Building cat/not-a-cat classifier (Deep Neural Network for Image Classification)

- Build and train a deep L-layer neural network, and apply it to supervised learning

first step: Building a deep neural network for image classification.

- Using non-linear units like ReLU to improve our model
- Building a deeper neural network (with more than 1 hidden layer)
- Implementing an easy-to-use neural network class

Here's an outline of the steps (first step):

- Initialize the parameters an L-layer neural network
- Implement the forward propagation module
- The ACTIVATION function is: relu/sigmoid
- Stack the [LINEAR->RELU] forward function L-1 time (for layers 1 through L-1) and add a [LINEAR->SIGMOID] at the end (for the final layer L, resulting in $Z^{[l]}$). This gives you a new L_model_forward function.
- Compute the loss
- Implement the backward propagation module
- The gradient of the ACTIVATION function is: relu_backward/sigmoid_backward
- Stack [LINEAR->RELU] backward L-1 times and add [LINEAR->SIGMOID] backward in a new L_model_backward function
- Finally, update the parameters

In [18]:
```
import time
import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage
from dnn_app_utils_v3 import *
from dnn_app_utils_v3 import load_data
from public_tests import *

%matplotlib inline
plt.rcParams['figure.figsize'] = (5.0, 4.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

%load_ext autoreload
%autoreload 2

np.random.seed(1)
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

1- Load and Process the Dataset

- a training set of `m_train` images labelled as cat (1) or non-cat (0)
- a test set of `m_test` images labelled as cat and non-cat
- each image is of shape (num_px, num_px, 3) where 3 is for the 3 channels (RGB).

In [19]:
```
train_x_orig, train_y, test_x_orig, test_y, classes = load_data()
```

In [20]:
```
# Explore our dataset
m_train = train_x_orig.shape[0]
num_px = train_x_orig.shape[1]
m_test = test_x_orig.shape[0]

print ("Number of training examples: " + str(m_train))
print ("Number of testing examples: " + str(m_test))
print ("Each image is of size: (" + str(num_px) + ", " + str(num_px) + ", 3)")
print ("train_x_orig shape: " + str(train_x_orig.shape))
print ("train_y shape: " + str(train_y.shape))
print ("test_x_orig shape: " + str(test_x_orig.shape))
print ("test_y shape: " + str(test_y.shape))
```

```
Number of training examples: 209
Number of testing examples: 50
Each image is of size: (64, 64, 3)
train_x_orig shape: (209, 64, 64, 3)
train_y shape: (1, 209)
test_x_orig shape: (50, 64, 64, 3)
test_y shape: (1, 50)
```

In [21]:
```
# Reshape the training and test examples
#combining three channels into a vector (red, blue, green)
#standardize the images before feeding them to the network
train_x_flatten = train_x_orig.reshape(train_x_orig.shape[0], -1).T   # The "-1" makes r
test_x_flatten = test_x_orig.reshape(test_x_orig.shape[0], -1).T

# Standardize data to have feature values between 0 and 1.
train_x = train_x_flatten/255.
test_x = test_x_flatten/255.

print ("train_x's shape: " + str(train_x.shape))
print ("test_x's shape: " + str(test_x.shape))
```

```
train_x's shape: (12288, 209)
test_x's shape: (12288, 50)
```

2- Model Architecture

L-layer Deep Neural Network:

- The model can be summarized as: [LINEAR -> RELU] $\times$ (L-1) -> LINEAR -> SIGMOID

1. Initializing parameters / Define hyperparameters
2. Loop for num_iterations: a. Forward propagation b. Compute cost function c. Backward propagation d. Update parameters (using parameters, and grads from backprop)
3. Using trained parameters to predict labels

In [22]:
```
### CONSTANTS ###
layers_dims = [12288, 20, 7, 5, 1] #  4-layer model
```

In [23]:
```
# L_layer_model
```

```
def L_layer_model(X, Y, layers_dims, learning_rate = 0.0075, num_iterations = 3000, prin
    """
    Implements a L-layer neural network: [LINEAR->RELU]*(L-1)->LINEAR->SIGMOID.

    Arguments:
    X -- input data, of shape (n_x, number of examples)
    Y -- true "label" vector (containing 1 if cat, 0 if non-cat), of shape (1, number of
    layers_dims -- list containing the input size and each layer size, of length (number
    learning_rate -- learning rate of the gradient descent update rule
    num_iterations -- number of iterations of the optimization loop
    print_cost -- if True, it prints the cost every 100 steps

    Returns:
    parameters -- parameters learnt by the model. They can then be used to predict.
    """

    np.random.seed(1)
    costs = []                         # keep track of cost

    # Parameters initialization.
    parameters = initialize_parameters_deep(layers_dims)

    # Loop (gradient descent)
    for i in range(0, num_iterations):

        # Forward propagation: [LINEAR -> RELU]*(L-1) -> LINEAR -> SIGMOID.
        AL, caches = L_model_forward(X, parameters)
        cost = compute_cost(AL, Y)

        # Backward propagation
        grads = L_model_backward(AL, Y, caches)

        # Update parameters.
        parameters = update_parameters(parameters, grads, learning_rate)

        # Print the cost every 100 iterations
        if print_cost and i % 100 == 0 or i == num_iterations - 1:
            print("Cost after iteration {}: {}".format(i, np.squeeze(cost)))
        if i % 100 == 0 or i == num_iterations:
            costs.append(cost)

    return parameters, costs
```

In [24]:
```
parameters, costs = L_layer_model(train_x, train_y, layers_dims, num_iterations = 1, pri

print("Cost after first iteration: " + str(costs[0]))

L_layer_model_test(L_layer_model)
```

```
Cost after iteration 0: 0.7717493284237686
Cost after first iteration: 0.7717493284237686
Cost after iteration 1: 0.707070900891257
Cost after iteration 1: 0.707070900891257
Cost after iteration 1: 0.707070900891257
Cost after iteration 2: 0.7063462654190897
 All tests passed.
```

3 - Train the model

training your model as a 4-layer neural network, and the cost should decrease on every iteration.

In [25]:
```
parameters, costs = L_layer_model(train_x, train_y, layers_dims, num_iterations = 2500,
```

```
Cost after iteration 0: 0.7717493284237686
Cost after iteration 100: 0.6720534400822914
Cost after iteration 200: 0.6482632048575212
Cost after iteration 300: 0.6115068816101354
Cost after iteration 400: 0.5670473268366111
Cost after iteration 500: 0.54013766345478
Cost after iteration 600: 0.5279299569455267
Cost after iteration 700: 0.46547737717668514
Cost after iteration 800: 0.3691258524959279
Cost after iteration 900: 0.39174697434805344
Cost after iteration 1000: 0.3151869888600617
Cost after iteration 1100: 0.2726998441789385
Cost after iteration 1200: 0.23741853400268137
Cost after iteration 1300: 0.19960120532208647
Cost after iteration 1400: 0.18926300388463305
Cost after iteration 1500: 0.16118854665827753
Cost after iteration 1600: 0.14821389662363316
Cost after iteration 1700: 0.13777487812972944
Cost after iteration 1800: 0.1297401754919012
Cost after iteration 1900: 0.12122535068005211
Cost after iteration 2000: 0.1138206066863371
Cost after iteration 2100: 0.10783928526254133
Cost after iteration 2200: 0.10285466069352679
Cost after iteration 2300: 0.10089745445261787
Cost after iteration 2400: 0.09287821526472397
Cost after iteration 2499: 0.088439943441702
```

In [26]:
```
pred_train = predict(train_x, train_y, parameters)
```

```
Accuracy: 0.9856459330143539
```

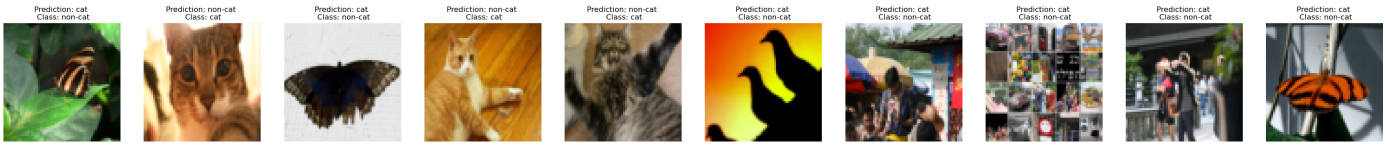
In [27]:
```
pred_test = predict(test_x, test_y, parameters)
```

```
Accuracy: 0.8
```

4 - Results Analysis

The following code will show a few mislabeled images.

In [28]:
```
print_mislabeled_images(classes, test_x, test_y, pred_test)
```



A few types of images the model tends to do poorly on include:

- Cat body in an unusual position
- Cat appears against a background of a similar color
- Unusual cat color and species
- Camera Angle
- Brightness of the picture
- Scale variation (cat is very large or small in image)