In [2]:
```
#para este experimento se utilizó la función de activación relu,no. de clases,ADAM
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]:
```
learning_rate = 0.001
epochs = 20
batch_size = 120
```

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

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

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

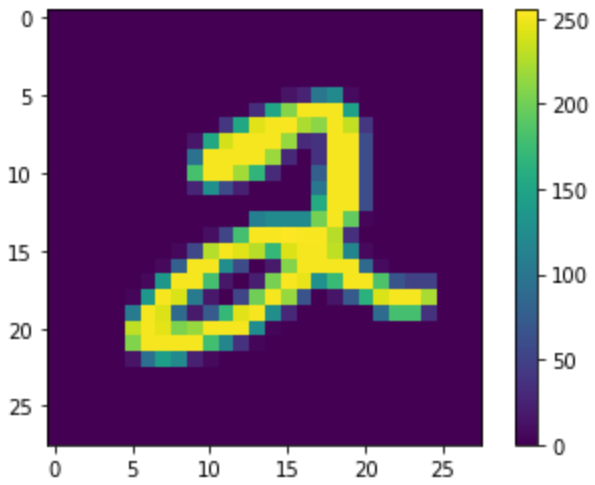
In [6]:
```
x_trainv = X_train.reshape(60000, 784)
x_testv = X_test.reshape(10000, 784)
x_trainv = x_trainv.astype('float32')
x_testv = x_testv.astype('float32')

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

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

```
3
```

In [8]:
```
num_classes=10
y_trainc = keras.utils.to_categorical(Y_train, num_classes)
y_testc = keras.utils.to_categorical(Y_test, num_classes)
```

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



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

```
test_images = Y_train / 255.0
```

In [13]:
```
model = Sequential()
model.add(Dense(512, activation='relu6', input_shape=(784,)))
model.add(Dense(num_classes, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 512)               401920

 dense_3 (Dense)             (None, 24)                12312

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

In [15]:
```
#model.compile(optimizer='adam',
           #   loss='sparse_categorical_crossentropy',  metrics=['accuracy'])
```

In [16]:
```
model.compile(loss='categorical_crossentropy',optimizer=SGD(learning_rate=learning_rate)
```

In [17]:
```
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 4ms/step - loss: 2.6181 - accuracy: 0.3119
- val_loss: 2.1142 - val_accuracy: 0.5961
Epoch 2/20
500/500 [==============================] - 2s 3ms/step - loss: 1.8258 - accuracy: 0.6911
- val_loss: 1.5546 - val_accuracy: 0.7541
Epoch 3/20
500/500 [==============================] - 2s 3ms/step - loss: 1.3956 - accuracy: 0.7735
- val_loss: 1.2180 - val_accuracy: 0.8043
Epoch 4/20
500/500 [==============================] - 2s 3ms/step - loss: 1.1292 - accuracy: 0.8078
- val_loss: 1.0059 - val_accuracy: 0.8257
Epoch 5/20
500/500 [==============================] - 2s 3ms/step - loss: 0.9575 - accuracy: 0.8251
- val_loss: 0.8661 - val_accuracy: 0.8457
Epoch 6/20
500/500 [==============================] - 2s 3ms/step - loss: 0.8413 - accuracy: 0.8372
- val_loss: 0.7694 - val_accuracy: 0.8546
Epoch 7/20
500/500 [==============================] - 2s 3ms/step - loss: 0.7588 - accuracy: 0.8461
- val_loss: 0.6992 - val_accuracy: 0.8594
Epoch 8/20
500/500 [==============================] - 2s 3ms/step - loss: 0.6975 - accuracy: 0.8518
- val_loss: 0.6465 - val_accuracy: 0.8648
Epoch 9/20
500/500 [==============================] - 2s 3ms/step - loss: 0.6502 - accuracy: 0.8574
- val_loss: 0.6051 - val_accuracy: 0.8686
Epoch 10/20
500/500 [==============================] - 2s 3ms/step - loss: 0.6128 - accuracy: 0.8615
- val_loss: 0.5721 - val_accuracy: 0.8723
```

```
Epoch 11/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5823 - accuracy: 0.8653
- val_loss: 0.5447 - val_accuracy: 0.8747
Epoch 12/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5569 - accuracy: 0.8687
- val_loss: 0.5219 - val_accuracy: 0.8789
Epoch 13/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5355 - accuracy: 0.8717
- val_loss: 0.5026 - val_accuracy: 0.8813
Epoch 14/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5172 - accuracy: 0.8744
- val_loss: 0.4859 - val_accuracy: 0.8829
Epoch 15/20
500/500 [==============================] - 2s 3ms/step - loss: 0.5013 - accuracy: 0.8765
- val_loss: 0.4714 - val_accuracy: 0.8852
Epoch 16/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4873 - accuracy: 0.8786
- val_loss: 0.4586 - val_accuracy: 0.8869
Epoch 17/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4748 - accuracy: 0.8805
- val_loss: 0.4472 - val_accuracy: 0.8887
Epoch 18/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4638 - accuracy: 0.8824
- val_loss: 0.4373 - val_accuracy: 0.8900
Epoch 19/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4539 - accuracy: 0.8844
- val_loss: 0.4280 - val_accuracy: 0.8911
Epoch 20/20
500/500 [==============================] - 2s 3ms/step - loss: 0.4449 - accuracy: 0.8857
- val_loss: 0.4197 - val_accuracy: 0.8920
```

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.]
```

In [ ]: