Assignment-7: PCA

Rezwan-Ul-Alam (ID: 2011659042)
Md. Nur Alam Jowel (ID: 2012355042)
Raian Ruku (ID: 2013409642)

Email addresses:
rezwan.alam1@northsouth.edu
alam.jowel@northsouth.edu
raian.ruku@northsouth.edu

June 3, 2024

I. Introduction

In this assignment, our aim to classify hand signs using various machine learning models. We utilize a dataset containing hand sign images, preprocess these images, and apply different classifiers to evaluate their performance. The primary goal is to determine the effectiveness of dimensionality reduction using PCA (Principal Component Analysis) and compare the performance of several classifiers including Neural Networks (NN), Naive Bayes (NB), K-Nearest Neighbors (KNN), and Support Vector Machines (SVM).

II. Methodology

A. *Data Loading and Preprocessing*

The dataset consists of hand sign images stored in NumPy arrays. We first load the data from the Google Drive, resize the images, and preprocess them for further analysis. Here is the code,

```
X = np.load("/content/drive/MyDrive/data/handsignX.npy")
y = np.load("/content/drive/MyDrive/data/handsigny.npy").ravel()

train_images_resized = []
for img_array in X:
    img = Image.fromarray(img_array.reshape(20, 20))
    img = img.resize((150, 150), Image.ANTIALIAS)  # Resize to 150x150 pixels
    train_images_resized.append(np.array(img).flatten())

X_resized = np.array(train_images_resized)
```

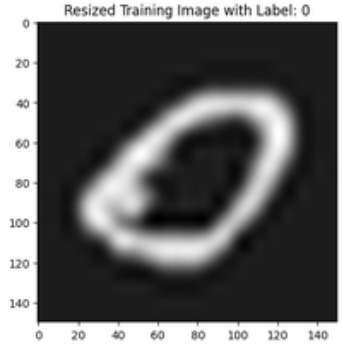

Fig. 1: Here is an image of resized training images.

We also load and preprocess the test images, including inverting their colors.

```
import json
json_path = '/content/drive/MyDrive/Test/Test/dd.json'

with open(json_path) as f:
    data = json.load(f)
test_images = []
test_labels = []
for filename, image_data in data.items():
    img = Image.open("/content/drive/MyDrive/Test/Test/" + image_data["filename"])
    img = img.convert('L').resize((150, 150))
    img = ImageOps.invert(img)  # Invert the colors
    img_array = np.array(img).flatten()
    test_images.append(img_array)
    test_labels.append(int(image_data["regions"][0]["region_attributes"]["shape"]))

X_test = np.array(test_images)
y_test = np.array(test_labels)
```

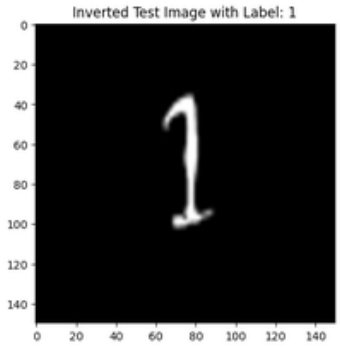

Fig. 2: Here is the photo of an inverted image from a white background to black.

B. Standardization and PCA

The data is standardized to have a mean of zero and a standard deviation of one. PCA is then applied to reduce the dimensionality while retaining 95

```
scaler = StandardScaler()
X_std = scaler.fit_transform(X_resized)
X_test_std = scaler.transform(X_test)

pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_std)
X_test_pca = pca.transform(X_test_std)
```

C. *Model Training and Evaluation*

*1) **Neural Network**:* We train a neural network with and without PCA to evaluate the impact of dimensionality reduction on performance.

```
# NN without PCA
nn_model = Sequential(
    [
        InputLayer((X_resized.shape[1],)),
        Dense(50, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
        Dropout(0.2),
        Dense(25, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
        Dropout(0.2),
        Dense(15, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
        Dropout(0.2),
        Dense(10, activation="linear")
    ]
)

nn_model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                 optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                 metrics=['accuracy'])

nn_model.fit(X_resized, y, epochs=100)

# Test the NN model without PCA
nn_loss, nn_acc = nn_model.evaluate(X_test, y_test)
print("NN Test accuracy without PCA:", nn_acc)

# NN with PCA
nn_model_pca = Sequential(
    [
        InputLayer((X_pca.shape[1],)),
        Dense(50, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
        Dropout(0.2),
        Dense(25, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
        Dropout(0.2),
        Dense(15, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
        Dropout(0.2),
        Dense(10, activation="linear")
    ]
)

nn_model_pca.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy
(from_logits=True),
                     optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                     metrics=['accuracy'])

nn_model_pca.fit(X_pca, y, epochs=100)

# Test the NN model with PCA
nn_loss_pca, nn_acc_pca = nn_model_pca.evaluate(X_test_pca, y_test)
print("NN Test accuracy with PCA:", nn_acc_pca)
```

Here is the loss vs accuracy graph for both With and without PCA.

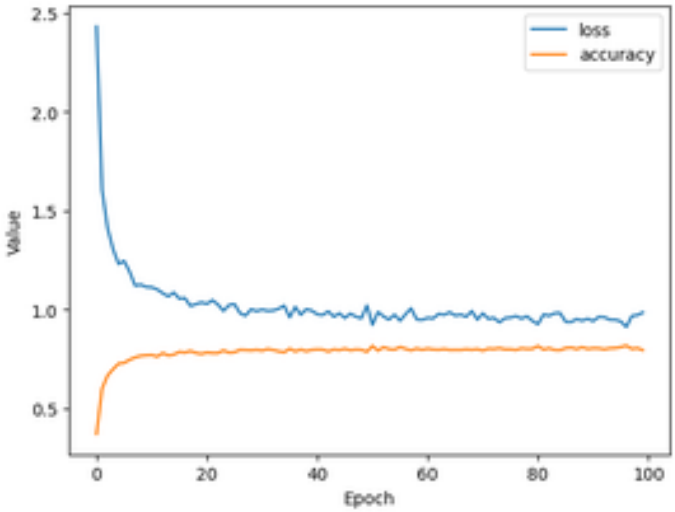

Fig. 3: Without PCA

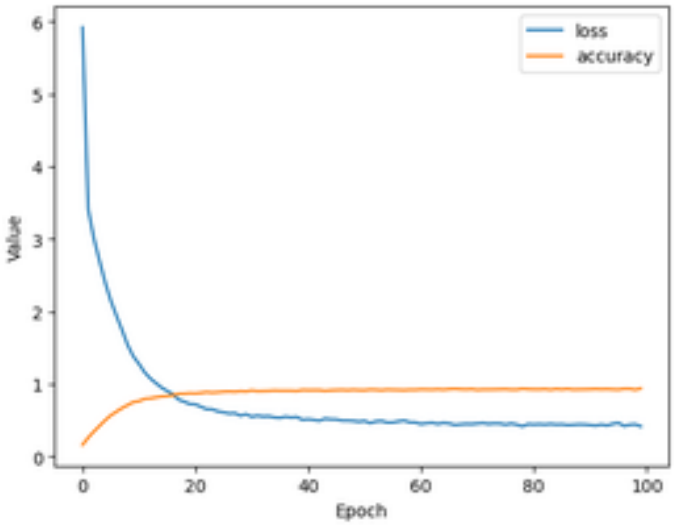

Fig. 4: With PCA

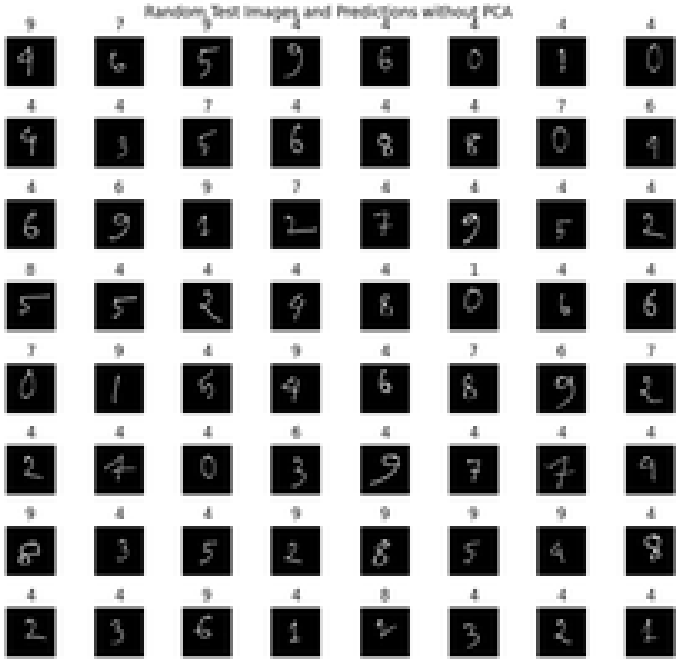

Fig. 5: Here is the output of 64 randomly chosen test images with predictions

*2) **Naive Bayes**:* Naive Bayes classifiers are trained with and without PCA. Here is the code,

```
# NB without PCA
nb_model = GaussianNB()
nb_model.fit(X_std, y)
y_pred_nb = nb_model.predict(X_test_std)
nb_acc = accuracy_score(y_test, y_pred_nb)
print("Naive Bayes Test accuracy without PCA:", nb_acc)

# NB with PCA
nb_model_pca = GaussianNB()
nb_model_pca.fit(X_pca, y)
y_pred_nb_pca = nb_model_pca.predict(X_test_pca)
nb_acc_pca = accuracy_score(y_test, y_pred_nb_pca)
print("Naive Bayes Test accuracy with PCA:", nb_acc_pca)
```

*3) **K-Nearest Neighbors**:* We evaluate K-Nearest Neighbors with and without PCA. Here is the code,

```
# KNN without PCA
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_std, y)
y_pred_knn = knn_model.predict(X_test_std)
knn_acc = accuracy_score(y_test, y_pred_knn)
print("K-Nearest Neighbors Test accuracy without PCA:", knn_acc)

# KNN with PCA
knn_model_pca = KNeighborsClassifier(n_neighbors=5)
knn_model_pca.fit(X_pca, y)
y_pred_knn_pca = knn_model_pca.predict(X_test_pca)
knn_acc_pca = accuracy_score(y_test, y_pred_knn_pca)
print("K-Nearest Neighbors Test accuracy with PCA:", knn_acc_pca)
```

*4) **Support Vector Machine**:* SVM classifiers are trained with and without PCA. Here is the code,

```
# SVM without PCA
svm_model = SVC(kernel='linear')
svm_model.fit(X_std, y)
y_pred_svm = svm_model.predict(X_test_std)
svm_acc = accuracy_score(y_test, y_pred_svm)
print("SVM Test accuracy without PCA:", svm_acc)

# SVM with PCA
svm_model_pca = SVC(kernel='linear')
svm_model_pca.fit(X_pca, y)
y_pred_svm_pca = svm_model_pca.predict(X_test_pca)
svm_acc_pca = accuracy_score(y_test, y_pred_svm_pca)
print("SVM Test accuracy with PCA:", svm_acc_pca)
```

III. Results

The results of the different classifiers with and without PCA are as follows:

Here is the code:

```
# SVM without PCA
print("NN Test accuracy without PCA:", nn_acc)
print("NN Test accuracy with PCA:", nn_acc_pca)
print("Naive Bayes Test accuracy without PCA:", nb_acc)
print("Naive Bayes Test accuracy with PCA:", nb_acc_pca)
print("K-Nearest Neighbors Test accuracy without PCA:", knn_acc)
print("K-Nearest Neighbors Test accuracy with PCA:", knn_acc_pca)
print("SVM Test accuracy without PCA:", svm_acc)
print("SVM Test accuracy with PCA:", svm_acc_pca)
```

Output:

Classifier	Accuracy Without PCA	Accuracy With PCA
Neural Network	0.07	0.10
Naïve Bayes	0.065	0.09
K-Nearest Neighbors	0.075	0.065
Support Vector Machine	0.085	0.105

Additionally, we evaluate the per-class accuracy using the classification report:

Here is the code:

```
print("\nPer-class accuracy for Naive Bayes without PCA:")
print(classification_report(y_test, y_pred_nb))

print("\nPer-class accuracy for Naive Bayes with PCA:")
print(classification_report(y_test, y_pred_nb_pca))

print("\nPer-class accuracy for K-Nearest Neighbors without PCA:")
print(classification_report(y_test, y_pred_knn))

print("\nPer-class accuracy for K-Nearest Neighbors with PCA:")
print(classification_report(y_test, y_pred_knn_pca))

print("\nPer-class accuracy for SVM without PCA:")
print(classification_report(y_test, y_pred_svm))

print("\nPer-class accuracy for SVM with PCA:")
print(classification_report(y_test, y_pred_svm_pca))
```

Here is Output:

Per-class accuracy for Naive Bayes without PCA:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	19
1	0.00	0.00	0.00	20
2	0.00	0.00	0.00	20
3	0.20	0.05	0.08	20
4	0.09	0.24	0.13	21
5	0.00	0.00	0.00	20
6	0.05	0.10	0.06	20
7	0.08	0.05	0.06	20
8	0.20	0.20	0.20	20
9	0.00	0.00	0.00	20
accuracy			0.07	200
macro avg	0.06	0.06	0.05	200
weighted avg	0.06	0.07	0.05	200

Per-class accuracy for Naive Bayes with PCA:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	19
1	0.00	0.00	0.00	20
2	0.10	0.90	0.17	20
3	0.00	0.00	0.00	20
4	0.00	0.00	0.00	21
5	0.00	0.00	0.00	20
6	0.00	0.00	0.00	20
7	0.00	0.00	0.00	20
8	0.00	0.00	0.00	20
9	0.00	0.00	0.00	20
accuracy			0.09	200
macro avg	0.01	0.09	0.02	200
weighted avg	0.01	0.09	0.02	200

Per-class accuracy for K-Nearest Neighbors without PCA:

	precision	recall	f1-score	support
0	0.08	0.16	0.11	19
1	0.00	0.00	0.00	20
2	0.00	0.00	0.00	20
3	0.12	0.05	0.07	20
4	0.10	0.33	0.15	21
5	0.00	0.00	0.00	20
6	0.05	0.05	0.05	20
7	0.11	0.10	0.10	20
8	0.00	0.00	0.00	20
9	0.04	0.05	0.04	20
accuracy			0.07	200
macro avg	0.05	0.07	0.05	200
weighted avg	0.05	0.07	0.05	200

Per-class accuracy for K-Nearest Neighbors with PCA:

	precision	recall	f1-score	support
0	0.08	0.16	0.11	19
1	0.00	0.00	0.00	20
2	0.00	0.00	0.00	20
3	0.17	0.05	0.08	20
4	0.07	0.24	0.10	21
5	0.00	0.00	0.00	20
6	0.05	0.05	0.05	20
7	0.11	0.10	0.11	20
8	0.08	0.05	0.06	20
9	0.00	0.00	0.00	20
accuracy			0.07	200
macro avg	0.06	0.06	0.05	200
weighted avg	0.06	0.07	0.05	200

Per-class accuracy for SVM without PCA:

	precision	recall	f1-score	support
0	0.20	0.05	0.08	19
1	0.00	0.00	0.00	20
2	0.00	0.00	0.00	20
3	0.00	0.00	0.00	20
4	0.10	0.57	0.17	21
5	0.00	0.00	0.00	20
6	0.00	0.00	0.00	20
7	0.09	0.05	0.06	20
8	0.00	0.00	0.00	20
9	0.06	0.15	0.09	20
accuracy			0.09	200
macro avg	0.05	0.08	0.04	200
weighted avg	0.04	0.09	0.04	200

Per-class accuracy for SVM with PCA:

	precision	recall	f1-score	support
0	0.33	0.11	0.16	19
1	0.00	0.00	0.00	20
2	0.00	0.00	0.00	20
3	0.00	0.00	0.00	20
4	0.10	0.71	0.18	21
5	0.00	0.00	0.00	20
6	0.00	0.00	0.00	20
7	0.00	0.00	0.00	20
8	0.00	0.00	0.00	20
9	0.11	0.20	0.15	20
accuracy			0.10	200
macro avg	0.06	0.10	0.05	200
weighted avg	0.05	0.10	0.05	200

IV. Discussion

From the results, we observe that PCA affects the accuracy of different classifiers in different ways. For the Neural Network, accuracy dropped from 0.10 to 0.07, which means PCA might have removed important features needed for learning complex patterns. On the other hand, Naïve Bayes improved from 0.065 to 0.09 with PCA, showing that PCA helped by simplifying the data. K-Nearest Neighbors had a small accuracy drop from 0.075 to 0.065, possibly because PCA changed the data in a way that made it harder for the classifier to work properly. Support Vector Machine accuracy went up from 0.085 to 0.105, meaning PCA made the data easier to separate. These results show that PCA's usefulness depends on the type of classifier and the data used.