In [18]:
```
#para este experimento se utilizó la función de activación relu,no. de clases,NADAM
import tensorflow as tf
import keras as keras
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,Activation
from tensorflow.keras.optimizers import RMSprop, SGD,Adam,Nadam
```

In [4]:
```
learning_rate = 0.001
epochs = 20
batch_size = 120
```

In [5]:
```
from tensorflow.keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

In [6]:
```
X_train.shape
```

Out[6]:
```
(60000, 28, 28)
```

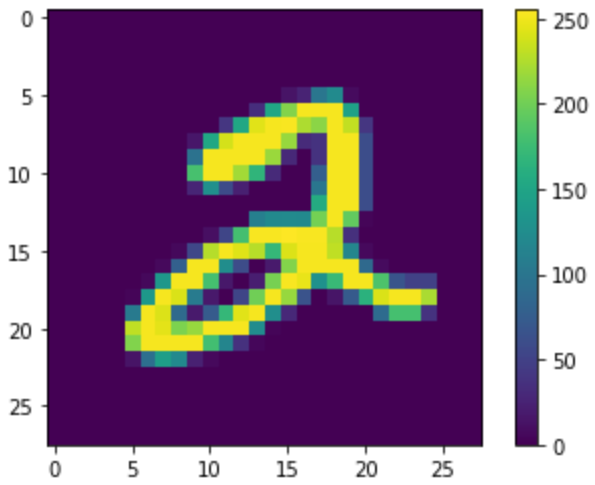
In [7]:
```
x_trainv = X_train.reshape(60000, 784)
x_testv = X_test.reshape(10000, 784)
x_trainv = x_trainv.astype('float32')
x_testv = x_testv.astype('float32')

x_trainv /= 255  # x_trainv = x_trainv/255
x_testv /= 255
```

In [8]:
```
print(Y_train[10000])
```

```
3
```

In [9]:
```
num_classes=10
y_trainc = keras.utils.to_categorical(Y_train, num_classes)
y_testc = keras.utils.to_categorical(Y_test, num_classes)
```

In [10]:
```
plt.figure()
plt.imshow(X_train[5])#número de imagen en el mnist
plt.colorbar()
plt.grid(False)
plt.show()
```



In [11]:
```
#pre-procesamiento
train_images = X_train / 255.0#escalara los valores
```

```
test_images = Y_train / 255.0
```

In [12]:
```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dense(num_classes, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               401920

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```
#model.compile(optimizer='adam',
           #    loss='sparse_categorical_crossentropy',  metrics=['accuracy'])
```

In [21]:
```
model.compile(loss='categorical_crossentropy',optimizer=Nadam(learning_rate=learning_rat
```

In [22]:
```
history = model.fit(x_trainv, y_trainc,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_testv, y_testc)
                    )
```

```
Epoch 1/20
500/500 [==============================] - 3s 5ms/step - loss: 0.2682 - accuracy: 0.9254
- val_loss: 0.1328 - val_accuracy: 0.9614
Epoch 2/20
500/500 [==============================] - 2s 4ms/step - loss: 0.1058 - accuracy: 0.9694
- val_loss: 0.0944 - val_accuracy: 0.9712
Epoch 3/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0682 - accuracy: 0.9803
- val_loss: 0.0773 - val_accuracy: 0.9757
Epoch 4/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0479 - accuracy: 0.9861
- val_loss: 0.0725 - val_accuracy: 0.9759
Epoch 5/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0345 - accuracy: 0.9904
- val_loss: 0.0640 - val_accuracy: 0.9789
Epoch 6/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0259 - accuracy: 0.9926
- val_loss: 0.0616 - val_accuracy: 0.9805
Epoch 7/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0192 - accuracy: 0.9948
- val_loss: 0.0623 - val_accuracy: 0.9793
Epoch 8/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0145 - accuracy: 0.9964
- val_loss: 0.0616 - val_accuracy: 0.9808
Epoch 9/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0109 - accuracy: 0.9974
- val_loss: 0.0626 - val_accuracy: 0.9817
Epoch 10/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0082 - accuracy: 0.9982
- val_loss: 0.0617 - val_accuracy: 0.9813
```

```
Epoch 11/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0070 - accuracy: 0.9984
- val_loss: 0.0671 - val_accuracy: 0.9812
Epoch 12/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0090 - accuracy: 0.9975
- val_loss: 0.0678 - val_accuracy: 0.9821
Epoch 13/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0068 - accuracy: 0.9981
- val_loss: 0.0666 - val_accuracy: 0.9812
Epoch 14/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0040 - accuracy: 0.9991
- val_loss: 0.0696 - val_accuracy: 0.9828
Epoch 15/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0088 - accuracy: 0.9974
- val_loss: 0.0754 - val_accuracy: 0.9797
Epoch 16/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0033 - accuracy: 0.9994
- val_loss: 0.0645 - val_accuracy: 0.9843
Epoch 17/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0027 - accuracy: 0.9994
- val_loss: 0.0701 - val_accuracy: 0.9827
Epoch 18/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0055 - accuracy: 0.9983
- val_loss: 0.0806 - val_accuracy: 0.9801
Epoch 19/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0035 - accuracy: 0.9991
- val_loss: 0.0730 - val_accuracy: 0.9831
Epoch 20/20
500/500 [==============================] - 2s 4ms/step - loss: 0.0011 - accuracy: 0.9999
- val_loss: 0.0865 - val_accuracy: 0.9803
```

In [23]:

```
score = model.evaluate(x_testv, y_testc, verbose=1) #evaluar la eficiencia del modelo
print(score)
a=model.predict(x_testv) #predicción de la red entrenada
print(a.shape)
print(a[1])
print("resultado correcto:")
print(y_testc[1])
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0865 - accuracy: 0.9803
[0.08651945739984512, 0.9803000092506409]
313/313 [==============================] - 0s 1ms/step
(10000, 10)
[1.7653883e-06 9.9964577e-01 1.0000000e+00 6.3476651e-03 4.6669262e-15
 2.5136352e-03 6.6876339e-05 3.3757734e-09 9.9799652e-03 3.8390063e-10]
resultado correcto:
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

In [ ]: