Image classification of number using tensorflow and MNIST dataset

MNIST database

MNIST is a database. The acronym stands for “Modified National Institute of Standards and Technology.” The MNIST database contains handwritten digits (0 through 9), and can provide a baseline for testing image processing systems. MNIST is divided into two datasets: the training set has 60,000 examples of hand-written numerals, and the test set has 10,000. MNIST is a subset of a larger dataset available at the National Institute of Standards and Technology. All of its images are the same size, and within them, the digits are centered and size normalized.

Tensorflow

TensorFlow is a Python-friendly open source library for numerical computation that makes machine learning and developing neural networks faster and easier. Created by the Google Brain team and initially released to the public in 2015, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms (aka neural networks) and makes them useful by way of common programmatic metaphors. It uses Python or JavaScript to provide a convenient front-end API for building applications, while executing those applications in high-performance C++.

Neural Network

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

IMPORTING LIBRARIES

In [1]:

```
import tensorflow as tf
```

IMPORTING DATASETS

In [18]:

```
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

DIMENSION OF ARRAY

In [19]:

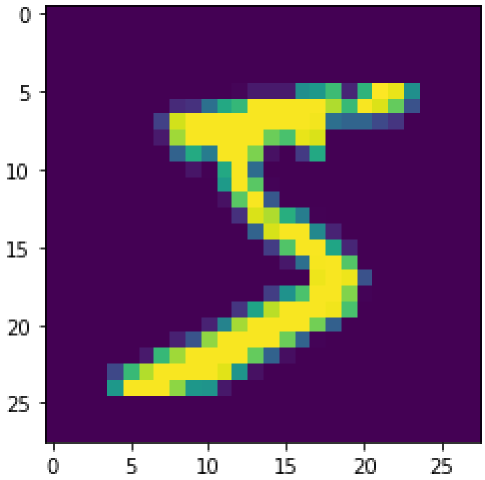
```
print("x_train shape: ",x_train.shape)
print("y_train shape: ",y_train.shape)
print("x_test shape: ",x_test.shape)
print("y_test shape: ",y_test.shape)
```

```
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 TRAINING DATASET

In [20]:

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



In [21]:

```
y_train[0]
```

Out[21]: 5

ENCODING CLASSES

In [22]:

```
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 [23]:

```
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 [24]:

```
y_train_enc[0]
```

Out[24]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)

RESHAPE TRAINING AND TESTING DATASET

In [25]:

```
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 ARRAY

In [26]:

```
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 [27]:

```
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, 1.179642859530
7615, 1.8033104860561113, 1.6887592893473735, 2.821543345689335, 2.7197200597260127,
1.192370770276177, 1.53602436040239, 1.7396709323290347, 2.7960875241985046, 2.65608
05059989363, 2.18514780841857, 2.4906176663085375, -0.10587612575618353, 2.681536327
489767, 0.03413089244338476, -0.19497150097409063, 0.7723497156774721, 0.93781255536
78709, -0.245883143955751 8, 2.210603629909401, 1.9051337720194337, 1.268738234748668
6, 1.7651267538198654, -0.424073894391566, 0.41596821480584373, -0.2840668761919977,
0.27596119660627544, 1.4596568959298981, 1.2941940562394993, 2.096052433200663, 1.95
60454150010949, 2.7579037919622587, 1.4851127174207288, -0.09314821501076823, 2.7833
59613453089, 2.286971094381893, 2.4524339340722916, 1.3451056992211605, -0.042236572
029107036, 2.643352595253521, -0.13133194724701414, 0.7596218049320568, 0.288689107 3
5169073, 0.6068868759870732, 0.6196147867324885, -0.4113459836461507, 0.466879857787
50496, 0.9505404661132862, 0.14868208915212244, 0.5687031437508273, 1.23055450251242
27, 0.5941589652416579, 2.3633385588543843, 0.12322626766129186, 1.5614801818932207,
1.0905474843128544, 0.19959373213378365, -0.08042030426535293, -0.22042732246492122,
1.8924058612740184, 1.2560103240032534, 2.057868700964417, 1.7523988430744502, 2.388
794380345215, 0.39051239331501314, -0.3986180729007354, -0.3095226976828283, 1.61239
18248748819, 1.9433175042556796, 0.02140298169796946, -0.11860403650159883, 2.439706
023326876, 2.7451758812168436, 2.2742431836364774, 0.16140999989753776, 2.6051688630
172753, 2.770631702707674, 2.134236165436909, 1.026907930585778, 0.072314624679630 6
7, 1.9942291472373408, 2.630624684508106, 0.721438072695811}
Standardized testing set:  {-0.42680526933869534, 0.6341696780260173, 1.480423505090
7284, 1.581468738173082, 1.909865745690731, 0.3310339787789565, 2.377199948696 6164,
2.7813808810260308, 2.617182377267206, 2.0740642494495556, 1.7204059336613182, 2.440
3532193730877, 2.2130014449377917, 0.12894351261424936, 2.71822761034956, -0.3131293
821210476, 2.743488918620148, 2.3645692945613224, 2.6298130314025006, 2.187740136667
2036, 0.4194485577260159, 0.621539023890723, 0.35629528704954494, 0.3057726705083681
5, 2.415091911102499, 2.7687502268907367, 2.794011535161325, 2.7055969562142654, 0.0
5315958780248416 4, 0.9120440690024896, 0.2931420163730739, -0.24997611144457657, -0.
19945349490339978, -0.148930878362223, -0.1868228407681056, -0.41417461520340115, 1.
2025491141142561, 1.9351270539613195, 1.0257199562201373, 2.730858264484854, 2.46561
4527643676, 0.22998874569660294, 0.40681790359072173, 0.31840332464366233, 2.5540291
065907352, -0.212084149038694, -0.3889133069328128, -0.03525499114457522, 0.01526762
5396601573, 1.6698833171201415, 0.5204937908083694, 0.343664632914250 7, 0.0784208960
7307257, -0.36365199866222436, 1.3414863096024923, 1.2530717306554329, 1.16465715170
83735, 1.8719737832848486, 2.5161371441848526, 2.579290414861324, 2.162478828396615,
2.604551723131912, 2.137217520126027, 2.3393079862907338, 0.545755099078957 9, 2.4024
```

```
61256967205, -0.1615615324975172, 1.013089302084843, 0.4826018284024869, 0.027898279
531895772, 1.6319913547142588, 1.1015038810319024}
```

CREATING NEURAL NETWORK

In [28]:

```python
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 [29]:

```python
model.compile(
optimizer = 'sgd',
loss = 'categorical_crossentropy',
metrics = ['accuracy']
)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 532)               417620
_________________________________________________________________
dense_4 (Dense)              (None, 532)               283556
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5330
=================================================================
Total params: 706,506
Trainable params: 706,506
Non-trainable params: 0
_________________________________________________________________
```

FITTING MODEL WITH TRAINING DATASET AND TARGET VARIABLE

In [30]:

```python
model.fit(
x_train_std,
y_train_enc,
epochs = 12
)
```

```
Epoch 1/12
1875/1875 [==============================] - 12s 6ms/step - loss: 0.3208 - accuracy:
0.9087 0s - l
Epoch 2/12
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1570 - accuracy:
0.9543
Epoch 3/12
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1145 - accuracy:
0.9670
Epoch 4/12
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0900 - accuracy:
0.9741
Epoch 5/12
1875/1875 [==============================] - 10s 6ms/step - loss: 0.0725 - accuracy:
0.9796
Epoch 6/12
```

```
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0594 - accuracy: 
0.9835
Epoch 7/12
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0499 - accuracy: 
0.9862
Epoch 8/12
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0423 - accuracy: 
0.9887
Epoch 9/12
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0359 - accuracy: 
0.9909
Epoch 10/12
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0305 - accuracy: 
0.9927
Epoch 11/12
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0264 - accuracy: 
0.9940
Epoch 12/12
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0226 - accuracy: 
0.9955
```

Out[30]:
```
<tensorflow.python.keras.callbacks.History at 0x2d801cc4af0>
```

ACCURACY OF THE MODEL

In [31]:
```
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 [==============================] - 1s 3ms/step - loss: 0.0639 - accuracy: 0.
9796: 0s - loss: 0.0805 - ac
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0182 - accuracy: 
0.9974
test set accuracy:  97.96000123023987
train set accuracy:  99.73666667938232
```

In [ ]: