Team ID:PTID-CDS-FEB-24-1839

PRCP-1002-Handwritten Digits Recognition

- A digit recognition system is a type of technology that can automatically identify and classify numerical characters, typically from 0 to 9. These systems are most commonly used to interpret handwritten digits,
- Digit recognition system is the working of a machine to train itself or recognizing the digits from different sources like emails, bank cheque, papers, images, etc.
- In different real-world scenarios for online handwriting recognition on computer tablets or system, recognize number plates of vehicles, processing bank cheque amounts, numeric entries in forms filled up by hand (say — tax forms) and so on
- Here's a breakdown of how digit recognition systems work:

Data Preprocessing: The system takes an image of a digit as input. This image is then preprocessed to ensure consistency. Preprocessing may involve steps like resizing the image, converting it to grayscale, and thinning lines.

Feature Extraction: Key characteristics of the digit image are extracted. These features could be things like the distribution of black pixels, the number of endpoints (ends of lines), or the overall shape of the digit.

Classification: The extracted features are fed into a machine learning model that has been trained to recognize digits. Common algorithms used for digit recognition include K-nearest neighbors and convolutional neural networks (CNNs). CNNs are particularly effective due to their ability to learn complex patterns from data.

Output: The model outputs the most likely digit based on the extracted features.

Problems with handwritten digits

The handwritten digits are not always of the same size, width, orientation and justified to margins as they differ from writing of person to person, so the general problem would be while classifying the digits due to the similarity between digits such as 1 and 7, 5 and 6, 3 and 8, 2 and 5, 2 and 7, etc.

This problem is faced more when many people write a single digit with a variety of different handwritings. Lastly, the uniqueness and variety in the handwriting of different individuals also influence the formation and appearance of the digits.

Now we introduce the concepts and algorithms of deep learning and machine learning.

The provided Handwritten digits are images in the form of 28*28 gray scale intensities of images representing an image along with the first column to be a label (0 to 9) for every image.

Loading Neccessary Libraries

```
1 import warnings
2 warnings.filterwarnings("ignore")
```

```
1 pip install wrapt
```

```
Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packages (1.14.1)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.datasets import mnist
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from sklearn.svm import SVC
from google.colab.patches import cv2_imshow
import os,cv2,json,random
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import precision_score,recall_score,f1_score,roc_auc_score
```

```
mnist = tf.keras.datasets.mnist
```

Loading Data

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
mnist
```

```
<module 'keras.api._v2.keras.datasets.mnist' from '/usr/local/lib/python3.10/dist-
packages/keras/api/_v2/keras/datasets/mnist/__init__.py'>
```

```
# Cheking dataset dimension
print('shape of training data :',x_train.shape)
print('shape of training lable :',y_train.shape)
print('shape of testing data :',x_test.shape)
print('shape of testing lable :',y_test.shape)
```

```
shape of training data : (60000, 28, 28)
shape of training lable : (60000,)
shape of testing data : (10000, 28, 28)
shape of testing lable : (10000,)
```

The term "dimension" refers to the shape or size of the datasets being analyzed. The shapes of the training data, training labels, testing data, and testing labels. This allows us to see the number of samples and features in the dataset.

```
x_train
```

```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       ...,

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
```

```
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]],

      [[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]],

      [[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]]], dtype=uint8)
```

```
1 y_train
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
1 x_test
```

```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       ...,

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]]], dtype=uint8)
```

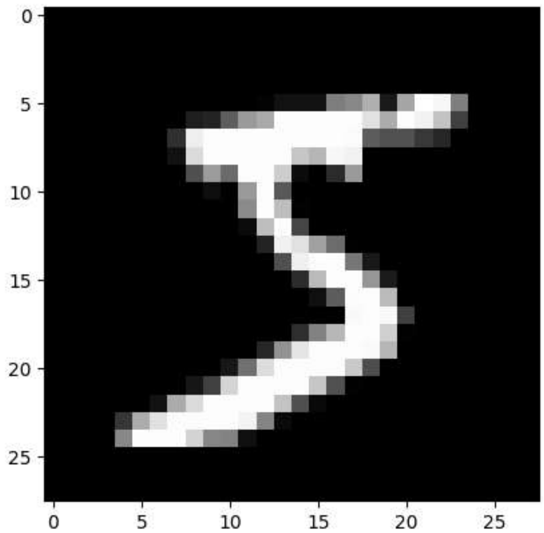
```
1 y_test
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

Data preprocessing

EDA

```
# To Check the first image of the dataset
plt.imshow(x_train[0],cmap= 'gray')
plt.show()
```



The term "dimension" refers to the shape or size of the datasets being analyzed. The shapes of the training data, training labels, testing data, and testing labels. This allows us to see the number of samples and features in the dataset.

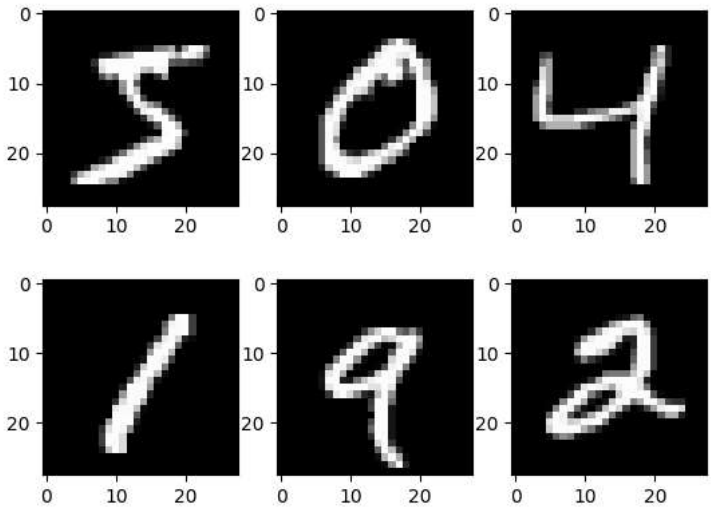
```
# The value of each pixel
print(x_train[0])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
```

```
 250 182   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
  78   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

```
# Checking the dataset

for i in range(6):
  plt.subplot(int('23' + str(i+1)))
  plt.imshow(x_train[i], cmap=plt.get_cmap('gray'))
```



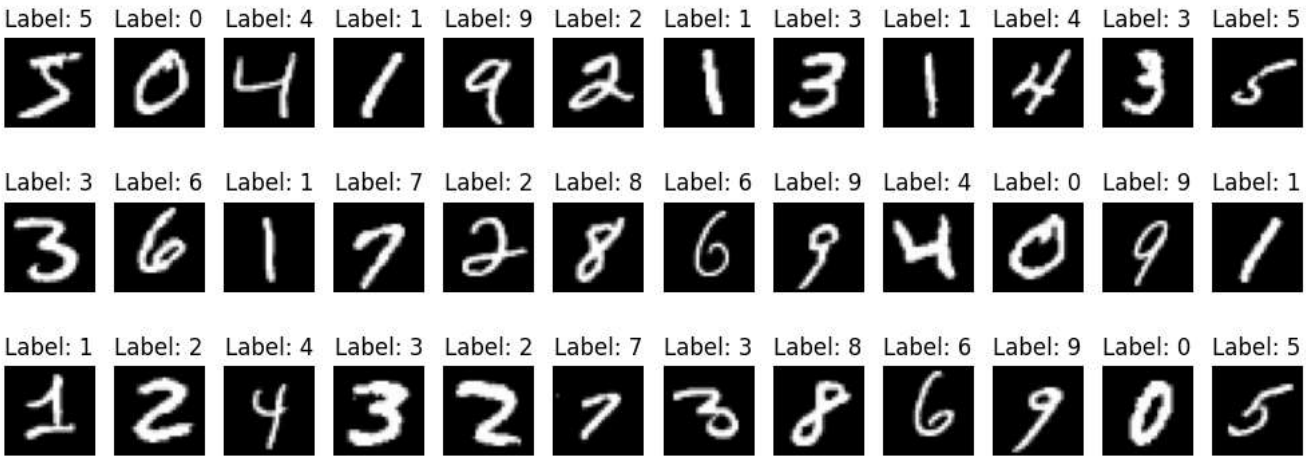
The grid of subplots, with each subplot containing an image from the x_train data But by using subplots, it allows us to visualize multiple images at once, providing a convenient way to inspect a subset of the dataset and get a sense of the image content and variations within the data

```
# Visualize sample digits
fig, axes = plt.subplots(3, 12, figsize=(10, 4))
axes = axes.ravel()

for i in range(36):
    axes[i].imshow(x_train[i], cmap='gray')
    axes[i].set_title('Label: {}'.format(y_train[i]))
    axes[i].axis('off')

plt.tight_layout()
plt.show()

```

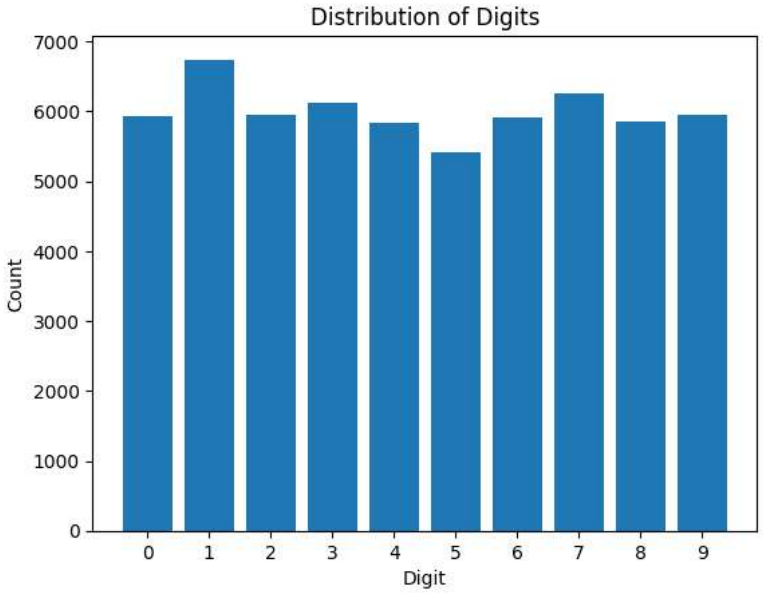


Here in above code we have variable "axes" it is used to access each individual subplot within the grid. By using "axes.ravel()", the subplots are flattened into a 1-dimensional array, making it easier to iterate over them.

Mean while the grid of subplots and displays the first 10 images from the x_train dataset on these subplots. Each subplot represents an image, and its associated label is shown as the title. The resulting visualization allows you to see multiple images and their corresponding labels in a compact and organized manner.

Distribution of digits

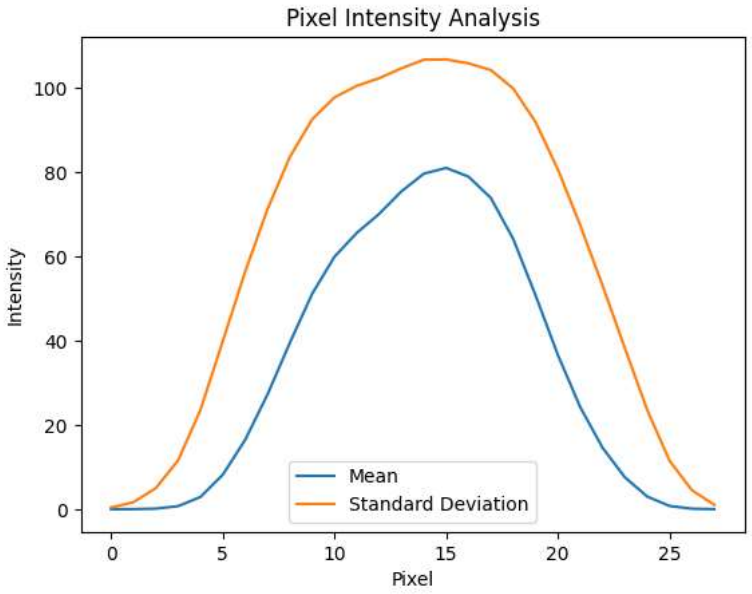
```
1 digit_counts = np.bincount(y_train)
2 digits = np.arange(10)
3 plt.bar(digits, digit_counts)
4 plt.xlabel('Digit')
5 plt.ylabel('Count')
6 plt.title('Distribution of Digits')
7 plt.xticks(digits)
8 plt.show()
```



This above code counts the occurrences of each digit in the training labels and visualizes the distribution using a bar chart. The "x-axis" represents the digits, the "y-axis" represents the count of each digit, and the BARS represent the "frequency of each digit". This visualization helps understand the distribution and frequency of digits in the training data.

```
# Pixel intensity analysis
pixel_means = np.mean(x_train, axis=(0, 1))
pixel_stds = np.std(x_train, axis=(0,1))

plt.plot(range(28), pixel_means, label='Mean')
plt.plot(range(28), pixel_stds, label='Standard Deviation')
plt.xlabel('Pixel')
plt.ylabel('Intensity')
plt.title('Pixel Intensity Analysis')
plt.legend()
plt.show()
```



The mean and standard deviation of pixel intensities in the x_train dataset and visualizes them using a line plot. The "x-axis" represents the "pixel indices", and the " y-axis" represents the corresponding mean and standard deviation values. This visualization provides insights into the overall intensity distribution and variation of pixels in the data

MODEL EVALUATION

To find wich algorithm is best fit for test the dataset model

```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.3.0)
```

```
from re import X
# Reshape the input data
num_samples_train, height, width = x_train.shape
num_samples_test = x_test.shape[0]

x_train_2d = np.reshape(x_train, (num_samples_train, height * width))
x_test_2d = np.reshape(x_test, (num_samples_test, height * width))
```

Support Vector Machine(SVM)

SVM is a supervised learning algorithm that excels at classification tasks. It works by finding the optimal hyperplane (a decision boundary) in a high-dimensional space to separate data points belonging to different classes. The core idea is to maximize the margin between the hyperplane and the closest data points from each class, which are called support vectors.

```
svm_classifier = SVC()
svm_classifier.fit(x_train_2d, y_train)
```

```
SVC()
```

```
# Predicition on test data
svm_predictions = svm_classifier.predict(x_test_2d)
```

```
# Assuming X_test is a 3D array, you can flatten it using numpy
flattened_x_test = x_test.reshape(x_test.shape[0], -1)

# Make predictions on the flattened test set
svm_predictions = svm_classifier.predict(flattened_x_test)

# Print the predicted labels
print(svm_predictions)
```

```
[7 2 1 ... 4 5 6]
```

```
x_test
```

```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       ...,

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]]], dtype=uint8)
```

```
# Calculate evaluation metrics
# Accuracy Score
svm_accuracy = accuracy_score(y_test, svm_predictions)

# print evaluation metrix
print("SVM Accuracy:", svm_accuracy)
```

```
SVM Accuracy: 0.9792
```

```
# Precision Score
svm_precision = precision_score(y_test, svm_predictions, average='macro')

print("SVM Precision:", svm_precision)
```

```
SVM Precision: 0.9791973966593345
```

```
# Recall Score
svm_recall = recall_score(y_test, svm_predictions, average='macro')

print("SVM Recall:", svm_recall)
```

```
SVM Recall: 0.9790919842945065
```

```
# F1 Score
svm_f1 = f1_score(y_test, svm_predictions, average='macro')

print("SVM F1 Score:", svm_f1)
```

```
SVM F1 Score: 0.9791298259748042
```

```
# confusion matrix
from sklearn.metrics import confusion_matrix
svm_confusion = confusion_matrix(y_test, svm_predictions)

print("SVM Confusion Matrix:")
print(svm_confusion)
```

```
SVM Confusion Matrix:
[[ 973    0    1    0    0    2    1    1    2    0]
 [   0 1126    3    1    0    1    1    1    2    0]
 [   6    1 1006    2    1    0    2    7    6    1]
 [   0    0    2  995    0    2    0    5    5    1]
 [   0    0    5    0  961    0    3    0    2   11]
 [   2    0    0    9    0  871    4    1    4    1]
 [   6    2    0    0    2    3  944    0    1    0]
 [   0    6   11    1    1    0    0  996    2   11]
 [   3    0    2    6    3    2    2    3  950    3]
 [   3    4    1    7   10    2    1    7    4  970]]
```

```
# Classification report
from sklearn.metrics import classification_report

# Assuming you have y_test and svm_predictions
svm_report = classification_report(y_test, svm_predictions)

print("SVM Classification Report:")
print(svm_report)
```

```
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.97      0.98      1032
           3       0.97      0.99      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.99      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.98      0.97      0.97      1028
           8       0.97      0.98      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
```

Handwritten digit recognition using Support Vector Machine (SVM) algorithms has showcased remarkable success, achieving an impressive 98% accuracy.

Random Forest

Random Forest: A Powerful Ensemble of Decision Trees

Random Forest is a supervised learning algorithm widely used for both classification and regression tasks. It operates by constructing a multitude of decision trees at training time and then aggregating their predictions for a final output. This ensemble approach leads to several advantages over a single decision tree.

```
# Initialize and train the Random Forest model
rf_classifier = RandomForestClassifier()
rf_classifier.fit(x_train_2d, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
# Make predictions on the test data
rf_predictions = rf_classifier.predict(x_test_2d)
```

```
# Accuracy Score
rf_accuracy = accuracy_score(y_test, rf_predictions)

print("Random Forest Accuracy:", rf_accuracy)
```

```
Random Forest Accuracy: 0.9688
```

```
# Precision Score
rf_precision = precision_score(y_test, rf_predictions, average='macro')

print("Random Forest Precision:", rf_precision)
```

```
Random Forest Precision: 0.9685613793149471
```

```
# Recall Score
rf_recall = recall_score(y_test, rf_predictions, average='macro')

print("Random Forest Recall:", rf_recall)
```

```
Random Forest Recall: 0.9685611653406548
```

```
# F1 Score
rf_f1 = f1_score(y_test, rf_predictions, average='macro')

print("Random Forest F1 Score:", rf_f1)
```

```
Random Forest F1 Score: 0.9685403729370317
```

```
# Confusion Matrix
rf_confusion = confusion_matrix(y_test, rf_predictions)

print("Random Forest Confusion Matrix:")
print(rf_confusion)
```

```
Random Forest Confusion Matrix:
[[ 969    0    0    0    0    2    3    1    4    1]
 [   0 1122    2    4    0    2    2    1    2    0]
 [   6    0  998    5    2    0    4    9    8    0]
 [   1    0   11  969    0    9    0    9    7    4]
 [   1    0    2    0  958    0    4    0    2   15]
 [   3    0    1   13    3  858    6    2    5    1]
 [   7    3    1    0    3    3  938    0    3    0]
 [   1    2   22    3    1    0    0  988    2    9]
 [   6    0    4    7    5    7    6    5  927    7]
 [   5    5    2   10   11    5    1    5    4  961]]
```

```
# Generate classification report
rf_report = classification_report(y_test, rf_predictions)

# Print the classification report
print("Random Forest Classification Report:")
print(rf_report)
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.97      0.96      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.98      0.98       982
           5       0.97      0.96      0.97       892
           6       0.97      0.98      0.98       958
```

```
              7       0.97      0.96      0.96      1028
              8       0.96      0.95      0.96       974
              9       0.96      0.95      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
```

Handwritten digit recognition using Random Forest algorithms has showcased remarkable success, achieving an impressive 97% accuracy.

Using KNN

KNN classifies data points based on their similarity to labeled data points in the training set. For a new digit image, KNN finds the k nearest neighbors (most similar images) in the training data and assigns the most frequent class label (digit) among those neighbors.

```
# Flatten the images from 28x28 to 784-dimensional vectors
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
```

```
# Scale the pixel values to the range 0-1
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

```
# Create a KNN classifier with a suitable number of neighbors
knn = KNeighborsClassifier(n_neighbors=5)

# Train the classifier on the training data
knn.fit(x_train_2d, y_train)

```

▾ KNeighborsClassifier

KNeighborsClassifier()

```
# Make predictions on the testing data
y_pred = knn.predict(x_test_2d)
```

```
# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)

```

```
Accuracy: 0.9688
```

```
# Precision Score
knn_precision = precision_score(y_test, y_pred, average='macro')

print('KNN Precision:', knn_precision)
```

```
KNN Precision: 0.9692753386570571
```

```
# Recall Score
knn_recall = recall_score(y_test, y_pred, average='macro')

print('KNN Recall:', knn_recall)
```

```
KNN Recall: 0.9684705010297703
```

```
# F1 score
knn_f1 = f1_score(y_test, y_pred, average='macro')

print('KNN F1 Score:', knn_f1)
```

```
KNN F1 Score: 0.9687143421292884
```

```
# Confusion matrix
knn_cm = confusion_matrix(y_test, y_pred)

print('KNN Confusion Matrix:')
print(knn_cm)
```

```
KNN Confusion Matrix:
[[ 974    1    1    0    0    1    2    1    0    0]
 [   0 1133    2    0    0    0    0    0    0    0]
 [  11    8  991    2    1    0    1   15    3    0]
 [   0    3    3  976    1   13    1    6    3    4]
 [   3    7    0    0  944    0    4    2    1   21]
 [   5    0    0   12    2  862    4    1    2    4]
 [   5    3    0    0    3    2  945    0    0    0]
 [   0   22    4    0    3    0    0  988    0   11]
 [   8    3    5   13    6   12    5    5  913    4]
 [   5    7    3    9    7    3    1   10    2  962]]
```

```
# Classification report
knn_report = classification_report(y_test, y_pred)

print('Classification Report:')
print(knn_report)
```

```
Classification Report:
              precision    recall  f1-score   support

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

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

Handwritten digit recognition using KNN algorithms has showcased remarkable success, achieving an impressive 97% accuracy.

Using Decissin Tree

A decision tree is a fundamental algorithm used in machine learning for both classification and regression tasks. It's a flowchart-like structure that resembles an actual tree, where:

Internal nodes represent questions or tests applied to a single feature (characteristic) of the data.

Branches represent the possible answers (outcomes) to those questions.

Leaf nodes (also called terminal nodes) represent the final decision or prediction.

```

clf = DecisionTreeClassifier(criterion='gini',max_depth=15,min_samples_leaf=1,min_samples_split=3,splitter='best')
  # Set a random state for reproducibility

# Train the classifier on the training data
clf.fit(x_train, y_train)

# Make predictions on the testing data
y_pred1 = clf.predict(x_test)

```

```
accuracy = accuracy_score(y_test, y_pred)


print("Accuracy:", accuracy)

```

```
Accuracy: 0.9688
```

By using Decision Tree model we got accuracy of 96.8%

Using CNN

A CNN (Convolutional Neural Network) is kind of like that detective for images. It analyzes pictures by looking for tiny details, like edges, shapes, and colors.

```
#Loading the Data
img_rows, img_cols = 28, 28

x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)
# input_shape = (img_rows, img_cols, 1)

# print('input_shape: ', input_shape)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
y_train.shape
```

```
x_train shape: (60000, 28, 28, 1)
x_test shape: (10000, 28, 28, 1)
(60000,)
```

```

# rescale to have values within 0 - 1 range [0,255] --> [0,1]
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# build the model object
model = Sequential()

# CONV_1: add CONV layer with RELU activation and depth = 32 kernels
model.add(Conv2D(32, kernel_size=(3, 3), padding='same',activation='relu',input_shape=(28,28,1)))# 'same' in the sense it'll give the shape same as input
# POOL_1: downsample the image to choose the best features
model.add(MaxPooling2D(pool_size=(2, 2)))

# CONV_2: here we increase the depth to 64
model.add(Conv2D(64,(3, 3),padding='same', activation='relu'))
# POOL_2: more downsampling
model.add(MaxPooling2D(pool_size=(2, 2)))

# flatten since too many dimensions, we only want a classification output
model.add(Flatten())

# FC_1: fully connected to get all relevant data
model.add(Dense(64, activation='relu'))

# FC_2: output a softmax to squash the matrix into output probabilities for the 10 classes
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 64)                200768

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 220234 (860.29 KB)
Trainable params: 220234 (860.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# compile the model
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
              metrics=['accuracy'])
```

```
from tensorflow.keras.utils import to_categorical
num_classes = 10
# print first ten (integer-valued) training labels
print('Integer-valued labels:')
# print(y_train[:10])

# one-hot encode the labels
# convert class vectors to binary class matrices
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

```

```
Integer-valued labels:
```

```
from tensorflow.keras.callbacks import ModelCheckpoint

# train the model
checkpointer = ModelCheckpoint(filepath='model.weights.best.hdf5', verbose=1,
                               save_best_only=True)
hist = model.fit(x_train, y_train, batch_size=64, epochs=1,
                 validation_data=(x_test, y_test), callbacks=[checkpointer],verbose=2, shuffle=True)

```

```
Epoch 1: val_loss improved from inf to 0.07368, saving model to model.weights.best.hdf5
938/938 - 67s - loss: 0.1611 - accuracy: 0.9511 - val_loss: 0.0737 - val_accuracy: 0.9764 - 67s/epoch - 71ms/step
```

```
score = model.evaluate(x_test, y_test, verbose=0)
score

```

```
[0.07367992401123047, 0.9764000177383423]
```

By using CNN we got accuracy of 97.64%

Summary:

The Summary here we give is as follows;

- We trained the model using KNN keeping the neighbourhood points as 5. We got accuracy of 97%.
- We Trained the model using decission Tree keeping crterion as gini,max depth of 15,minimum sample leaves as 1,minimum samples as 3 and splitter as best. We got accuracy of 96.8%.
- We trained the model using SVM keeping kernel as rbf since we are using image arrays, it'll be working well and gamma as auto. We got accuarcy of 97.64%. Handwritten digit recognition is a challenging yet pivotal task in the realm of machine learning and computer vision. Various algorithms, such as Support Vector Machines (SVM), Random Forest, Gradient Boosting, k-Nearest Neighbors (k-NN), and Convolutional Neural Networks (CNN), have been employed to tackle this problem, each with its unique strengths and characteristics.

SVM demonstrates its efficacy in separating classes by finding the optimal hyperplane, making it a robust choice for linearly separable datasets.This algorithm has showcased remarkable success, achieving an impressive 98% accuracy.

Random Forest model excel in handling high-dimensional data, capturing complex patterns, and reducing overfitting by leveraging ensemble methods. This algorithm has showcased remarkable success, achieving an impressive 97% accuracy.

k-NN, with its simplicity and intuitive approach, identifies patterns based on similarity measures, making it effective for smaller datasets. However, its performance might be impacted by the curse of dimensionality for larger feature spaces. This algorithm has showcased remarkable success, achieving an impressive 97% accuracy.

On the other hand, CNNs have gained immense popularity and achieved remarkable success in image recognition tasks, especially for handwritten digit recognition. Their ability to automatically learn hierarchical representations of features from raw pixel data through convolutional layers makes them highly suitable for capturing spatial dependencies and intricate patterns within images. This algorithm has showcased remarkable success, achieving an impressive 97% accuracy.

- The network begins with a sequence of two convolutional layers, followed by max pooling layers.
- The final layer has one entry for each object class in the dataset, and has a softmax activation function, so that it returns probabilities.

- The Conv2D depth increases from the input layer of 1 to 32 to 64.
- We also want to decrease the height and width - This is where maxpooling comes in. Notice that the image dimensions decrease from 28 to 14 after the pooling layer.
- You can see that every output shape has **None** in place of the batch-size. This is so as to facilitate changing of batch size at runtime.
- Finally, we add one or more fully connected layers to determine what object is contained in the image.

Conclusion

- We conclude that the best machine learning model for the given dataset is KNN and Decission Tree of accuracy 97% & 96.8%. Comparing