EXPERIMENT 1

HANDWRITTEN DIGIT CLASSIFICATION USING CNN (MNIST DATASET - GRAY SCALE)

AIM:

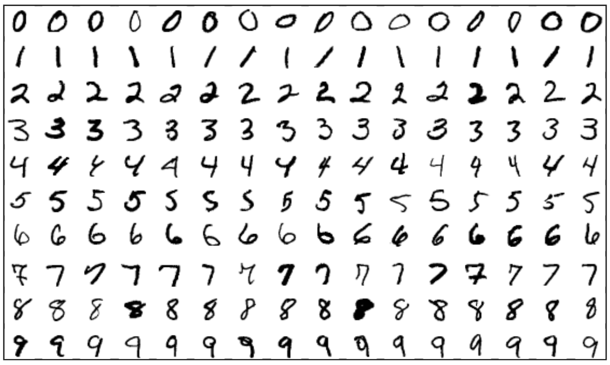
To build and train a Convolutional Neural Network (CNN) model using the MNIST dataset to classify grayscale images of handwritten digits (0–9).

PRE-REQUISITES:

1. Basics of Machine Learning Basics
2. Python Programming
3. Knowledge on Numpy, Pandas, Matplotlib, TensorFlow/ Keras
4. Jupyter Notebook
5. Data Pre-Processing Techniques
6. Knowledge on Neural Networks

MNIST Dataset

- The MNIST data set contains handwritten single digits from 0 to 9.
- This data set can be easily accessed with Keras.
- The data set ha 60,000 Training and 10,000 Testing Images.
- Each digit image is a 28x28 Matrix.



1. Importing the Basic Libraries

In [49]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# It tells Jupyter to display Matplotlib plots directly below the code cell that produced them, inside the noteboo
# You don’t need to call plt.show()
%matplotlib inline
```

2. Importing the Built-in MNIST dataset from the Keras

In [50]:
```
from tensorflow.keras.datasets import mnist
```

In [51]:
```
# Load the MNIST dataset as Training and Testing data
(X_train,y_train),(X_test,y_test) = mnist.load_data()
```

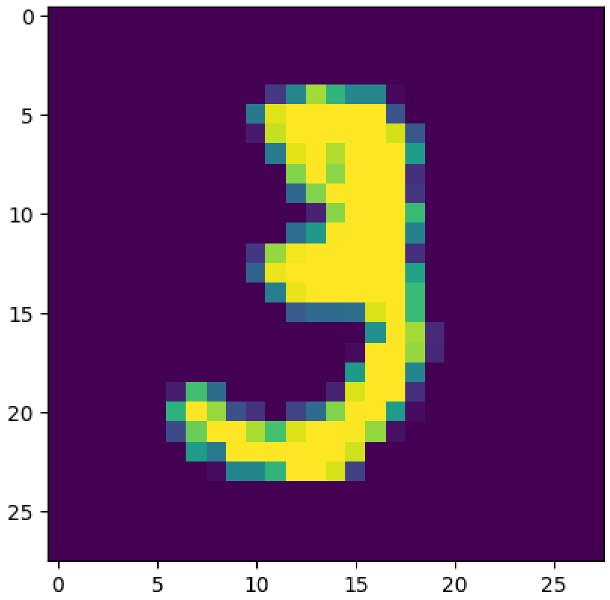
In [52]:
```
X_train.shape, y_train.shape
# 60,000 Images, each image is 28x28 pixel
```

Out[52]:
```
((60000, 28, 28), (60000,))
```

In [ ]:
```
# Reading one Image of the MNIST X_train data
X_train[10]
```

In [54]:
```
# Viewing the Image 8 of X_train
plt.imshow(X_train[10])
```

Out[54]:
```
<matplotlib.image.AxesImage at 0x309010610>
```



In [55]:
```
# Checking y_train data
y_train
```

Out[55]:
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

3. Pre-Process the Data as required

Since, this is classification problem, we need to encode the y_train data, If not, the model assume the y label is a continuous data

In [56]:
```
# Import the library
from tensorflow.keras.utils import to_categorical
```

In [57]:
```
# Shape of the y_train
y_train.shape
```

Out[57]:
```
(60000,)
```

One-hot Encoding the y

In [58]:
```
# Convert class labels to one-hot encoding
# num_classes=10 tells the function that your classification task has 10 different classes
y_train_cat = to_categorical(y_train, num_classes=10)
y_test_cat = to_categorical(y_test, num_classes=10)
```

In [59]:
```
# the index of one represents the actual output digit
# the 8th row belongs to digit 1
y_train_cat[10]
```

Out[59]:
```
array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0.])
```

Scaling the Data

In [60]:
```
# Each pixel value of every image is ranging from 0 to 255
# So, normalize every value in between 0 to 1
```

In [61]:
```
# Normalize the pixel values to range [0, 1]
# the max value of any pixel is 255, so dividing each value with 255 will normalize the value to maximum 1
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Re-shaping the Data

In [62]:
```
X_train.shape, X_test.shape
```

Out[62]:
```
((60000, 28, 28), (10000, 28, 28))
```

- We reshape MNIST images from (28, 28) to (28, 28, 1) because Convolutional Neural Networks (CNNs) require input data to include a channel dimension—and since MNIST images are grayscale, the channel value is 1, making the input shape compatible with **CNN layers that expect 3D input: height, width, and channels.**

```
In [63]: # Reshape data to add channel dimension (1 for grayscale)

         # batch size, height, width, colour channel)
         X_train = X_train.reshape(60000,28,28,1)
         X_test = X_test.reshape(10000,28,28,1)
```

4. Build the Model

```
In [64]: # import the libraries
         from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
```

Create the Model

```
In [65]: # Model Instance
         model = Sequential()
```

```
In [66]: # Convolution Layer
         model.add(Conv2D(filters=32,kernel_size=(4,4),input_shape=(28,28,1),activation='relu'))
```

```
/Users/srinutupakula/Library/Python/3.9/lib/python/site-packages/keras/src/layers/convolutional/base_conv.py:107: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
In [67]: # Pooling Layer
         model.add(MaxPool2D(pool_size=(2,2)))
```

```
In [68]: # Flatten Layer
         model.add(Flatten())
```

```
In [69]: # Dense Layers (Fully Connected Layers)
         model.add(Dense(128,activation='relu'))
```

```
In [70]: # Output Layer (For multiclass use softmax)
         model.add(Dense(10,activation='softmax'))
```

Compile the Model

```
In [71]: model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
         # loss - Optimizer that adjusts weights to minimize loss
         # optimizer - Suitable for multi-class classification with one-hot labels
         # accuracy - Track model performance using accuracy metric
```

4. Train the Model

```
In [72]: # Train the model with Early Stopping
         from tensorflow.keras.callbacks import EarlyStopping
```

```
In [73]: early_stop = EarlyStopping(monitor='val_loss', patience=1)
```

```
In [74]: # Train the model
         model.fit(X_train, y_train_cat, epochs=10, validation_data=(X_test,y_test_cat), callbacks=[early_stop])
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9109 - loss: 0.2971 - val_accuracy: 0.9810 - val_loss: 0.0
578
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9841 - loss: 0.0526 - val_accuracy: 0.9826 - val_loss: 0.0
492
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9896 - loss: 0.0330 - val_accuracy: 0.9865 - val_loss: 0.0
379
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9923 - loss: 0.0229 - val_accuracy: 0.9821 - val_loss: 0.0
561
```

```
Out[74]: <keras.src.callbacks.history.History at 0x3091054f0>
```

5. Evaluate the Model

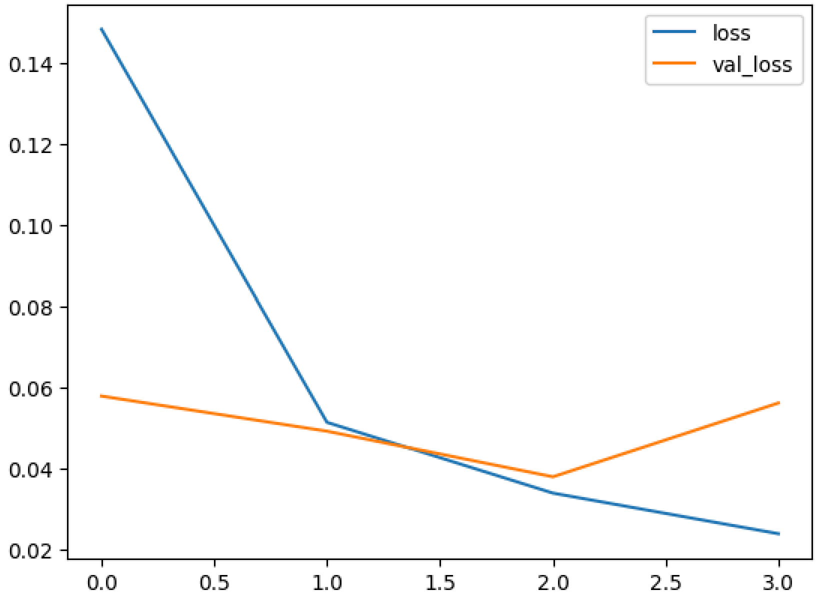
```
In [75]: # Plot the accuracy because we used accuracy metric while compiling the model
         metrics = pd.DataFrame(model.history.history)
         metrics
```

Out[75]:

	accuracy	loss	val_accuracy	val_loss
0	0.955933	0.148289	0.9810	0.057840
1	0.984433	0.051357	0.9826	0.049171
2	0.989317	0.033886	0.9865	0.037911
3	0.992200	0.023861	0.9821	0.056133

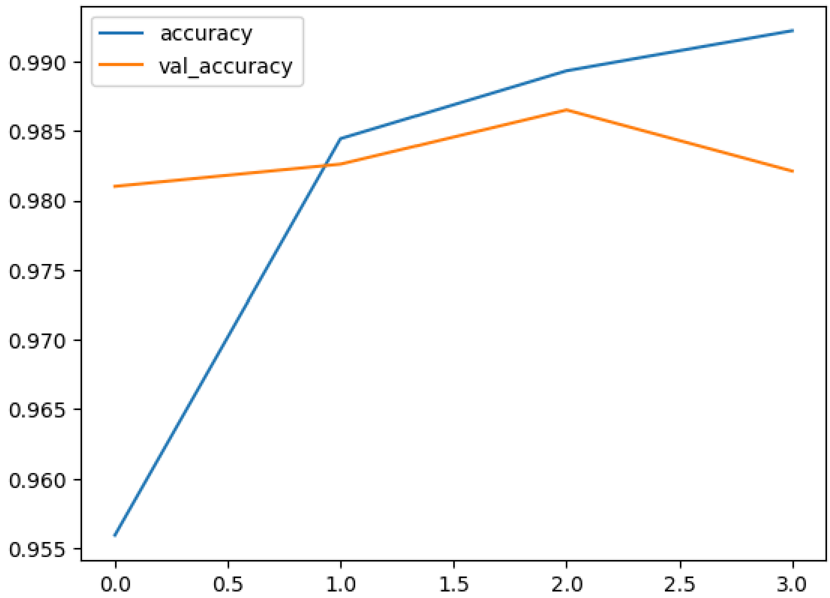
In [76]:
```
# Plot loss
metrics[['loss', 'val_loss']].plot()
```

Out[76]: <Axes: >



In [77]:
```
# Plot accuracy
metrics[['accuracy', 'val_accuracy']].plot()
```

Out[77]: <Axes: >



Classification report

In [78]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [79]:
```
# Get the Classifications on test data
y_pred = model.predict(X_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 872us/step
```

In [80]:
```
# y_test is one-hot encoded, convert it to class labels too
y_pred = np.argmax(y_pred, axis=1)
```

In [81]:
```python
# Classification Report
print(classification_report(y_test,y_pred))
```
```
              precision    recall  f1-score   support

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

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

In [82]:
```python
# Confusion Matrix
print(confusion_matrix(y_test,y_pred))
```
```
[[ 975    0    3    1    0    1    0    0    0    0]
 [    0 1123    6    1    2    0    1    1    1    0]
 [    0    0 1021    6    0    0    0    5    0    0]
 [    0    0    1 1008    0    1    0    0    0    0]
 [    0    1    2    0  973    0    0    0    0    6]
 [    1    0    0   14    0  877    0    0    0    0]
 [    6    3    4    0    2   33  909    0    1    0]
 [    1    0    8    2    0    0    0 1014    0    3]
 [    2    1    7   18    3    4    0    2  932    5]
 [    1    1    2    2    5    1    0    7    1  989]]
```

Classiying the new image

In [83]:
```python
from tensorflow.keras.preprocessing import image
from PIL import Image

# Convert to grayscale
new_image = Image.open('two.png').convert('L')
```

In [84]:
```python
# Resize to 28x28
new_image = new_image.resize((28, 28))
```

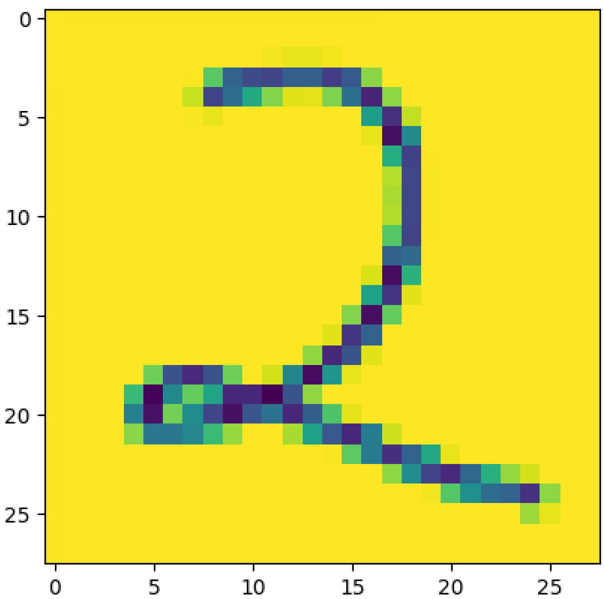
In [85]:
```python
new_image
```

Out[85]: 

In [86]:
```python
# Convert to NumPy array and normalize
img_array = np.array(new_image)
img_array = img_array / 255.0
```

In [87]:
```python
plt.imshow(img_array)
```

Out[87]: <matplotlib.image.AxesImage at 0x309236dc0>



In [88]:
```
# Reshape to match input shape of model: (1, 28, 28, 1)
img_array = img_array.reshape(1, 28, 28, 1)
```

In [89]:
```
pred = model.predict(img_array)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

In [90]:
```
np.argmax(pred, axis=1)
```

Out[90]:
```
array([2])
```

RESULT:

A Convolutional Neural Network (CNN) model was successfully developed and trained using the MNIST dataset to classify grayscale images of handwritten digits (0–9) and the model achieved high accuracy and able to correctly predict digits from custom input images.