In [3]:
```
#para este experimento se utilizó la función de activación softmax,no. de clases,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 [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)
```

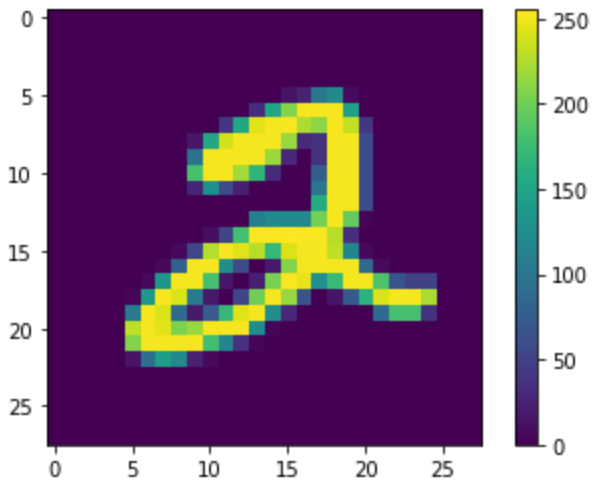
In [7]:
```
x_trainv = X_train.reshape(60000, 784)
x_testv = X_test.reshape(10000, 784)
x_trainv = x_trainv.astype('float32')
x_testv = x_testv.astype('float32')

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='softmax', input_shape=(784,)))
model.add(Dense(num_classes, activation='softmax'))

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 [14]:
```
model.compile(loss='categorical_crossentropy',optimizer=SGD(learning_rate=learning_rate)
```

In [15]:
```
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: 2.3024 - accuracy: 0.1186
- val_loss: 2.3023 - val_accuracy: 0.1493
Epoch 2/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3023 - accuracy: 0.1318
- val_loss: 2.3022 - val_accuracy: 0.1135
Epoch 3/20
500/500 [==============================] - 2s 3ms/step - loss: 2.3021 - accuracy: 0.1124
- val_loss: 2.3021 - val_accuracy: 0.1135
Epoch 4/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3020 - accuracy: 0.1124
- val_loss: 2.3019 - val_accuracy: 0.1135
Epoch 5/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3019 - accuracy: 0.1124
- val_loss: 2.3018 - val_accuracy: 0.1135
Epoch 6/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3018 - accuracy: 0.1124
- val_loss: 2.3018 - val_accuracy: 0.1135
Epoch 7/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3018 - accuracy: 0.1124
- val_loss: 2.3017 - val_accuracy: 0.1135
Epoch 8/20
500/500 [==============================] - 2s 5ms/step - loss: 2.3017 - accuracy: 0.1124
- val_loss: 2.3016 - val_accuracy: 0.1135
Epoch 9/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3016 - accuracy: 0.1124
- val_loss: 2.3015 - val_accuracy: 0.1135
Epoch 10/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3015 - accuracy: 0.1124
- val_loss: 2.3015 - val_accuracy: 0.1135
```

```
Epoch 11/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3015 - accuracy: 0.1124
- val_loss: 2.3014 - val_accuracy: 0.1135
Epoch 12/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3014 - accuracy: 0.1124
- val_loss: 2.3013 - val_accuracy: 0.1135
Epoch 13/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3014 - accuracy: 0.1124
- val_loss: 2.3013 - val_accuracy: 0.1135
Epoch 14/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3013 - accuracy: 0.1124
- val_loss: 2.3012 - val_accuracy: 0.1135
Epoch 15/20
500/500 [==============================] - 2s 3ms/step - loss: 2.3013 - accuracy: 0.1124
- val_loss: 2.3012 - val_accuracy: 0.1135
Epoch 16/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3013 - accuracy: 0.1124
- val_loss: 2.3012 - val_accuracy: 0.1135
Epoch 17/20
500/500 [==============================] - 2s 3ms/step - loss: 2.3012 - accuracy: 0.1124
- val_loss: 2.3011 - val_accuracy: 0.1135
Epoch 18/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3012 - accuracy: 0.1124
- val_loss: 2.3011 - val_accuracy: 0.1135
Epoch 19/20
500/500 [==============================] - 2s 4ms/step - loss: 2.3012 - accuracy: 0.1124
- val_loss: 2.3011 - val_accuracy: 0.1135
Epoch 20/20
500/500 [==============================] - 2s 5ms/step - loss: 2.3011 - accuracy: 0.1124
- val_loss: 2.3010 - val_accuracy: 0.1135
```

In [16]:

```
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: 2.3010 - accuracy: 0.1135
[2.301039934158325, 0.11349999904632568]
313/313 [==============================] - 0s 1ms/step
(10000, 10)
[0.0990569  0.10797299 0.09962327 0.10147    0.09814073 0.09368893
 0.09917755 0.10297299 0.09829407 0.09960259]
resultado correcto:
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

In [ ]: