In [4]:
```
#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
from tensorflow.keras import regularizers
```

In [5]:
```
learning_rate = 0.001
epochs = 20
batch_size = 120
```

In [6]:
```
from tensorflow.keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

In [7]:
```
X_train.shape
```

Out[7]:
```
(60000, 28, 28)
```

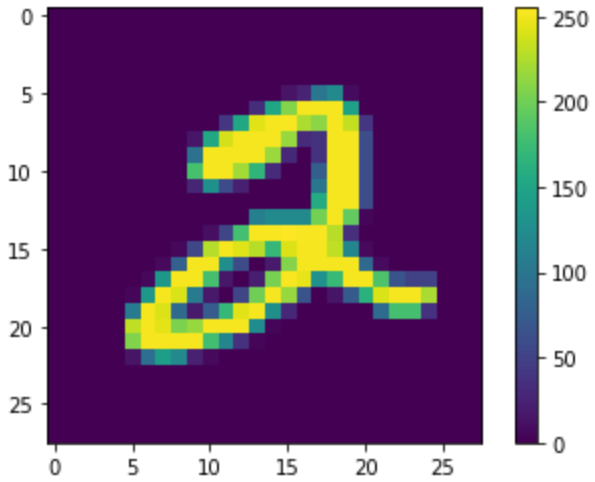
In [8]:
```
x_trainv = X_train.reshape(60000, 784)
x_testv = X_test.reshape(10000, 784)
x_trainv = x_trainv.astype('float32')
x_testv = x_testv.astype('float32')

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

In [9]:
```
print(Y_train[10000])
```

```
3
```

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

In [11]:
```
plt.figure()
plt.imshow(X_train[5])#número de imagen en el mnist
plt.colorbar()
plt.grid(False)
plt.show()
```



In [12]:
```
#pre-procesamiento
train_images = X_train / 255.0#escalara los valores
```

```
test_images = Y_train / 255.0
```

In [20]:
```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,),kernel_regularizer=regularize
model.add(Dense(num_classes, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 512)               401920

 dense_4 (Dense)             (None, 10)                5130

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

In [21]:
```
#model.compile(optimizer='adam',
           #    loss='sparse_categorical_crossentropy',  metrics=['accuracy'])
```

In [22]:
```
model.compile(loss='categorical_crossentropy',optimizer=Nadam(learning_rate=learning_rat
```

In [23]:
```
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: 9.3671 - accuracy: 0.7455
- val_loss: 1.5700 - val_accuracy: 0.8247
Epoch 2/20
500/500 [==============================] - 2s 4ms/step - loss: 1.4870 - accuracy: 0.8304
- val_loss: 1.3833 - val_accuracy: 0.8487
Epoch 3/20
500/500 [==============================] - 2s 4ms/step - loss: 1.3511 - accuracy: 0.8478
- val_loss: 1.2889 - val_accuracy: 0.8625
Epoch 4/20
500/500 [==============================] - 2s 4ms/step - loss: 1.2780 - accuracy: 0.8564
- val_loss: 1.2169 - val_accuracy: 0.8677
Epoch 5/20
500/500 [==============================] - 2s 5ms/step - loss: 1.2295 - accuracy: 0.8619
- val_loss: 1.1811 - val_accuracy: 0.8723
Epoch 6/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1950 - accuracy: 0.8665
- val_loss: 1.1590 - val_accuracy: 0.8727
Epoch 7/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1684 - accuracy: 0.8694
- val_loss: 1.1289 - val_accuracy: 0.8778
Epoch 8/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1459 - accuracy: 0.8709
- val_loss: 1.1205 - val_accuracy: 0.8740
Epoch 9/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1273 - accuracy: 0.8745
- val_loss: 1.1021 - val_accuracy: 0.8755
Epoch 10/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1120 - accuracy: 0.8759
- val_loss: 1.0961 - val_accuracy: 0.8769
```

```
Epoch 11/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0990 - accuracy: 0.8777
- val_loss: 1.0635 - val_accuracy: 0.8825
Epoch 12/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0872 - accuracy: 0.8793
- val_loss: 1.0653 - val_accuracy: 0.8802
Epoch 13/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0765 - accuracy: 0.8808
- val_loss: 1.0420 - val_accuracy: 0.8884
Epoch 14/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0677 - accuracy: 0.8816
- val_loss: 1.0305 - val_accuracy: 0.8892
Epoch 15/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0594 - accuracy: 0.8828
- val_loss: 1.0325 - val_accuracy: 0.8900
Epoch 16/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0526 - accuracy: 0.8842
- val_loss: 1.0186 - val_accuracy: 0.8901
Epoch 17/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0454 - accuracy: 0.8846
- val_loss: 1.0170 - val_accuracy: 0.8900
Epoch 18/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0398 - accuracy: 0.8848
- val_loss: 1.0062 - val_accuracy: 0.8928
Epoch 19/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0330 - accuracy: 0.8864
- val_loss: 1.0106 - val_accuracy: 0.8887
Epoch 20/20
500/500 [==============================] - 2s 5ms/step - loss: 1.0281 - accuracy: 0.8867
- val_loss: 1.0103 - val_accuracy: 0.8956
```

In [24]:

```
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: 1.0103 - accuracy: 0.8956
[1.0103362798690796, 0.8956000208854675]
313/313 [==============================] - 0s 1ms/step
(10000, 10)
[5.9736925e-01 1.0826539e-02 9.4250488e-01 3.9392906e-01 3.7208243e-05
 3.3870685e-01 5.7399839e-01 4.8424645e-06 1.9046415e-01 1.4579527e-04]
resultado correcto:
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

In [ ]: