GE461 Project: Dimensionality Reduction and Visualization Report

Efe Tarhan
22002840
Electrical and Electronics Engineering

Contents

Introduction

This project covers the application of dimensionality reduction using different tools employing machine learning and statistical techniques. The fashion-MNIST dataset used for the project includes 10 different classes where each class is labeled with a number from 0 to 9. The numbers correspond to following labels:

```
class_names = {
    0: "T-shirt/top",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle boot"
}
```

Fig.1 Label corresponds of the original class labels.

The dataset consists of greyscale images consist of 28x28 pixels with integer range 0 to 255. Some samples from the dataset can be seen blow as a combined image.

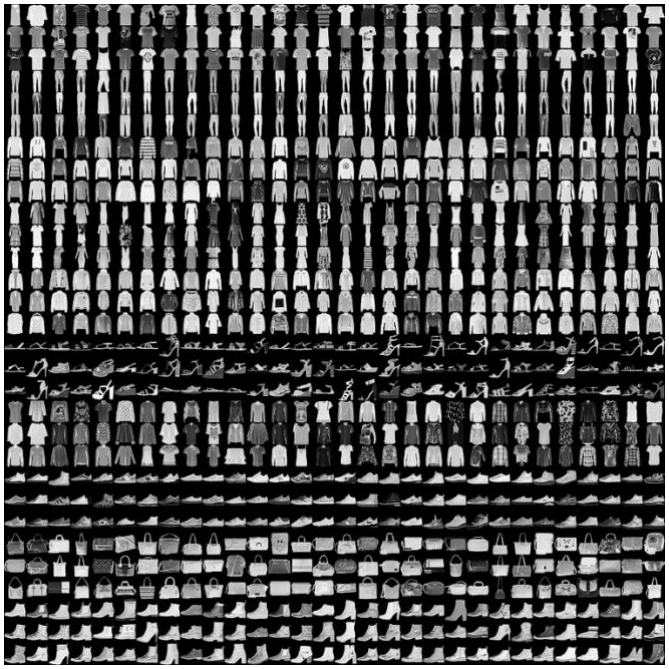

Fig.2 Visualization of the data inside the fashion-MNIST dataset.

Throughout this project the images are flattened to a vector with 784 dimensions. Dimensionality reduction techniques like PCA, random projection, Isomap and t-SNE are applied to reduce the dimensionality of the data in a strategic way to encapsulate as much

information as possible. After applying dimensionality techniques, the data is classified using multivariate Gaussian probability functions. As a simple machine learning technique, the class that gives the highest probability is assigned to each data and the predicted results are compared with real results to obtain training and testing error.

Question 1: Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical technique used to simplify the complexity in high-dimensional data while retaining as much of the original information as possible. It operates on the premise that in any high-dimensional dataset, many dimensions are often correlated and therefore redundant. PCA identifies new, orthogonal axes (principal components) that maximize the variance in the original data. The first principal component captures the most variance, the second captures the second most, and so on, under the constraint that each is orthogonal to the others. Mathematically, PCA involves calculating the eigenvalues and eigenvectors of the data's covariance matrix. The covariance matrix is formulated as following:

$$\Sigma = (X - \mu)(X - \mu)^T$$

where μ is the mean of the columns in the matrix where each column represents a feature for the dataset. The eigenvectors represent the directions of the maximum variance (the principal components), and the eigenvalues indicate the magnitude of variance captured by each eigenvector. By projecting the original data onto these new axes, PCA effectively reduces its dimensionality. This is beneficial for visualization, noise reduction, and improving the efficiency of other machine learning algorithms. The transformation from high-dimensional space to a lower-dimensional subspace is given by $Y = XP$, where X is the original data matrix, P is the matrix of eigenvectors, and Y represents the data in the new subspace. This process of dimensionality reduction while preserving essential data characteristics makes PCA a powerful tool in data science and analytics.

Part 1: Centralizing the Data

In accordance with the guidelines, the dataset, denoted as X undergoes a centering transformation. This procedure involves the subtraction of the mean value of the entire dataset

from each data point within X. It is imperative that this transformation is applied to the dataset in its entirety prior to the partitioning process that segregates the dataset into training and testing subsets.

```
#%% Centralize the Dataset

X_centered = X_list - np.mean(X_list,axis = 0)
```

Fig.3 Centralization code

Part 2: Applying PCA

The "**train_test_split()**" function, as described in [1] and residing within the "**sklearn.model_selection**" module, facilitates the division of a dataset into two distinct subsets. This division is governed by the "**test_size**" parameter, which, for the purposes of this project, is set to 0.5, thereby equally distributing the dataset comprising 10,000 data points into two segments of 5,000 instances each. This segmentation precedes the application of Principal Component Analysis (PCA). Additionally, the function allows for the specification of a seed value for the random number generator to ensure reproducibility of results. For this project, the seed value has been selected as 42.

```
#%% Train-Test Split the Data

X_train, X_test, y_train, y_test = train_test_split(X_centered, y_list, test_size=0.5, random_state=42)
```

Fig.4 Code for splitting the dataset into training and testing sets.

Upon partitioning the dataset into training and testing subsets, an analytical procedure leveraging Principal Component Analysis (PCA) was applied to the training data. The PCA methodology involves the utilization of the "**PCA()**" class, along with its associated functionalities, which are encapsulated within the "**sklearn.decomposition**" library [2]. The "PCA()" object facilitates the derivation of principal components and their respective explained variances by employing the "**.fit()**"method. This process enables the extraction of eigenvalues and eigenvectors, as well as the transformation of the dataset into a specified dimensional space. The explained variance values, derived as eigenvalues of the covariance matrix, quantify the variance that each principal component accounts for within the dataset.

The eigenvalues pertinent to the PCA performed on the training dataset are depicted in the figure.

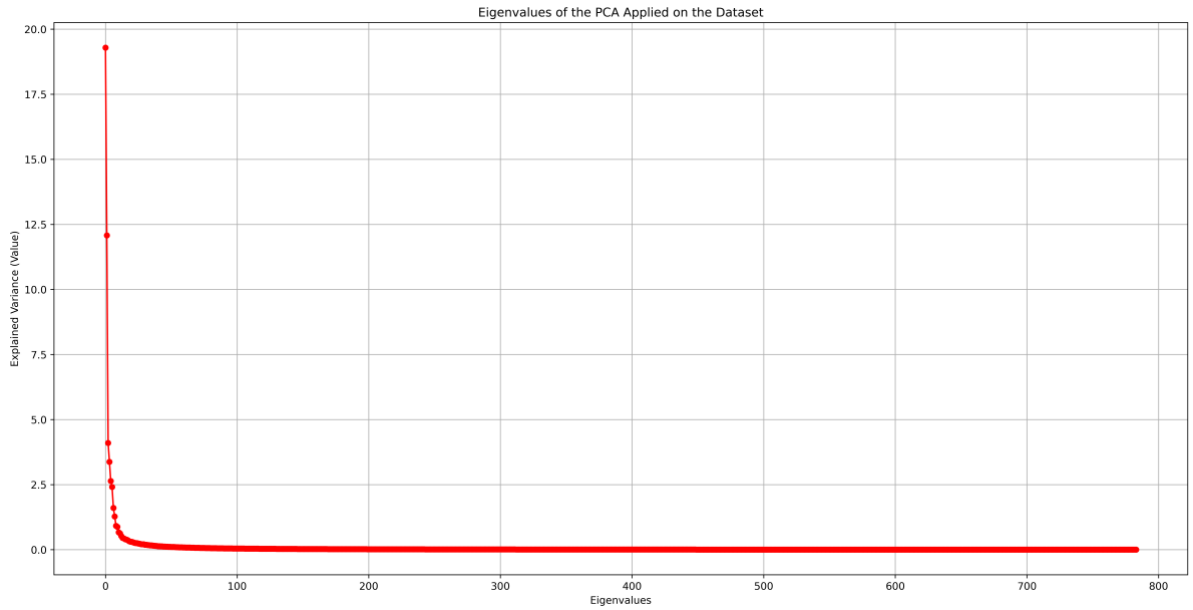

Fig.5 Eigenvalues of the PCA applied on training set.

From the analysis of the figure, it has been determined that the cumulative sum of the largest eigenvalues reveals that the initial 150 components account for approximately 95% of the total variance within the dataset. Similarly, to explain 90% of the dataset's variance, only the first 80 components are required. Given this, the preference to select the initial 150 components is a meaningful choice. This approach enables preserving 95% of the dataset's information content while reducing the dataset’s dimensionality from 784 to 150.

Part 3: Eigenvector Visualization

The sample mean of the training set before centralizing the dataset is shown in the figure below.

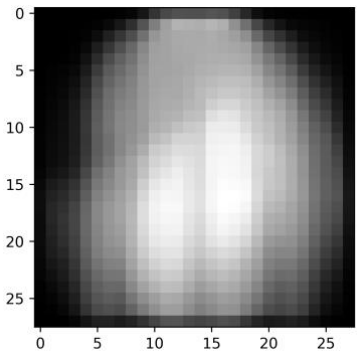

Fig.6 Mean of the training set before centralization.

Also, the first 81 eigenvectors are displayed in the figure below that are cumulatively responsible for %90 of the variance inside the training set.

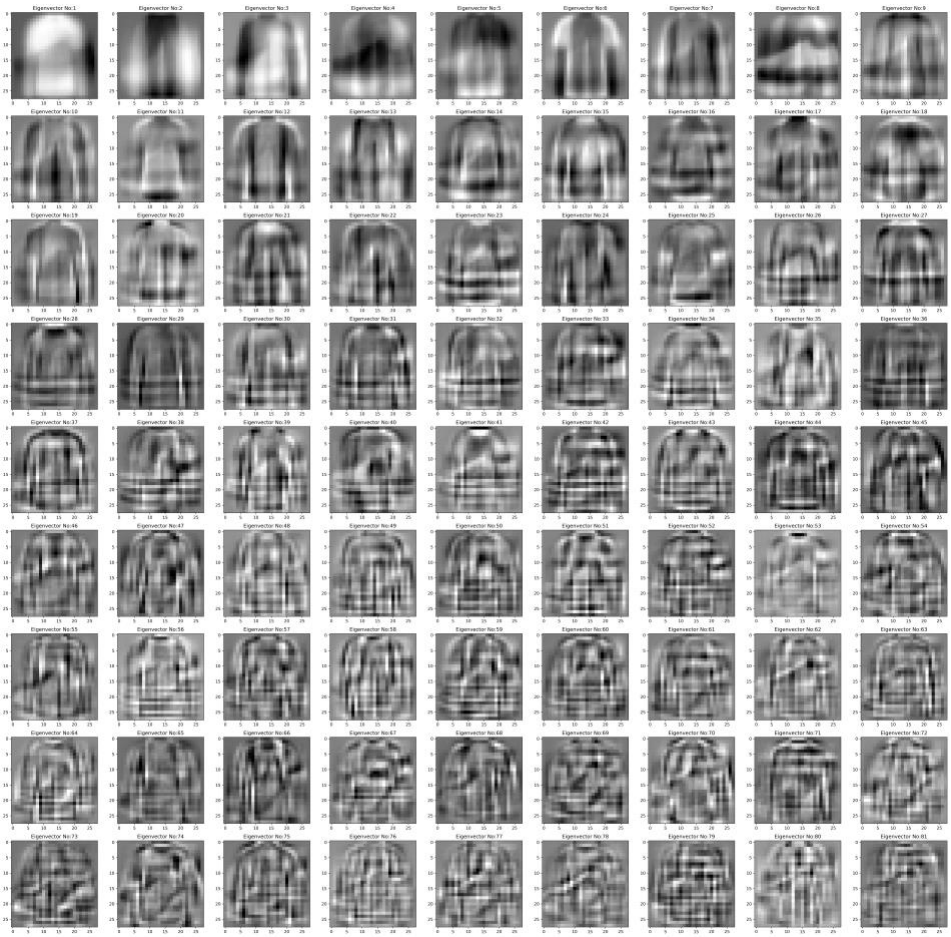

Fig.7 First 81 eigenvectors of the dataset displayed as images.

As observed from the figure, the initial eigenvectors, corresponding to the larger eigenvalues, demonstrate a smoother appearance, like the mean vector of the training set. This phenomenon arises from the capacity of the first eigenvectors to encapsulate high variance within the dataset utilizing relatively simple patterns that are reflective of the data itself. Conversely, as we progress to eigenvectors associated with smaller eigenvalues, there is a notable transition in their complexity. These eigenvectors begin to capture more sophisticated aspects of the data, containing the specific details of individual data instances.

Part 4: Application of Gaussian Classifier on Transformed Data

The employment of a Gaussian classifier for both the training and testing phases of the dataset analysis involves the creation of a handwritten function, denoted as "**GaussianPred()**". This function is designed to take the training set, the test set, and their respective labels as inputs. T "**multivariate_normal()**" method used for the function is sourced from the "**scipy.stats**" library [3]. This method is used for the computation of multivariate Gaussian probabilities for a given data instance across each class. It achieves this by requiring the covariance matrix and mean of the training set, in addition to the specific data instance under consideration, as inputs. The "**multivariate_normal()**" method is used for the evaluation of how likely a data instance is to belong to each class based on the Gaussian distribution parameters derived from the training data. The code snippet provided for the "GaussianPred()" function can be seen below.

```
def GaussianPred(X_train, X_test, y_train, y_test):
    classes = np.unique(y_train)
    models = {}

    for cls in classes:
        indices = np.where(y_train == cls)[0]
        mean = np.mean(X_train[indices], axis=0)
        cov = np.cov(X_train[indices].T)
        models[cls] = multivariate_normal(mean=mean, cov=cov)

    def predict(X):
        probs = np.array([models[cls].pdf(X) for cls in classes])
        return classes[np.argmax(probs, axis=0)]

    y_pred_train = predict(X_train)
    y_pred_test = predict(X_test)

    train_err = np.mean(y_pred_train != y_train.squeeze()) * 100
    test_err = np.mean(y_pred_test != y_test.squeeze()) * 100


    return train_err, test_err
```

Fig.8 The code for obtaining test and training error rates for Gaussian Classifier

The Gaussian classification has been applied for dataset with dimensionality reduction using components in the range **10 to 380 components with a step of 5**.

Part 5: Results

The result of the model with training and testing errors can be seen in the figure below.

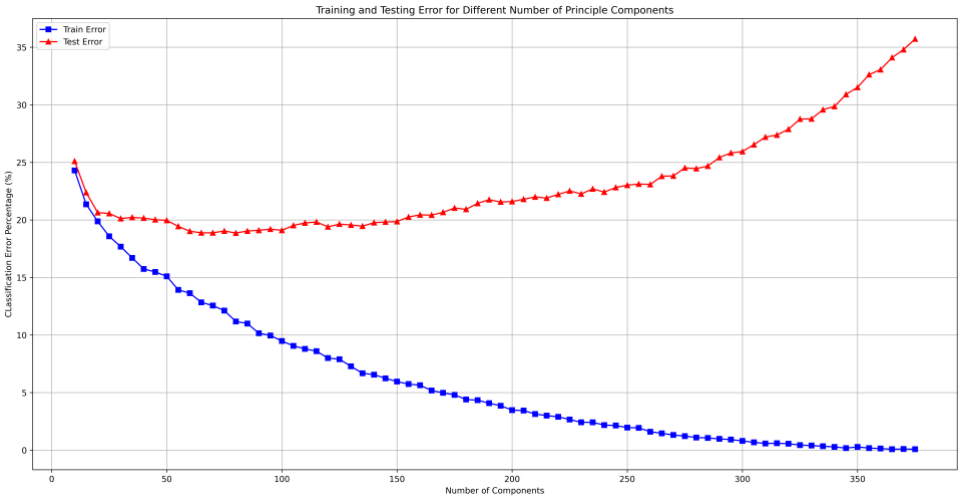

Fig.9 Training and testing errors for different number of principle components

As it can be seen from the plot, using more than **nearly 70 components** causes overfitting of the model to the training set and while the training error rate decreases the test error increases from that point.

Question 2: Random Subspace Projection

For the random transformation part, a matrix has been generated with the dimensions $\#original\_dim\ x\ \#reduced\_dim$. Each entry of this matrix has been initialized as an element from the normal distribution. Then the matrix is normalized column wise to obtain the final transformation matrix.

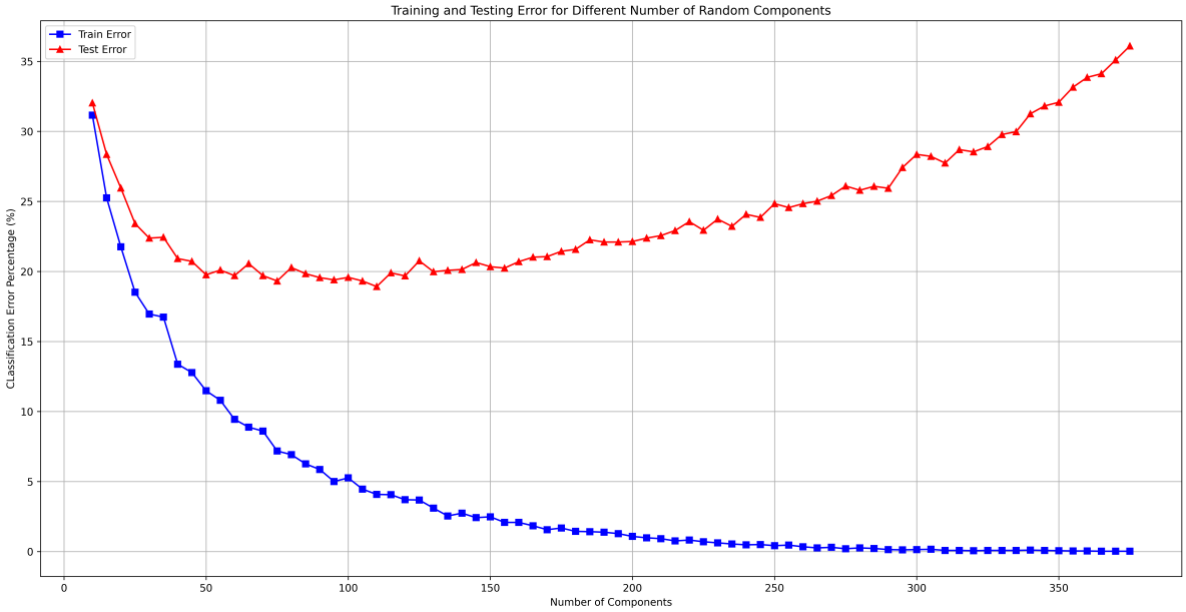

Fig.10 Training and testing errors for different number of random components.

Compared to PCA, random transformation achieves similar test accuracy. However, the improvements don't decrease as they do with PCA because components are not selected based on their variance explanation capacity. Therefore, the performance is less predictable and robust than PCA results.

Question 3: Manifold Based Dimensionality Reduction (Isomap)

Isomap, short for Isometric Mapping, is a nonlinear dimensionality reduction technique that seeks to preserve the global geometry of the original data. It's particularly effective for datasets lying on a curved manifold within a higher-dimensional space. Isomap starts by constructing a neighborhood graph, where each point is connected to its nearest neighbors, reflecting the local structure of the data. This step is crucial as it lays the foundation for preserving the manifold's geometric properties. The core of Isomap's methodology involves estimating the geodesic distances between all pairs of points on the manifold. In contrast to linear techniques like PCA, which consider straight-line Euclidean distances, Isomap uses the constructed graph to approximate the true geodesic distances—the shortest paths along the manifold. This is typically achieved by computing the shortest path distances in the neighborhood graph, often using Dijkstra's algorithm or the Floyd-Warshall algorithm. With the geodesic distances estimated, Isomap then applies classical Multidimensional Scaling (MDS) to these distances. MDS positions each data point in a new, lower-dimensional space in such a way that the between-point distances are preserved as well as possible. The result is a representation of the data in a lower-dimensional space that maintains the intrinsic geometric relationships of the original high-dimensional data. Mathematically, Isomap seeks to minimize the difference between the geodesic distances d_G on the manifold and the Euclidean distances d_Y in the lower-dimensional embedding, aiming for $d_{Y_{i,j}}$ approx $d_{G_{i,j}}$for all points i and j. This process effectively unfolds the manifold, revealing the underlying structure of the data in a way that is often more interpretable and amenable to further analysis or visualization. [4]

Part 1: Settings of the Isomap Method

In the third part of the analysis, a manifold-based dimensionality reduction technique known as “Isomap” was applied to the dataset, and its performance was evaluated in conjunction with a Gaussian classifier. The “**Isomap()**” class function, as implemented in the “**sklearn.manifold**” library [5], aligns closely with the methodology described in the corresponding paper. This function also includes hyperparameters beyond the number of components, notably the number of neighbors. Due to the significant computation time required by this function, only one parameter setting for the number of neighbors was tested, specifically, a value of 3.

Part 2-3: Results

The Gaussian classification was then performed on the dataset after dimensionality reduction, considering a range of 10 to 380 components in steps of 5, like the PCA setup. The outcomes of employing this method are illustrated in the figure below.

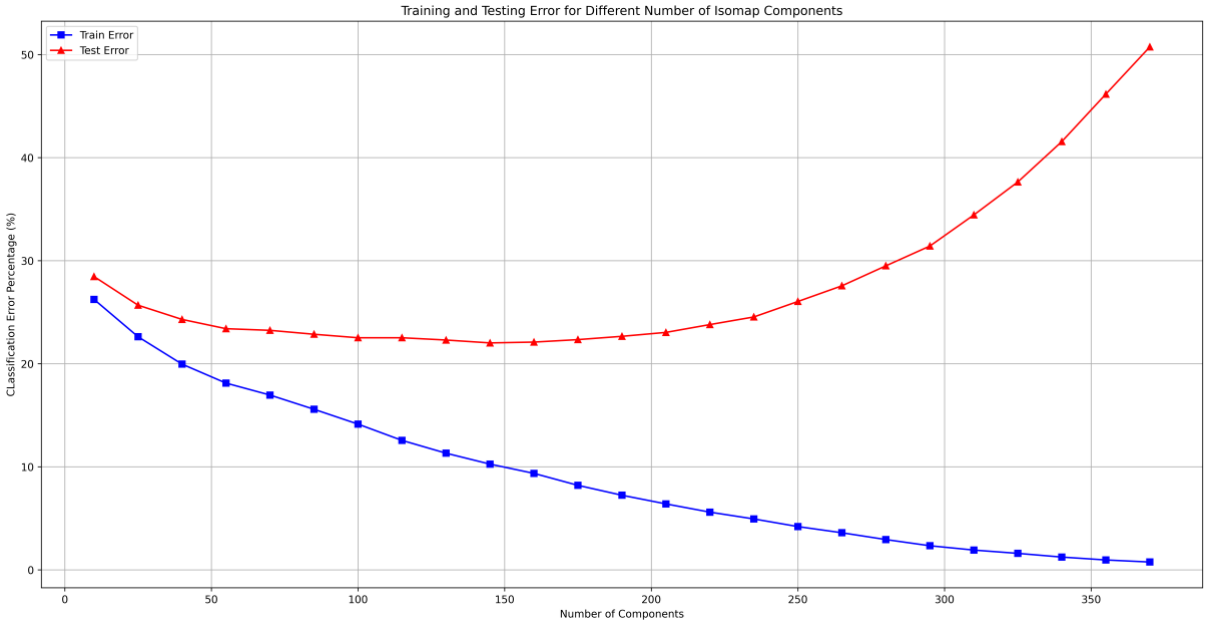

Fig.11 Training and testing errors for different number of components obtained with Isomap technique.

Compared to PCA, the Isomap method exhibited lower accuracy at its point of lowest error on the test set. Although the test error curve for Isomap was higher than that for PCA, the technique demonstrated a more robust and consistent decrease in the training error curve, along with more stable behavior in the test error curve. The underperformance of Isomap

relative to PCA could potentially be attributed to inappropriate hyperparameter selection. Given the nonlinear structure of the data, it was anticipated that Isomap would outperform PCA. This expectation stems from Isomap's design to preserve the intrinsic geometric properties of data lying on a nonlinear manifold, which should ideally make it more adept at handling complex, nonlinear relationships within the data. However, the selection of hyperparameters, particularly the number of neighbors in the Isomap algorithm, is crucial for accurately capturing the manifold's structure. An incorrect choice can significantly impact the algorithm's ability to discern and maintain the true underlying relationships in the data, leading to suboptimal performance.

Question 4: t Distributed Stochastic Neighbour Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a sophisticated machine learning algorithm designed for visualizing high-dimensional data in a low-dimensional space, typically two or three dimensions. Developed by Laurens van der Maaten and Geoffrey Hinton, t-SNE excels at uncovering patterns in the data by preserving local structures and relationships among points. Unlike linear techniques such as PCA, t-SNE focuses on the probability distributions of points in both the high-dimensional and low-dimensional spaces. It converts the Euclidean distances between points in the high-dimensional space into conditional probabilities that represent similarities. The similarity of point x_i to point x_j is the conditional probability $p_{j|i}$ which is high when x_j is close to x_i.

For this part the t-SNE algorithm has been applied to obtain a 2D visualization of the dataset. For this purpose, the "**TSNE()**" class from the "**sklearn.manifold**" library [6] has been used. This function can manage to do the same operations given in the article stated in the guideline. As hyperparameters, the value of perplexity has been set to 30 which is a hyperparameter like number of neighbors in the k-nearest neighbors' regression. For other hyperparameters, the default parameters of the function were used. The result obtained using the t-SNE algorithm is scattered on a two-dimensional plot where each color represents different classes. This plot can be seen from the figure below.

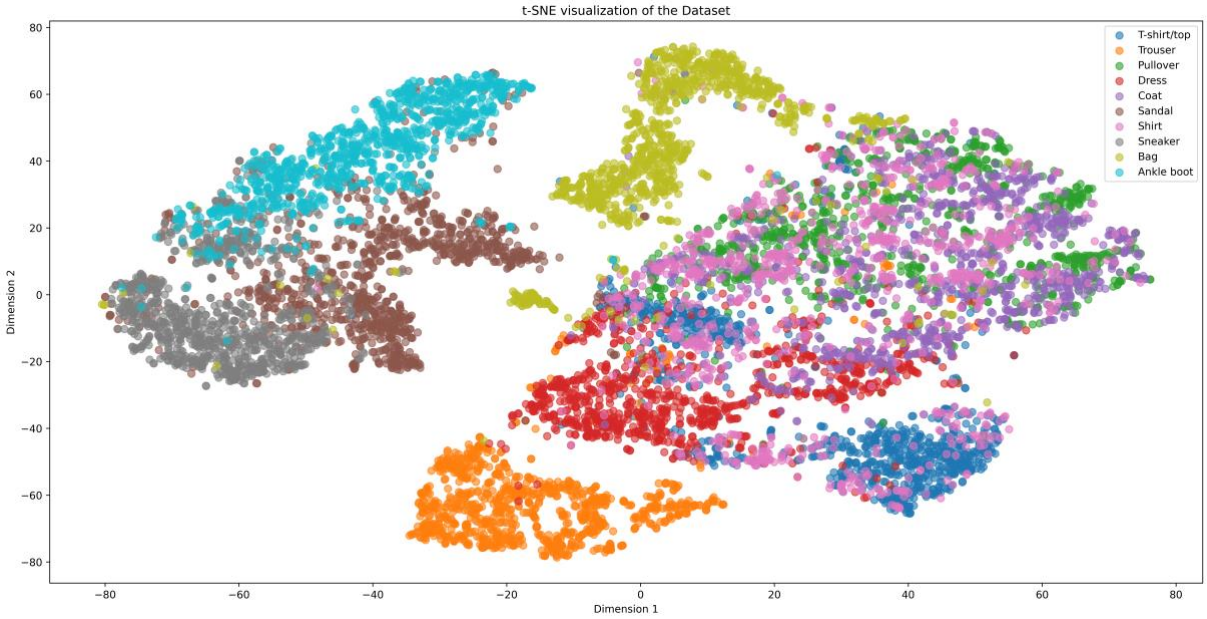

Fig.12 Scatter plot for the results of the t-SNE algorithm

From the dimensionality reduced image the datapoints that belong to sandal, sneaker and ankle boots are mapped closer but still can be mostly separable because of their individual distinctions. Since trousers and bags are totally different categories among the given 10 categories they can easily be separated and visualized individually. For the categories of the clothes that belong to top wear, the separation couldn't be achieved as good as the other classes since the similarity between the classes are higher compared to others. But overall, it could be argued that the t-SNE algorithm has managed to separate the classes well.

References

[1] Scikit-learn developers, "train_test_split," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [Accessed: 6 April 2024].

[2] Scikit-learn developers, "PCA," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. [Accessed: 6 April 2024].

[3] SciPy developers, "multivariate_normal," *SciPy*. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.multivariate_normal.html. [Accessed: 6 April 2024].

[4] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319-2323, Dec. 2000, doi: 10.1126/science.290.5500.2319.

[5] Scikit-learn developers, "Isomap," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html. [Accessed: 6 April 2024].

[6] Scikit-learn developers, "t-SNE," *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html. [Accessed: 6 April 2024].