluster Label')
plt.show()
```

```
Train Clustering Accuracy: 0.09649122807017543
```



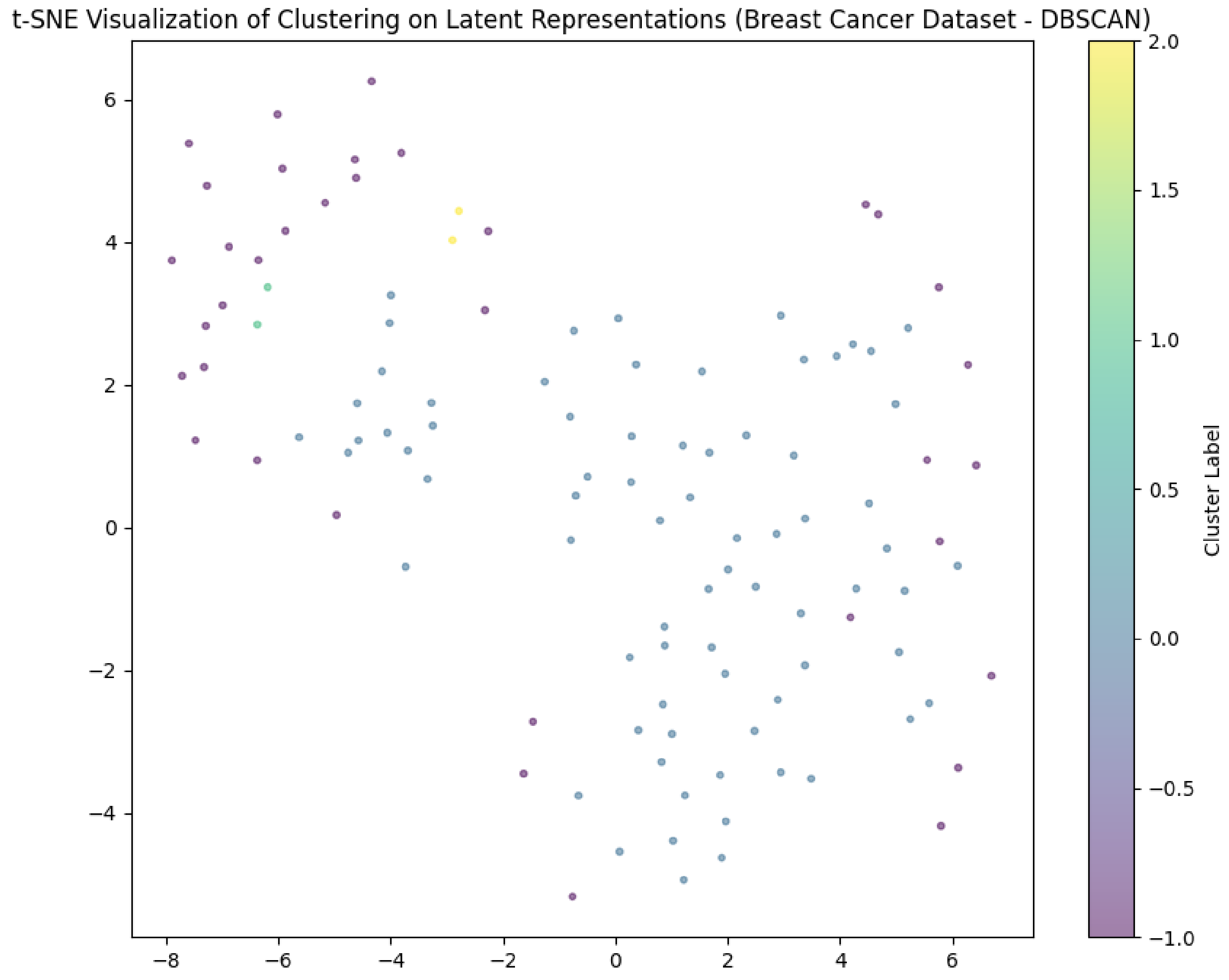
I also used DBSCAN, but the accuracy was only 14%.

[5]

```
# Use DBSCAN clustering on the raw test data
dbscan = DBSCAN(eps=3, min_samples=2)
clustered_labels_dbscan = dbscan.fit_predict(x_test_scaled)

# Apply t-SNE to reduce dimensionality for visualization
tsne = TSNE(n_components=2, random_state=42)
embedded_dbscan = tsne.fit_transform(x_test_scaled)

# Evaluate clustering accuracy using true labels
train_accuracy_dbscan = accuracy_score(y_test, clustered_labels_dbscan)

print("Train Clustering Accuracy (DBSCAN):", train_accuracy_dbscan)
```

```
# Plot the clustered points
plt.figure(figsize=(10, 8))
scatter_dbscan = plt.scatter(embedded_dbscan[:, 0], embedded_dbscan[:, 1],
c=clustered_labels_dbscan, cmap='viridis', s=10, alpha=0.5)
plt.title('t-SNE Visualization of Clustering on Latent Representations
(Breast Cancer Dataset - DBSCAN)')
plt.colorbar(scatter_dbscan, label='Cluster Label')
plt.show()
```

```
Train Clustering Accuracy (DBSCAN): 0.14912280701754385
```



Autoencoder Architecture

The autoencoder architecture consists of an input layer with a shape matching the input dimension. This is followed by three dense layers with decreasing units: 128, 64, and 32, each using the rectified linear unit (ReLU) activation function. The encoder part of the autoencoder captures hierarchical features in a reduced-dimensional latent space. Subsequently, there are three dense layers in the decoder section, mirroring the encoder's structure but in reverse order. The final layer uses the sigmoid activation function to produce the reconstructed output. The autoencoder aims to learn a compact representation of the input data through the encoder and then reconstruct the original data through the decoder.

[6]

```
# Define the autoencoder model
input_dim = x_train_scaled.shape[1]

autoencoder = keras.Sequential([
    layers.Input(shape=(input_dim,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(input_dim, activation='sigmoid')
])
autoencoder.compile(optimizer='adam', loss='mse')

autoencoder.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 128)               3968

 dense_21 (Dense)            (None, 64)                8256

 dense_22 (Dense)            (None, 32)                2080

 dense_23 (Dense)            (None, 64)                2112

 dense_24 (Dense)            (None, 128)               8320

 dense_25 (Dense)            (None, 30)                3870

=================================================================
Total params: 28606 (111.74 KB)
Trainable params: 28606 (111.74 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training and loss

The Mean Squared Error (MSE) is chosen as the loss function for the autoencoder due to its ability to measure the average squared difference between the predicted and true values, emphasizing pixel-wise accuracy in reconstruction tasks. MSE is less sensitive to outliers, providing robustness during training, and its mathematical simplicity facilitates efficient computation and differentiation. The squared nature of MSE gives higher penalty to larger errors, aligning with the goal of accurately reconstructing significant features in the data. Additionally, MSE's interpretation as minimizing the Euclidean distance and its compatibility with the Gaussian assumption make it a common and effective choice for training autoencoders, ensuring the model learns to produce reconstructions that closely match the input data.

Then I trained the network using the train data and set the test data for validation, with 50 epochs and 32 for batch size. I assigned the result to history so that I can plot the result of train loss and validation loss.

[7]
```
# Train the autoencoder

history = autoencoder.fit(x_train_scaled, x_train_scaled, epochs=50,
batch_size=32, shuffle=True, validation_data=(x_test_scaled,
x_test_scaled))
```

```
Epoch 1/50
15/15 [==============================] - 2s 16ms/step - loss: 1.1909 -
val_loss: 0.9649
Epoch 2/50
15/15 [==============================] - 0s 7ms/step - loss: 0.9051 -
val_loss: 0.7260
Epoch 3/50
15/15 [==============================] - 0s 6ms/step - loss: 0.7690 -
val_loss: 0.6683
Epoch 4/50
15/15 [==============================] - 0s 7ms/step - loss: 0.7183 -
val_loss: 0.6310
Epoch 5/50
15/15 [==============================] - 0s 8ms/step - loss: 0.6851 -
val_loss: 0.6149
Epoch 6/50
15/15 [==============================] - 0s 7ms/step - loss: 0.6672 -
val_loss: 0.6012
Epoch 7/50
15/15 [==============================] - 0s 7ms/step - loss: 0.6518 -
val_loss: 0.5908
Epoch 8/50
```

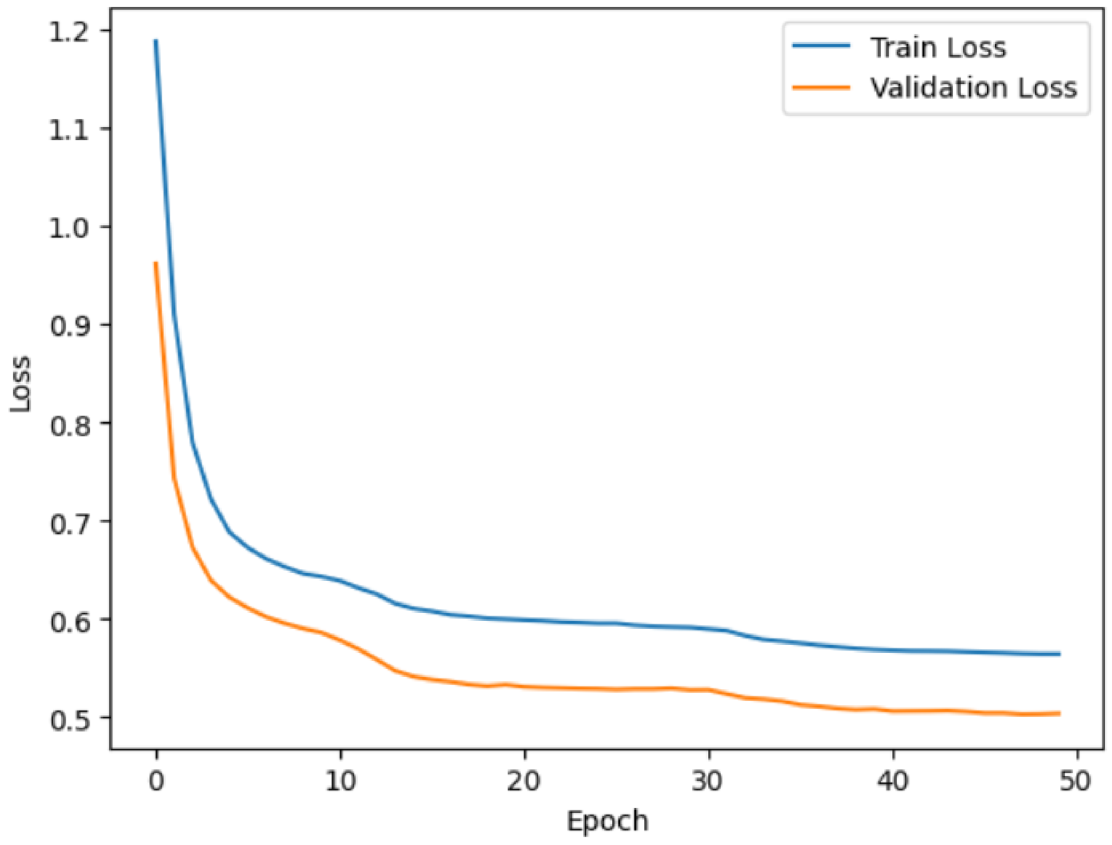
```
15/15 [==============================] - 0s 7ms/step - loss: 0.6435 -
val_loss: 0.5825
Epoch 9/50
15/15 [==============================] - 0s 6ms/step - loss: 0.6377 -
val_loss: 0.5793
Epoch 10/50
15/15 [==============================] - 0s 7ms/step - loss: 0.6328 -
val_loss: 0.5713
Epoch 11/50
15/15 [==============================] - 0s 6ms/step - loss: 0.6264 -
val_loss: 0.5623
Epoch 12/50
15/15 [==============================] - 0s 6ms/step - loss: 0.6169 -
val_loss: 0.5495
Epoch 13/50
15/15 [==============================] - 0s 6ms/step - loss: 0.6070 -
val_loss: 0.5366
Epoch 14/50
15/15 [==============================] - 0s 9ms/step - loss: 0.6013 -
val_loss: 0.5321
Epoch 15/50
15/15 [==============================] - 0s 10ms/step - loss: 0.5979 -
val_loss: 0.5302
Epoch 16/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5962 -
val_loss: 0.5273
Epoch 17/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5936 -
val_loss: 0.5247
Epoch 18/50
15/15 [==============================] - 0s 10ms/step - loss: 0.5921 -
val_loss: 0.5224
Epoch 19/50
15/15 [==============================] - 0s 8ms/step - loss: 0.5904 -
val_loss: 0.5221
Epoch 20/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5878 -
val_loss: 0.5217
Epoch 21/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5869 -
val_loss: 0.5200
Epoch 22/50
15/15 [==============================] - 0s 10ms/step - loss: 0.5846 -
val_loss: 0.5193
Epoch 23/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5839 -
val_loss: 0.5180
Epoch 24/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5834 -
val_loss: 0.5169
Epoch 25/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5811 -
val_loss: 0.5158
Epoch 26/50
```

```
15/15 [==============================] - 0s 10ms/step - loss: 0.5797 -
val_loss: 0.5159
Epoch 27/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5784 -
val_loss: 0.5131
Epoch 28/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5768 -
val_loss: 0.5125
Epoch 29/50
15/15 [==============================] - 0s 10ms/step - loss: 0.5759 -
val_loss: 0.5116
Epoch 30/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5755 -
val_loss: 0.5123
Epoch 31/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5749 -
val_loss: 0.5109
Epoch 32/50
15/15 [==============================] - 0s 10ms/step - loss: 0.5743 -
val_loss: 0.5110
Epoch 33/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5740 -
val_loss: 0.5105
Epoch 34/50
15/15 [==============================] - 0s 10ms/step - loss: 0.5742 -
val_loss: 0.5119
Epoch 35/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5728 -
val_loss: 0.5105
Epoch 36/50
15/15 [==============================] - 0s 9ms/step - loss: 0.5724 -
val_loss: 0.5105
Epoch 37/50
15/15 [==============================] - 0s 8ms/step - loss: 0.5719 -
val_loss: 0.5074
Epoch 38/50
15/15 [==============================] - 0s 7ms/step - loss: 0.5710 -
val_loss: 0.5083
Epoch 39/50
15/15 [==============================] - 0s 6ms/step - loss: 0.5703 -
val_loss: 0.5079
Epoch 40/50
15/15 [==============================] - 0s 6ms/step - loss: 0.5701 -
val_loss: 0.5083
Epoch 41/50
15/15 [==============================] - 0s 7ms/step - loss: 0.5701 -
val_loss: 0.5076
Epoch 42/50
15/15 [==============================] - 0s 6ms/step - loss: 0.5692 -
val_loss: 0.5074
Epoch 43/50
15/15 [==============================] - 0s 7ms/step - loss: 0.5690 -
val_loss: 0.5074
Epoch 44/50
```

```
15/15 [==============================] - 0s 6ms/step - loss: 0.5687 -
val_loss: 0.5075
Epoch 45/50
15/15 [==============================] - 0s 6ms/step - loss: 0.5683 -
val_loss: 0.5072
Epoch 46/50
15/15 [==============================] - 0s 7ms/step - loss: 0.5678 -
val_loss: 0.5066
Epoch 47/50
15/15 [==============================] - 0s 6ms/step - loss: 0.5677 -
val_loss: 0.5095
Epoch 48/50
15/15 [==============================] - 0s 6ms/step - loss: 0.5676 -
val_loss: 0.5071
Epoch 49/50
15/15 [==============================] - 0s 7ms/step - loss: 0.5669 -
val_loss: 0.5060
Epoch 50/50
15/15 [==============================] - 0s 6ms/step - loss: 0.5662 -
val_loss: 0.5066
```

[8]

```
# Plot the training and validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Autoencoder is made of an encoder and then a decoder. The loss function, calculate the distance between the original data and the reconstructed one.

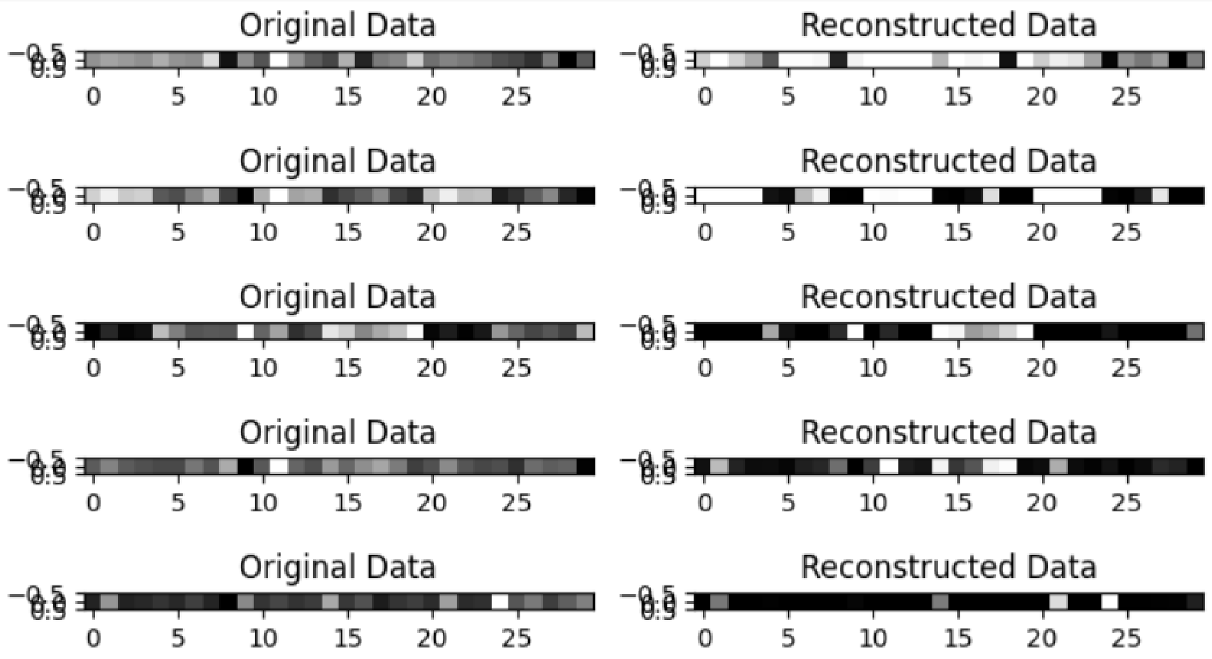
Visualization of Original vs. Reconstructed Data

After that I plot some of the original data and the corresponding reconstructed data. The smoothness and denoising effect are visible.

[9]

```
# Reconstruct data using the trained autoencoder
reconstructed_data = autoencoder.predict(x_test_scaled)

# Choose random indices to visualize samples
sample_indices = np.random.choice(range(len(x_test)), size=5,
replace=False)

# Plot original and reconstructed data for selected samples
for idx in sample_indices:
    plt.figure(figsize=(8, 4))
```

```
# Original data
plt.subplot(1, 2, 1)
plt.imshow(np.reshape(x_test_scaled[idx], (1, -1)), cmap='gray')
plt.title('Original Data')

# Reconstructed data
plt.subplot(1, 2, 2)
plt.imshow(np.reshape(reconstructed_data[idx], (1, -1)), cmap='gray')
plt.title('Reconstructed Data')

plt.show()
```



Clustering with Latent Layer

And finally, I used the output of latent layer for the test data for clustering. The latent layer is the last layer of the encoder section of autoencoder.

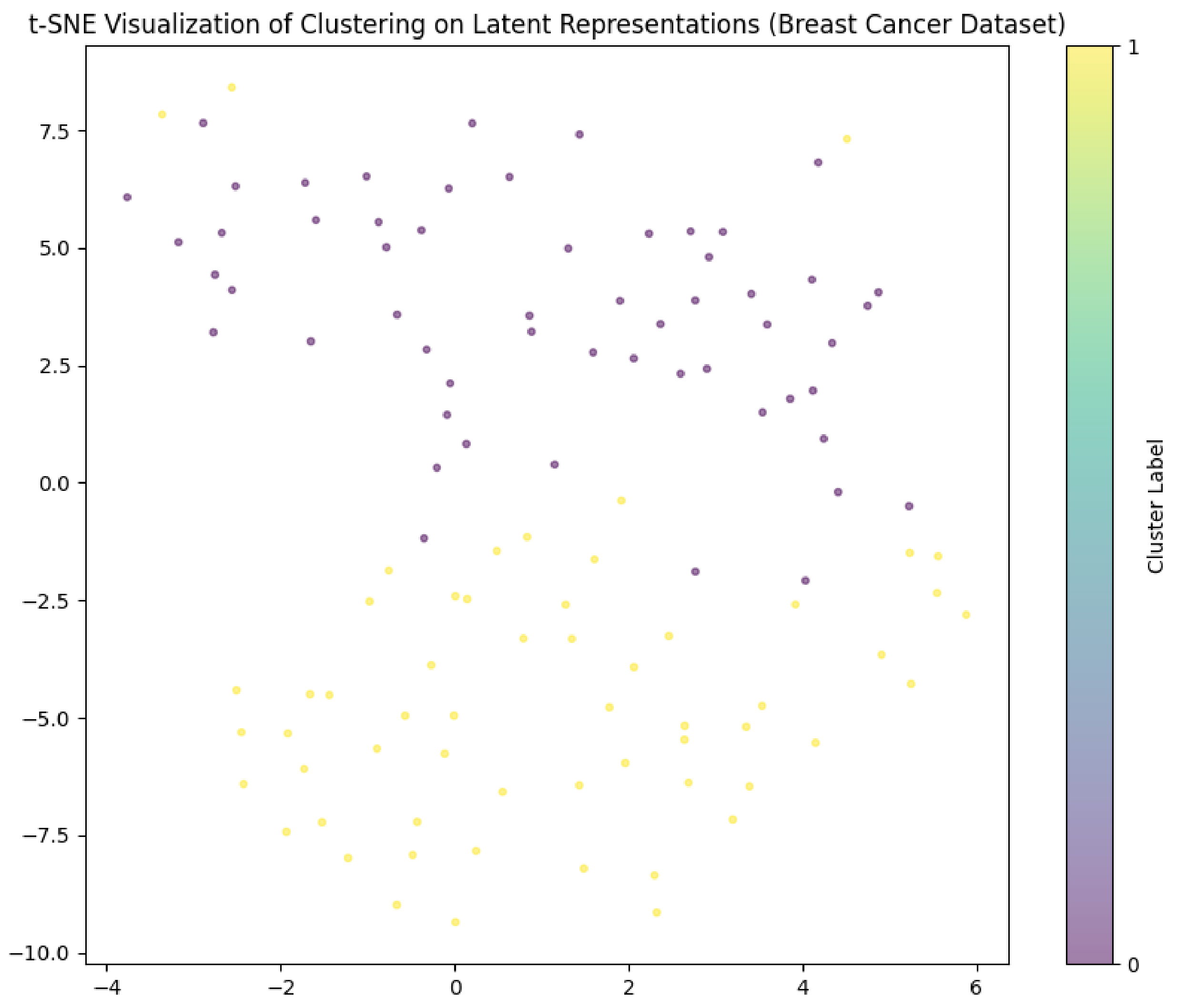
The accuracy raised to 85%.

The observed improvement in accuracy can be attributed to the autoencoder's capacity to learn a more compact and informative representation of the input data. By training the autoencoder to encode and subsequently decode the features of the breast cancer dataset, the model captures essential patterns and structures within the data. The latent layer, acting as a bottleneck in the autoencoder architecture, serves as a condensed representation that emphasizes key characteristics relevant to the clustering task. This learned representation likely highlights intrinsic patterns associated with benign and malignant tumors. Consequently, when applying clustering algorithms to this enriched latent space, the model exhibits higher accuracy as it leverages the refined features encoded by the autoencoder. The denoising and feature-enhancing capabilities of the autoencoder contribute to a more discriminative representation, enabling clustering algorithms to discern subtle differences between benign and malignant tumors with increased precision.

[10]

```
# Extract features from the latent layer
latent_features = autoencoder.layers[3].output

# Define a new model with the latent layer as output
feature_model = keras.Model(inputs=autoencoder.input,
outputs=latent_features)

latent_test = feature_model.predict(x_test_scaled)

# Use K-Means clustering on the latent representations
kmeans = KMeans(n_clusters=2, random_state=42)
clustered_labels = kmeans.fit_predict(latent_test)

# Apply t-SNE to reduce dimensionality for visualization
tsne = TSNE(n_components=2, random_state=42)
embedded = tsne.fit_transform(latent_test)

# Calculate additional evaluation metrics
accuracy = accuracy_score(y_test, clustered_labels)
precision = precision_score(y_test, clustered_labels)
recall = recall_score(y_test, clustered_labels)
f1 = f1_score(y_test, clustered_labels)
conf_matrix = confusion_matrix(y_test, clustered_labels)

# Print the evaluation metrics
print("Clustering Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
# | TN  FP |
# | FN  TP |

# Plot the clustered points
plt.figure(figsize=(10, 8))
scatter = plt.scatter(embedded[:, 0], embedded[:, 1], c=clustered_labels,
cmap='viridis', s=10, alpha=0.5)
plt.title('t-SNE Visualization of Clustering on Latent Representations
(Breast Cancer Dataset)')
plt.colorbar(scatter, ticks=range(2), label='Cluster Label')
plt.show()
```

```
Clustering Accuracy: 0.8508771929824561
Precision: 0.9655172413793104
Recall: 0.7887323943661971
F1 Score: 0.8682170542635659
Confusion Matrix:
[[41  2]
 [15 56]]
```



Conclusion

In this analysis, the Breast Cancer Dataset, comprising 569 instances and 30 features describing cell nuclei characteristics in breast cancer biopsies, was explored. Initial clustering attempts on raw test data using K-Means and DBSCAN yielded low accuracy (9% and 14%, respectively).

Subsequently, an autoencoder architecture, featuring three dense layers in both the encoder and decoder sections, was employed to learn a compact representation of the input data. The model, trained for 50 epochs using Mean Squared Error (MSE) as the loss function, exhibited a decreasing loss over epochs.

The autoencoder demonstrated effective data reconstruction, providing a smooth and denoised effect on the original data.

The latent layer extracted from the autoencoder was then utilized for clustering with K-Means, resulting in a significant accuracy improvement to 85%.

Evaluation metrics, including precision, recall, F1 score, and a confusion matrix, were calculated to comprehensively assess clustering performance.

The findings indicate that the autoencoder's feature learning capabilities enhanced the discriminative power of the data, leading to improved clustering accuracy. This approach holds promise for identifying patterns associated with benign and malignant tumors, offering valuable insights into breast cancer characteristics. Further exploration, including hyperparameter tuning and experimentation with alternative clustering algorithms, could enhance the overall performance of the proposed methodology.