```
import numpy as np
import h5py
import pickle
import scipy
from scipy import ndimage



def load_data():
"""
Load Data Function and training data are taken from coursera's machine learning course.
"""
    train_dataset = h5py.File('datasets/train_catvnoncat.h5', "r")
    train_x = np.array(train_dataset["train_set_x"][:])
    train_y = np.array(train_dataset["train_set_y"][:])

    test_dataset = h5py.File('datasets/test_catvnoncat.h5', "r")
    test_x = np.array(test_dataset["test_set_x"][:])
    test_y = np.array(test_dataset["test_set_y"][:])



    train_y = train_y.reshape((1, train_y.shape[0]))
    test_y = test_y.reshape((1, test_y.shape[0]))

    return train_x, train_y, test_x, test_y

def save_obj(obj, name):
    with open(name + '.pkl', 'wb') as f:
        pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)
        print("Saved file")
```

```
def sigmoid(Z):
    return 1 / (1 + np.exp(-Z))

def initialize(dimensions):
    w = np.zeros(shape=(dimensions, 1))
    b = 0

    assert(w.shape == (dimensions, 1))
    assert(isinstance(b, float) or isinstance(b, int))

    return w, b

def cost(A, Y, x_shape):
    return (- 1 / x_shape)*np.sum(Y*np.log(A) + (1 - Y)*(np.log(1 - A)))

def forward_propagation(w, b, X, Y):
    x_shape= X.shape[1]

    A = sigmoid(np.dot(w.T, X) + b)

    return np.squeeze(cost(A, Y, x_shape)), A

def backward_propagation(w, b, X, A, Y):
    x_shape = X.shape[1]
    dw = (1 / x_shape) * np.dot(X, (A - Y).T)
    db = (1 / x_shape) * np.sum(A - Y)

    return dw, db
```

```
def gradient_descent(w, b, X, Y, iterations, alpha):
    costs = []

    for i in range(iterations):
        cost, A = forward_propagation(w, b, X, Y)

        dw, db = backward_propagation(w, b, X, A, Y)

        w = w- alpha * dw
        b = b- alpha * db

        if i % 100 == 0:
            costs.append(cost)
            print ("The cost after {} steps is: {}".format(i, cost))

    return w, b, dw, db, costs
```

```
def predict(w, b, X):
    x_shape = X.shape[1]
    prediction  = np.zeros((1, x_shape))


    w = w.reshape(X.shape[0], 1)
    A = np.dot(w.T, X)+b
    A = sigmoid(A)


    for i in range(A.shape[1]):
```

```
        A[0, i] = 1 if A[0,i] > 0.5 else 0

    return A


def predict_accuracy(X_train, Y_train, X_test, Y_test, parameters):
    w = parameters["w"]
    b = parameters["b"]

    test_prediction = predict(w, b, X_test)
    train_prediction = predict(w, b, X_train)

    test_accuracy = 100 - (np.mean(np.abs(test_prediction - Y_test)) * 100)
    train_accuracy = 100 - (np.mean(np.abs(train_prediction - Y_train)) * 100)

    return test_accuracy, train_accuracy

def model(x_train, y_train, iterations=2000, alpha=0.005):

    w, b = initialize(x_train.shape[0])


    w, b, dw, db, costs = gradient_descent(w, b, x_train, y_train, iterations, alpha)
    print("Model finished")

    parameters = {"w": w, "b" : b, "costs" : costs, "alpha" : alpha, "iterations": iterations}

    return parameters
```

```
x_train, y_train, x_test, y_test = load_data()

num_px = x_train.shape[1]

x_train = x_train.reshape(x_train.shape[0], -1).T
x_test = x_test.reshape(x_test.shape[0], -1).T
print("Reshaped train data of size: " + str(x_train.shape[0]))
print("Reshaped test data of size: " + str(x_test.shape[0]) + '\n')



x_train = x_train / 255.
x_test = x_test / 255.



parameters = model(x_train, y_train, iterations = 2000, alpha = 0.005)

test_accuracy, train_accuracy = predict_accuracy(x_train, y_train, x_test, y_test, parameters)

print("The train set had an accuracy of: {}".format(train_accuracy))
print("The test set had an accuracy of: {}".format(test_accuracy))

save_obj(parameters, "datasets/parameters")
```

```
# Test your own image
image = "image.jpg"

image = np.array(ndimage.imread(image, flatten = False))
image = scipy.misc.imresize(image, size=(num_px, num_px)).reshape((1, num_px ** 2 * 3)).T

numprediction = predict(parameters["w"], parameters["b"], image)
prediction = "Not a cat" if int(np.squeeze(numprediction)) == 0 else "A cat"

print("The algorithm predicts: " + prediction, np.squeeze(numprediction))
```