Develop A Neural Network That Can Read Handwriting

LETS GROW MORE - Virtual Internship 2023

Name : Shiva Dagdu Mehenge

Data Science Intern

#LGMVIP

In [1]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
```

In [2]:

```python
mnist= tf.keras.datasets.mnist
```

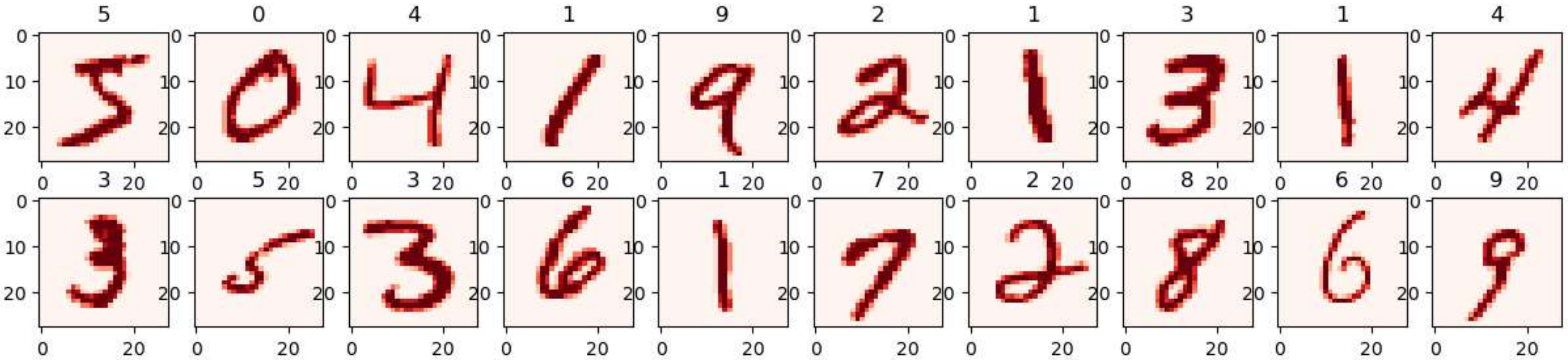
In [3]:

```python
(x_train,y_train),(x_test,y_test)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11490434/11490434 [==============================] - 10s 1us/step
```

In [4]:

```python
fig=plt.figure(figsize=(15,3))
for i in range(20):
  ax=fig.add_subplot(2,10,i+1)
  ax.imshow(np.squeeze(x_train[i]),cmap='Reds')
  ax.set_title(y_train[i])
```



In [5]:

```
print(x_train.shape)
print(x_train[0])
```

```
(60000, 28, 28)
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```

In [6]:

```python
xtrain = x_train/255.0
xtest = x_test/255.0
```

In [7]:

```python
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28,28)),
                                    tf.keras.layers.Dense(128, activation ='relu'),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

In [8]:

```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 784)               0         
                                                                 
 dense (Dense)               (None, 128)               100480    
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

```
model.compile(optimizer = tf.keras.optimizers.Adam(),
              loss ='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

In [10]:

```
model.fit(xtrain,y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2665 - accuracy: 0.9240
Epoch 2/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1147 - accuracy: 0.9664
Epoch 3/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0795 - accuracy: 0.9758
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0588 - accuracy: 0.9822
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0458 - accuracy: 0.9863
```

Out[10]:

```
<keras.callbacks.History at 0x2035c0e3700>
```

In [11]:

```
model.fit(xtrain,y_train, epochs=9)
```

```
Epoch 1/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0368 - accuracy: 0.9886
Epoch 2/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0290 - accuracy: 0.9915
Epoch 3/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0229 - accuracy: 0.9932
Epoch 4/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0209 - accuracy: 0.9935
Epoch 5/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0167 - accuracy: 0.9949
Epoch 6/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0124 - accuracy: 0.9963
Epoch 7/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0127 - accuracy: 0.9961
Epoch 8/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0100 - accuracy: 0.9971
Epoch 9/9
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0093 - accuracy: 0.9970
```

Out[11]:

```
<keras.callbacks.History at 0x203786060d0>
```

In [12]:

