Artificial Intelligence

LAB PROJECT SUBMISSION

Submitted to: Ms. Navdeep Kaur

Submitted by: 102103392, 102103816,102103395,102103394

Problem statement:

To recognize the handwritten digits using KNN algorithm.

Description of problem:

Handwritten digit recognition is a challenging field in computer vision and artificial intelligence. Despite significant advancements in this area, current systems still need help recognizing handwritten text accurately and efficiently.

The problem lies in the variability of handwriting styles and the complexity of identifying different shapes and sizes of handwritten numbers. As a result, there is a need to develop a robust system that can accurately recognize handwritten text in various contexts, such as forms, documents, and digital notes.

This project aims to address the problem of handwritten digit recognition by developing an efficient and accurate system using deep learning algorithms and image processing techniques. The ultimate goal is to provide a reliable and user-friendly solution for identifying handwritten digits that can be applied to various real-world applications.

Code and Explanation

PART-I Training our model for K = 3, 5, 7, 9, 11 or any 5 ‘K’ values using KNN classifier on the following training data: Training Dataset to be used: train.csv.

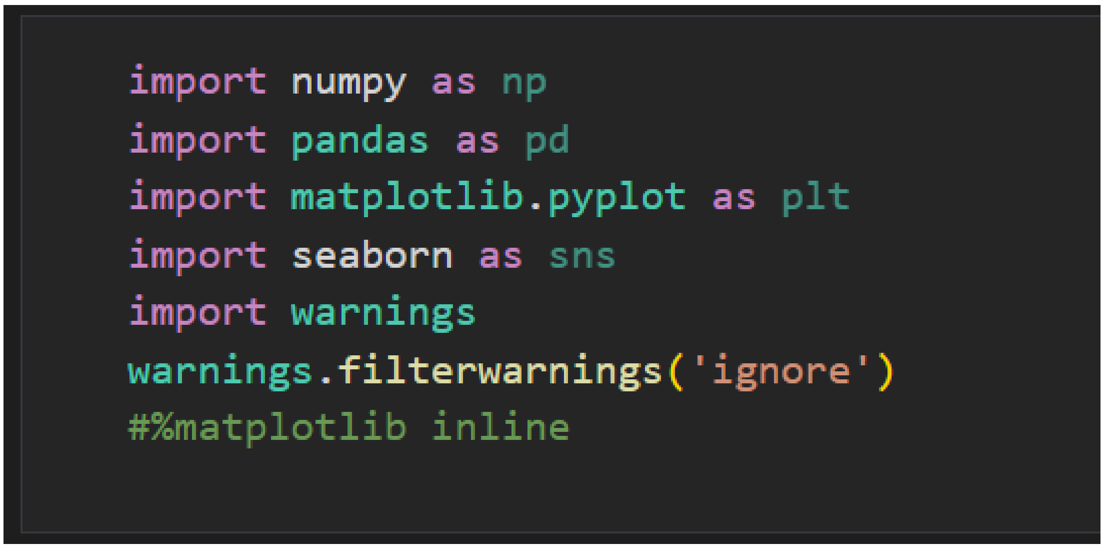
PART-II Testing our trained model on the following data set and creating a confusion matrix for the same: Test Dataset to be used: test.csv Also evaluate the testing performance of your model using precision, recall, and F1 score.

We used sklearn library to implement KNN and evaluate the accuracy, precision, recall, F1 score, and Confusion matrix. We evaluated these using five values for K i.e. 3,5,7,9,11.

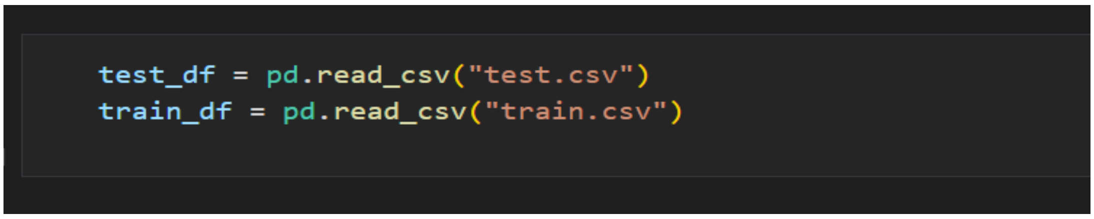
IMPORTING USED MODULES

Brief description of the imported modules:

1. **Numpy:** A Python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices.
2. **Pandas:** APython library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data
3. **Seaborn:** Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
4. **Matplotlib:** Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
#%matplotlib inline
```



```
test_df = pd.read_csv("test.csv")
train_df = pd.read_csv("train.csv")
```

The two lines of code read in data from two separate CSV files ("test.csv" and "train.csv") and store the data as pandas data frames named test_df and train_df, respectively.

The PD.read_csv() function is part of the pandas library in Python and is used to read CSV files into a pandas data frame. The function takes in the file path as its argument and returns a data frame that contains the data from the CSV file.

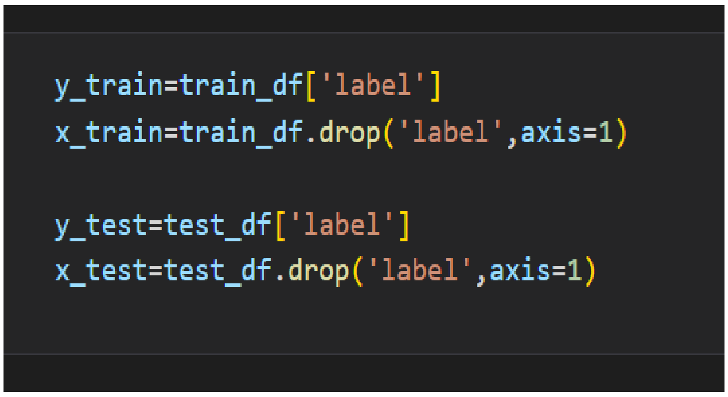
Preparation of the training and testing data for a supervised machine learning model for handwritten text recognition.

The training data is stored in the train_df DataFrame, which has two columns: label and pixels. The label column contains the ground truth label for each image in the training set, while the pixel's column contains the pixel values for each image.

1. The first line of the code extracts the label column from the train_df DataFrame and stores it in the y_train variable.

2. The second line extracts all columns except the label column from the train_df DataFrame and stores them in the x_train variable. This creates two separate variables, one containing the target values and the other containing the input features.

3. Similarly, the third and fourth lines of code extract the label column and the input features from the test_df DataFrame and store them in the y_test and x_test variables, respectively. These variables are used for testing the trained model's performance on unseen data



```
y_train=train_df['label']
x_train=train_df.drop('label',axis=1)

y_test=test_df['label']
x_test=test_df.drop('label',axis=1)
```

Visualizing the image data and verifying that it corresponds to the expected labels.

This code block plots and displays one of the handwritten digits from the training set and prints its corresponding label.

1. The first line sets the size of the figure to be plotted using plt.figure() function with figsize parameter.

2. The second line sets the variable some_digit to the digit index we want to plot. In this case, it's set to 4.

3. The third line selects the some_digit row of the training set using the iloc method

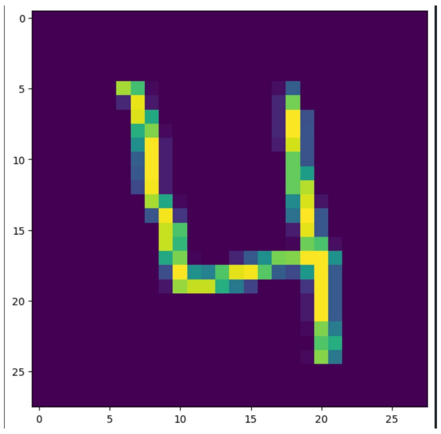
and extracts the pixel's column as a numpy array using the to_numpy() method.

4. The fourth line reshapes the 1D numpy array into a 2D array of shape (28,28) since each image is 28x28 pixels in size.

5. The fifth line plots the image using the imshow() function from matplotlib.

6. The sixth line prints the label of the digit at index some_digit from the y_train variable. In this case, it would print the label corresponding to the handwritten digit image we plotted.

```
plt.figure(figsize=(7,7))
some_digit=4
some_digit_image = x_train.iloc[some_digit].to_numpy()
plt.imshow(np.reshape(some_digit_image, (28,28)))
print(y_train[some_digit])
```

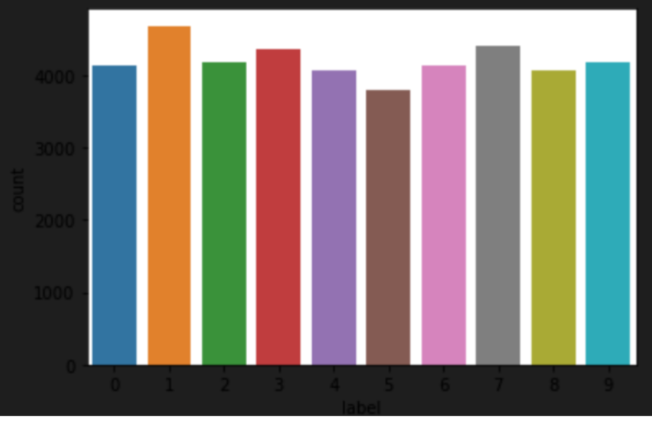


```
sns.countplot( x='label', data=train_df)
```

The code **sns.countplot(x='label', data=train_df)** uses the Python visualization library Seaborn to create a countplot of the label column in the train_df DataFrame.

A countplot is a type of bar plot that shows the count of each category in a categorical variable. In this case, the label is a categorical variable and the countplot will show the frequency of each category in the dataset. The x parameter specifies the column for the x-axis, which is labeled in this case.
Assuming that train_df is a Pandas DataFrame with a label column containing categorical data, this code will generate a bar plot with the count of each category on the y-axis and the category names on the x-axis. The resulting plot can be used to quickly visualize the distribution of labels in the dataset

```
<AxesSubplot:xlabel='label', ylabel='count'>
```


count
4000
3000
2000
1000
0
0
1
2
3
4
5
6
7
8
9
label

Training of a K-nearest neighbors (KNN) classifier on the handwritten digit dataset and Evaluation of its performance on the test set.

1. The first line imports the KNeighborsClassifier class from the sklearn.neighbors module. This class implements the KNN algorithm, which is a simple and effective classification algorithm that classifies data points based on their nearest neighbors.

2. The second line creates an instance of the KNeighborsClassifier class with n_neighbors parameter set to 3 (in the below attached codes). This means the algorithm will consider 3 nearest neighbors to make classification decisions. It can be modified to try different values of n neighbors and compare the performance of the model (like shown in the codes below)

3. The third line fits the classifier to the training data using the fit() method of the KNeighborsClassifier class.
4. The fourth line uses the trained model to make predictions on the test set, and stores the predicted labels in the y_pred variable.
5. The fifth to seventh lines compute and print the accuracy, classification report, and confusion matrix of the KNN classifier.

The accuracy_score() function from sklearn.metrics module calculates the accuracy of the model by comparing the predicted labels (y_pred) to the actual labels in the test set (y_test). The classification_report() function generates a report that includes precision, recall, and F1-score for each class. The confusion_matrix() function computes the confusion matrix, which is a table that shows the true positives, false positives, false negatives, and true negatives for each class

```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3)
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

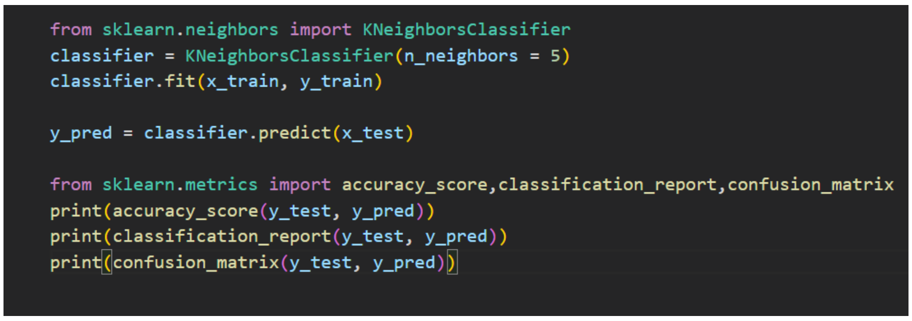
```
0.9774
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      1.00      0.98      1135
           2       0.99      0.97      0.98      1032
           3       0.97      0.98      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.97      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.97      0.97      0.97      1028
           8       1.00      0.95      0.97       974
           9       0.97      0.96      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

[[ 973    1    1    0    0    1    3    1    0    0]
 [   0 1134    1    0    0    0    0    0    0    0]
 [   6    6 1005    1    0    0    1   11    2    0]
 [   1    2    3  989    1    6    0    6    0    2]
 [   0    4    0    0  961    0    2    2    0   13]
 [   3    0    0    8    2  871    2    1    1    4]
 [   5    4    0    0    2    2  945    0    0    0]
 [   0   15    3    0    1    0    0 1000    0    9]
 [   5    2    3   13    5   11    2    4  923    6]
 [   2    6    2    6    8    3    1    8    0  973]]
```

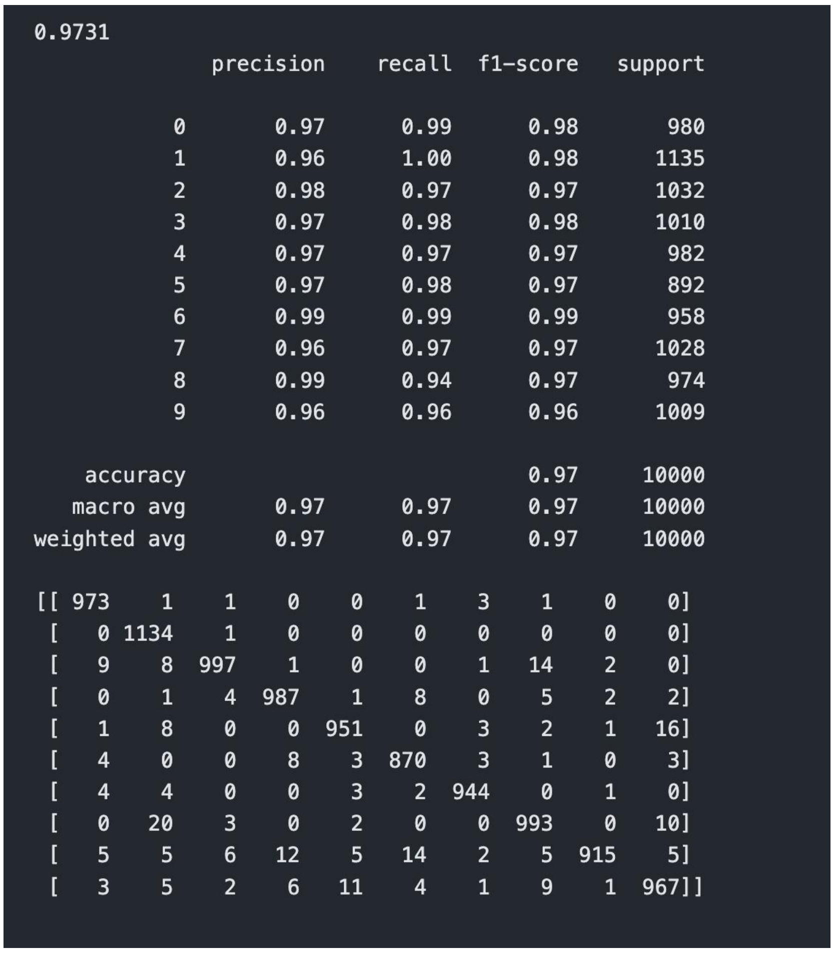
#



```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```



```
0.9731
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.96      1.00      0.98      1135
           2       0.98      0.97      0.97      1032
           3       0.97      0.98      0.98      1010
           4       0.97      0.97      0.97       982
           5       0.97      0.98      0.97       892
           6       0.99      0.99      0.99       958
           7       0.96      0.97      0.97      1028
           8       0.99      0.94      0.97       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

[[ 973    1    1    0    0    1    3    1    0    0]
 [    0 1134    1    0    0    0    0    0    0    0]
 [    9    8  997    1    0    0    1   14    2    0]
 [    0    1    4  987    1    8    0    5    2    2]
 [    1    8    0    0  951    0    3    2    1   16]
 [    4    0    0    8    3  870    3    1    0    3]
 [    4    4    0    0    3    2  944    0    1    0]
 [    0   20    3    0    2    0    0  993    0   10]
 [    5    5    6   12    5   14    2    5  915    5]
 [    3    5    2    6   11    4    1    9    1  967]]
```

```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 7)
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
0.9699
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.95      1.00      0.97      1135
           2       0.98      0.95      0.97      1032
           3       0.97      0.98      0.97      1010
           4       0.98      0.96      0.97       982
           5       0.97      0.98      0.97       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.95      0.97       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

[[ 972    1    1    0    0    2    3    1    0    0]
 [   0 1131    2    1    0    0    1    0    0    0]
 [  13   10  981    2    0    0    2   18    6    0]
 [   0    2    2  985    1    9    0    5    2    4]
 [   1   10    0    0  944    0    4    2    1   20]
 [   4    1    0    6    2  870    4    1    0    4]
 [   4    4    0    0    3    3  944    0    0    0]
 [   0   27    4    0    2    0    0  984    0   11]
 [   5    4    5   11    5    9    2    6  921    6]
 [   3    6    2    6   10    2    1   11    1  967]]
```

#

```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 9)
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
0.9684
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.94      1.00      0.97      1135
           2       0.98      0.95      0.96      1032
           3       0.97      0.97      0.97      1010
           4       0.98      0.96      0.97       982
           5       0.97      0.97      0.97       892
           6       0.98      0.98      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.93      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

[[ 971    1    1    0    0    2    4    1    0    0]
 [    0 1131    2    1    0    0    1    0    0    0]
 [   13   11  979    2    0    0    2   19    6    0]
 [    0    3    2  983    1    9    0    6    3    3]
 [    0   11    0    0  946    0    5    1    1   18]
 [    4    1    0    8    1  869    3    1    0    5]
 [    4    5    0    0    3    3  943    0    0    0]
 [    0   27    5    0    2    0    0  985    0    9]
 [    7    4    6   12    6   12    3    7  910    7]
 [    4    6    3    4   11    1    1   11    1  967]]
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 11)
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```