Sri Lanka Institute Of Information Technology

Image Classification Model using Deep Learning Neural network

Project Report

Deep Learning

IT16001480

H.K.D.C.Jayalath

Table of Contents

List of Figures

1. Introduction

- Can you identify the below image?

Figure 1: Image of a cat

You will have directly recognized it .It's a cat. Take a step back and discover how you came to this conclusion. You were showed an image and you classified the class it suitable to (a cat in this example).This is what image classification.

There are theoretically n number of clusters in which a particular image can be classified. Manually testing and classifying images is a very uninteresting process. The job becomes near intolerable when we're tackled with a massive number of images, approximately 50,000 or even 100,000. How valuable would it be if we could systematize this entire procedure and quickly label images per their corresponding class?

In today world, Self-driving cars are an extreme example to know where image classification is used in the real-life. To let autonomous driving, we can form an image classification model that identify many objects, such as vehicles, persons, moving stuffs, etc. on the streets.

2. Proposed Solution

According to this assignment I have trained a deep learning neural network to classify cats or non-cats using a dataset. For that I have created a deep learning neural network classifier model using supervised learning to predict a specified image is a cat or a non-cat.
Let's dive into how an image classification model is made, what are the essentials for it, and how it can be implemented in Python. Let's we understand how an image classification model is naturally designed. We can divide this process sketchily into 4 phases

- Loading and pre-processing Data
- Defining Model architecture
- Training the model
- Estimation of performance

3. Dataset

3.1 Description of a Dataset

Before beginning to predict whether the particular image is dropped under which group.(Cat or Non-cat),We need to find out a data set which is related to this background. For prediction purposes I have used a data set in a Kaggle learning repository. Resulting link contains "Cat vs non-Cat" data set.
The dataset contains of cats and non-cats images you'll preprocess the images, then train a deep neural network can used to classify the images as cat or non-cat image

The data set comprising:

- A training set of m_train images taken as cat (1) or non-cat (0)
- A test set of m_test images taken as cat and non-cat
- Each image is of form (num_px, num_px, 3) where 3 is for the 3 channels (RGB).

3.2 Data Preprocessing

We've got the data, but we can't precisely just stuff raw images right through our deep neural network. First, we need all of the images to be the same size. Also, the labels of "cat" and "non-cat" are not beneficial, we need them to be one-hot arrays.

```
In [5]: # Explore your 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)
```

Figure 2: Explore the dataset

3.3 Image to vector conversion

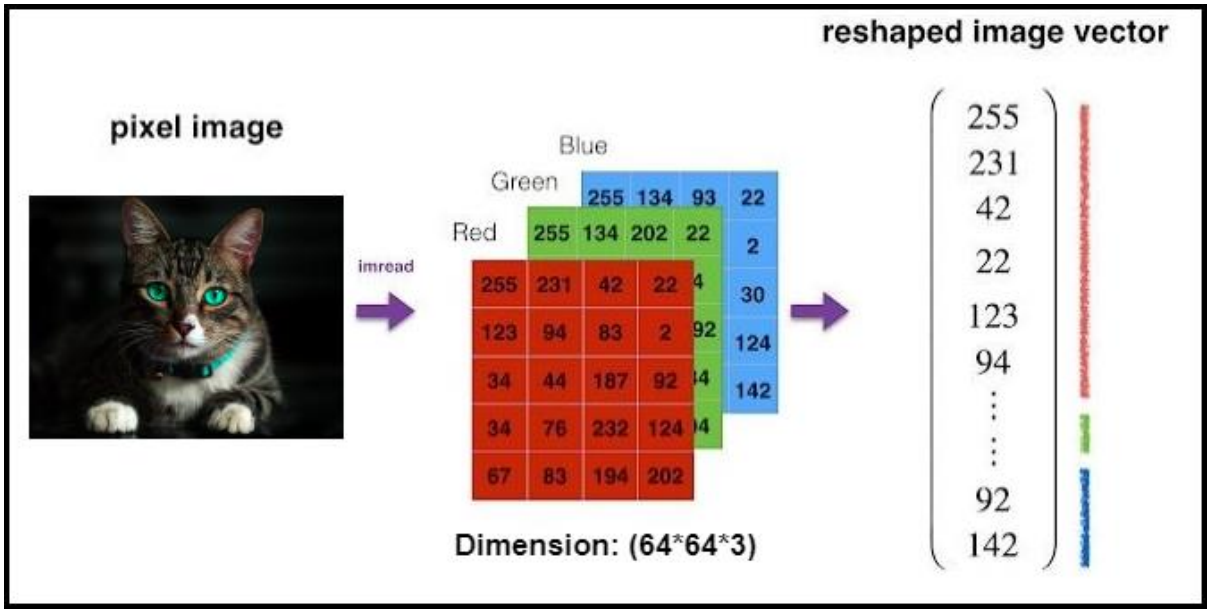


Figure 3: Image to vector conversion

The row vector (12288) has the exact same number of elements if you calculate 64*64*3==12288. In order to reshape the row vector, (12288), there are two steps required. The first step is involved with using reshape function in numpy, and the second step is elaborate with using transpose function in numpy as well. By definition from the official web

site, reshape function provides a new shape to an array without changing its data. Here, the phrase without changing its data is a significant part. Reshape operations should be delivered in extra detailed step. The following path is described in a logical concept.

Split the row vector (12288) into 3 pieces. Each part corresponds to the each channels.

- this outcomes in (3 x 4096) dimension of tensor

Split the resulting tensor from the earlier step with 64. 64 here means width of an image.

- this outcomes in (3 x 64 x 64)

In order to implement the guidelines written in logical sense in numpy, reshape function should be called in the subsequent arguments, (209, 3, 64, 64). As you noticed, reshape function doesn't spontaneously divide further when the third value (64, width) is provided. We essential to explicitly specify the value for the last value (64, height)

As typically, you reshape and standardize the images earlier feeding them to the network. The code is given in the screenshot underneath.

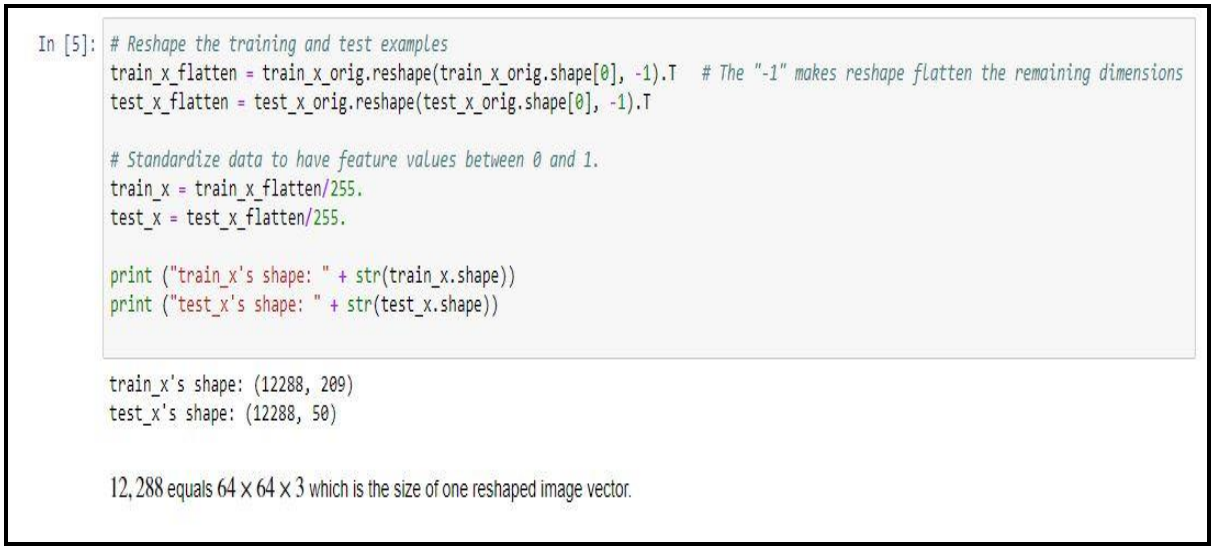


Figure 4: Reshape the training and testing data

Normalize function takes an image data, x, and returns it as a normalized Numpy array. The values in the original data is going to be converted in range of 0 to 1, wide-ranging without modification the shape of the array. A simply answer to why normalization should be accomplished is somewhat related to activation function.

```
In [7]:
def normalize(x):
    """
        argument
            - x: input image data in numpy array [64, 64, 3]
        return
            - normalized x
    """
    min_val = np.min(x)
    max_val = np.max(x)
    x = (x-min_val) / (max_val-min_val)
    return x
```

Figure 5: Normalize function

For specimen, sigmoid activation function takes an input value and outputs a novel value ranging from 0 to 1. When the input value is somewhat big, the output value easily reaches the max value 1. Likewise, when the input value is slightly small, the output value effortlessly reaches the max value 0.

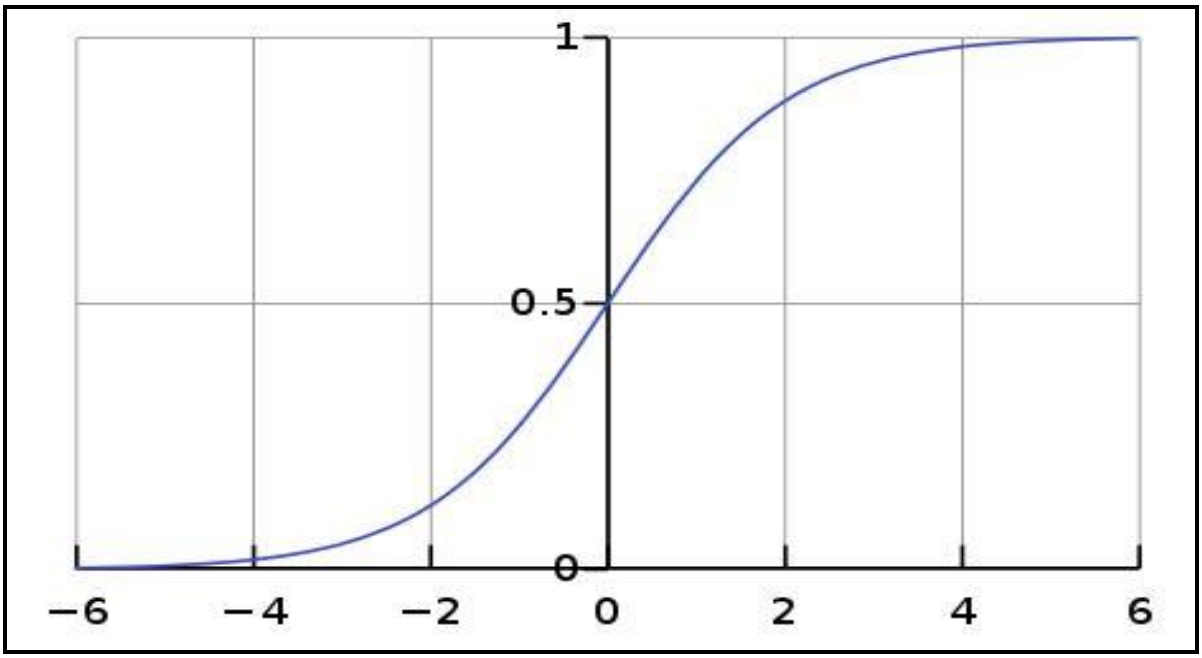


Figure 6: Sigmoid activation function graph

For one more example, ReLU activation function takes an input value and outputs a novel value ranging from 0 to infinity. When the input value is fairly large, the output value increases linearly. However, when the input value is slightly small, the output value effortlessly reaches the max value 0.

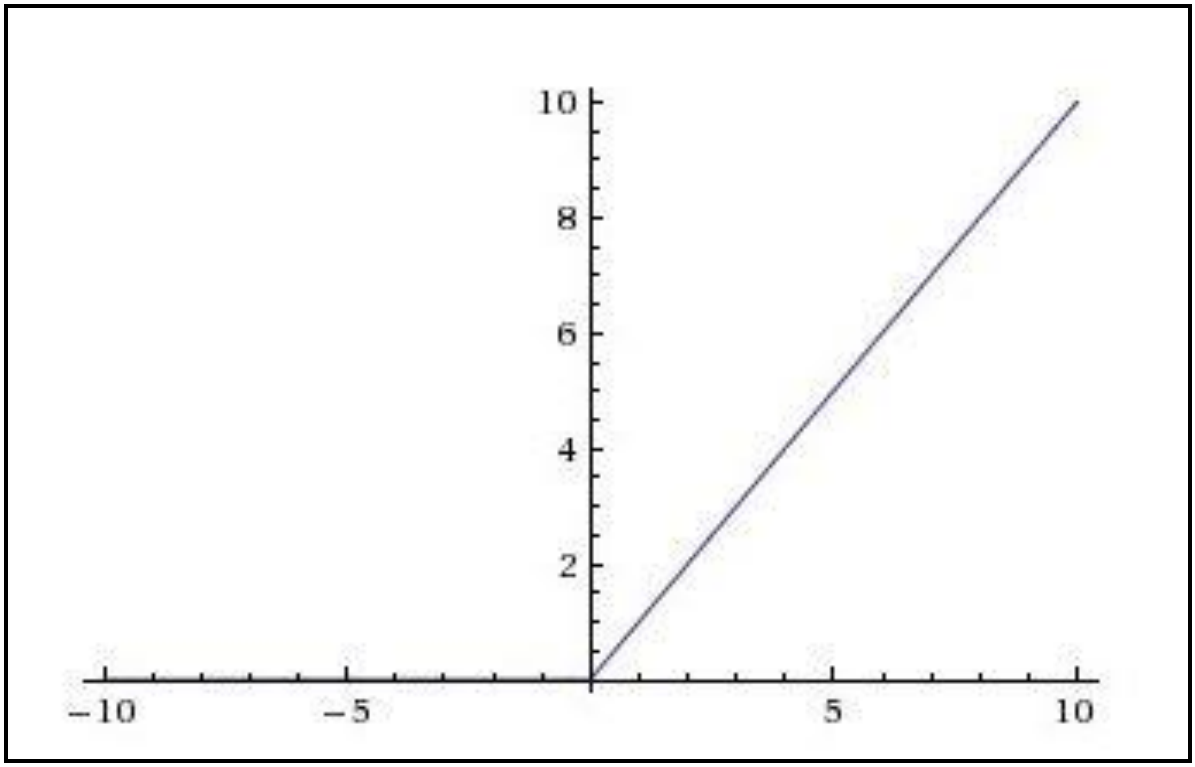


Figure 7: ReLU activation function

Now, when we consider about the image data, all values initially ranges from 0 to 255. This sounds like when it is passed into sigmoid function, the output is nearly always 1, and when it is passed into ReLu function, the output could be very large. When backpropagation procedure is performed to optimize the networks, this could lead to an exploding gradient which leads to a terrible learning steps. In order to avoid this matter, ideally, it is better let all the values be around 0 and 1.

3.5 Data Visualization

Visualizing how many are of them are cat and non-cat images in the dataset

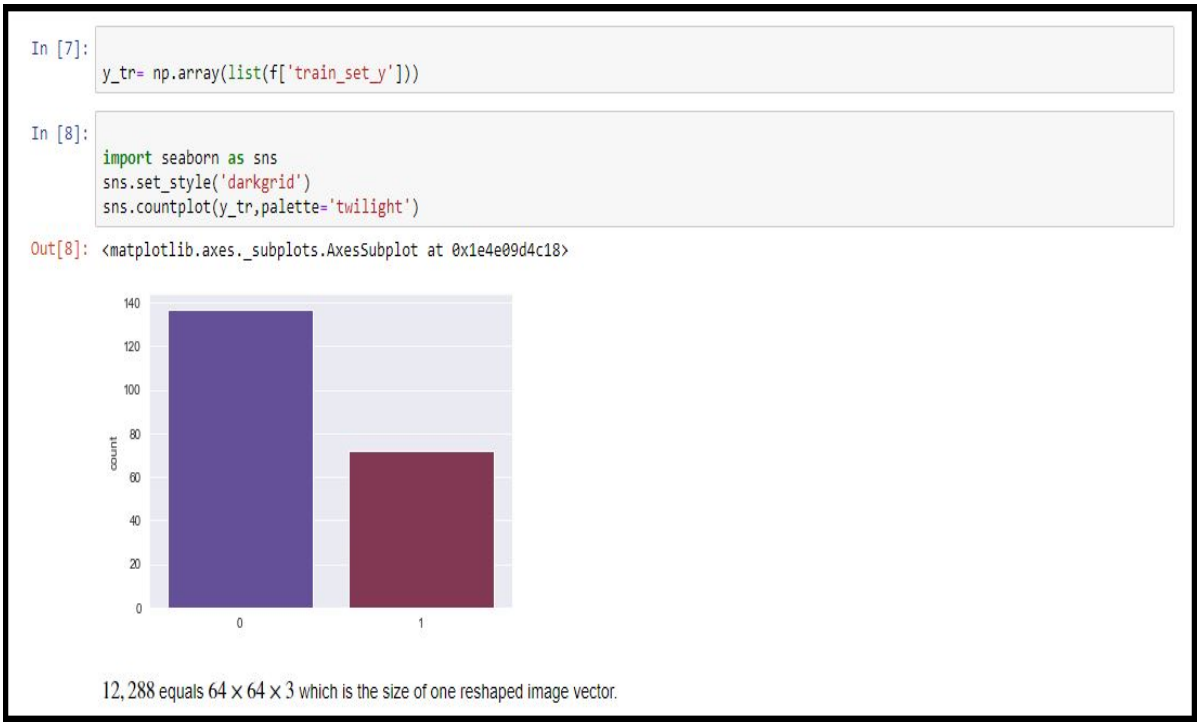


Figure 8: Data Visualization

4. Methodology

4.1 Architecture of your model

To identify cat images from non-cat images, build a deep neural network which contain of L-layered architecture. It is tough to represent an L-layer deep neural network with the above representation. However, here is a basic network representation.

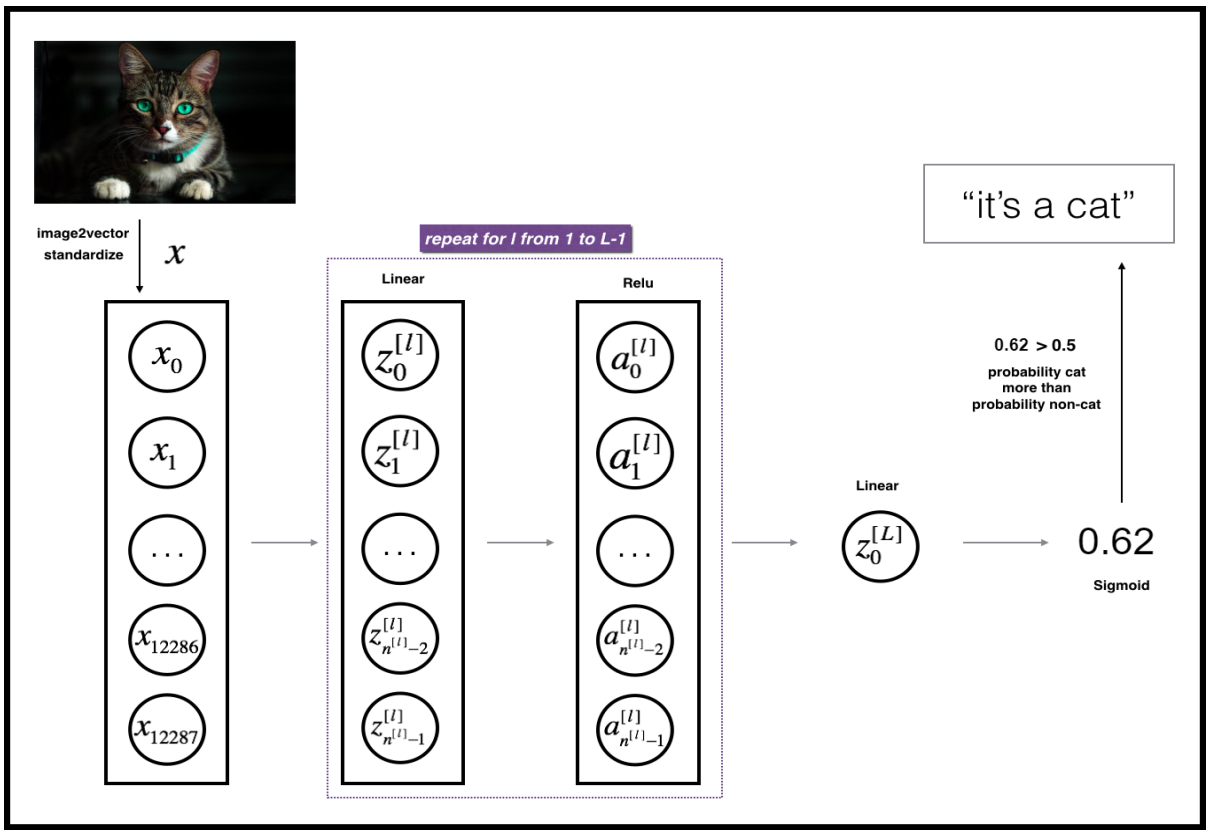


Figure 9: L-layer neural network.

The model can be shortened as: [LINEAR -> RELU] ×× (L-1) -> LINEAR -> SIGMOID

- The input is a (64,64,3) image which is flattened to a trajectory(vector) of size (12288,1).

- The conforming vector: [x0,x1,...,x12287]T is then multiplied by the weight matrix W[1]W[1] and then you add the intercept b[1]. The result is called the linear unit.

- Then, you take the relu of the linear unit. This method could be repeated numerous times for each (W[l],b[l]) depending on the model architecture.

- Finally, you take the sigmoid of the last linear unit. If it is greater than 0.5, you categorize it to be a cat.

4.2 Implementation

As typically I will follow the Deep Learning methodology to build the model:

1. Initialize parameters / Define hyper parameters

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. Use trained parameters to predict labels

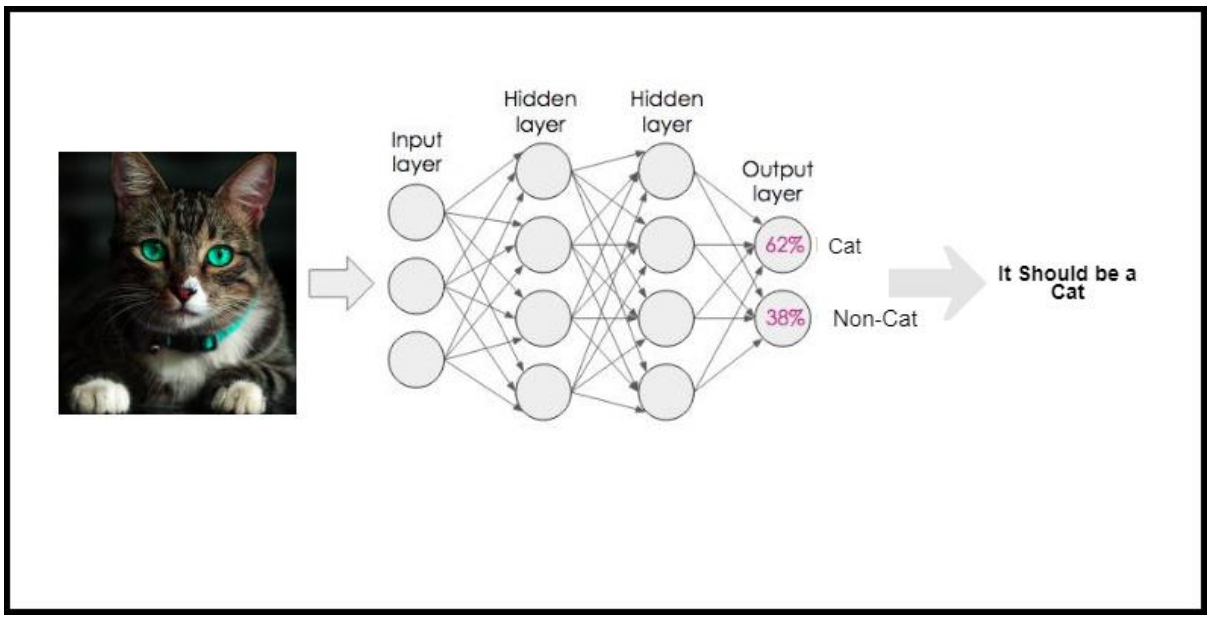


Figure 10: Layered Architecture of neural network

Implementation of layered Model

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

Figure 11: Layered model dimension

```
def initialize_parameters_deep(layers_dims):
...
return parameters

def L_model_forward(X, parameters):
...
return AL, caches

def compute_cost(AL, Y):
...
return cost

def L_model_backward(AL, Y, caches):
...
return grads

def update_parameters(parameters, grads, learning_rate):
...
return parameters
```

Develop an L-layer deep neural network: [LINEAR->RELU]*(L-1)→LINEAR→ SIGMOID.
Arguments:

Data (features), numpy array of method
(Amount of instances, number_px * number_px * 3) → X

If true "label" trajectory (holding 0 if cat, 1 if non-cat), of form (1, number of samples) →Y

This is contain list of details with the input size and correspondingly layer size, of length and width (amount of layers + 1). → Layer_dimension

This is demonstrate the learning rate of the gradient descent update regulation → Learning rate

This is indicate an often indicate quantity of the optimization cycles →Number of Iteration

If true, it subjects the cost for all hundred (100) phases → Print cost

Returns:

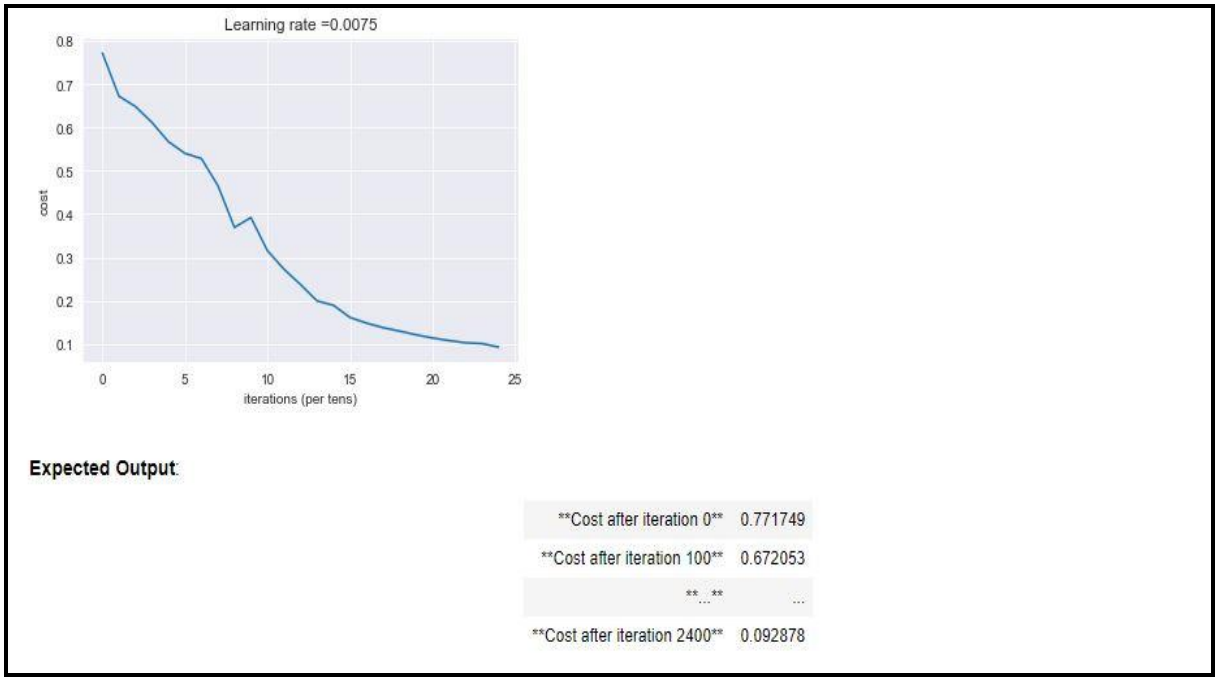
This is parameters learnt by the trained model. They can then be used to forecast. → Parameters

```
In [7]: # GRADED FUNCTION: L_layer_model
def L_layer_model(X, Y, layers_dims, learning_rate = 0.0075, num_iterations = 3000, print_cost=False):#lr was 0.009

    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)
        # Compute cost.
        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 training example
        if print_cost and i % 100 == 0:
            print ("Cost after iteration %i: %f" %(i, cost))
        if print_cost and i % 100 == 0:
            costs.append(cost)
    # plot the cost
    plt.plot(np.squeeze(costs))
    plt.ylabel('cost')
    plt.xlabel('iterations (per tens)')
    plt.title("Learning rate =" + str(learning_rate))
    plt.show()
    return parameters
```

Figure 12: Implement the deep neural network

In here to categorize Cat vs Non-Cats image we will train the model as a 4-layer neural network. We can train your model by running the below cell. The cost should reduce on every iteration. It may take approximately up to 5 minutes to run 2500 iterations. Check if the "Cost after iteration 0" matches the predictable output below.

```
In [11]: parameters = L_layer_model(train_x, train_y, layers_dims, num_iterations = 2500, print_cost = True)

Cost after iteration 0: 0.771749
Cost after iteration 100: 0.672053
Cost after iteration 200: 0.648263
Cost after iteration 300: 0.611507
Cost after iteration 400: 0.567047
Cost after iteration 500: 0.540138
Cost after iteration 600: 0.527930
Cost after iteration 700: 0.465477
Cost after iteration 800: 0.369126
Cost after iteration 900: 0.391747
Cost after iteration 1000: 0.315187
Cost after iteration 1100: 0.272700
Cost after iteration 1200: 0.237419
Cost after iteration 1300: 0.199601
Cost after iteration 1400: 0.189263
Cost after iteration 1500: 0.161189
Cost after iteration 1600: 0.148214
Cost after iteration 1700: 0.137775
Cost after iteration 1800: 0.129740
Cost after iteration 1900: 0.121225
Cost after iteration 2000: 0.113821
Cost after iteration 2100: 0.107839
Cost after iteration 2200: 0.102855
Cost after iteration 2300: 0.100897
Cost after iteration 2400: 0.092878
```



Expected Output:

Cost after iteration 0	0.771749
Cost after iteration 100	0.672053
...	...
Cost after iteration 2400	0.092878

Figure 13: Trained the neural network model

Train set and Test set outputs

```
In [12]: print("train_x shape = "+str(train_x.shape))
         print("train_y shape = "+str(train_y.shape))
         pred_train = predict(train_x, train_y, parameters)

         train_x shape = (12288, 209)
         train_y shape = (1, 209)
         Accuracy: 0.9856459330143539
```

Expected Output:

Train Accuracy	0.985645933014

```
In [13]: print("test_x shape = "+str(test_x.shape))
         print("test_y shape = "+str(test_y.shape))
         pred_test = predict(test_x, test_y, parameters)

         test_x shape = (12288, 50)
         test_y shape = (1, 50)
         Accuracy: 0.8
```

Expected Output:

Test Accuracy	0.8

Figure 14: Train set and test set outputs

The above images show the model perform against the training set and test data set. It seems that your 4-layer neural network has decent performance (80%).This is satisfied performance for this task.

5. Results Analysis

Primary, let's take an aspect at some images the L-layer model labeled wrongly. This will show a little mislabeled images.

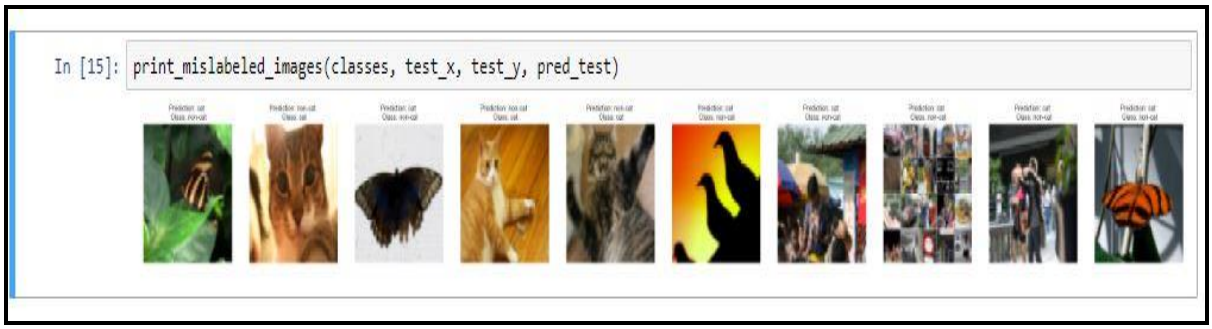


Figure 15: Mislabeled images

A few type of images the model have a tendency to do poorly on include:

- unusual position of the image
- background color and cats appearance color is mostly similar
- Unusual cat color and types
- Different and unusual camera angle
- Brightness level of the portrait

5.1 Test with your own image

To check how the model is performing we can test with our own portrait.

Figure 16: Testing the model

Output

Figure 17: Testing the model output

6. Evaluation

6.1 Future Works

- We can train this model with different training set and test data set.
- In future we can train this dataset with some other different deep learning Method. After that we can improve accuracy of our algorithm.
- Try different dataset with different features to do prediction.

6.2 Discussion

Under this document I demonstrate how to do image classification using the Jupyter Notebook. In the process, we covered pre-processing of the image, construct the deep neural network layer and testing the model for an input image.
Exactness of the model can be further developed by 'Hyper parameter' tuning like experimenting with 'Adam optimizer' parameters, adding additional layers to the model, etc.

7. Appendix

7.1 Code

Importing Libraries

```
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 *
import matplotlib.pyplot as plt
import seaborn as sns


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

%load_ext autoreload
%autoreload 2
np.random.seed(1)
```

Load the dataset

```
train_x_orig, train_y, test_x_orig, test_y, classes = load_data()
filename = 'train_catvnoncat.h5'
f = h5py.File(filename, 'r')
list(f.keys())
data = list(f['train_set_x'])
```

Example of a pictures

```
index = 10
plt.imshow(train_x_orig[index])
print ("y = " + str_(train_y_index_[0,index]) + ". It's well define" +
classes_[train_y_index[0,index_]].decode("utf-8_") +  " picture_details.")
for i in range(50,55):
    print(data[i].shape)
    plt.imshow(data[i])
    plt.show()
```

Explore your 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 ("Every portrait is of scale: (" + str(number_px_) + ", " + str(number_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))
```

Reshape the training and test examples

```
train_x_flatten = train_x_orig.reshape(train_x_orig.shape[0], -1).T  # The "-1" makes reshape flatten the remaining dimensions
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))
```

Normalization

```
def normalize(x):
    """
    argument
        - x: input image data in numpy array [64, 64, 3]
    return
        - normalized x
 """
 min_val = np.min(x)
 max_val = np.max(x)
 x = (x-min_val) / (max_val-min_val)
 return x
```

Data Visualization

```
y_tr= np.array(list(f['train_set_y']))
sns.set_style('darkgrid')
sns.countplot(y_tr,palette='twilight')
```

CONSTANTS

```
layers_dimensiion = [12288, 20, 7, 5, 1] # 4-layer model
```

GRADED FUNCTION: 4-layer_model

```
def L_layer_model_(X, Y, layers_dimensiion, learning_rate = 0.0075, num_iterations = 3000, print_cost=False)
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)
       # Compute cost.
       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 training example
if print_cost and i % 100 == 0:
    print ("Cost after iteration %i: %f" %(i, cost))
if print_cost and i % 100 == 0:
    costs.append(cost)

 # plot the cost
 plt.plot(np.squeeze(costs))
 plt.ylabel('cost')
 plt.xlabel('iterations (per tens)')
 plt.title("Learning rate =" + str(learning_rate))
 plt.show()
 return parameters
```

Train the model

```
Parameters_learn = L_layer_model_architecture_(train_x, train_y, layers_dimension_,
num_iterations = 2500, print_cost = True)
```

Train and Test set outputs

```
print("train_x shape = "+str(train_x.shape))
print("train_y shape = "+str(train_y.shape))
pred_train = predict(train_x, train_y, parameters)

print("test_x shape = "+str(test_x.shape))
print("test_y shape = "+str(test_y.shape))
pred_test = predict(test_x, test_y, parameters)
```

Find mislabeled images

```
print_mislabeled_images(classes, test_x, test_y, pred_test)
```

Test with your own image

```
my_image_ = "my_image.jpg" # modify this to the name of your portrait image file
my_label_y _= [1] # the correct class of image (1→ cat, 0 → non-cat)

fname = "images/" + my_image
image = np.array(ndimage.imread(fname, flatten=False))
my_image = scipy.misc.imresize(image,
size=(num_px,num_px)).reshape((num_px*num_px*3,1))
my_predicted_image = predict(my_image, my_label_y, parameters)
```

```
plt.imshow(image)
print ("[y] = " + _str_(np1.squeeze_(my_predicted_image_)) +", your L-layer model
forecasts"\" + classes[_int(np1.squeeze_(my_predicted_image_)),].decode_("utf-8_") +  "\"
image_cat_vs_non-cat.")
```

References

[1] - Dataset- https://www.kaggle.com/sshreshth/cat-noncat

[2] - For auto-reloading external module: http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython