NITTE EDUCATION TRUST | N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India

(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC
☎: 08258 - 281039 – 281263, Fax: 08258 – 281265

Department of Computer Science and Engineering
B.E. CSE Program Accredited by NBA, New Delhi from 1-7-2018 to 30-6-2021

Report on Mini Project

"Cat vs Non-Cat Classifier"

Course Code: 18CS605

Course Name: Machine Learning Lab

Semester: VIth Section: B

Submitted To
Mrs. Shabari Shedthi B

Asst Prof Gd II, Dept of CSE, NMAMIT

Submitted By

Name: Madhav Bhat K	USN: 4NM18CS085
Name: Manish J Bangera	USN: 4NM18CS090

Date of submission: 26/05/2021

Signature of Course Instructor

NITTE EDUCATION TRUST | N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India
(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC
☎: 08258 - 281039 – 281263, Fax: 08258 – 281265

CERTIFICATE

"Cat vs Non-Cat Classifier" is a bonafide work carried out by Madhav Bhat K (4NM18CS085) and Manish J Bangera (4NM18CS090) and in partial fulfilment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering prescribed by Visvesvaraya Technological University, Belagavi during the year 2020-2021.

It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The Mini project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.

Signature of Guide Signature of HOD

ABSTRACT

Machine learning, especially its subfield of Deep Learning, had many amazing advances in the recent years, and important research papers may lead to breakthroughs in technology that get used by billions of people. The research in this field is developing very quickly and to help our readers monitor the progress we present the list of most important recent scientific papers published since 2014

We will be covering up the concepts of using the logistic regression along with neural networks, applying forward and backward propagation and then applying them to the practice in order to build your image recognition system i.e. a cat classifier in this case .This cat classifier takes an image as an input and then based on regression and neural network techniques

TABLE OF CONTENTS

INTRODUCTION

Image classification has become one of the key pilot use-cases for demonstrating machine learning. Nowadays, there is plenty of open source frameworks that can easily train a powerful algorithm to model a large image dataset, such as Keras, Pytorch, TensorFlow, Theano, etc... But every machine learning engineer should be able to build a logistic regression model and simple ANN with a neural network mindset from scratch so he can later debug all problems related to model training and evaluation. For someone who is fairly new to this realm, it might seem very challenging at first. But we believe this is a great approach to begin understanding the fundamental building blocks behind a neural network.

In this project, we will explain how to use logistic regression and multilayer neural network to build an image classification model to distinguish cat images from non-cat images where Logistic regression is a binary classification method, whereas multilayer neural network can be used to classify multiclass images.

LITERATURE SURVEY

Classification is the frequently (most commonly) applied data mining mechanism, which explains a set of pre- classified examples to develop a (procedure) model that can (identifies or categories) classify the population (Dataset) of records at large. During Fraud detection in network and Transactional (credit) risk applications are particularly well suited to this type of analysis. This way frequently applies by decision tree or neural network- based classification algorithms. The data (categorization) classification process involves learning (Trained previously) and then classification will apply on given data. Learning process can be done by analyzing data using classification Mechanism. In classification test data are used to find out (estimate) the accuracy of the classification rules which we applied. If the accuracy is acceptable the rules can be applied to the new data tuples.
E.g.

o Classification by decision tree induction
o Bayesian Classification
o Neural Networks
o Support Vector Machines (SVM)
o Classification Based on Associations

In this project we would like to introduce how Neural Networks are comparatively better than traditional ML models which are based on probabilistic models

DESIGN

The system has been designed in the following manner:

- Importing the necessary Libraries
  - Numpy (Array Manipulation)
  - Matplotlib (Graph Visualization)
  - h5py (Dataset/Weights manipulation)
  - Scipy (To read the Images from file and convert to np array)
  - PIL (Image Manipulation

- Loading the dataset
- Visualization and pre-processing of the dataset
- Initialization of Parameters
- Calculating the cost function and its gradient
- Using an optimization algorithm (gradient descent)
- Gather all three functions above into one main model function, in the right order to form a model

Tools used:

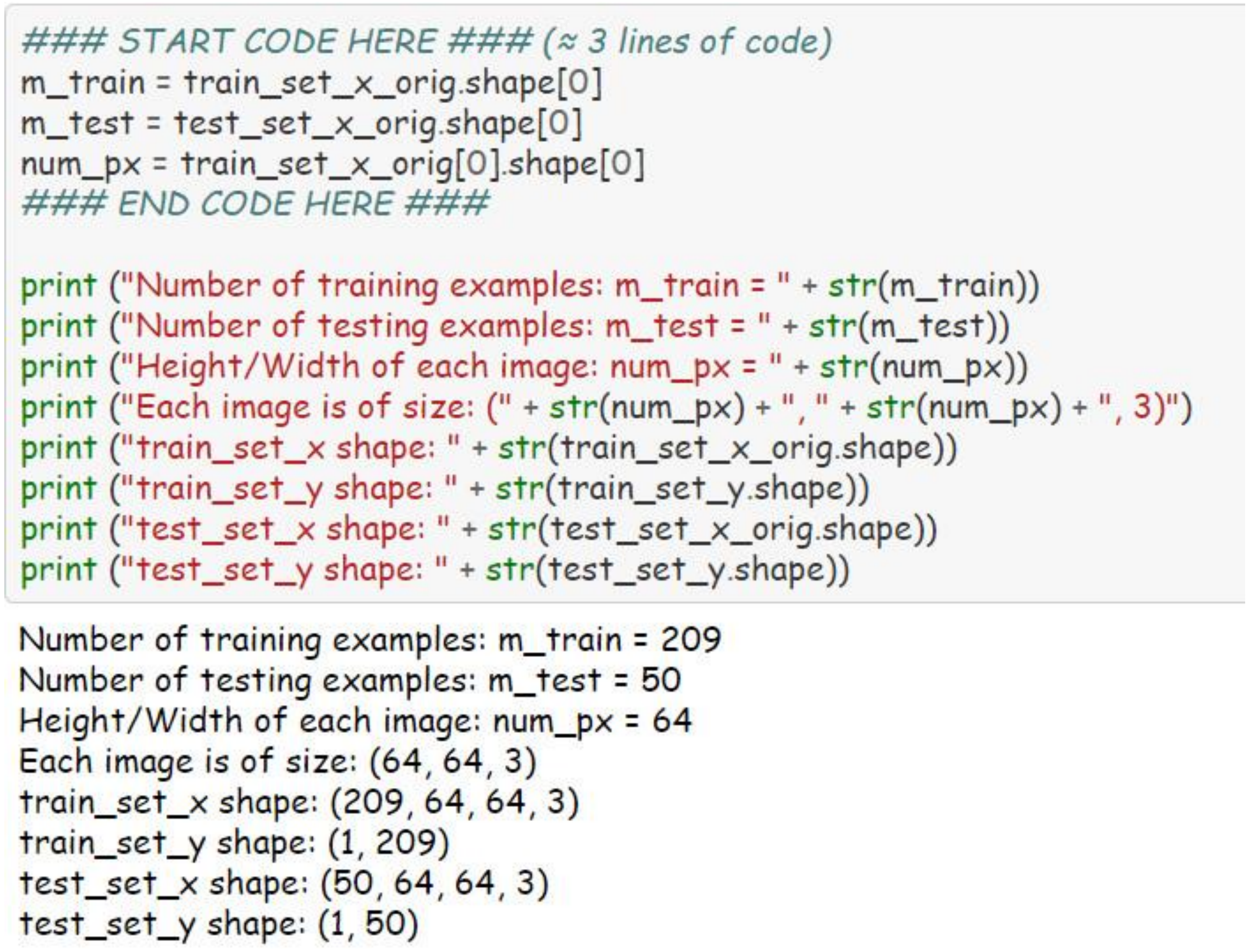
1. **Jupyter notebooks:** An advanced tool useful for creating and editing Python notebooks.

IMPLEMENTATION

The following steps are taken to build a machine learning model.

Data Exploration:

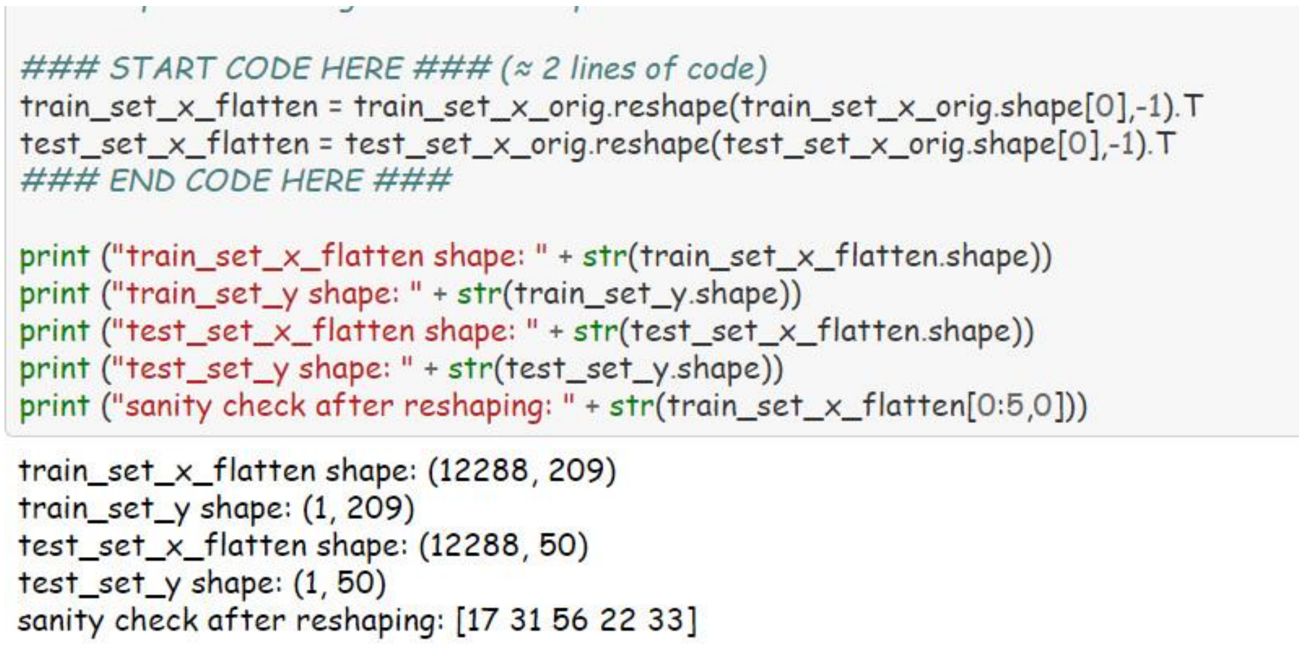
The following figures describe the image dataset that's used in this project



```
### START CODE HERE ### (≈ 3 lines of code)
m_train = train_set_x_orig.shape[0]
m_test = test_set_x_orig.shape[0]
num_px = train_set_x_orig[0].shape[0]
### END CODE HERE ###

print ("Number of training examples: m_train = " + str(m_train))
print ("Number of testing examples: m_test = " + str(m_test))
print ("Height/Width of each image: num_px = " + str(num_px))
print ("Each image is of size: (" + str(num_px) + ", " + str(num_px) + ", 3)")
print ("train_set_x shape: " + str(train_set_x_orig.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x shape: " + str(test_set_x_orig.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
```

```
Number of training examples: m_train = 209
Number of testing examples: m_test = 50
Height/Width of each image: num_px = 64
Each image is of size: (64, 64, 3)
train_set_x shape: (209, 64, 64, 3)
train_set_y shape: (1, 209)
test_set_x shape: (50, 64, 64, 3)
test_set_y shape: (1, 50)
```



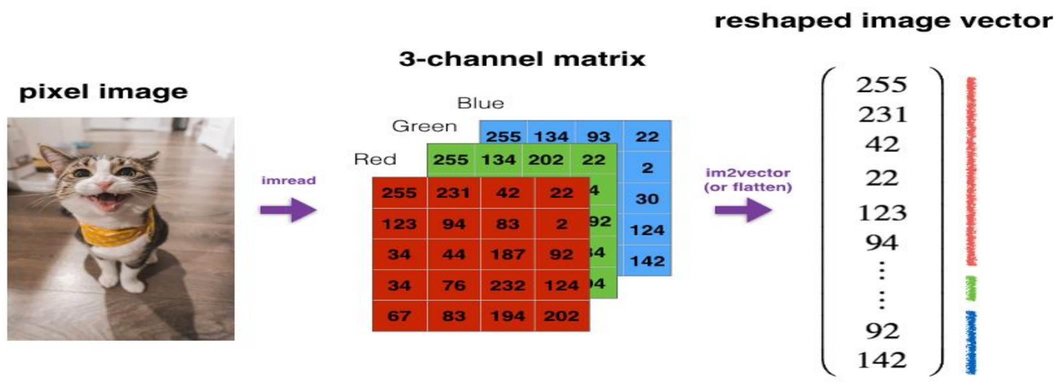
```
### START CODE HERE ### (≈ 2 lines of code)
train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0],-1).T
test_set_x_flatten = test_set_x_orig.reshape(test_set_x_orig.shape[0],-1).T
### END CODE HERE ###

print ("train_set_x_flatten shape: " + str(train_set_x_flatten.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x_flatten shape: " + str(test_set_x_flatten.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
print ("sanity check after reshaping: " + str(train_set_x_flatten[0:5,0]))
```

```
train_set_x_flatten shape: (12288, 209)
train_set_y shape: (1, 209)
test_set_x_flatten shape: (12288, 50)
test_set_y shape: (1, 50)
sanity check after reshaping: [17 31 56 22 33]
```

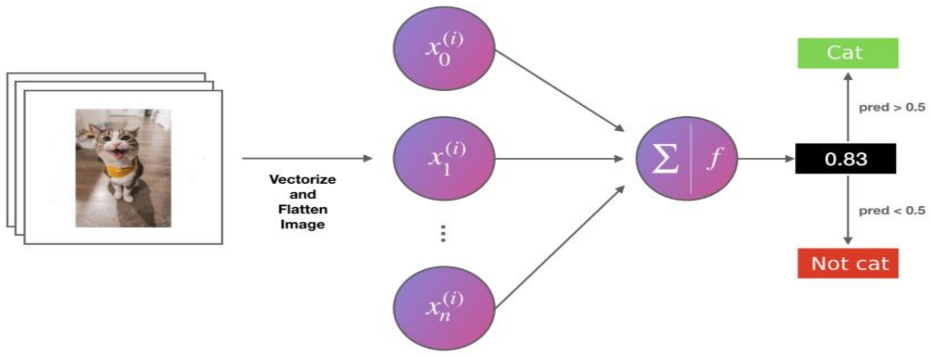
<u>Data Pre-processing:</u>

- As we are dealing with image dataset, there would be no null values and each image is labelled accordingly to its respective classes
- Each pixel is composed of RGB tuple which ranges from (0,0,0) to (255,255,255), so to reduce the complexity of the algorithm we normalize the dataset by dividing each pixel value in the tuple by 255
- Then each image is vectorized as shown in the figure



Model Selection and Training:

1. Logistic Regression:
   - We will use the sigmoid activation function. The main reason why we use sigmoid is that its output exists between (0 to 1) and it is especially used for models where we have to predict the probability as an output. Since the probability of anything exists only between the range of 0 and 1, sigmoid is the right choice



$$sigmoid(w^T x + b) = \frac{1}{1+e^{-(w^T x+b)}}$$

   - We need to initialize our network weights with a vector of zeros. Normally, neural network weights should be randomly initialized, but in the case of logistic regression, zeros ensure the values are always on the linear area, making the propagation easier
   - Now that our parameters are initialized, we can do the "forward" and "backward" propagation steps for learning the parameters. Thus, we will implement a function that computes the cost function and its gradient

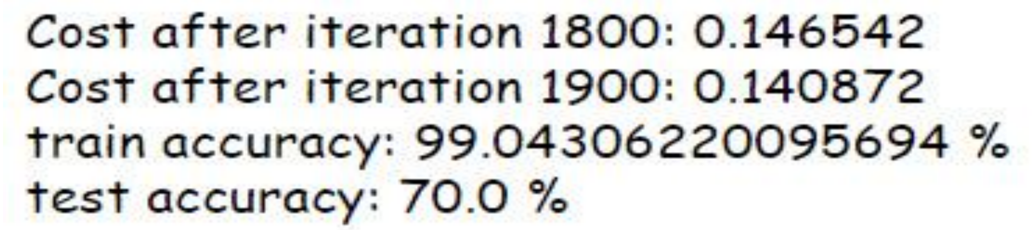
Forward Propagation:

- You get X
- You compute $A = \sigma(w^T X + b) = (a^{(1)}, a^{(2)}, \ldots, a^{(m-1)}, a^{(m)})$
- You calculate the cost function: $J = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})$

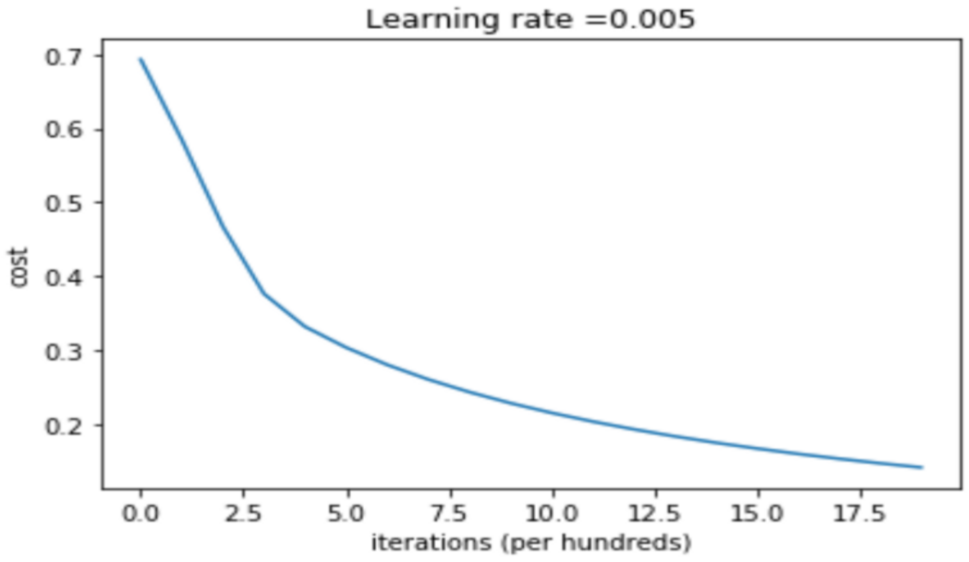
Here are the two formulas you will be using:

$$\frac{\partial J}{\partial w} = \frac{1}{m} X(A - Y)^T$$
$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)})$$

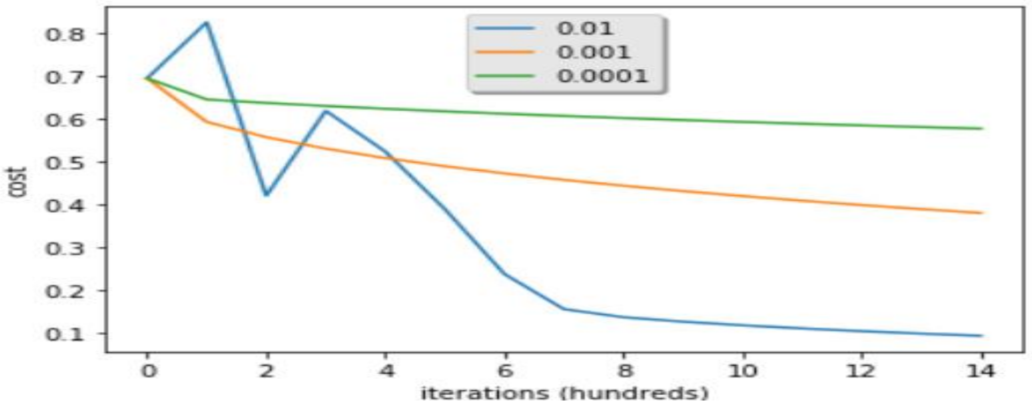
- We used optimization technique to update weights using the gradient descent method and finally, we used them to predict the labels for a given set of images
- By using all above points we built the model based on the image dataset and training and testing accuracy are as show below for learning rate = 0.005 and for 2000 iterations



```
Cost after iteration 1800: 0.146542
Cost after iteration 1900: 0.140872
train accuracy: 99.04306220095694 %
test accuracy: 70.0 %
```



- This figure corresponds to cost vs iteration graph for different learning rate

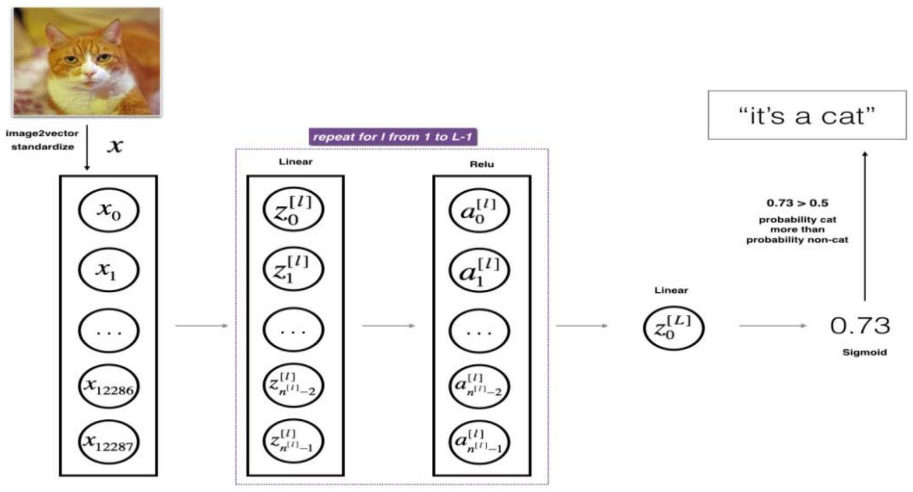


2. Artificial Neural Network:

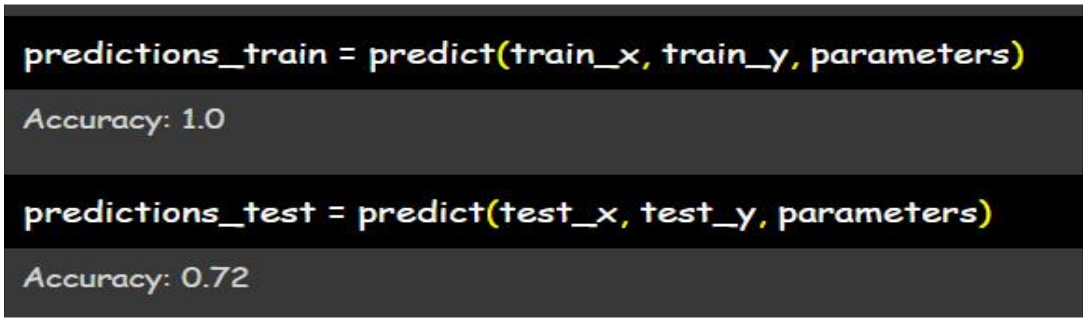
- As usual you will follow the Deep Learning methodology to build the model:

1. Initialize parameters / Define hyperparameters
2. Loop for number of 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

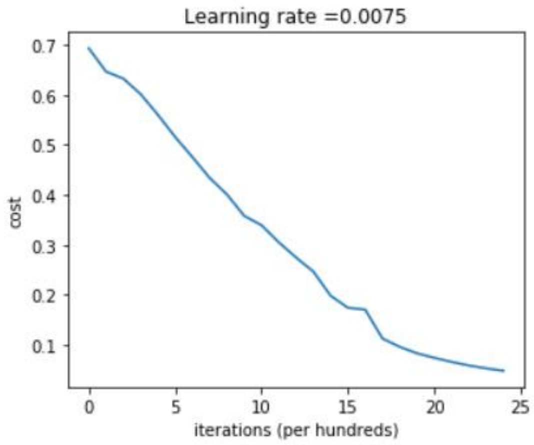
- Here we proceed as the same way as Logistic regression, but here we initialize weights at each layer L randomly such that their mean value is 1 and standard deviation is 0
- Similarly, we compute cost and perform forward propagation and backward propagation and by gradient descent we update the weights of each layer L



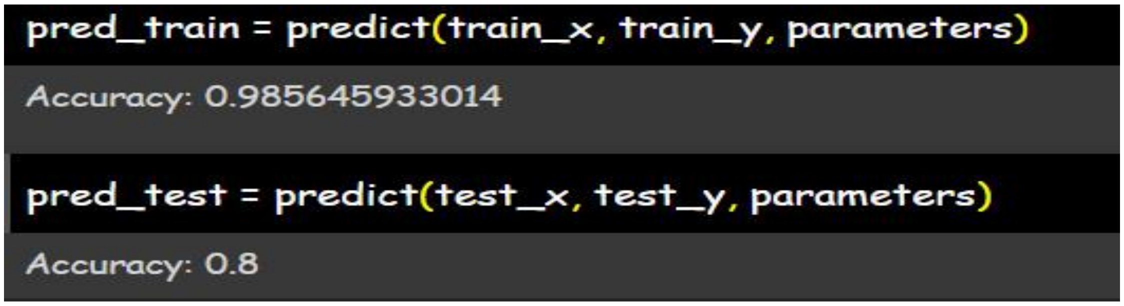
- First, we tried with 2Layer Neural Network which corresponds as follows (LINEAR -> RELU -> LINEAR -> SIGMOID)
- For the above model we got the following results



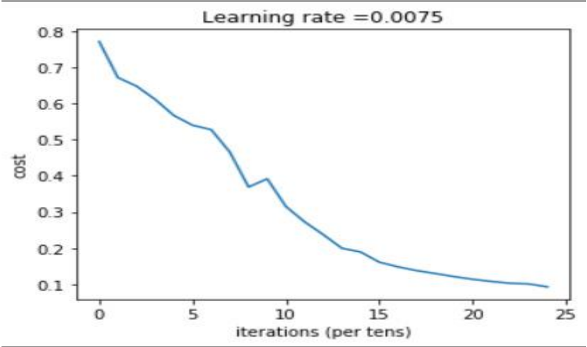
```
predictions_train = predict(train_x, train_y, parameters)
Accuracy: 1.0
predictions_test = predict(test_x, test_y, parameters)
Accuracy: 0.72
```



- These results describe there is overfitting of the data to the model, this can be improved by giving more data during testing phase or increasing the number of Layers or using some other optimization techniques
- By increasing the L value by 4 we got better results compared to the last model



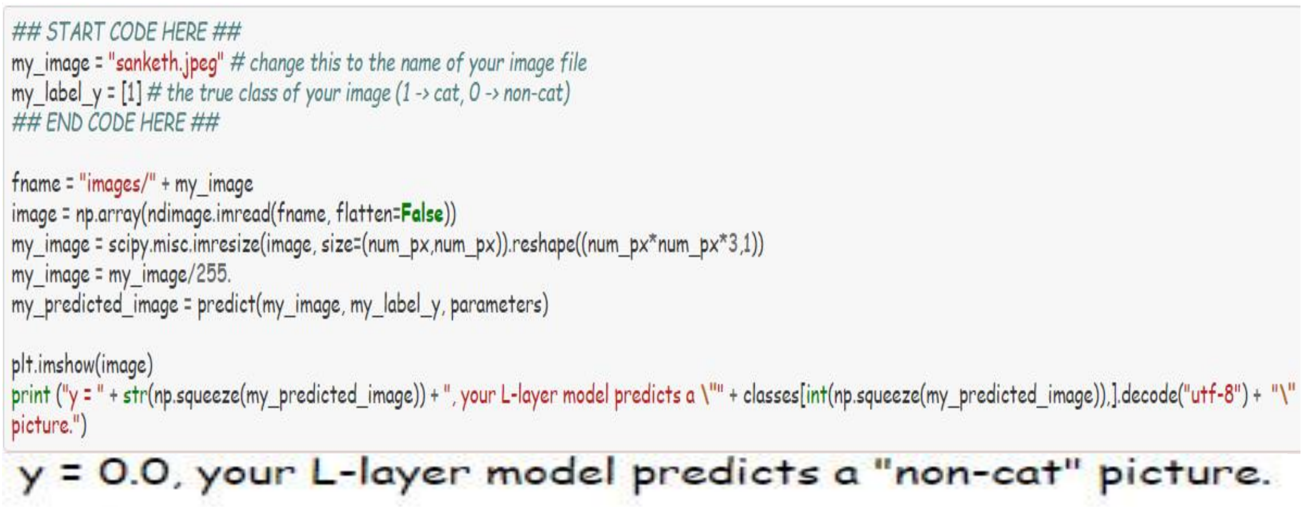
```
pred_train = predict(train_x, train_y, parameters)
Accuracy: 0.985645933014
pred_test = predict(test_x, test_y, parameters)
Accuracy: 0.8
```



