LGMVIP_DS_October_23_Task_Number_3-1

Neural Network That Can Read Hand Writing

By Mouli Nahal

In [2]:
```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import math
```

Train Test Splitting MNIST Data

In [3]:
```
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

X_train = X_train / 255.0
X_test = X_test / 255.0
```

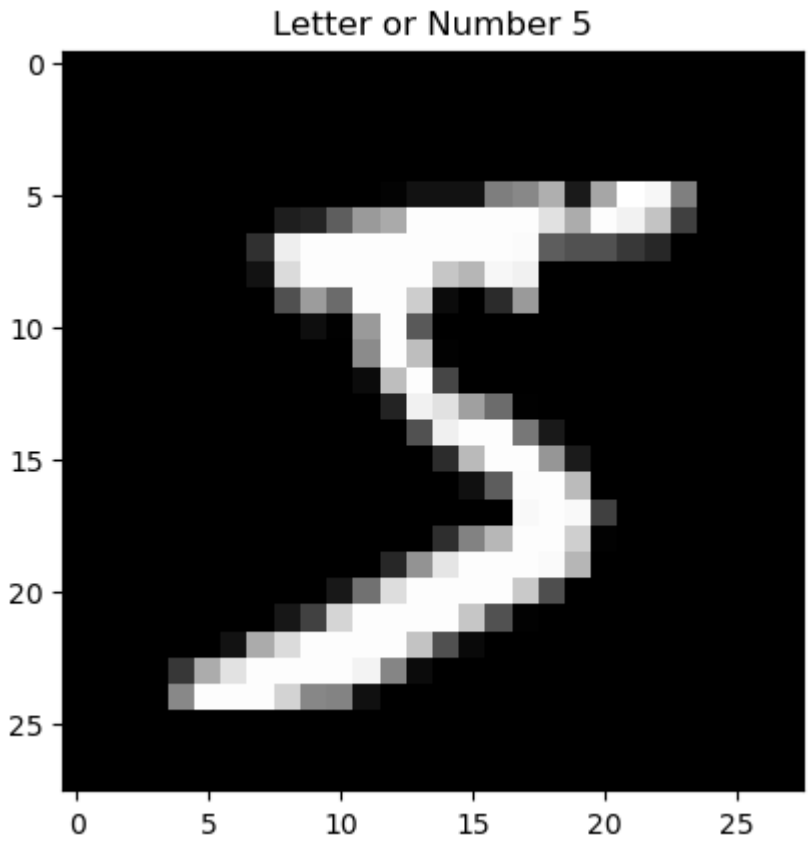
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
mnist.npz
11490434/11490434 [==============================] - 5s 0us/step
```

In [4]:
```
print("The Shape of Training Dataset is: ",X_train.shape)
print('The Shape of Test Dataset is: ',X_test.shape)
```

```
The Shape of Training Dataset is:  (60000, 28, 28)
The Shape of Test Dataset is:  (10000, 28, 28)
```

In [5]:
```
plt.imshow(X_train[0],cmap='gray')
plt.title('Letter or Number '+str(y_train[0]))
```

Out[5]:
```
Text(0.5, 1.0, 'Letter or Number 5')
```



In [6]:
```
image_size=[]
for i in range(len(X_train)):
    image_size.append(X_train[i].size)
```

In [7]:
```
print('The size of Image is: ',(math.sqrt(X_train[0].size),math.sqrt(X_train[0].siz
```

```
The size of Image is:  (28.0, 28.0)
```

In [8]:
```
image_width=28
image_height=28
batch_size=128
class_num=10
epochs=20
input_shape=(image_width,image_height,1)
```

Build the Neural Network Model

```
    # This is formatted as code
```

In [9]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation='softmax')
])
```

In [10]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [12]:
```
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
```

```
Epoch 1/10
750/750 [==============================] - 4s 5ms/step - loss: 0.3864 - accuracy:
0.8877 - val_loss: 0.1922 - val_accuracy: 0.9476
Epoch 2/10
750/750 [==============================] - 2s 3ms/step - loss: 0.1877 - accuracy:
0.9459 - val_loss: 0.1389 - val_accuracy: 0.9613
Epoch 3/10
750/750 [==============================] - 2s 2ms/step - loss: 0.1396 - accuracy:
0.9588 - val_loss: 0.1131 - val_accuracy: 0.9657
Epoch 4/10
750/750 [==============================] - 2s 2ms/step - loss: 0.1132 - accuracy:
0.9667 - val_loss: 0.1013 - val_accuracy: 0.9696
Epoch 5/10
750/750 [==============================] - 2s 2ms/step - loss: 0.0954 - accuracy:
0.9711 - val_loss: 0.0915 - val_accuracy: 0.9716
Epoch 6/10
750/750 [==============================] - 2s 2ms/step - loss: 0.0811 - accuracy:
0.9754 - val_loss: 0.0889 - val_accuracy: 0.9737
Epoch 7/10
750/750 [==============================] - 2s 2ms/step - loss: 0.0715 - accuracy:
0.9784 - val_loss: 0.0875 - val_accuracy: 0.9732
Epoch 8/10
750/750 [==============================] - 2s 2ms/step - loss: 0.0631 - accuracy:
0.9807 - val_loss: 0.0840 - val_accuracy: 0.9752
Epoch 9/10
750/750 [==============================] - 2s 2ms/step - loss: 0.0602 - accuracy:
0.9805 - val_loss: 0.0903 - val_accuracy: 0.9737
Epoch 10/10
750/750 [==============================] - 2s 2ms/step - loss: 0.0525 - accuracy:
0.9828 - val_loss: 0.0832 - val_accuracy: 0.9766
```

Out[12]:
```
<keras.callbacks.History at 0x21c43f3a6e0>
```

Model Performance on Test Dataset

In [13]:
```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0739 - accuracy:
0.9780
Test accuracy: 97.80%
```

Make Predictions

In [14]:
```
predictions = model.predict(X_test[:5])
```

```
1/1 [==============================] - 0s 155ms/step
```

In [ ]:
```

```