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))