```
print(model.evaluate(x_test,y_test))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 17.8906 - accuracy: 0.9808
[17.890552520751953, 0.9807999730110168]
```

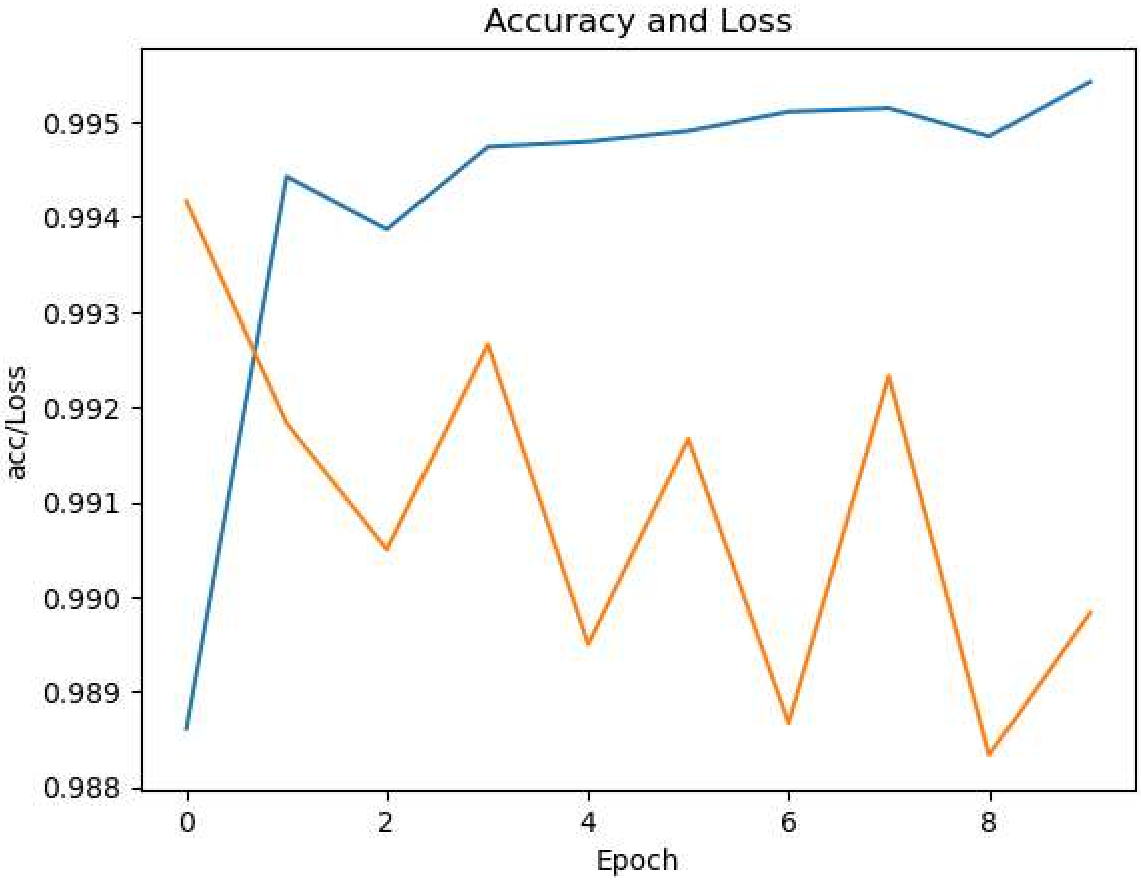
In [13]:

```
history=model.fit(x_train,y_train,epochs=10,batch_size=32,validation_split=0.1)
```

```
Epoch 1/10
1688/1688 [==============================] - 6s 4ms/step - loss: 5.2709 - accuracy: 0.9886 - val_loss: 2.567
0 - val_accuracy: 0.9942
Epoch 2/10
1688/1688 [==============================] - 6s 3ms/step - loss: 1.9537 - accuracy: 0.9944 - val_loss: 3.104
8 - val_accuracy: 0.9918
Epoch 3/10
1688/1688 [==============================] - 5s 3ms/step - loss: 2.1301 - accuracy: 0.9939 - val_loss: 4.378
9 - val_accuracy: 0.9905
Epoch 4/10
1688/1688 [==============================] - 5s 3ms/step - loss: 1.8090 - accuracy: 0.9947 - val_loss: 3.663
2 - val_accuracy: 0.9927
Epoch 5/10
1688/1688 [==============================] - 5s 3ms/step - loss: 1.6643 - accuracy: 0.9948 - val_loss: 4.273
2 - val_accuracy: 0.9895
Epoch 6/10
1688/1688 [==============================] - 5s 3ms/step - loss: 1.9391 - accuracy: 0.9949 - val_loss: 4.352
9 - val_accuracy: 0.9917
Epoch 7/10
1688/1688 [==============================] - 5s 3ms/step - loss: 1.6901 - accuracy: 0.9951 - val_loss: 5.487
5 - val_accuracy: 0.9887
Epoch 8/10
1688/1688 [==============================] - 5s 3ms/step - loss: 1.9068 - accuracy: 0.9951 - val_loss: 3.502
0 - val_accuracy: 0.9923
Epoch 9/10
1688/1688 [==============================] - 4s 3ms/step - loss: 1.5873 - accuracy: 0.9949 - val_loss: 5.849
1 - val_accuracy: 0.9883
Epoch 10/10
1688/1688 [==============================] - 5s 3ms/step - loss: 1.4900 - accuracy: 0.9954 - val_loss: 5.019
7 - val_accuracy: 0.9898
```

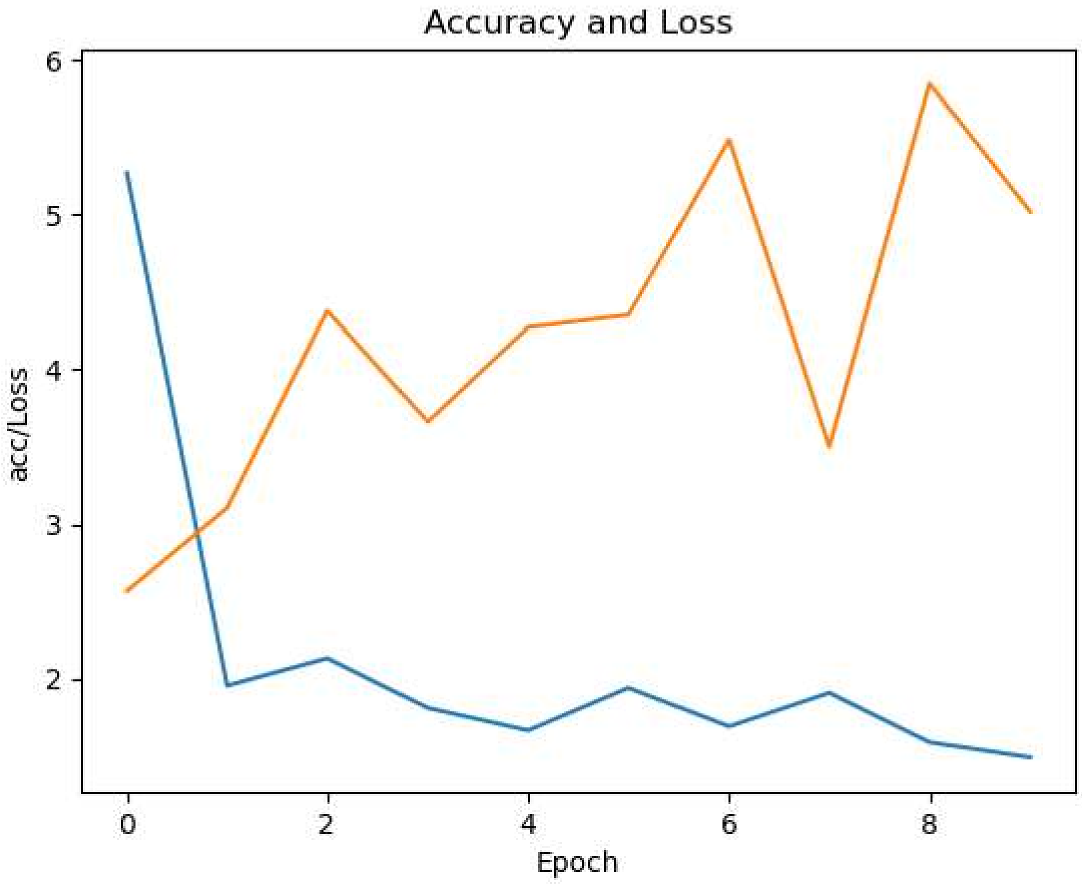
In [14]:

```
plt.title("Accuracy and Loss")
plt.xlabel("Epoch")
plt.ylabel("acc/Loss")
plt.plot(history.history['accuracy'],label='acc')
plt.plot(history.history["val_accuracy"],label='val')
plt.show()
```



In [15]:

```
plt.title("Accuracy and Loss")
plt.xlabel("Epoch")
plt.ylabel("acc/Loss")
plt.plot(history.history['loss'],label='acc')
plt.plot(history.history["val_loss"],label='val')
plt.show()
```

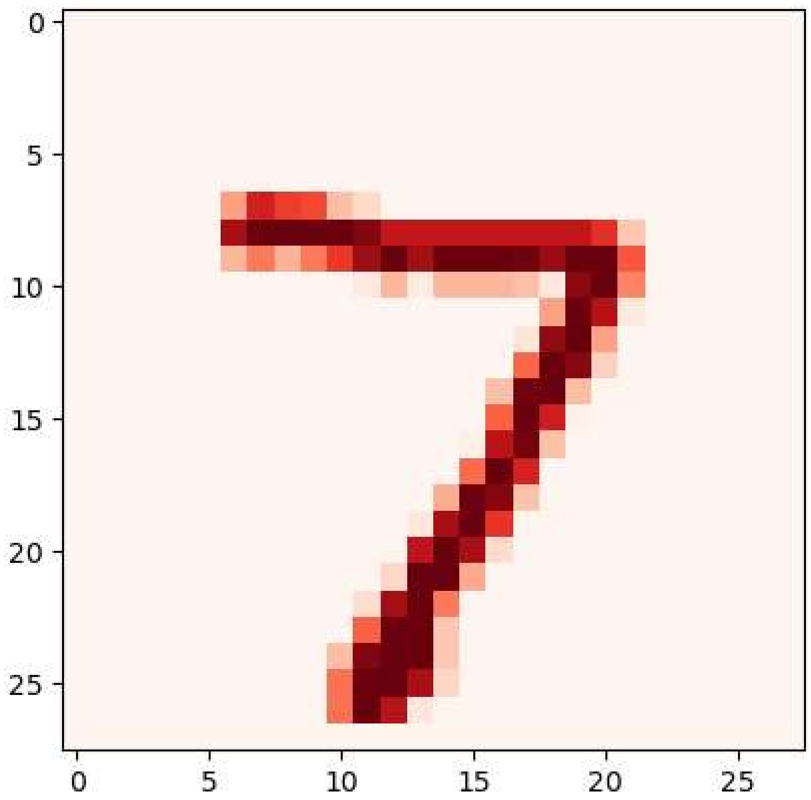


In [16]:

```
plt.imshow(np.squeeze(x_test[0]),cmap="Reds")
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x2037a835550>
```



In [17]:

```
prediction=model.predict(x_test)
print(np.argmax(prediction[0]))
```

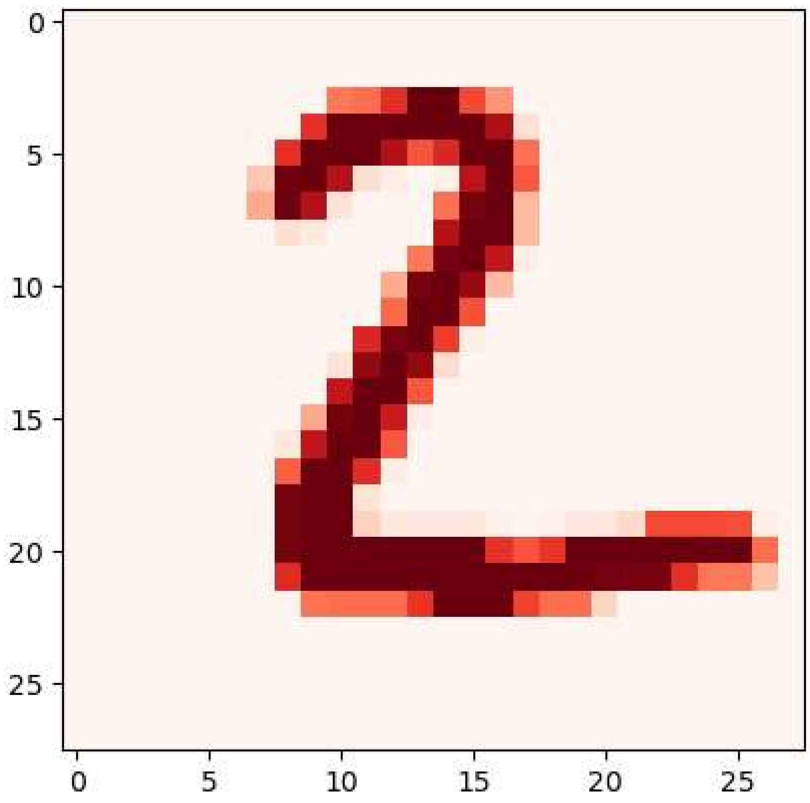
```
313/313 [==============================] - 1s 2ms/step
7
```

In [18]:

```python
plt.imshow(np.squeeze(x_test[1]),cmap="Reds")
```

Out[18]:

```
<matplotlib.image.AxesImage at 0x2037a8c07c0>
```



In [19]:

```python
prediction=model.predict(x_test)
print(np.argmax(prediction[1]))
```

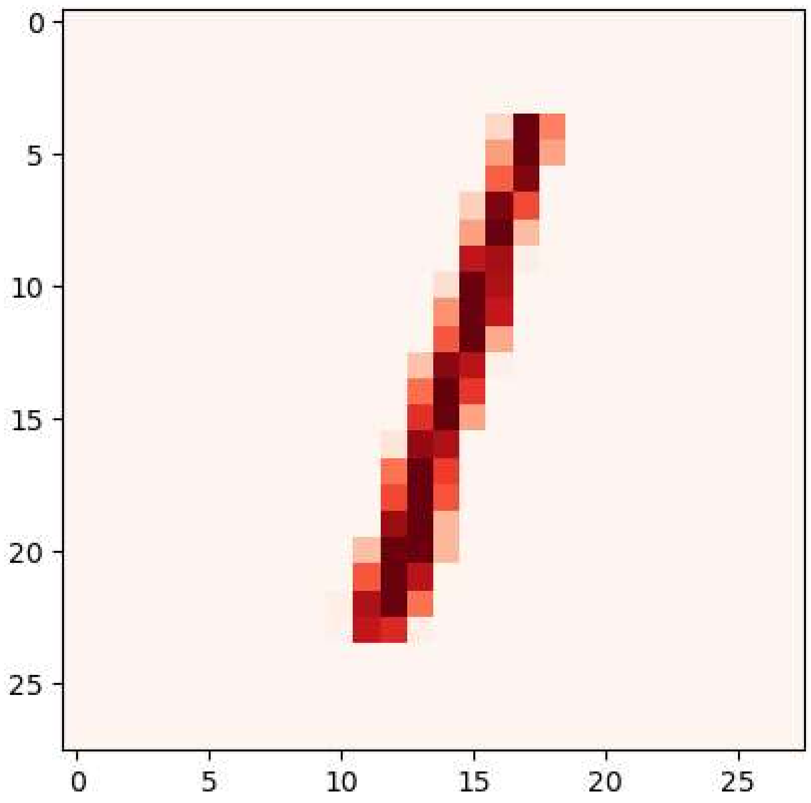
```
313/313 [==============================] - 1s 2ms/step
2
```

In [20]:

```
plt.imshow(np.squeeze(x_test[2]),cmap="Reds")
```

Out[20]:

```
<matplotlib.image.AxesImage at 0x2037aa7f8b0>
```



In [21]:

```
prediction=model.predict(x_test)
print(np.argmax(prediction[2]))
```

```
313/313 [==============================] - 1s 2ms/step
1
```

Thank you !

In [ ]:

```
```