Image classification of number using tensorflow and MNIST dataset

MNIST database

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.The database is also widely used for training and testing in the field of machine learning.It was created by "re-mixing" the samples from NIST's original datasets.The MNIST database contains 60,000 training images and 10,000 testing images.Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset

Tensorflow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow was developed by the Google Brain team for internal Google use in research and production.The initial version was released under the Apache License 2.0 in 2015.Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019. TensorFlow can be used in a wide variety of programming languages, most notably Python, as well as Javascript, C++, and Java

Neural Network

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another

Importing Libraries

In [3]:
```
import tensorflow as tf
```

Importing Datasets

In [4]:
```
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

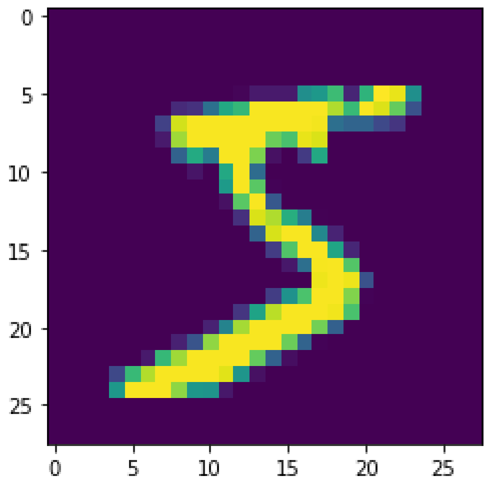
Dimension of the Array

In [5]:
```
print("x_train shape: ",x_train.shape)
print("y_train shape: ",y_train.shape)
print("x_test shape: ",x_test.shape)
print("y_test shape: ",y_test.shape)
```

```
x_train shape:  (60000, 28, 28)
y_train shape:  (60000,)
x_test shape:  (10000, 28, 28)
y_test shape:  (10000,)
```

Importing Module & Plotting First Image of the Training Dataset

In [7]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(x_train[0])
plt.show()
```



Checking With the Target Variable

In [8]:
```
y_train[0]
```

Out[8]:
```
5
```

Encoding Classes

In [9]:
```
from keras.utils import to_categorical
y_train_enc=to_categorical(y_train)
y_test_enc=to_categorical(y_test)
```

Checking Dimension of both the target variable

In [10]:
```
print("y_train shape: ",y_train_enc.shape)
print("y_test shape: ",y_test_enc.shape)
```

```
y_train shape:  (60000, 10)
y_test shape:  (10000, 10)
```

In [11]:
```
y_train_enc[0]
```

Out[11]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

Reshape Training and the Testing Datasets

In [12]:
```
import numpy as np
x_train_rs=np.reshape(x_train,(60000,784))
```

```
x_test_rs=np.reshape(x_test,(10000,784))
print("x_train reshaped: ",x_train_rs.shape)
print("x_test reshaped: ",x_test_rs.shape)
```

```
x_train reshaped:  (60000, 784)
x_test reshaped:  (10000, 784)
```

Standardization of the array

In [13]:
```
x_mean=np.mean(x_train_rs)
x_mean2=np.mean(x_test_rs)
x_std=np.std(x_train_rs)
x_std2=np.std(x_test_rs)
x_train_std=(x_train_rs-x_mean)/x_std
x_test_std=(x_test_rs-x_mean2)/x_std2
```

Standardization view of Training and Testing Datasets

In [14]:
```
print("Standardized training set: ",set(x_train_std[0]))
print("Standardized testing set: ",set(x_test_std[0]))
```

```
Standardized training set:  {-0.3858901621553201, 1.3069219669849146, 1.1796428595
307615, 1.8033104860561113, 1.6887592893473735, 2.821543345689335, 2.7197200597260
127, 1.192370770276177, 1.53602436040239, 1.7396709323290347, 2.7960875241985046,
2.6560805059989363, 2.18514780841857, 2.4906176663085375, -0.10587612575618353, 2.
681536327489767, 0.03413089244338476, -0.19497150097409063, 0.7723497156774721, 0.
9378125553678709, -0.2458831439557518, 2.210603629909401, 1.9051337720194337, 1.26
87382347486686, 1.7651267538198654, -0.424073894391566, 0.41596821480584373, -0.28
40668761919977, 0.27596119660627544, 1.4596568959298981, 1.2941940562394993, 2.096
052433200663, 1.9560454150010949, 2.7579037919622587, 1.4851127174207288, -0.09314
821501076823, 2.783359613453089, 2.286971094381893, 2.4524339340722916, 1.34510569
92211605, -0.042236572029107036, 2.643352595253521, -0.13133194724701414, 0.759621
8049320568, 0.28868910735169073, 0.6068868759870732, 0.6196147867324885, -0.411345
9836461507, 0.46687985778750496, 0.9505404661132862, 0.14868208915212244, 0.568703
1437508273, 1.2305545025124227, 0.5941589652416579, 2.3633385588543843, 0.12322626
766129186, 1.5614801818932207, 1.0905474843128544, 0.19959373213378365, -0.0804203
0426535293, -0.22042732246492122, 1.8924058612740184, 1.2560103240032534, 2.057868
700964417, 1.7523988430744502, 2.388794380345215, 0.39051239331501314, -0.39861807
29007354, -0.3095226976828283, 1.6123918248748819, 1.9433175042556796, 0.021402981
69796946, -0.11860403650159883, 2.439706023326876, 2.7451758812168436, 2.274243183
6364774, 0.16140999989753776, 2.6051688630172753, 2.770631702707674, 2.13423616543
6909, 1.026907930585778, 0.07231462467963067, 1.9942291472373408, 2.63062468450810
6, 0.721438072695811}
Standardized testing set:  {-0.42680526933869534, 0.6341696780260173, 1.4804235050
907284, 1.581468738173082, 1.909865745690731, 0.3310339787789565, 2.37719994869661
64, 2.7813808810260308, 2.617182377267206, 2.0740642494495556, 1.7204059336613182,
2.4403532193730877, 2.2130014449377917, 0.12894351261424936, 2.71822761034956, -0.
3131293821210476, 2.743488918620148, 2.3645692945613224, 2.6298130314025006, 2.187
7401366672036, 0.4194485577260159, 0.621539023890723, 0.35629528704954494, 0.30577
267050836815, 2.415091911102499, 2.7687502268907367, 2.794011535161325, 2.70559695
62142654, 0.053159587802484164, 0.9120440690024896, 0.2931420163730739, -0.2499761
1144457657, -0.19945349490339978, -0.148930878362223, -0.1868228407681056, -0.4141
7461520340115, 1.2025491141142561, 1.9351270539613195, 1.0257199562201373, 2.73085
8264484854, 2.465614527643676, 0.22998874569660294, 0.40681790359072173, 0.3184033
2464366233, 2.5540291065907352, -0.212084149038694, -0.3889133069328128, -0.035254
99114457522, 0.015267625396601573, 1.6698833171201415, 0.5204937908083694, 0.34366
46329142507, 0.07842089607307257, -0.36365199866222436, 1.3414863096024923, 1.2530
717306554329, 1.1646571517083735, 1.8719737832848486, 2.5161371441848526, 2.579290
414861324, 2.162478828396615, 2.604551723131912, 2.137217520126027, 2.339307986290
7338, 0.5457550990789579, 2.402461256967205, -0.1615615324975172, 1.01308930208484
3, 0.4826018284024869, 0.027898279531895772, 1.6319913547142588, 1.101503881031902
4}
```

Creating Neural Network

In [15]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(532, activation = 'relu', input_shape = (784,)),
    Dense(532, activation = 'relu'),
    Dense(10, activation = 'softmax')
])
```

Compiling Neural Network

In [16]:
```
model.compile(
    optimizer = 'sgd',
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 532)               417620    
                                                                 
 dense_1 (Dense)             (None, 532)               283556    
                                                                 
 dense_2 (Dense)             (None, 10)                5330      
                                                                 
=================================================================
Total params: 706,506
Trainable params: 706,506
Non-trainable params: 0
_________________________________________________________________
```

Fitting Model with a Training Dataset and Target Variable

In [17]:
```
model.fit(
    x_train_std,
    y_train_enc,
    epochs = 5
)
```

```
Epoch 1/5
1875/1875 [==============================] - 19s 7ms/step - loss: 0.3165 - accurac
y: 0.9106
Epoch 2/5
1875/1875 [==============================] - 14s 7ms/step - loss: 0.1567 - accurac
y: 0.9547
Epoch 3/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.1134 - accurac
y: 0.9675
Epoch 4/5
1875/1875 [==============================] - 16s 9ms/step - loss: 0.0890 - accurac
y: 0.9747
Epoch 5/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0722 - accurac
y: 0.9797
```

Out[17]:
```
<keras.callbacks.History at 0x1fab330ae80>
```

Accuracy of the model

In [18]:
```
loss1, accuracy1 = model.evaluate(x_test_std, y_test_enc)
loss2, accuracy2 = model.evaluate(x_train_std, y_train_enc)
print('test set accuracy: ', accuracy1 *100)
print('train set accuracy: ', accuracy2 * 100)
```

```
313/313 [==============================] - 2s 4ms/step - loss: 0.0909 - accuracy:
0.9728
1875/1875 [==============================] - 7s 3ms/step - loss: 0.0623 - accurac
y: 0.9829
test set accuracy:  97.28000164031982
train set accuracy:  98.29333424568176
```