Rede Neural Convolucional: aplicação

Bem-vindo a segunda tarefa desta semana! Nesta tarefa, você irá:

- Implementar funções auxiliares que serão utilizadas na implementação de um modelo usando TensorFlow.
- Implementar uma convNet completa utilizando TensorFlow.

Após esta tarefa você deve ser capaz de:

- Construir e treinar uma convNet usando TensorFlow para um problema de classificação.

1.0 - Modelo usando TensorFlow

Na tarefa anterior, você construiu funções auxiliares utilizando numpy para compreender os mecanismos por trás de uma rede neural convolucional. A maioria das aplicações práticas de aprendizado profundo hoje em dia são implementadas utilizando frameworks de programação, que já possuem diversas funções implementadas e que podem simplesmente ser chamadas.

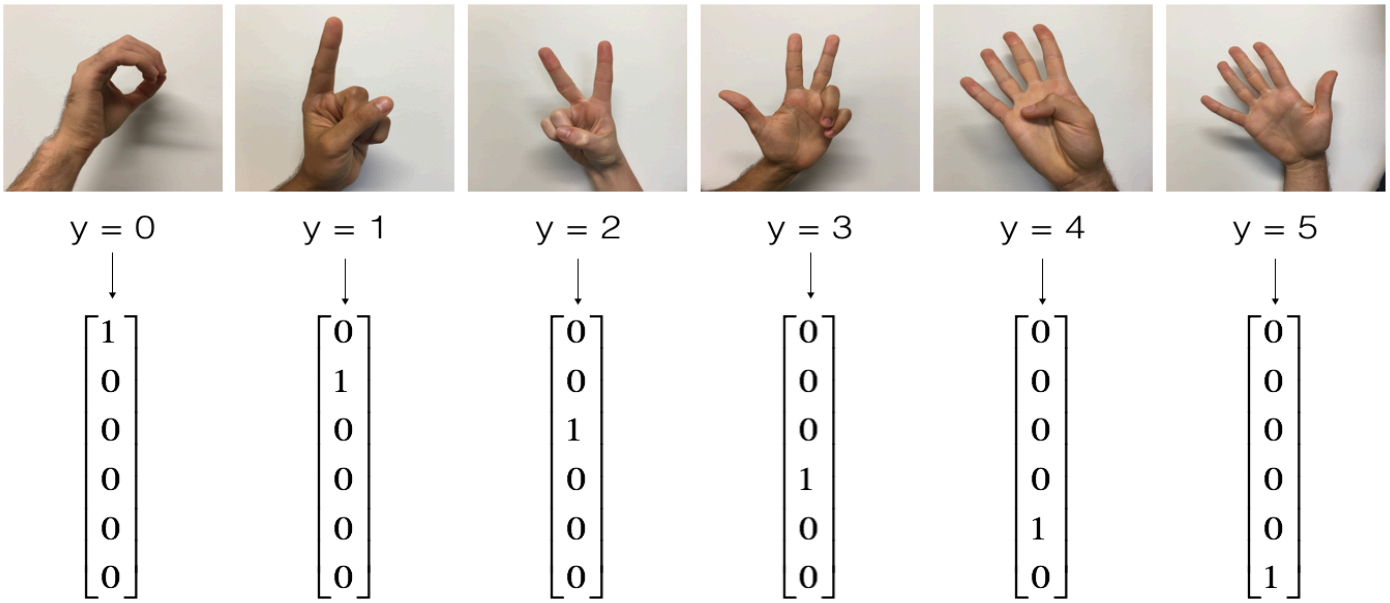
Como sempre vamos começar carregando os pacotes necessários para esta tarefa.

In [20]:
```
import math
import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage
import tensorflow as tf
from tensorflow.python.framework import ops
from cnn_utils import *

%matplotlib inline
np.random.seed(1)
```

Execute a célula abaixo para carregar a base de dados "SIGNS" que será usada nesta tarefa.

In [21]:
```
# Carregando a base de dados SIGNS
X_train_orig, Y_train_orig, X_test_orig, Y_test_orig, classes = loa
d_dataset()
```

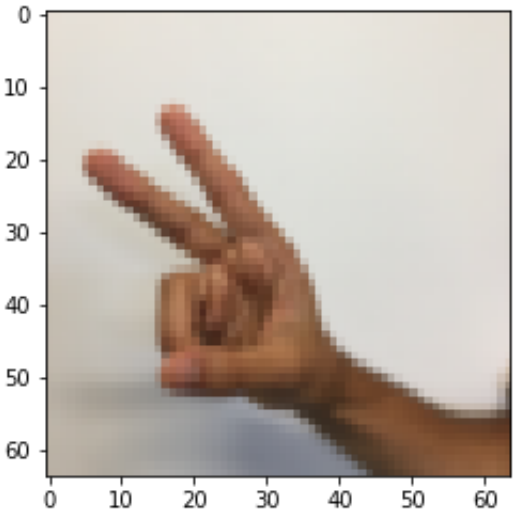
Lembrando que, a base de dados SIGNS é uma coleção de 6 sinais representando números de 0 até 5.



A próxima célula irá mostrar um exemplo de imagem da base de dados. Modifique o valor da variável `index` e execute novamente a célula para ver outros exemplos.

In [22]:
```
# Exemplo de imagem
index = 6
plt.imshow(X_train_orig[index])
print ("y = " + str(np.squeeze(Y_train_orig[:, index])))
```

```
y = 2
```



No curso anterior você construiu uma rede neural totalmente conectada para esta base de dados. Porém, como esta é uma base de dados de imagens, parece mais natural utilizar uma convNet para ele.

Vamos começar examinando o formato dos dados.

In [23]:
```
X_train = X_train_orig/255.
X_test = X_test_orig/255.
Y_train = convert_to_one_hot(Y_train_orig, 6).T
Y_test = convert_to_one_hot(Y_test_orig, 6).T
print ("número de exemplos de treinamento = " + str(X_train.shape[0
]))
print ("número de exemplos de teste = " + str(X_test.shape[0]))
print ("Formato do X_train: " + str(X_train.shape))
print ("Formato do Y_train: " + str(Y_train.shape))
print ("Formato do X_test: " + str(X_test.shape))
print ("Formato do Y_test: " + str(Y_test.shape))
conv_layers = {}
```

```
número de exemplos de treinamento = 1080
número de exemplos de teste = 120
Formato do X_train: (1080, 64, 64, 3)
Formato do Y_train: (1080, 6)
Formato do X_test: (120, 64, 64, 3)
Formato do Y_test: (120, 6)
```

1.1 - Criando placeholders

TensorFlow requer que sejam criados placeholders (variáveis) para os dados de entrada que serão utilizados no modelo, quando a sessão for executada.

Exercício: Implemente a função abaixo para criar os placeholders para as imagens de entrada X e a saída Y. Você não deve definir ainda o número de exemplos. Portanto, utilize o valor "None" como sendo o tamanho do batch, isto te dará flexibilidade para escolher este valor depois. Logo, defina as dimensões de X como **[None, n_H0, n_W0, n_C0]** e Y deve ter as dimensões **[None, n_y]**. Dica (https://www.tensorflow.org/api_docs/python/tf/placeholder).

In [24]:
```
# FUNÇÃO DE AVALIAÇÃO: create_placeholders

def create_placeholders(n_H0, n_W0, n_C0, n_y):
    """
    Cria os placeholders para a sessão do tensorflow.

    Argumentos:
    n_H0 -- um escalar, altura da imagem de entrada.
    n_W0 -- um escalar, largura da imagem de entrada.
    n_C0 -- um escalar, número de canais da entrada.
    n_y -- um escalar, número de classes

    Retorna:
    X -- o placeholder para os dados de entrada, no formato [None,
n_H0, n_W0, n_C0] e dtype "float"
    Y -- o placeholder para os rótulos de saída, no formato [None,
n_y] e dtype "float"
    """

    ### INICIE O SEU CÓDIGO AQUI ### (≈2 linhas)
    X = tf.placeholder(tf.float32, shape=(None, n_H0, n_W0, n_C0),
name="X")
    Y = tf.placeholder(tf.float32, shape=(None, n_y), name="Y")
    ### TÉRMINO DO CÓDIGO ###

    return X, Y
```

In [25]:
```
X, Y = create_placeholders(64, 64, 3, 6)
print ("X = " + str(X))
print ("Y = " + str(Y))
```

```
X = Tensor("X_1:0", shape=(?, 64, 64, 3), dtype=float32)
Y = Tensor("Y_1:0", shape=(?, 6), dtype=float32)
```

Saída esperada

X = Tensor("Placeholder:0", shape=(?, 64, 64, 3), dtype=float32)
Y = Tensor("Placeholder_1:0", shape=(?, 6), dtype=float32)

Nota: os valores dos placeholders podem ser diferentes de 0 e 1.

1.2 - Inicialização de parâmetros

Vamos inicializar os filtros/pesos $W1$ e $W2$ utilizando `tf.contrib.layers.xavier_initializer(seed = 0)`. Voc6e não precisa se preocupar com as variáveis de bias, em breve você verá que o tensorFlow cuidará disto. Note também que você deve inicializar apenas os filtros/pesos das camadas CONV. O tensorFlow incializa as camadas da parte totalmente conectada automaticamente. Falaremos mais sobre isto ainda nesta tarefa.

Exercício: Implemente initialize_parameters(). As dimensões para cada grupo de filtros são dadas abaixo. Lembre-se, para inicializar um parâmetro W no formato [1,2,3,4] em Tensorflow, use:

```
W = tf.get_variable("W", [1,2,3,4], initializer = ...)
```

mais info (https://www.tensorflow.org/api_docs/python/tf/get_variable).

In [26]:
```
# FUNÇÃO DE AVALIAÇÃO: initialize_parameters

def initialize_parameters():
    """
    Inicializa os filtros/pesos para construir uma convNet com tens
orflow. Os formatos são:
                        W1 : [4, 4, 3, 8]
                        W2 : [2, 2, 8, 16]
    Retorna:
    parameters -- um dicionário python de tensores contendo W1 e W2
    """

    tf.set_random_seed(1)                              # usado para
manter consistência nos dados

    ### INICIE O SEU CÓDIGO AQUI ### (aprox. 2 linhas)
    W1 = tf.get_variable("W1", [4, 4, 3, 8], initializer = tf.contr
ib.layers.xavier_initializer(seed = 0))
    W2 = tf.get_variable("W2", [2, 2, 8, 16], initializer = tf.cont
rib.layers.xavier_initializer(seed = 0))
    ### TÉRMINO DO CÓDIGO ###

    parameters = {"W1": W1,
                  "W2": W2}

    return parameters
```

In [27]:
```
tf.reset_default_graph()
with tf.Session() as sess_test:
    parameters = initialize_parameters()
    init = tf.global_variables_initializer()
    sess_test.run(init)
    print("W1 = " + str(parameters["W1"].eval()[1,1,1]))
    print("W2 = " + str(parameters["W2"].eval()[1,1,1]))
```

```
W1 = [ 0.00131723  0.1417614  -0.04434952  0.09197326  0.14984085
-0.03514394
 -0.06847463  0.05245192]
W2 = [-0.08566415  0.17750949  0.11974221  0.16773748 -0.0830943
-0.08058
 -0.00577033 -0.14643836  0.24162132 -0.05857408 -0.19055021  0.13
45228
 -0.22779644 -0.1601823  -0.16117483 -0.10286498]
```

Saída esperada:

W1 =	[0.00131723 0.14176141 -0.04434952 0.09197326 0.14984085 -0.03514394 -0.06847463 0.05245192]
W2 =	[-0.08566415 0.17750949 0.11974221 0.16773748 -0.0830943 -0.08058 -0.00577033 -0.14643836 0.24162132 -0.05857408 -0.19055021 0.1345228 -0.22779644 -0.1601823 -0.16117483 -0.10286498]

1.2 - Propagação para frente

Em TensorFlow, existem funçòes prontas que executam os passos da convolução.

- **tf.nn.conv2d(X,W1, strides = [1,s,s,1], padding = 'SAME'):** a entrada X e um grupo de filtros $W1$, esta função faz a convolução do filtro $W1$ sobre X. A terceira entrada ([1,s,s,1]) representa o valor de stride para cada dimensão da entrada (m, n_H_prev, n_W_prev, n_C_prev). Você pode ler a documentação completa desta função aqui (https://www.tensorflow.org/api_docs/python/tf/nn/conv2d)
- **tf.nn.max_pool(A, ksize = [1,f,f,1], strides = [1,s,s,1], padding = 'SAME'):** dada uma entrada A, esta função utiliza uma janela de tamanho (f, f) e stride = (s, s) para executar o max pooling em cada janela. Você pode ler a documentação completa desta função aqui (https://www.tensorflow.org/api_docs/python/tf/nn/max_pool)
- **tf.nn.relu(Z1):** computa o valor de ReLU de Z1 elemento a elemento (que pode estar em qualquer formato). Você pode ler a documentação completa desta função aqui. (https://www.tensorflow.org/api_docs/python/tf/nn/relu)
- **tf.contrib.layers.flatten(P)**: dada uma entrada P, esta função coverte P em um vetor de 1D mantendo o tamanho do batch. Ela retorna um tensor vetorizado no formato [batch_size, k]. Você pode ler a documentação completa desta função aqui. (https://www.tensorflow.org/api_docs/python/tf/contrib/layers/flatten)

- **tf.contrib.layers.fully_connected(F, num_outputs):** dado um vetor de entrada F, esta função retorna a saída de uma camada totalmente conectada. Você pode ler a documentação completa desta função aqui. (https://www.tensorflow.org/api_docs/python/tf/contrib/layers/fully_connected)

Na última função acima (`tf.contrib.layers.fully_connected`), a camada totalmente conectada inicializa os pesos no grafo de computação automaticamente, e mantém os pesos atualizados conforme o modelo é treinado. Portanto, você não precisa se preocupar com a inicialização destes parâmetros.

Exercício:

Implemente a função `forward_propagation` abaixo para o modelo: `CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN -> FULLYCONNECTED`. Você deve usar as funções descritas acima.

Em detalhes, utilize os seguintes conjuntos de parâmetros para todos os passos do modelo:

```
- Conv2D: stride = 1, padding = "SAME"
- ReLU
- Max pool: filtro = 8x8, stride = 8, padding = "SAME"
- Conv2D: stride = 1, padding = "SAME"
- ReLU
- Max pool: filtro = 4x4, stride = 4, padding = "SAME"
- Vetorize a saída anterior.
- camada FULLYCONNECTED (FC): aplique uma camada totalmente conectada
sem uma função de ativação não-linear. Não chame o softmax aqui. Isto i
rá resultar em 6 neurônios na camada de saída, que será passada para o
softmax. Em tensorFlow a softmax e o custo funcionam juntos em uma únic
a função, que será chamada quando o custo for computado.
```

In [32]:
```
# FUNÇÃO DE AVALIAÇÃO: forward_propagation

def forward_propagation(X, parameters):
    """
    Implementa a propagação para frente do modelo:
    CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN -> FULLYCONNECTED

    Argumentos:
    X -- o placeholder dos dados de entrada, no formato (tamanho da entrada, número de exemplos)
    parameters -- dicionário python contendo os parâmetros "W1"e "W2"
                  os formatos dos parâmetros são dados pela função initialize_parameters

    Retorna:
    Z3 -- a saída da última unidade LINEAR.
    """

    # Recupera os parâmetros do dicionário "parameters"
    W1 = parameters['W1']
    W2 = parameters['W2']

    ### INICIE O SEU CÓDIGO AQUI ###
    # CONV2D: filtro = W1, stride = 1, padding = 'SAME'
    Z1 = tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME')
    # RELU
    A1 = tf.nn.relu(Z1)
    # MAXPOOL: janela = 8x8, stride = 8, padding = 'SAME'
    P1 = tf.nn.max_pool(A1, ksize=[1, 8, 8, 1], strides=[1, 8, 8, 1], padding='SAME')
    # CONV2D: filtro W2, stride = 1, padding = 'SAME'
    Z2 = tf.nn.conv2d(P1, W2, strides=[1, 1, 1, 1], padding='SAME')
    # RELU
    A2 = tf.nn.relu(Z2)
    # MAXPOOL: janela = 4x4, stride = 4, padding = 'SAME'
    P2 = tf.nn.max_pool(A2, ksize=[1, 4, 4, 1], strides=[1, 4, 4, 1], padding='SAME')
    # Vetorizando a saída
    P2 = tf.contrib.layers.flatten(P2)
    # FULLY-CONNECTED sem uma função de ativação não-linear (não chama a softmax).
    # 6 neurôni na camada de saída. Dica: um dos argumentos deve ser "activation_fn=None"
    Z3 = tf.contrib.layers.fully_connected(P2, 6, activation_fn=None)
    ### TÉRMINO DO CÓDIGO AQUI ###

    return Z3
```

In [33]:
```
tf.reset_default_graph()

with tf.Session() as sess:
    np.random.seed(1)
    X, Y = create_placeholders(64, 64, 3, 6)
    parameters = initialize_parameters()
    Z3 = forward_propagation(X, parameters)
    init = tf.global_variables_initializer()
    sess.run(init)
    a = sess.run(Z3, {X: np.random.randn(2,64,64,3), Y: np.random.r
andn(2,6)})
    print("Z3 = " + str(a))
```

```
Z3 = [[ 1.4416987  -0.24909692  5.450499   -0.2618962  -0.20669901
1.3654671 ]
 [ 1.4070845  -0.02573182  5.0892797  -0.4866991  -0.4094069   1.2
624857 ]]
```

Saída esperada:

Z3 =	[[1.4416984 -0.24909666 5.450499 -0.2618962 -0.20669907 1.3654671] [1.4070846 -0.02573211 5.08928 -0.48669922 -0.40940708 1.2624859]]

1.3 - Computando custo

Implemente a função compute_cost abaixo. As seguintes funçòes podem ser úteis:

- **tf.nn.softmax_cross_entropy_with_logits(logits = Z3, labels = Y):** determina a perda por entropia da softmax. A função faz as duas coisas, a ativação da softmax e também determina a perda resultante. Veja a documentação completa da função aqui. (https://www.tensorflow.org/api_docs/python/tf/nn/softmax_cross_entropy_with_logits)
- **tf.reduce_mean:** computa a média dos elementos sobre todas as dimensões de um tensor. Utiliza esta soma para determinar a perda sobre todos os exemplos e obter o custo total. Veja a documentação completa da função aqui. (https://www.tensorflow.org/api_docs/python/tf/reduce_mean)

Exercício: Compute o custo abaixo utilizando as funções acima.

In [36]:
```
# FUNÇÃO DE AVALIAÇÃO: compute_cost

def compute_cost(Z3, Y):
    """
    Computa o custo.

    Argumentos:
    Z3 -- saída da propagação para frente (saída da última unidade linear), no formato (6, número de exmplos)
    Y -- vetor de rótulos verdadeiros do placeholder, mesmo formato que Z3

    Retorna:
    cost - Tensor da função custo
    """

    ### INICIE O SEU CÓDIGO AQUI ### (1 linha de código)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits = Z3, labels = Y))
    ### TÉRMINO DO CÓDIGO ###

    return cost
```

In [37]:
```
tf.reset_default_graph()

with tf.Session() as sess:
    np.random.seed(1)
    X, Y = create_placeholders(64, 64, 3, 6)
    parameters = initialize_parameters()
    Z3 = forward_propagation(X, parameters)
    cost = compute_cost(Z3, Y)
    init = tf.global_variables_initializer()
    sess.run(init)
    a = sess.run(cost, {X: np.random.randn(4,64,64,3), Y: np.random.randn(4,6)})
    print("cost = " + str(a))
```

```
cost = 4.664871
```

Saída esperada:

cost =	4.6648693

1.4 Modelo

Finalmente vamos juntar as funções auxiliares implementadas acima para construir um modelo. O modelo será treinado na base de dados SIGNS.

Já implementamos, na parte anterior do curso, a função `random_mini_batches()`. Lembre-se de que esta função retorna uma lista de mini-batches.

Exercício: Complete a função abaixo.

O modelo deverá:

- criar os placeholders
- inicializar os parâmetros
- fazer a propagação para frente
- computar o custo
- criar um otimizador

Vamos criar uma sessão e executar um loop por num_epochs, obter os mini-batches e, então, otimizar a função para cada mini-batch. Dica para inicializar as variáveis (https://www.tensorflow.org/api_docs/python/tf/global_variables_initializer)

In [38]:
```
# FUNÇÃO DE AVALIAÇÃO: model

def model(X_train, Y_train, X_test, Y_test, learning_rate = 0.0001,
          num_epochs = 3000, minibatch_size = 64, print_cost = True
):
    """
    Implementa uma convNet de três camadas usando:
    CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATT
EN -> FULLYCONNECTED

    Argumentos:
    X_train -- conjunto de treinamento, no formato (None, 64, 64, 3
)
    Y_train -- saídas do conjunto de treinamento, no formato (None,
n_y = 6)
    X_test -- conjunto de teste, no formato (None, 64, 64, 3)
    Y_test -- saídas do conjunto de teste, no formato (None, n_y =
6)
    learning_rate -- taxa de aprendizado da otimização
    num_epochs -- númerode épocas do loop de otimização
    minibatch_size -- tamanho do mini-batch
    print_cost -- Se verdade imprime o custo a cada 100 épocas

    Retorna:
    train_accuracy -- número real que indica a precisão no conjunto
de treinamento (X_train)
    test_accuracy -- número real que indica a precisão no conjunto
```

```
de teste (X_test)
    parameters -- parâmetros aprendidos pelo modelo. Podem ser util
izados para previsão.
    """

    ops.reset_default_graph()                         # para poder
rodar o modelo várias vezes sem sobrescrever variáveis do tf
    tf.set_random_seed(1)                             # para manter
os resultados consistentes (semente do tensorflow)
    seed = 3                                          # para manter
os resultados consistentes (semente do numpy)
    (m, n_H0, n_W0, n_C0) = X_train.shape
    n_y = Y_train.shape[1]
    costs = []                                        # para armaze
nar os valores do custo

    # Criar Placeholders com os formatos corretos
    ### INÍCIO DO CÓDIGO ### (1 linha)
    X, Y = create_placeholders(n_H0, n_W0, n_C0, n_y)
    ### TÉRMINO DO CÓDIGO ###

    # Inicializa os parâmetros
    ### INÍCIO DO CÓDIGO ### (1 linha)
    parameters = initialize_parameters()
    ### TÉRMINO DO CÓDIGO ###

    # Propagação para frente: Construa a propagação para frente no
grafo de computação do tensorFlow
    ### INÍCIO DO CÓDIGO ### (1 linha)
    Z3 = forward_propagation(X, parameters)
    ### TÉRMINO DO CÓDIGO ###

    # Função de custo: Adicione a função de custo ao grafo de compu
tação do tensorFlow
    ### INÍCIO DO CÓDIGO ### (1 linha)
    cost = compute_cost(Z3, Y)
    ### TÉRMINO DO CÓDIGO ###

    # Propagação para trás: Defina o otimizador a ser utilizado no
tensorFlow. Use o AdamOptimizer para minimizar o custo.
    ### INÍCIO DO CÓDIGO ### (1 linha)
    optimizer = tf.train.AdamOptimizer(learning_rate = learning_rat
e).minimize(cost)
    ### TÉRMINO DO CÓDIGO ###

    # Inicialize todas as variáveis globalmente
    init = tf.global_variables_initializer()

    # Inicie a sessão para computar  grafo do tensorFlow
    with tf.Session() as sess:

        # Executa  inicialização
        sess.run(init)
```

```
        # Faz o loop de treinamento
        for epoch in range(num_epochs):

            minibatch_cost = 0.
            # número de minibatches de tamanho minibatch_size no co
njunto de treinamento
            num_minibatches = int(m / minibatch_size)
            seed = seed + 1
            minibatches = random_mini_batches(X_train, Y_train, min
ibatch_size, seed)

            for minibatch in minibatches:

                # Seleciona um minibatch
                (minibatch_X, minibatch_Y) = minibatch
                # IMPORTANTE: A linha que executa o grafo de comput
ação em um minibatch.
                # Execute a sessão para otimizar o custo, o feed_di
ct deve conter um minibatch para (X,Y).
                ### INÍCIO DO CÓDIGO ### (1 linha)
                _ , temp_cost = sess.run([optimizer,cost], feed_dic
t ={X: minibatch_X, Y: minibatch_Y})
                ### TÉRMINO DO CÓDIGO ###

                minibatch_cost += temp_cost / num_minibatches


            # Imprime o custo a cada 100 épocas.
            if print_cost == True and epoch % 100 == 0:
                print ("Custo após a época %i: %f" % (epoch, miniba
tch_cost))
            if print_cost == True and epoch % 1 == 0:
                costs.append(minibatch_cost)


        # plota o custo
        plt.plot(np.squeeze(costs))
        plt.ylabel('custo')
        plt.xlabel('interações (por dez)')
        plt.title("taxa de aprendizado =" + str(learning_rate))
        plt.show()

        # Calcula as predições corretas
        predict_op = tf.argmax(Z3, 1)
        correct_prediction = tf.equal(predict_op, tf.argmax(Y, 1))

        # Calcula a precisão no conjunto de teste
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, "floa
t"))
        print(accuracy)
        train_accuracy = accuracy.eval({X: X_train, Y: Y_train})
        test_accuracy = accuracy.eval({X: X_test, Y: Y_test})
        print("Precisão no treinamento:", train_accuracy)
```

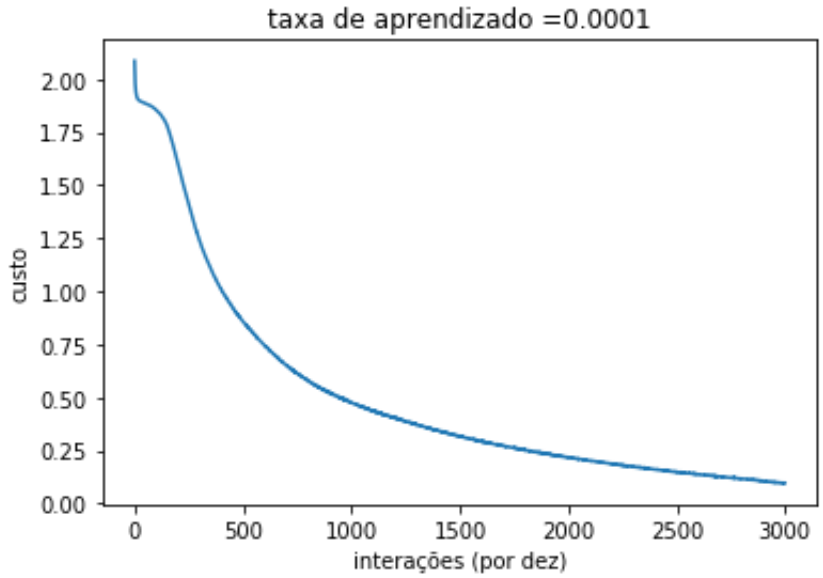
```
        print("Precisão no teste:", test_accuracy)

        return train_accuracy, test_accuracy, parameters
```

Execute a célula abaixo para treinar seu modelo por 3000 épocas. Verifique se o custo após a época 0 e 100 estão próximos. Senão estiverem interrompa o processamento e verifique o seu código!

O processo de treinamento é demorado. Pode ir tomar um café.

In [41]:
```
_, _, parameters = model(X_train, Y_train, X_test, Y_test)
```

```
Custo após a época 0: 2.086161
Custo após a época 100: 1.855229
Custo após a época 200: 1.603419
Custo após a época 300: 1.239312
Custo após a época 400: 1.007343
Custo após a época 500: 0.857656
Custo após a época 600: 0.746005
Custo após a época 700: 0.651712
Custo após a época 800: 0.578634
Custo após a época 900: 0.522095
Custo após a época 1000: 0.475677
Custo após a época 1100: 0.439807
Custo após a época 1200: 0.404230
Custo após a época 1300: 0.373715
Custo após a época 1400: 0.344911
Custo após a época 1500: 0.317373
Custo após a época 1600: 0.293020
Custo após a época 1700: 0.272584
Custo após a época 1800: 0.250823
Custo após a época 1900: 0.236341
Custo após a época 2000: 0.217011
Custo após a época 2100: 0.202638
Custo após a época 2200: 0.186904
Custo após a época 2300: 0.174336
Custo após a época 2400: 0.160785
Custo após a época 2500: 0.148457
Custo após a época 2600: 0.137603
Custo após a época 2700: 0.127130
Custo após a época 2800: 0.117846
Custo após a época 2900: 0.105421
```




```
Tensor("Mean_1:0", shape=(), dtype=float32)
Precisão no treinamento: 0.9898148
Precisão no teste: 0.89166665
```

Saída esperada: os seus resultados podem não bater com estes perfeitamente, mas devem estar próximos e a função de custo deve decrescer.

Custo após a época 0 =	2.086161
Custo após a época 100 =	1.855311
Precisão no treinamento =	0.9675926
Precisão no teste =	0.9

Parabéns! Você terminou a tarefa e construiu um modelo que reconhece sinais da base de dados SIGN com uma precisão de 90% no conjunto de teste. Se você quiser, brinque com os parâmetros. Note que o resultado obtido foi superior ao resultado da rede totalmente conectada feito anteriormente com esta mesma base de dados.

In [40]:
```
fname = "images/thumbs_up.jpg"
image = np.array(ndimage.imread(fname, flatten=False))
my_image = scipy.misc.imresize(image, size=(64,64))
plt.imshow(my_image)
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: De
precationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0.
Use ``matplotlib.pyplot.imread`` instead.

/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: De
precationWarning: `imresize` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.
2.0.
Use ``skimage.transform.resize`` instead.
  This is separate from the ipykernel package so we can avoid doin
g imports until
```

Out[40]: <matplotlib.image.AxesImage at 0x1c5da0cb00>