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

```
import tensorflow as tf
```

Importing Datasets

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist_
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

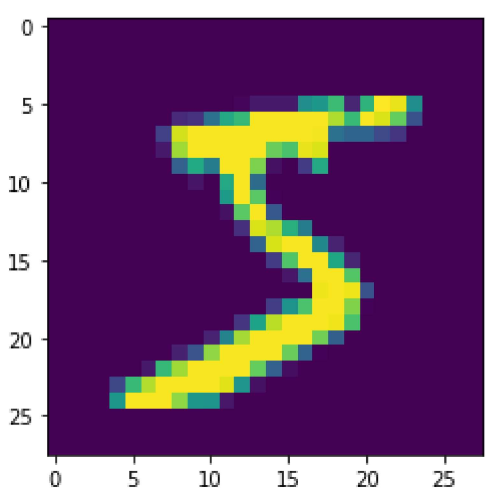
Dimension of the Array

```
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

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



Checking With the Target Variable

```
y_train[0]
```

```
5
```

Encoding Classes

```
from tensorflow.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

```
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)
```

```
y_train_enc[0]
```

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

Reshape Training and the Testing Datasets

```
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

```
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

```
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, 1.179642859530761
Standardized testing set:  {-0.42680526933869534, 0.6341696780260173, 1.48042350509072
```

Creating Neural Network

```
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

```
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

```
model.fit(
    x_train_std,
```

```
    y_train_enc,
    epochs = 5
)
```

```
Epoch 1/5
1875/1875 [==============================] - 21s 11ms/step - loss: 0.3169 - accuracy: 0
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1541 - accuracy: 0.9
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1116 - accuracy: 0.9
Epoch 4/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0879 - accuracy: 0.9
Epoch 5/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0709 - accuracy: 0.9
<keras.callbacks.History at 0x7f71152de0d0>
```

Accuracy of the model

```
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: 0.0904 - accuracy: 0.9724
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0589 - accuracy: 0.98
test set accuracy:  97.24000096321106
train set accuracy:  98.46333265304565
```