INNOMATICS®
RESEARCH LABS

INNOVATION. AUTOMATION. ANALYTICS

MNIST HANDWRITTEN DIGIT RECOGNITION USING ML

Presented by
Nithya Santhoshini K
Batch - 263

ABOUT ME

Background:

- I am a recent B.Tech graduate in Computer Science with a strong interest in data analytics, software development, and cloud engineering.

Work Experience:

- Currently, I am gaining practical exposure at Innomatics Research Labs, where I am working on various data science and Deep Learning projects.

Career Goals:

- I am eager to further my career in Data Science and make meaningful contributions by leveraging my analytical skills, technical knowledge, and passion for solving complex problems.

TABLE OF CONTENT

INTRODUCTION

- The aim of this project is to implement a classification algorithm to recognize handwritten digits (0- 9).
- In pattern recognition, it is well established that no single classifier consistently outperforms others across all classification problems.
- The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.

OBJECTIVE

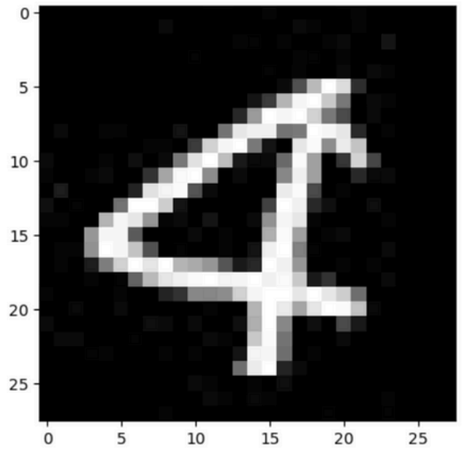
The objective of this project is to accurately recognize handwritten digits using the MNIST dataset by:

1. Importing and preprocessing the image data.
2. Extracting relevant features.
3. Implementing and evaluating the Machine Learning models
4. Assessing the classifier's performance.
5. Utilizing the model for accurate digit prediction

SAMPLE IMAGE OVERVIEW

```
In [3]: img
Out[3]: <image>

In [4]: img.mode
Out[4]: 'L'

In [5]: img.size
Out[5]: (28, 28)

In [6]: # Read all the images from each folder and convert into below forms:
        # -- numpy array
        # -- dataframe

In [7]: img_array = np.asarray(img)

In [8]: img_array
Out[8]: array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   4,
                  0,   1,   8,   0,   0,   2,   0,   3,   7,   0,   2,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,  11,   3,   0,   0,   0,
                  0,   0,   0,   0,  11,   8,   0,   0,   6,   0,   0,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   3,
                  6,   5,   1,   6,   0,   0,   6,  11,   0,   0,  24,   0,   0,
                  0,   0],
               [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   5,   0,   0,
```

```
[10]: <matplotlib.image.AxesImage at 0x16277fbd0>
```




```
[11]: pd.DataFrame(img_array)
```

[11]:

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	0	...	2	0	3	7	0	2	0	0	0	0
1	0	0	0	0	0	0	0	0	11	3	...	8	0	0	6	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	6	11	0	0	24	0	0	0	0
3	0	0	0	0	0	0	0	0	0	1	...	5	0	8	2	2	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	3	...	0	0	11	0	11	6	0	0	0	0
5	0	0	0	0	0	0	0	0	0	2	0	...	188	255	222	48	0	0	0	0	0	0

MODEL BUILDING & PROCESS

Step 1: Convert the image into Gray level image or Binary image

Step 2: Preprocessing the Gray Scale/Binary Image

Step 3: Convert the Gray Scale/Binary Image into a single Dimensional Array of [1,n]

Step 4: Keep the label of each Array along with it.

Step 5: Feed the classifier with the train_data set.

Step 6: Repeat the steps from 1 to 5 for all images in the Sample and Test Database.

Step 7: Feed the classifier with test_data set.

Step 8: Classify the input images into appropriate class label using minimum distance K-nearest neighbor classifier.

Step9: Prediction & End.

DATASET OVERVIEW

26]: X

26]:

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
0	0	0	0	0	0	0	0	0	7	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	3	7	1	0	0	3	7	0	...	9	0	0	1	9	10	0	0	0	0
2	0	7	0	0	2	0	0	0	4	0	...	0	0	0	12	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	7	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	2	0	...	2	0	0	4	3	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
41995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
41996	0	0	0	0	0	0	0	0	0	0	...	6	6	2	0	0	2	0	0	0	0
41997	0	0	0	0	0	0	0	0	3	7	...	0	0	0	0	0	0	0	0	0	0
41998	0	0	0	0	0	0	0	0	0	6	...	0	0	0	0	0	0	0	0	0	0
41999	0	0	0	0	0	0	0	0	5	0	...	0	1	0	0	0	0	0	0	0	0

42000 rows × 784 columns

```
In [20]: X

Out[20]: array([[[0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 [0, 0, 0, ..., 0, 0, 0],
                 ...,
                 [0, 0, 2, ..., 0, 0, 0],
                 [0, 1, 2, ..., 0, 0, 0],
                 [0, 1, 2, ..., 0, 0, 0]],
```

```
In [31]: y.value_counts()

Out[31]: 1    4684
         7    4401
         3    4351
         9    4188
         2    4177
         6    4137
         0    4132
         4    4072
         8    4063
         5    3795
         Name: count, dtype: int64
```

DATA PREPROCESSING

- The MinMax Scaler is a normalization technique used in machine learning and data preprocessing to rescale features to a specified range, usually [0, 1].
- This helps ensure that all features contribute equally to the model's performance, especially when features are on different scale.

```
In [38]: X_train

Out[38]: array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                 0.        ],
                [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                 0.        ],
                [0.        , 0.        , 0.22727273, ..., 0.        , 0.        ,
                 0.        ],
                ...,
                [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                 0.        ],
                [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                 0.        ],
                [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                 0.        ]])
```

```
In [40]: X_test

Out[40]: array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                 0.        ],
                [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
                 0.        ],
                [0.1       , 0.        , 0.27272727, ..., 0.        , 0.        ,
                 0.        ],
                ...,
```

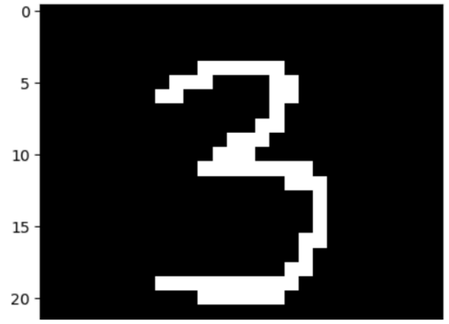
IMAGE MANIPULATIONS:

Binary and Gray scale conversions

```
In [60]: img
```

```
In [62]: img_thrsh
```

```
In [63]: plt.imshow(img_thrsh,cmap="gray")
Out[63]: <matplotlib.image.AxesImage at 0x162913890>
```

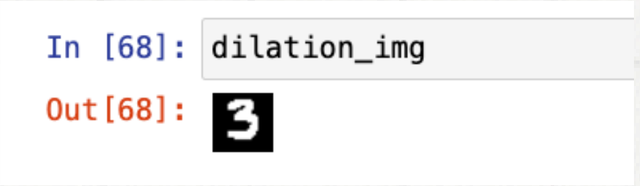


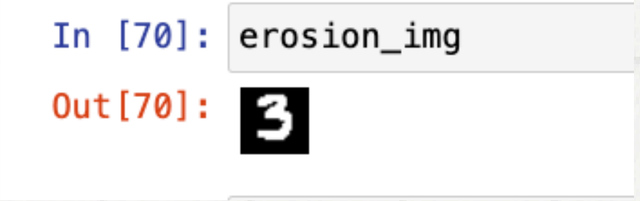

```
In [65]: prediction
Out[65]: array(['3'], dtype=object)
```

IMAGE MANIPULATIONS:

Dilation and Erosion Techniques

- Dilation is an operation that increases the size of objects in a binary image. It adds pixels to the boundaries of objects, which can help in connecting disjoint parts of an object or filling small holes.
- Erosion is an operation that decreases the size of objects in a binary image. It removes pixels from the boundaries of objects, which can help in removing noise or separating objects that are close to each other.





ML MODELS AND ACCURACIES

Logistic Regression:

- Simple, interpretable, and works well with linearly separable data.

Support Vector Machines (SVM):

- Effective in high-dimensional spaces, particularly with a good choice of kernel (e.g., RBF kernel).

Decision Trees:

- Simple, interpretable, and capable of capturing non-linear patterns.

Random Forests:

- Robust, reduces overfitting by averaging multiple decision trees.

```
DecisionTree accuracy: 0.8427380952380953
```

```
Logistic Regression accuracy: 0.919404761904762
```

```
Support Vector Machine accuracy: 0.9332142857142857
```

```
RandomForestClassifier accuracy: 0.9615476190476191
```

MODEL BUILDING

K Nearest Neighbours

Accuracy Achieved:

- Test Set Accuracy: 97%.
- The model demonstrated strong performance, accurately predicting the digit in 97 out of 100 cases.

Conclusion:

Effectiveness of KNN:

- KNN proved to be highly effective for this image recognition task.
- Simple yet powerful, KNN offers high accuracy with minimal preprocessing.

```
Out[52]: 97.71130952380952
```

```
In [56]: predict_digit(r"/Users/nithyasanthoshini/Desktop/
```

```
Out[56]: array(['4'], dtype=object)
```

```
n [91]: predict_digit([r"/Users/nithyasanthoshini/Desktop/I
                       r"/Users/nithyasanthoshini/Desktop/I
                       r"/Users/nithyasanthoshini/Desktop/In
```

```
ut[91]: array(['9', '8', '7'], dtype=object)
```

NEURAL NETWORK FOR MNIST DIGIT PREDICTION

Architecture:

- **Input Layer:** Flattened 28x28 images (784 features)
- **Hidden Layer:** Dense layer with 512 neurons and ReLU activation
- **Output Layer:** Dense layer with 10 neurons and Softmax activation for classification

Compilation:

- **Optimizer:** RMSprop
- **Loss Function:** Categorical Crossentropy
- **Metrics:** Accuracy

Performance:

- **Training Accuracy:** 99%
- **Test Accuracy:** 97%

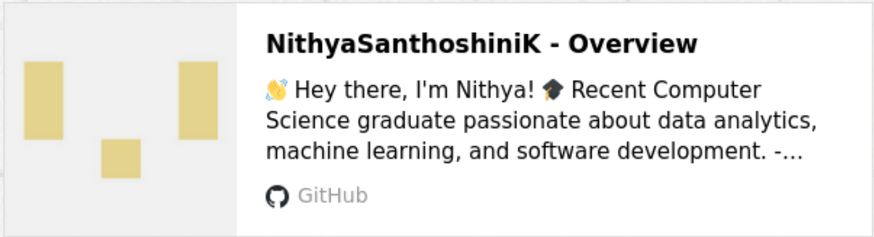
```
accuracy: 0.8721 - loss: 0.4355
accuracy: 0.9665 - loss: 0.1138
accuracy: 0.9787 - loss: 0.0713
accuracy: 0.9843 - loss: 0.0503
accuracy: 0.9900 - loss: 0.0355
```

```
test_acc: 0.9793999791145325
```

CONCLUSION

- This project successfully demonstrates the use of the K-Nearest Neighbors (KNN) algorithm and Neural Networks for handwritten digit recognition using the MNIST dataset, achieving an impressive accuracy of 97.7% and 98%
- Through systematic steps of importing and preprocessing the image data, extracting relevant features, and implementing the KNN classifier and Artificial Neural Netowrks , we achieved highly accurate predictions of handwritten digits.

THANK YOU



https://www.linkedin.com/in/nithya21966/