ELC- Handwritten Text Recognition

Importing libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#%matplotlib inline
```

Loading the MNIST datasets

In [2]:

```
data_df = pd.read_csv("data.csv")
#test_df = pd.read_csv("test.csv")
```

In [3]:

```
data_df.head()
```

Out[3]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [4]:

```
#test_df.head()
```

For train and test both we will use train.csv (Taking train data as complete data)

In [5]:

```
data_df.shape
```

Out[5]:

```
(42000, 785)
```

Data Preparation for Model Building

In [6]:

```
y=data_df['label']
x=data_df.drop('label',axis=1)
```

In [7]:

```
#x_for_test_data=test_df[:]
```

In [8]:

```
type(x)
```

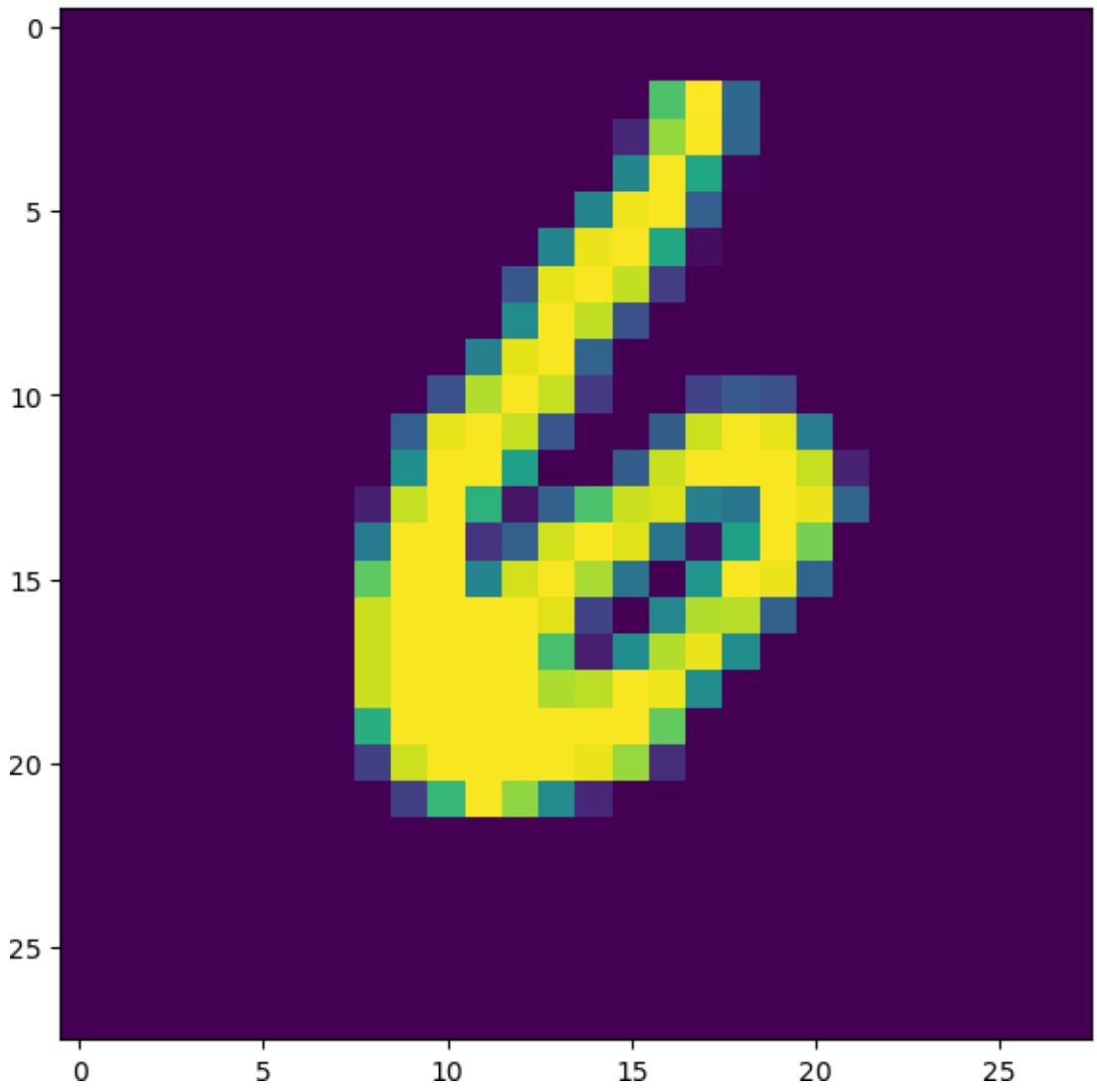
Out[8]:

```
pandas.core.frame.DataFrame
```

In [9]:

```
plt.figure(figsize=(7,7))
some_digit=1266
some_digit_image = x.iloc[some_digit].to_numpy()
plt.imshow(np.reshape(some_digit_image, (28,28)))
print(y[some_digit])
```

```
6
```

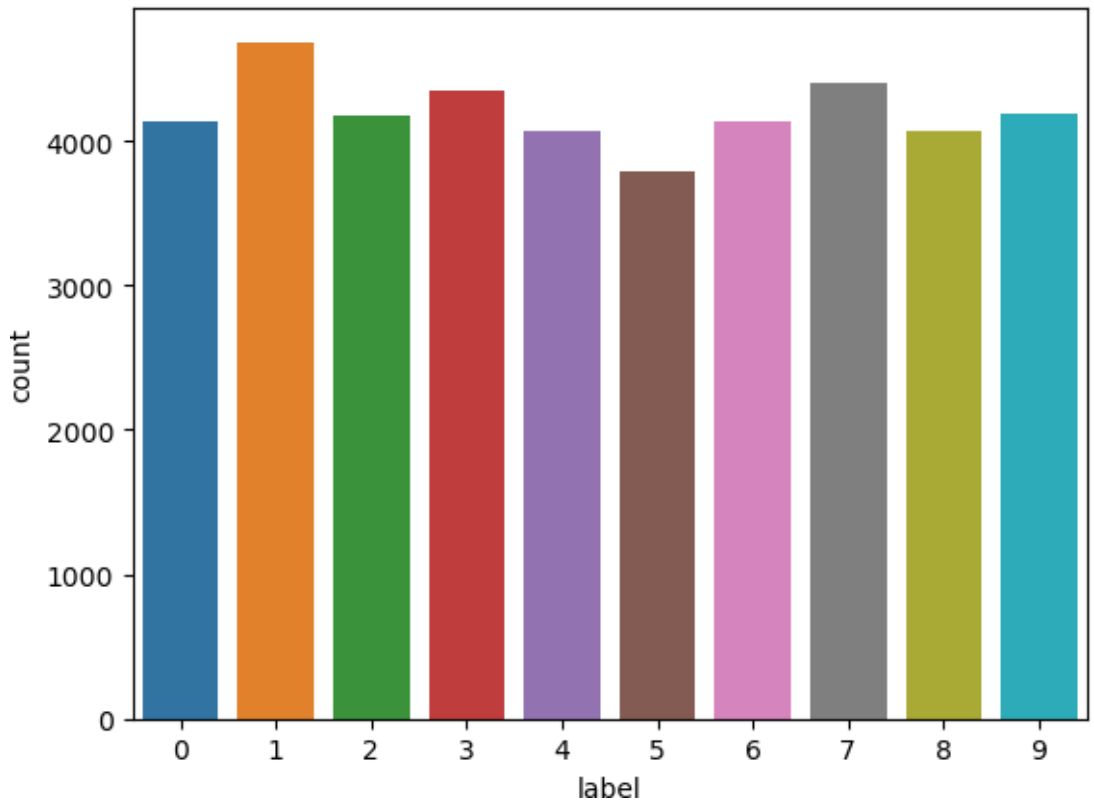


In [10]:

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

Out[10]:

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



we can conclude that our dataset is balanced

Splitting the train data into train and test

In [11]:

```
from sklearn.model_selection import  train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30,
random_state = 40)
```

In [12]:

```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[12]:

```
((29400, 784), (29400,), (12600, 784), (12600,))
```

Models

KNN

In [13]:

```
#from sklearn.preprocessing import StandardScaler
#scaler = StandardScaler()
#scaler.fit(x_train,y_train)
#x_train = scaler.transform(x_train)
#x_train.shape
```

k=3

In [14]:

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

Out[14]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [15]:

```
y_pred = classifier.predict(x_test)
y_pred
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classificatio
n.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `ku
rtosis`), the default behavior of `mode` typically preserves the axis it ac
ts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken
will be eliminated, and the value None will no longer be accepted. Set `kee
pdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[15]:

```
array([0, 2, 1, ..., 2, 4, 7], dtype=int64)
```

In [16]:

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

```
0.9636507936507936
```

In [17]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1236
           1       0.96      1.00      0.98      1370
           2       0.98      0.96      0.97      1252
           3       0.95      0.96      0.95      1369
           4       0.97      0.96      0.97      1215
           5       0.95      0.95      0.95      1132
           6       0.97      0.99      0.98      1216
           7       0.96      0.96      0.96      1326
           8       0.98      0.92      0.95      1197
           9       0.94      0.94      0.94      1287

    accuracy                           0.96     12600
   macro avg       0.96      0.96      0.96     12600
weighted avg       0.96      0.96      0.96     12600
```

In [18]:

```
print(confusion_matrix(y_test, y_pred))
```

```
[[1224    0    2    0    0    1    6    0    1    2]
 [   0 1364    0    0    0    0    2    2    1    1]
 [   5   10 1204    6    1    1    2   18    2    3]
 [   3    4    6 1315    0   22    1    7    9    2]
 [   2   12    1    0 1165    0    5    1    0   29]
```

```
[   3    1    1   27    2 1075   16    0    2    5]
[  10    1    0    0    1    3 1201    0    0    0]
[   1   17    5    0    0    0    0 1278    1   24]
[   5    8    7   25   11   21    2    4 1102   12]
[   7    5    1   12   18    6    0   21    3 1214]]
```

In [19]:

```
#y_pred_on_test_data = classifier.predict(x_for_test_data)
#y_pred_on_test_data
```

Working

This Python script trains and evaluates a K-Nearest Neighbours (KNN) classifier using a training dataset and a test dataset.

1. The script starts by importing required libraries like Numpy, Pandas, Matplotlib, and Scikit-Learn.

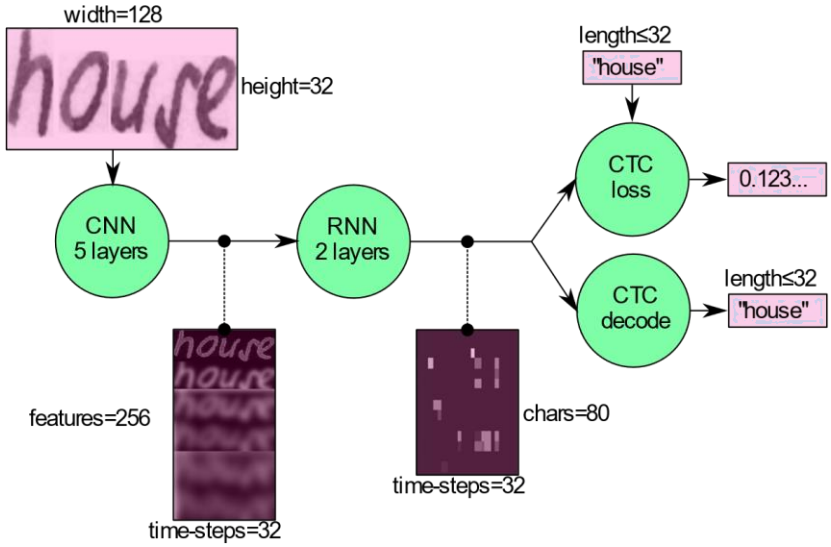
2. It then loads the training and test data into Pandas dataframes and splits the training data into features and labels.

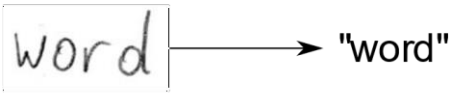
3. The next step is to further split the training data into train and validation sets, with 20% of the data reserved for validation.

4. Using the training data, a KNN classifier model is created with 5 neighbours and then trained using the fit() method.

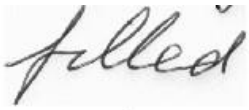
5. The script then makes predictions on the validation data, and uses the confusion_matrix(), accuracy_score(), and f1_score() functions to calculate the confusion matrix, accuracy, and F1 score.

6. Finally, the script makes predictions on the test data and saves the predictions to a CSV file named "y_test_pred.csv".





Best path decoding	"fuleid" ✗
Vanilla beam search	"fuleid" ✗
Word beam search	"filled" ✓



Best path decoding	"fuleid" ✗
Vanilla beam search	"fuleid" ✗
Word beam search	"filled" ✓