SCREENSHOTS

Predictions:

To make predictions we have made a simple UI which takes the image path string as the parameter and from the image we can test our own images

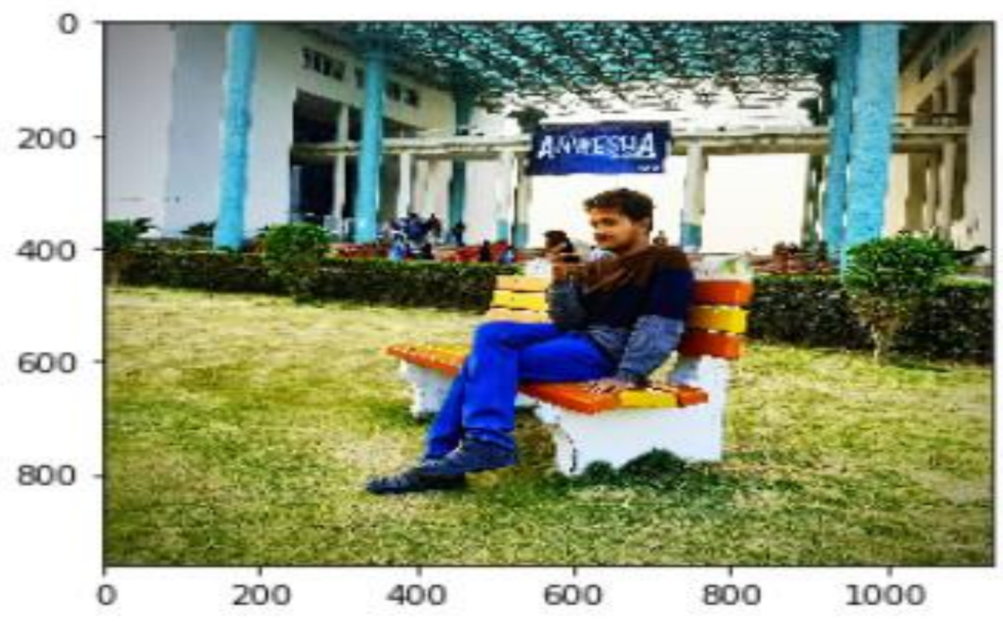


```
## START CODE HERE ##
my_image = "sanketh.jpeg" # change this to the name of your image file
my_label_y = [1] # the true class of your image (1 -> cat, 0 -> non-cat)
## END CODE HERE ##

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_image = my_image/255.
my_predicted_image = predict(my_image, my_label_y, parameters)

plt.imshow(image)
print ("y = " + str(np.squeeze(my_predicted_image)) + ", your L-layer model predicts a \"" + classes[int(np.squeeze(my_predicted_image)),].decode("utf-8") + "\" picture.")
```

```
y = 0.0, your L-layer model predicts a "non-cat" picture.
```



CONCLUSION

As we can see above, our model learned to successfully identify both cat and non-cat images. Its predictions are very confident in the majority of the cases and they are lower only for anything that’s unusual. It’s definitely expected behavior because our model has been trained on less amount of such uncommon data.

Furthermore, we can enhance the machine learning model which we built by playing with the learning rates, increasing the number of layers, changing optimization techniques and many more

Finally, we would like to conclude that, all machine learning algorithms can be built from scratch and can be optimizable without using external libraries

REFERENCES

- Coursera Deep Learning specialization: Click here
- Logistic Regression Models: Click here
- GitHub code for Logistic Regression: Click here
- GitHub code for Ann: Click here