In [2]:
```python
#para este experimento se utilizó la función de activación relu,SGD
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
```

In [3]:
```python
learning_rate = 0.001
epochs = 20
batch_size = 120
```

In [4]:
```python
from tensorflow.keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

In [5]:
```python
X_train.shape
```

Out[5]:
```
(60000, 28, 28)
```

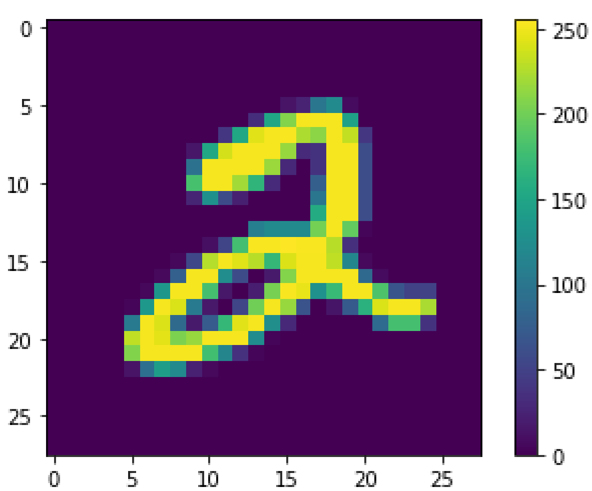
In [6]:
```python
x_trainv = X_train.reshape(60000, 784)
x_testv = X_test.reshape(10000, 784)
x_trainv = x_trainv.astype('float32')
x_testv = x_testv.astype('float32')

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

In [7]:
```python
print(Y_train[10000])
```

```
3
```

In [8]:
```python
num_classes=24
y_trainc = keras.utils.to_categorical(Y_train, num_classes)
y_testc = keras.utils.to_categorical(Y_test, num_classes)
```

In [9]:
```python
plt.figure()
plt.imshow(X_train[5])#número de imagen en el mnist
plt.colorbar()
plt.grid(False)
plt.show()
```



In [10]:
```python
#pre-procesamiento
train_images = X_train / 255.0#escalara los valores
```

```
test_images = Y_train / 255.0
```

In [19]:
```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dense(num_classes, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 512)               401920

 dense_5 (Dense)             (None, 24)                12312

=================================================================
Total params: 414,232
Trainable params: 414,232
Non-trainable params: 0
_________________________________________________________________
```

In [20]:
```
#model.compile(optimizer='adam',
           #   loss='sparse_categorical_crossentropy',  metrics=['accuracy'])
```

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

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 [==============================] - 2s 3ms/step - loss: 2.6466 - accuracy: 0.3401
- val_loss: 2.1135 - val_accuracy: 0.5943
Epoch 2/20
500/500 [==============================] - 2s 3ms/step - loss: 1.8213 - accuracy: 0.6824
- val_loss: 1.5435 - val_accuracy: 0.7500
Epoch 3/20
500/500 [==============================] - 2s 3ms/step - loss: 1.3896 - accuracy: 0.7650
- val_loss: 1.2147 - val_accuracy: 0.7986
Epoch 4/20
500/500 [==============================] - 2s 3ms/step - loss: 1.1300 - accuracy: 0.7990
- val_loss: 1.0106 - val_accuracy: 0.8190
Epoch 5/20
500/500 [==============================] - 2s 3ms/step - loss: 0.9638 - accuracy: 0.8160
- val_loss: 0.8763 - val_accuracy: 0.8330
Epoch 6/20
500/500 [==============================] - 2s 3ms/step - loss: 0.8509 - accuracy: 0.8284
- val_loss: 0.7824 - val_accuracy: 0.8440
Epoch 7/20
500/500 [==============================] - 2s 3ms/step - loss: 0.7700 - accuracy: 0.8372
- val_loss: 0.7135 - val_accuracy: 0.8524
Epoch 8/20
500/500 [==============================] - 2s 4ms/step - loss: 0.7095 - accuracy: 0.8442
- val_loss: 0.6608 - val_accuracy: 0.8579
Epoch 9/20
500/500 [==============================] - 2s 3ms/step - loss: 0.6624 - accuracy: 0.8513
- val_loss: 0.6195 - val_accuracy: 0.8621
Epoch 10/20
500/500 [==============================] - 2s 3ms/step - loss: 0.6246 - accuracy: 0.8558
- val_loss: 0.5860 - val_accuracy: 0.8670
```

```
Epoch 11/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5938 - accuracy: 0.8604
- val_loss: 0.5580 - val_accuracy: 0.8708
Epoch 12/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5679 - accuracy: 0.8644
- val_loss: 0.5345 - val_accuracy: 0.8747
Epoch 13/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5460 - accuracy: 0.8673
- val_loss: 0.5145 - val_accuracy: 0.8774
Epoch 14/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5271 - accuracy: 0.8701
- val_loss: 0.4973 - val_accuracy: 0.8791
Epoch 15/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5106 - accuracy: 0.8731
- val_loss: 0.4819 - val_accuracy: 0.8820
Epoch 16/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4961 - accuracy: 0.8755
- val_loss: 0.4687 - val_accuracy: 0.8844
Epoch 17/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4832 - accuracy: 0.8777
- val_loss: 0.4565 - val_accuracy: 0.8864
Epoch 18/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4716 - accuracy: 0.8797
- val_loss: 0.4458 - val_accuracy: 0.8879
Epoch 19/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4612 - accuracy: 0.8816
- val_loss: 0.4362 - val_accuracy: 0.8894
Epoch 20/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4518 - accuracy: 0.8832
- val_loss: 0.4274 - val_accuracy: 0.8914
```

In [18]:

```
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: 0.4197 - accuracy: 0.8920
[0.419666051864624, 0.8920000195503235]
313/313 [==============================] - 0s 1ms/step
(10000, 24)
[0.99576706 0.9036107  0.9996108  0.9961907  0.24783836 0.9948703
 0.9974698  0.19674288 0.98993236 0.33200213 0.18935655 0.15482433
 0.08822184 0.20797437 0.23764935 0.1874314  0.15654801 0.07323575
 0.1985506  0.16675599 0.15783876 0.14446746 0.11733217 0.08760193]
resultado correcto:
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [ ]: