

End Semester Report
on

**Using Deep Learning Techniques to Predict Images from
Encrypted Data**

by

ADITYA SHARMA 2018A7PS0367P

for the course
CS F266 STUDY PROJECT

Under the guidance of

Dr. AMITESH SINGH RAJPUT

At



**Birla Institute of Technology and Science, Pilani For
the Session 2020-2021, 2nd Semester**

TABLE OF CONTENTS

ACKNOWLEDGMENTS	2
LIST OF FIGURES	3
LIST OF TABLES	5
1. Literature review	7
1.1 Neural Networks	7
1.2 Homomorphic Encryption	7
1.3 Applying Deep Neural Networks over Encrypted Data	8
1.3.1 Various methods already used in the process are:	10
1.3.2 Comparison of various methods	13
1.3.3 Problems	14
1.3.4 Solutions	15
2. Background Work for Learning the Topics and Concepts required for the Project	16
2.1 Experiments performed	16
2.1.1 Task 1: Designing 1 Hidden Layer neural network	16
2.1.2 TASK 2: Implementation Of A Guided Project On Prediction Of Housing Prices Of California	17
2.1.3 TASK 3: Research Paper Analysis On: “Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data”.	19
2.1.5 TASK 5	21
2.1.6 TASK 6	22
2.2 Theory Learned	24
3. Problem Formulation	26
4. Implementing the solution	27
4.1 DATASET:	27
4.2 MODEL	28
4.3 Data Preprocessing	29
4.4 Results And Discussion	30
4.5 Analysis	36
5. Future Work And It’s Specific Applications	37
6. CONCLUSION	37
REFERENCES	38

ACKNOWLEDGMENTS

I would like to use this opportunity to thank the **Computer Science Department of BITS** Pilani to grant me this opportunity to present my experiments regarding 'Using Deep Learning Techniques to Predict Images from Encrypted Data' and special thanks to **Dr. Amitesh Singh Rajput**, Assistant Professor, Department of Computer Science for guiding and encouraging me through this study project.

LIST OF FIGURES

Figure 1: Basic outline of privacy preservation model	6
Figure 2: Architecture of a simple neural network described by the input, output, and a hidden layer in between	7
Figure 3: Fully homomorphic encryption	8
Figure 4: A typical scenario for model learning based on encrypted data	9
Figure 5: Workflow of the proposed privacy-preserving deep learning-based application	9
Figure 6: Stacked Auto Encoder	10
Figure 7: Image encryption results	11
Figure 8: Functions used in 1 hidden layer neural network	16
Figure 9: Cost after respective number of iterations	17
Figure 10: Plot depicting relationship between cost function and number of iterations	17
Figure 11: Median housing price plotting	18
Figure 12: Scatter_matrix	18
Figure 13: Comparison of predictions among Linear Regression Model, Decision Tree Regressor and Random Forest Regressor	19
Figure 14: Final predicted values	20
Figure 15: Final weights and biases	20
Figure 16: Final weight values of hidden layer value for the input layer shown by w_0	21
Figure 17: Final output values shown by ans and final weight	21
Figure 18: Cifar 10 dataset snapshot	22
Figure 19: Convolutional Neural Network Details	23
Figure 20: Data Augmentation Function Details	23
Figure 21: Result of CNN during various iterations	24

Figure 22: Implementation of sequential covering used in Ripper	25
Figure 23: Snapshot of the dataset used in the solution	27
Figure 24: Diagram of the neural network designed	28
Figure 25: Formulas used in the backpropagation algorithm	28
Figure 26: Functions used in L layer deep neural network	29
Figure 27: L2 regularisation Function Details	29
Figure 28: L2 regularisation Implementation Details	30
Figure 29: Cost after respective number of iterations	30
Figure 30: Plot depicting relationship between cost	30
Figure 31: Prediction results for 64 X 64 images	31
Figure 32: Prediction results for 50 X 50 images	32
Figure 33: Prediction results for 32 X 32 images	33
Figure 34: Prediction results for 16 X 16 images	34
Figure 35: Prediction results for 8 X 8 images	35
Figure 36: Analysis of the test set and training set predictions on different image sizes	36

LIST OF TABLES

Table 1: Validation Accuracies of Cifar Dataset	12
Table 2: Performance benchmarks during inference using a single data instance on MNIST (Models X and Y) and the Malaria dataset (Model Z).	13
Table 3: Comparison of neural network prediction on image dataset downscaled to respective dimensions	36

INTRODUCTION

There are many scenarios where the data needed is extremely sensitive. For example, in credit card transactions, the information should be available to the credit card company but not to the external developers. Similarly, patient-related health information is available in a hospital, but not for a researcher to identify trends in the data to understand the progression of cancer.

In addition, stringent rules and regulations regulating the collection and use of personal data, such as the General Data Protection Regulation (GDPR) in the European Union and the Health Insurance Portability and Transparency Act (HIPAA) in the United States, exist in some cases, such as medical applications. Often, confidentiality and privacy constraints prevent the sharing of the data with external service providers.

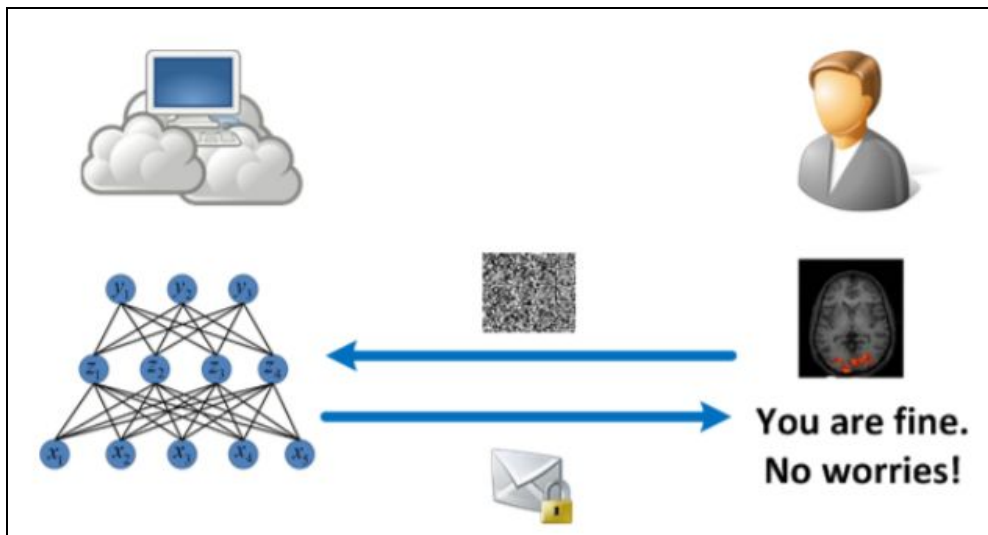


Figure1: Basic outline of privacy preservation model

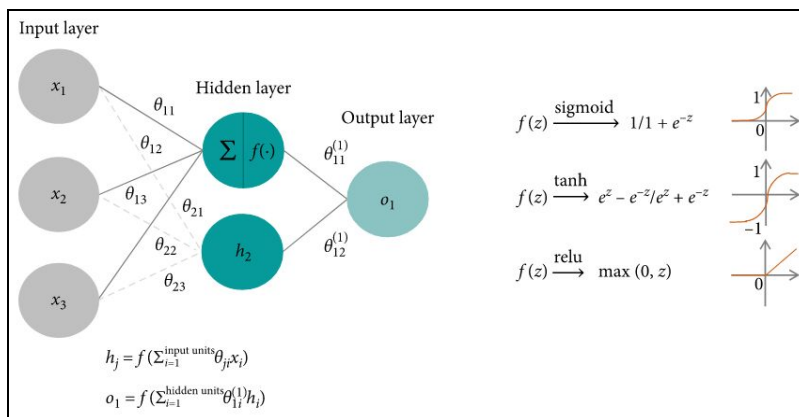
The model owners often consider the learned model and its parameters to be their own intellectual property and do not want to expose it for inference to clients or collaborators. We briefly refer to this purpose as privacy-preserving classification.

Through this project, we plan to use deep learning techniques to find solutions in this area of privacy protection.

1. Literature review

1.1 Neural Networks

A neural network can be represented as a computer model at a high level that maps inputs to outputs by composing layers with interconnected processing blocks (transformations and activation functions). Figure 2 portrays the design of a basic neural network. Usually, nonlinear activation functions are applied to each processing block to allow complicated arbitrary functional mapping. They filter the data that passes through the network, deciding the appropriate input signal to be forwarded to the next layer. In essence, they determine whether or not a certain neuron should be triggered and the neural network becomes a simple linear model without them. Several functions are used as activation functions, like



logistic sigmoid, hyperbolic tangent (tanh), and rectified linear unit, but not limited to (ReLU).

Figure 2: Architecture of a simple neural network described by the input, output, and a hidden layer in between

1.2 Homomorphic Encryption



Modern encryption algorithms are practically unbreakable because they need too much computational power to break them, at least before the advent of quantum computing. The method of splitting them, in other words, is too expensive and time consuming to be feasible. Homomorphic encryption, a development in cryptography technology, may alter that.

Homomorphic encryption is intended to allow encrypted data to be computed. Thus, as it is processed, data should remain confidential, allowing useful tasks to be performed with data existing in untrusted environments. This is a hugely useful power in an environment of distributed computing and heterogeneous networking.

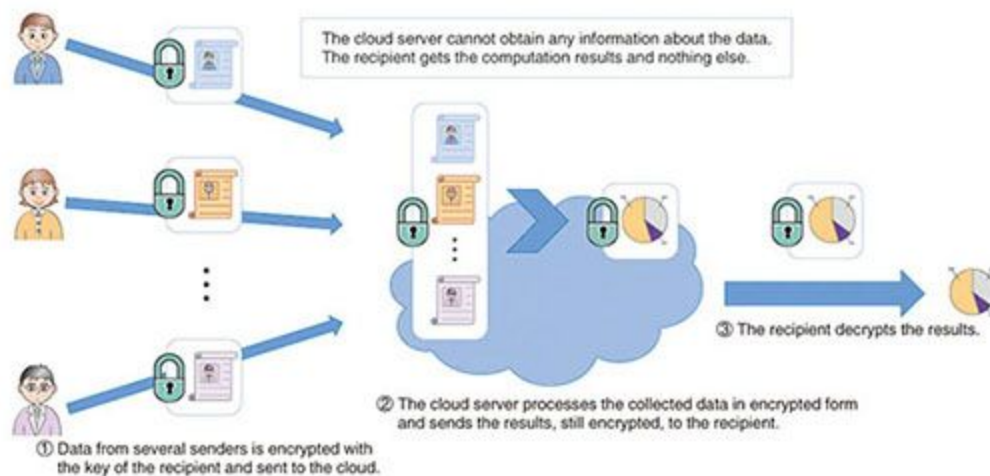


Figure 3 : Fully homomorphic encryption

A homomorphic cryptosystem is like other types of public encryption in that it uses a public key to encrypt data and enables only the person with the corresponding private key to access its unencrypted data. What sets it apart from other types of encryption, though is that it uses an algebraic method to allow you or others to perform a variety of encrypted data computations (or operations).

1.3 Applying Deep Neural Networks over Encrypted Data

Two steps consist of supervised learning algorithms:

1. The training process in which the algorithm learns from a data set of labeled examples and learns a model w

2. The classification process that runs a classifier C over a vector x function previously unseen, using the w model to produce a prediction C (x, w).

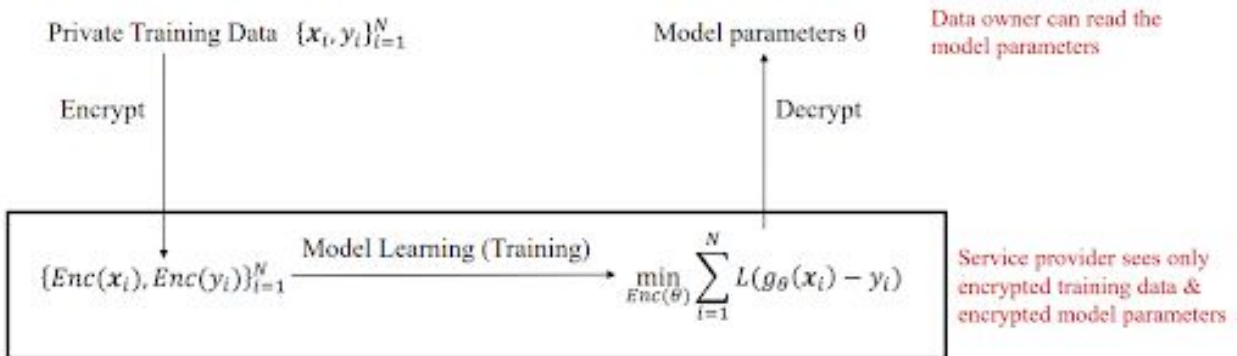


Figure 4: A typical scenario for model learning based on encrypted data

In applications that handle sensitive data, it is important that one or more of the parties involved remain secret to the function vector x and model w . Specifically, a client has a private input(key) represented as a function vector x , and a private input consisting of a private model w is accessible to the server. The classification needs to be privacy-preserving: the client should learn $C(x, w)$ but nothing else about the w model, while the server should not learn anything about the feedback of the client or the outcome of the classification.

Thereafter, the deep learning-based model will have access only to the encrypted version of the data (ciphertext), while the actual data (plaintext) are detached from the processing unit and remain private on the side of the data provider. Finally, the network can be trained directly on ciphertext data following the classical pipeline shown below.

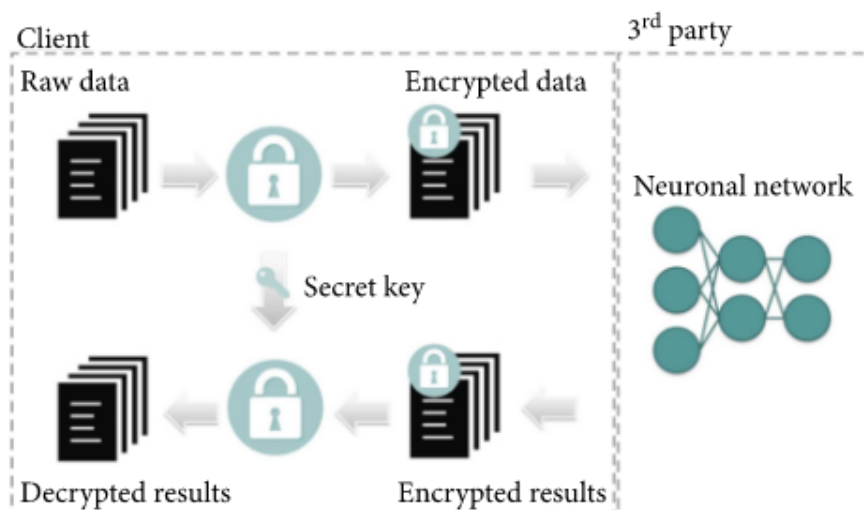


Figure 5 : Workflow of the proposed privacy-preserving deep learning-based application

1.3.1 Various methods already used in the process are:

1. Stacked Auto-Encoder (SAE)

Stacked Auto-Encoder (SAE) is a kind of deep learning algorithm for unsupervised learning. It has multilayers that project the input data vector representation into a lower vector space. These projection vectors are dense representations of the input data. As a consequence, for image compression, SAE can be used. The compression ones can further be encrypted using chaotic logistic maps.

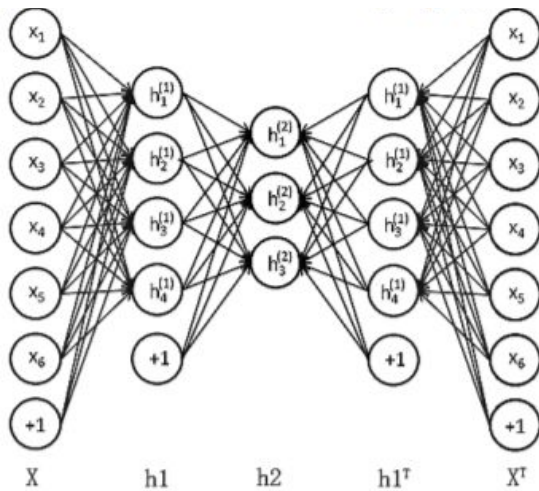


Figure 6 : Stacked Auto Encoder

Image reconstruction: The compressed image is recovered through the SAE model. See Fig. 6, if the compressed image came from the layer of $h2$, a new model is reconstructed with only the layers of $\{X, h1, h2\}$. The compressed image is normalized and is put into the layer of $h2$, then the output from the layer of X represents the recovered image.

2. Learnable Image Encryption

The learnable image encryption transforms an original image to the encrypted image after which people cannot recognize contents in the original image. However, anyone can identify the contents once the image is decrypted. This means that the decryption of the picture has the ability to breach privacy. But a new framework of image encryption and machine learning application is implemented via this approach. The main concept is to encrypt images for humans only, not for computers. With encrypted files, the network is explicitly learned in the suggested scheme, without decrypting images.

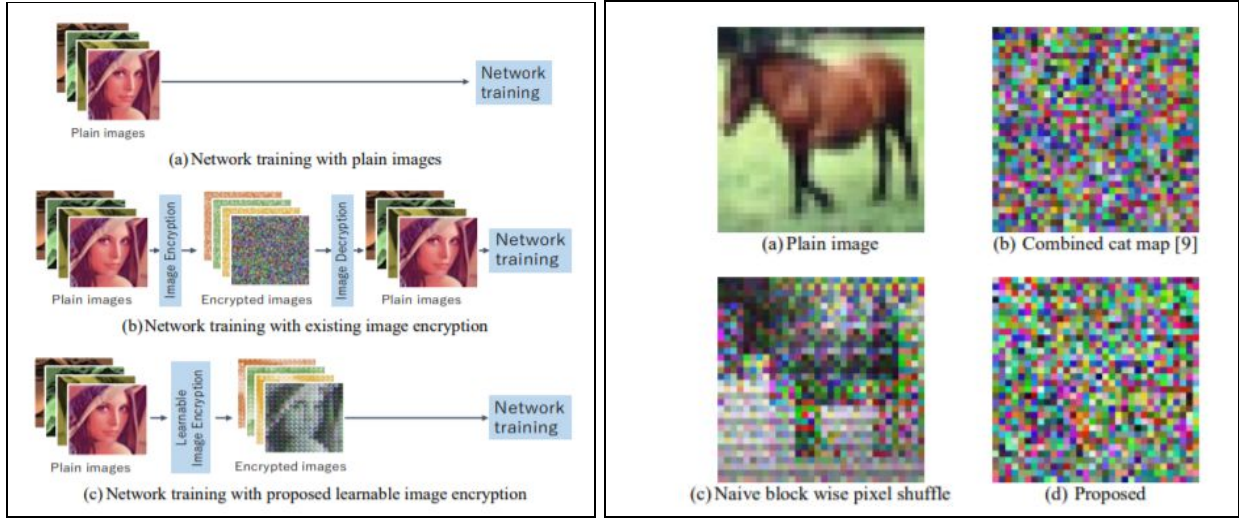


Figure 7: Image encryption results

A block-wise pixel shuffling algorithm is suggested for the learnable image encryption in this technique. The method for the proposed 8-bit RGB image block-wise pixel shuffling algorithm can be summarized as follows:

1. The 8-bit RGB image is divided $M \times M$ -sized blocks.
2. Each block is split to the upper 4-bit and the lower 4-bit images. Then, we have 6-channel image blocks.
3. Intensities of randomly selected pixel position are reversed.
4. Apply a random pixel shuffle.
5. Restore the encrypted picture.

VALIDATION ACCURACIES OF CIFAR DATASET		
	CIFAR 10	CIFAR 100
PLAIN IMAGE	0.884	0.591
COMBINED CAT MAP [9]	0.468	0.209
NAIVE BLOCK WISE PIXEL SHUFFLE	0.872	0.602
PROPOSED	0.863	0.568

Table 1: Validation Accuracies of Cifar Dataset

3. PySyft

A popular private learning library that implements private inference protocols from SPDZ and SecureNN. It is not possible to explicitly use floating point numbers, which is why PySyft uses fixed precision encoding. As either a virtual or a network worker, each group is represented. On the same computer, the former is instantiated and simulates network communications, while the latter can be run on the same or different devices and communicate through Web sockets. The same serialization and deserialization steps are applied by both worker groups.

4. CrypTen

A recent private inference library that can be used for both Private Training and Inference with PyTorch support. CrypTen uses both secret sharing (for matrix multiplications) and garbled circuits (for non-linearity evaluation) and includes translating protocols between these two secret sharing schemes. Using the Gloo backend, a PyTorch distributed backend where each party runs the same code in a different method, communication between parties is implemented.

5. TF-Trusted

A Tensorflow structure that enables models to be executed within a protected enclave of Intel SGX and is based on the Asylo library, a generic system for computing secure enclaves. For private inference, a Tensorflow pre-trained model, saved using the protobuf format, can be used.

6. HE-Transformer.

It is a library developed using Tensorflow, Intel nGraph's Deep Learning model compiler and Microsoft SEAL for private inference using Homomorphic Encryption (HE). It operates in a two-party environment: a server that is the owner of the model and the data-holding client. Additionally, non-linear activations such as ReLU and Max Pooling layers are also carried out by the client. The server sends the encrypted data to the client after computing a linear or a convolutionary layer, which decrypts it, applies the function, re-encrypts it and returns the result. The process helps control noise growth by refreshing the ciphertext after each

layer

7. SMPC

SMPC techniques offer a promising data privacy solution by enabling the analysis of sensitive data, distributed between various data providers, to be carried out in a way that does not reveal sensitive information beyond the results of the analysis.

8. Differential privacy

Methods based on differential privacy provide good protection and have been lately shown to achieve promising results when combined with machine learning techniques. Such approaches resolve the issues of privacy-preserving data processing by introducing random noise at various phases to the algorithm.

1.3.2 Comparison of various methods

- **Model architectures**

Model X and Y were evaluated on the MNIST dataset, while Model Z was applied to the Malaria dataset which is more complex and required a deeper model.

Performance benchmarks during inference using a single data instance on MNIST [17] (Models X and Y) and the Malaria dataset [20] (Model Z).				
Library	Model	Accuracy	Runtime (s)	Comm (MB)
PyTorch	X	97.66%	0.0002	-
PySyft		97.66%	0.5	5.71
CrypTen		97.66%	0.043	1.01
Tensorflow		97.63%	0.0003	-
TF-Trusted		97.63%	0.14	-
HE-Transformer		97.63%	15.4	1171.94
PyTorch	Y	98.47%	0.0003	-
PySyft		98.47%	0.66	1.28

CrypTen		98.47%	0.035	0.35
Tensorflow		98.05%	0.0003	-
TF-Trusted		98.05%	0.12	-
HE-Transformer		98.05%	12.6	2780.66
PyTorch	Z	94.85%	0.0011	-
PySyft		94.85%	8.26	38.56
CrypTen		94.85%	0.25	8.29
Tensorflow		91.5%	0.0008	-
TF-Trusted		91.5%	0.14	-
HE-Transformer		91.5%	548.94	56703.68

Table 2: Performance benchmarks during inference using a single data instance on MNIST (Models X and Y) and the Malaria dataset(Model Z).

1.3.3 Problems

- Computational overhead is the key limitation of these privacy-preserving neural network solutions. More calculations are required for deeper networks, resulting in longer running times. As the number of parties involved or the complexity of the model increases, the costs of communication and computation are greatly affected.
- Most of the Homomorphic Encryption solutions fail to maintain the highest prediction accuracy due to the polynomially approximated activation functions.
- Although techniques such as homomorphic encryption (HE), secure multiparty computing (SMPC) and differential privacy (DP) have shown promising results, their use in modern applications for machine learning remains limited because they are highly dependent on the scenario.
- In addition, a trade-off between privacy and efficiency or between privacy and utility between the techniques is often involved, since each comes with particular strengths and weaknesses.
- Although these systems are known for their efficiency in terms of proven

protection, the above limitations, in addition to their computational overhead, introduce noticeable limitations in the topology of the neural network, which in turn affect the performance of privacy-preserving neural networks

1.3.4 Solutions

- According to me, one of the solutions can be to identify a set of core operations over encrypted data that underlie many classification protocols. We can use efficient protocols for each one of these, either by improving existing schemes (e.g., for comparison) or by constructing new schemes (e.g., for argmax).
- Another technique can be to design these building blocks in a composable way, with regard to both functionality and security.

2. Background Work for Learning the Topics and Concepts required for the Project

2.1 Experiments performed

2.1.1 Task 1: Designing 1 Hidden Layer neural network

- **1 Hidden Layer neural network**

The following functions were written:

```
def initialize_parameters(n_x, n_h, n_y):  
    ...  
    return parameters  
def linear_activation_forward(A_prev, W, b, activation):  
    ...  
    return A, cache  
def compute_cost(AL, Y):  
    ...  
    return cost  
def linear_activation_backward(dA, cache, activation):  
    ...  
    return dA_prev, dW, db  
def update_parameters(parameters, grads, learning_rate):  
    ...  
    return parameters
```

Fig 8: Functions used in 1 hidden layer neural network

- With 1 Hidden Layer neural network, the following output was obtained:

```

Cost after iteration 0: 0.693049735659989
Cost after iteration 100: 0.6464320953428849
Cost after iteration 200: 0.6325140647912677
Cost after iteration 300: 0.6015024920354665
Cost after iteration 400: 0.5601966311605748
Cost after iteration 500: 0.5158304772764729
Cost after iteration 600: 0.47549013139433266
Cost after iteration 700: 0.43391631512257495
Cost after iteration 800: 0.4007977536203884
Cost after iteration 900: 0.3580705011323798
Cost after iteration 1000: 0.3394281538366413
Cost after iteration 1100: 0.30527536361962665
Cost after iteration 1200: 0.2749137728213018
Cost after iteration 1300: 0.2468176821061486
Cost after iteration 1400: 0.19850735037466102
Cost after iteration 1500: 0.17448318112556624
Cost after iteration 1600: 0.170807629780964
Cost after iteration 1700: 0.11306524562164737
Cost after iteration 1800: 0.09629426845937161
Cost after iteration 1900: 0.08342617959726871
Cost after iteration 2000: 0.07439078704319087
Cost after iteration 2100: 0.0663074813226794
Cost after iteration 2200: 0.05919329501038175
Cost after iteration 2300: 0.05336140348560564
Cost after iteration 2400: 0.04855478562877022

```

Fig 9: Cost after respective number of iterations

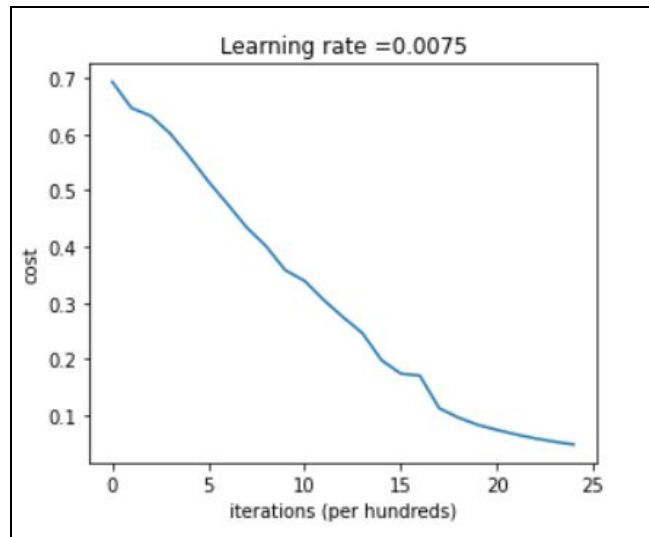


Fig 10: Plot depicting relationship between cost function and number of iterations

RESULT: On the 50 test sets, an accuracy of 72% was obtained.

2.1.2 TASK 2: Implementation Of A Guided Project On Prediction Of Housing Prices Of California

DATASET: The dataset contained attributes like longitude, latitude, house_age, total rooms, etc., along with its corresponding price. The mean cost of the houses was \$1,70,000.

AIM: To train a model on the dataset so that the unknown cost of new houses can be determined.

PROJECT DESCRIPTION:

- Scatter matrix of pandas.plotting and plot of matplotlib were used to analyze the data graphically-

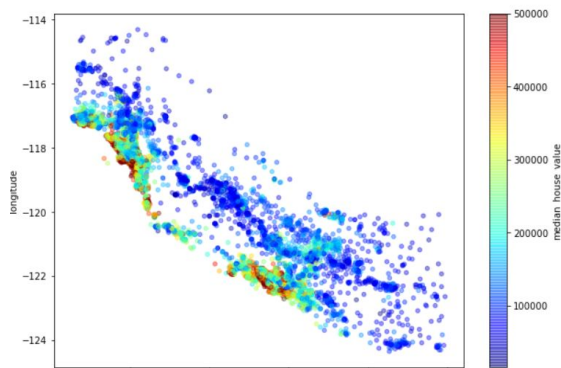


Fig 11: Median housing price plotting

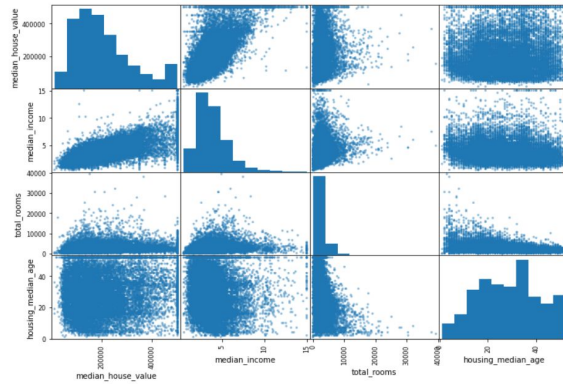


Fig 12: Scatter_matrix

- Used a linear regression model to predict the median house prices.
 - Root mean square error was 40% on training sets.
 - The root mean square error upon running the linear regression model on cross-validation test sets was also around 40%.
- Used decision tree regressor model to predict the median house prices.
 - The root mean square error was 0%.
 - This indicates that decision tree regression hugely overfits the data.
 - After running the decision tree regressor model on cross-validation test sets, the mean of the root mean square error was 45%.
 - Thus, in reality, it performs worse than linear regression.

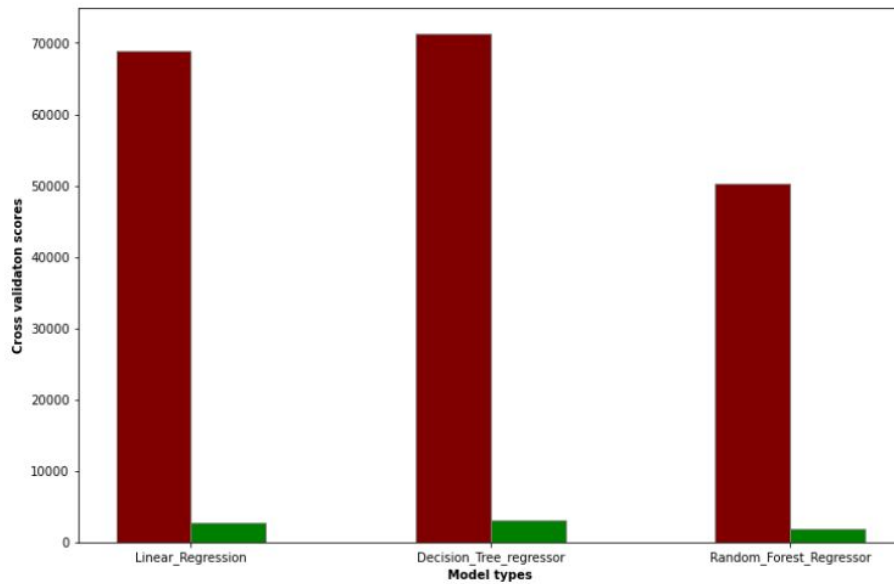


Fig 13: Comparison of predictions among Linear Regression Model, Decision Tree Regressor and Random Forest Regressor
 Red: Mean of the root mean square errors.
 Green: Standard deviation of the root-mean-square errors.

- Used random forest regressor model to predict the median house prices.
 - The root mean square error was 11%.
 - The root mean square error upon running the random forest regressor model on cross-validation test sets was 24%.

RESULT: Hence random forest regressor model was the best model for predictions.

2.1.3 TASK 3: Research Paper Analysis On: “Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data”.

[Reference:<https://www.sciencedirect.com/science/article/abs/pii/S0957417414007325>]

PAPER ANALYSIS:

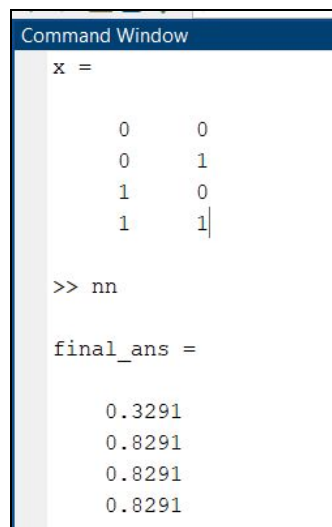
- Learned about the various limitations of the methods used in it.
- Learned about the various data cleansing techniques used in it.

- 4 techniques which were chosen to be analyzed and compared to classify the records were : C4.5, RIPPER, Naïve Bayesian, and AdaBoost.
- Two methods were used to test the models-
 - Three-way split with 10-folds: It was applied one C4.5, RIPPER, and Naïve Bayesian methods.
 - 10-fold cross-validation: The 10-fold cross-validation was applied to C4.5, RIPPER, Naïve Bayesian, and AdaBoost

CONCLUSION: It was concluded that Ripper was the best among them, outperforming all the other techniques.

2.1.4 TASK 4: Designing different types of neural network models from scratch in Matlab.

- **MODEL 1:** Designed a neural network with an input layer and an output layer (perceptron).
 - Trained the neural network for an OR Gate. The outputs were as follows:



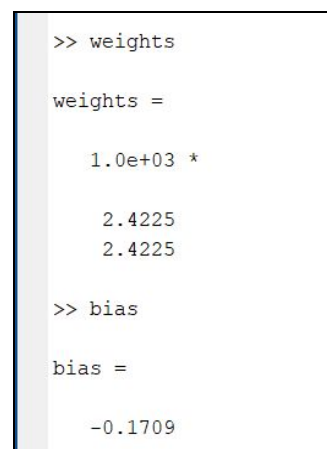
```

Command Window
x =
     0     0
     0     1
     1     0
     1     1

>> nn

final_ans =
    0.3291
    0.8291
    0.8291
    0.8291
  
```

Fig 14: Final predicted values



```

>> weights

weights =
    1.0e+03 *
    2.4225
    2.4225

>> bias

bias =
   -0.1709
  
```

Fig 15: Final weights and biases

- **MODEL 2:** Designed a neural network with an input layer, 1 hidden layer and an output layer.
 - Trained the neural network for an OR Gate. The outputs were as follows:

```
>> x
x =
    0    0
    0    1
    1    0
    1    1

>> nn1
>> wh
wh =
   -3.8229    2.3629    3.1384
   -3.7951    2.4515    3.1852
```

Fig 16: Final weight values of hidden layer weight

```
>> wo
wo =
   -13.2269
    1.7439
    3.5044

>> sigmoid_function(sigmoid_function(x*wh)*wo)

ans =
    0.0182
    0.9908
    0.9907
    0.9946
```

Fig 17: Final output values shown by ans and final value for the input layer shown by wo

CONCLUSION: The network with **1 hidden layer** gives more accurate predictions.

2.1.5 TASK 5

- Made a new working neural network model from scratch with regularisation for the CIFAR-10 dataset since it was not possible to extend the previous model for this dataset.
- Trained the neural network on a dataset of 50,000 images as training set. The output is returned in the form of 10 classes using multiclass classification.

```
Train data: (50000, 32, 32, 3)
Train filenames: (50000,)
Train labels: (50000,)
Test data: (10000, 32, 32, 3)
Test filenames: (10000,)
Test labels: (10000,)
Label names: (10,)
```

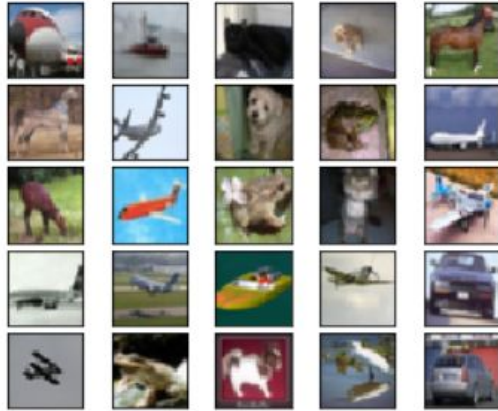


Figure 18: Cifar 10 dataset snapshot

- Converted the labels of the dataset into 10 classes using one hot encoding method so that neural network model can be used for classification.
- Found the cost using cross entropy cost method.

2.1.6 TASK 6

- Used Convolutional neural network to train neural network model on CIFAR-10 dataset.
- Defined a model using relu activation function in hidden layers.
- Used batch normalization.
- The model details are as follows:

model.summary()					
Model: "sequential"					
Layer (type)	Output Shape	Param #			
conv2d (Conv2D)	(None, 32, 32, 32)	896			
activation (Activation)	(None, 32, 32, 32)	0			
batch_normalization (Batch Normalization)	(None, 32, 32, 32)	128			
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248			
activation_1 (Activation)	(None, 32, 32, 32)	0			
batch_normalization_1 (Batch Normalization)	(None, 32, 32, 32)	128			
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0			
dropout (Dropout)	(None, 16, 16, 32)	0			
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496			
activation_2 (Activation)	(None, 16, 16, 64)	0			
batch_normalization_2 (Batch Normalization)	(None, 16, 16, 64)	256			
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928			
activation_3 (Activation)	(None, 16, 16, 64)	0			
batch_normalization_3 (Batch Normalization)	(None, 16, 16, 64)	256			
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0			
dropout_1 (Dropout)	(None, 8, 8, 64)	0			
batch_normalization_4 (Batch Normalization)	(None, 8, 8, 128)	512			
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584			
activation_5 (Activation)	(None, 8, 8, 128)	0			
batch_normalization_5 (Batch Normalization)	(None, 8, 8, 128)	512			
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0			
dropout_2 (Dropout)	(None, 4, 4, 128)	0			
flatten (Flatten)	(None, 2048)	0			
dense (Dense)	(None, 10)	20490			
Total params: 309,290					
Trainable params: 308,394					
Non-trainable params: 896					

Figure 19: Convolutional Neural Network Details

- Used data augmentation for further improving the accuracy.

```
#data augmentation
datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=False
)
datagen.fit(x_train)
```

Figure 20: Data Augmentation Function Details

Results during various iterations.

```
Epoch 24/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3874 - accuracy: 0.9017 - val_loss: 0.4909 - val_accuracy: 0.8821  
Epoch 25/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3887 - accuracy: 0.8998 - val_loss: 0.4624 - val_accuracy: 0.8857  
Epoch 26/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3920 - accuracy: 0.8995 - val_loss: 0.4517 - val_accuracy: 0.8895  
Epoch 27/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3852 - accuracy: 0.9008 - val_loss: 0.4869 - val_accuracy: 0.8828  
Epoch 28/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3850 - accuracy: 0.9021 - val_loss: 0.4583 - val_accuracy: 0.8874  
Epoch 29/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3852 - accuracy: 0.9001 - val_loss: 0.4512 - val_accuracy: 0.8906  
Epoch 30/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3865 - accuracy: 0.9007 - val_loss: 0.4563 - val_accuracy: 0.8895  
Epoch 31/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3831 - accuracy: 0.9009 - val_loss: 0.4597 - val_accuracy: 0.8874  
Epoch 32/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3816 - accuracy: 0.9026 - val_loss: 0.4784 - val_accuracy: 0.8818  
Epoch 33/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3828 - accuracy: 0.9010 - val_loss: 0.4336 - val_accuracy: 0.8959  
Epoch 34/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3866 - accuracy: 0.8998 - val_loss: 0.4579 - val_accuracy: 0.8903  
Epoch 35/35  
781/781 [=====] - 24s 31ms/step - loss: 0.3848 - accuracy: 0.9002 - val_loss: 0.4632 - val_accuracy: 0.8895
```

Figure 21: Result of CNN during various iterations

2.2 Theory Learned

- Did the course 'Neural Network and Deep Learning on Coursera by Andrew NG' to learn about neural network designing.
- Did the course 'Improving Deep Neural Networks on Coursera' to learn about neural network performance improvement.
- Learned about Linear SVM and the effect of the hyperparameter 'C' on its predictions.
- Learned about Non-Linear SVM and the effect of the polynomial degrees on underfitting and overfitting.
- Learned about Decision trees and RIPPER in detail:
 - Decision trees: Learned about Gini Impurity and Entropy calculation for each node of the tree, Probability calculation of the occurrence of each class in each node, the CART training algorithm and Pruning.

- RIPPER: Learned about the procedure of Sequential Covering and its subtopics:
 - Rule Growing
 - Instance Elimination
 - Rule Evaluation
 - Stopping Criterion
 - Rule Pruning
 - Learned about the various steps followed to implement the algorithm and the different variations that may arise.

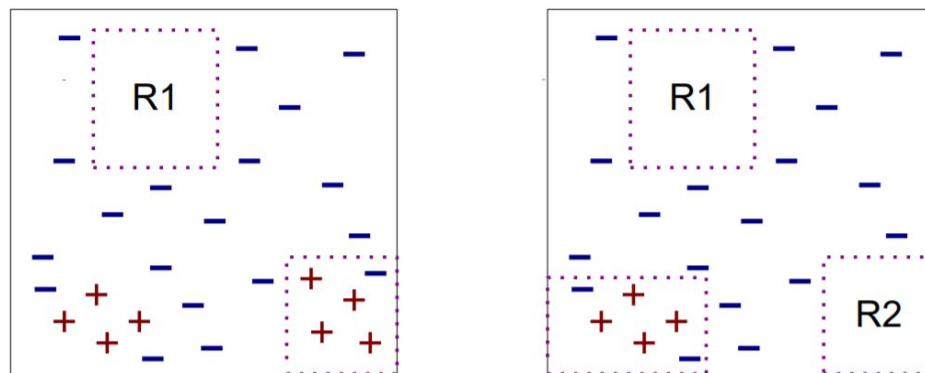


Fig 22: Implementation of sequential covering used in Ripper

- While designing my own neural network model, learned about the concepts of:
 - Hidden layers
 - Activation units
 - Weights matrix
 - Sigmoid function
 - Forward propagation
 - Multiclass Classification in neural networks
 - Cost function of a neural network
 - Backpropagation
 - Implementation of neural networks in Matlab using feedforwardnet.
- For improving the accuracy of predictions of the neural network model, learned about:
 - The concept of variance and overfitting.
 - Different kinds of regularisation techniques and their implementation such as:

- L2 regularisation
 - Dropout regularisation
 - The concept of Normalization and its implementation.
 - Weight Initialisation: Xavier Initialisation and HeLU methods.
 - Gradient Checking and it's implementation.
- Learned and implemented mini-batch gradient descent for improving the speed of the neural network model

3. Problem Formulation

This project is an attempt to train neural network models on encrypted data and produce appropriate predictions.

This would allow the users to encrypt the data and share the encrypted data with the service provider. The model can then be trained without ever letting the service provider see the underlying data. Because the resulting model would also be encrypted, nothing about the data or the learned model parameters is learned by the service provider. In addition, the resulting model is only useful for users with access to the key (used for encrypting the training data). Hence this will ensure privacy preservation.

In this project downscaled images would be used as input dataset upon which the neural network would perform its predictions

4. Implementing the solution

4.1 DATASET:

- The dataset consists of images divided into two classes, cat($y=1$) v/s non cat($y=0$).
- Each image is of shape (64, 64, 3) where 3 is for the 3 channels (RGB)
- The data set is divided into-
 - Training set of 259 images
 - Test set of 44 images

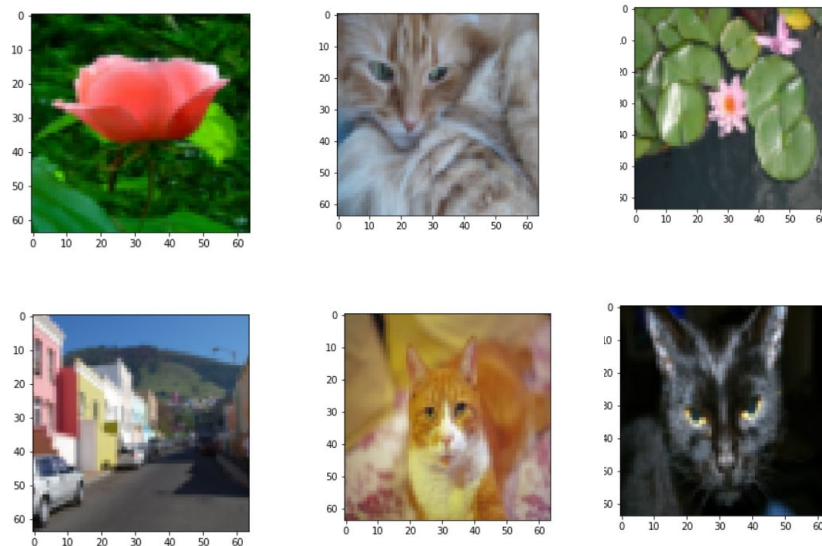


Figure 23: Snapshot of the dataset used in the solution

The link for the dataset is as follows-

<https://drive.google.com/drive/folders/18DMMTRak1LpOJu8yHsRaJaFJQrwfGmfz?usp=sharing>

4.2 MODEL

Designed L-layer deep neural network using NumPy.

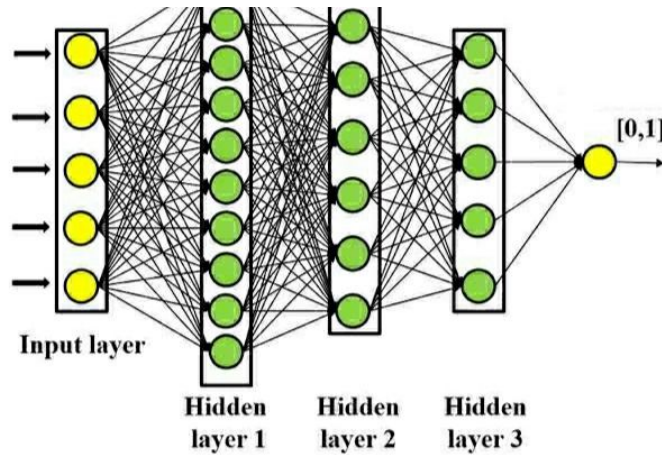
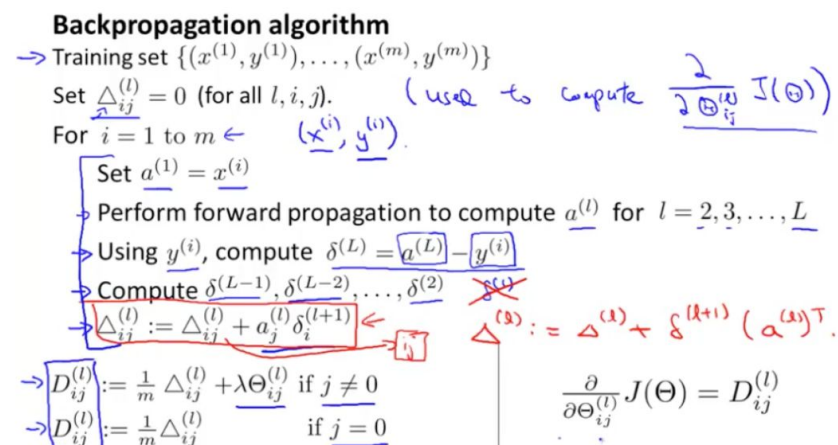


Figure 24 : Diagram of the neural network designed

- The the neural network contains 5 layer and its dimensions are: [number of pixels in image, 20, 7,5,1]
- The activation function of RELU is used for the hidden layers except the last layer where sigmoid is used to predict the output int he range of 0 to 1.
- The following algorithm was used to perform backpropagation:



Back propagation Algorithm

Fig

Figure 25: Formulas used in the backpropagation algorithm

- The following major functions are included in the model:

```
def initialize_parameters_deep(layer_dims):
    """
    Return parameters for deep neural network
    """
    return parameters

def L_model_forward(X, parameters):
    """
    Forward propagation through the model
    """
    return AL, caches

def compute_cost(AL, Y):
    """
    Compute the cost function
    """
    return cost

def L_model_backward(AL, Y, caches):
    """
    Backward propagation through the model
    """
    return grads

def update_parameters(parameters, grads, learning_rate):
    """
    Update parameters with the gradient descent
    """
    return parameters
```

Figure 26: Functions used in L layer deep neural network

4.3 Data Preprocessing

- Different methods like regularization, normalization, gradient checking, weight initialization, early stopping, etc., were tested to improve the accuracy of the model.
- Tried and tested the model with different regularisation parameters(lambda) and learning rates(alpha)
- Used L2 regularisation for regularisation purpose.

```
# GRADED FUNCTION: compute_cost_with_regularization

def compute_cost_with_regularization(AL, Y, parameters, lambda, L):
    """
    Implement the cost function with L2 regularization. See formula (2) above.

    Arguments:
    AL -- post-activation, output of forward propagation, of shape (output size, number of examples)
    Y -- "true" labels vector, of shape (output size, number of examples)
    parameters -- python dictionary containing parameters of the model

    Returns:
    cost - value of the regularized loss function (formula (2))
    """
    m = Y.shape[1]

    cross_entropy_cost = compute_cost(AL, Y) # This gives you the cross-entropy part of the cost

    ### START CODE HERE ### (approx. 1 line)
    for l in range(1,L):
        L2_regularization_cost = lambda * (np.sum(np.square(parameters["W"+str(l)])))

    L2_regularization_cost /= (2 * m)
    ### END CODER HERE ###

    cost = cross_entropy_cost + L2_regularization_cost

    return cost
```

Figure 27 : L2 regularisation Function Details

4.4 Results And Discussion

- Ran the L-layer deep neural network on plain data with L2 regularisation and got **98%** accuracy on training dataset and **82%** accuracy on test dataset.

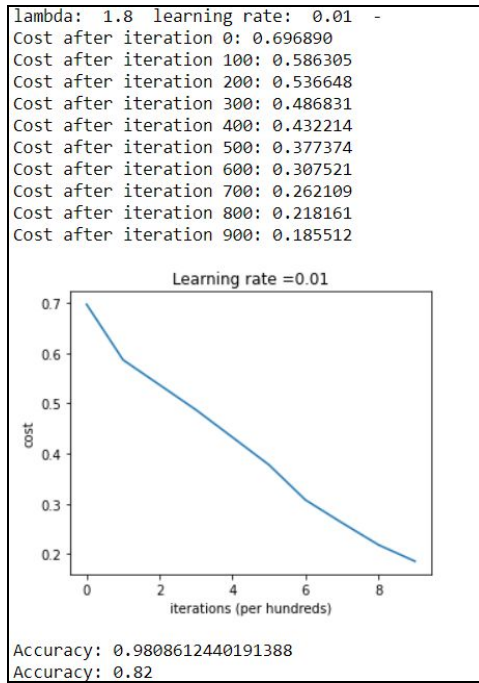


Figure 28 : L2 regularisation Implementation Details

- Ran the L-layer deep neural network on plain data for **2500** iterations and for learning rate(alpha value) of **0.0075**, and an accuracy of **98%** was obtained on training set and **93%** on test set.

```
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
```

Fig 29: Cost after respective number of iterations

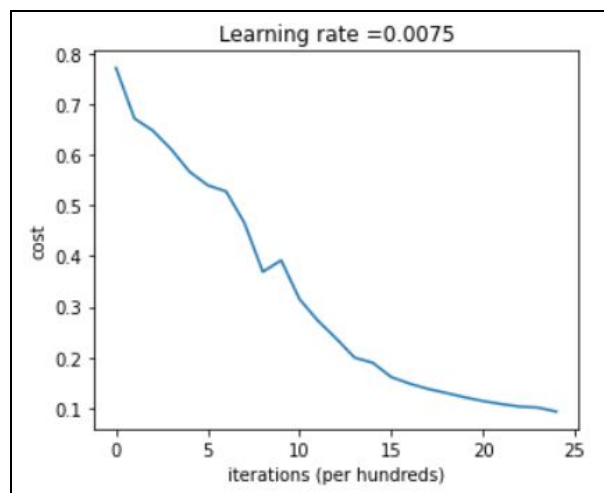
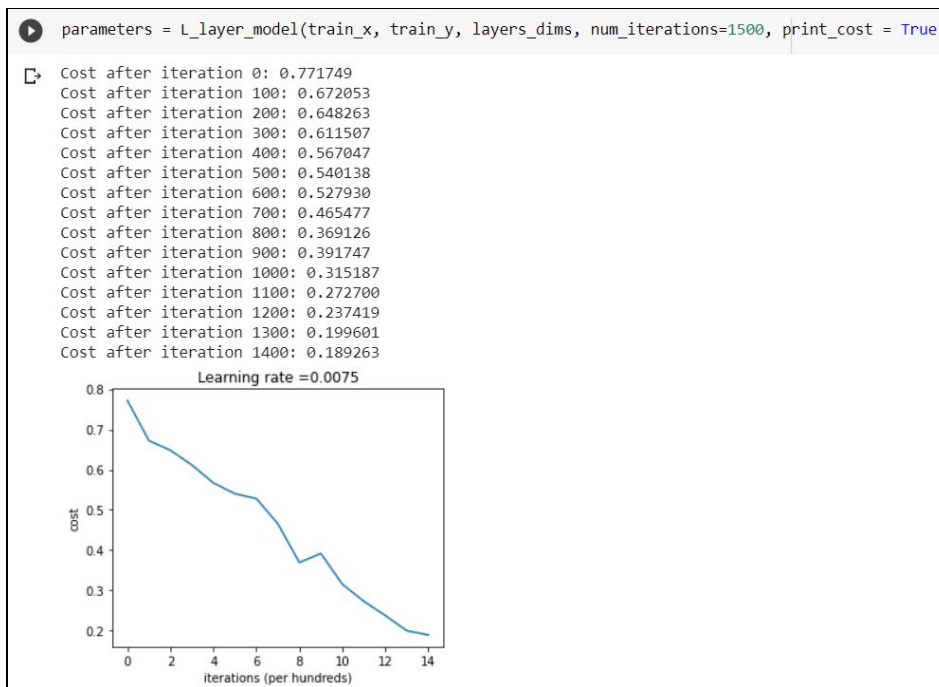


Fig 30: Plot depicting relationship between cost function and number of iterations

- The same dataset was then downsampled by various factors thus reducing the size of the images.
- The model was then tested each of the downsampled dataset and the results were recorded.

The **results** after various image resizing are as follows:

- ORIGINAL(64 X 64)



```

[517] pred_train = predict(train_x, train_y, parameters)

Accuracy: 0.9808612440191388

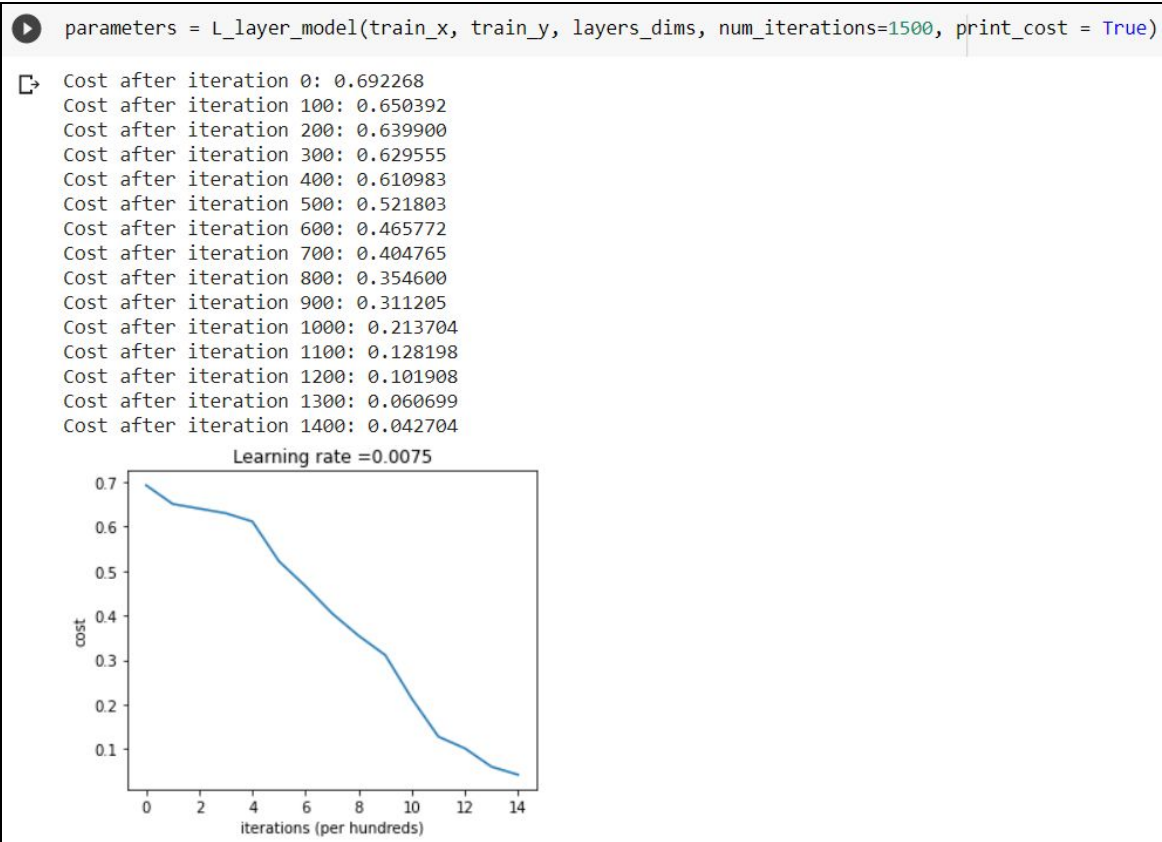
[518] pred_test = predict(test_x, test_y, parameters)

Accuracy: 0.9318181818181818

```

Figure 31 : Prediction results for 64 X 64 images

- 50 X 50



```
pred_train = predict(train_x, train_y, parameters)
```

```
Accuracy: 1.0
```

```
[485] pred_test = predict(test_x, test_y, parameters)
```

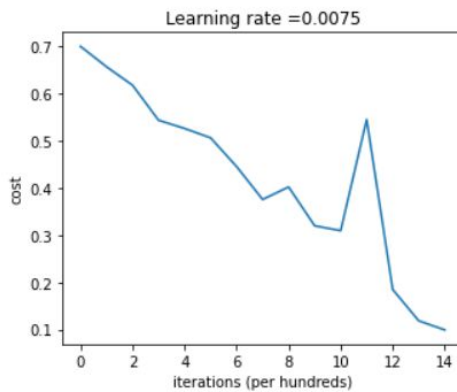
```
Accuracy: 0.8409090909090909
```

Figure 32 : Prediction results for 50 X 50 images

- 32 X 32

```
parameters = L_layer_model(train_x, train_y, layers_dims, num_iterations=1500, print_cost = True)
```

```
Cost after iteration 0: 0.699767
Cost after iteration 100: 0.656548
Cost after iteration 200: 0.617460
Cost after iteration 300: 0.543262
Cost after iteration 400: 0.525895
Cost after iteration 500: 0.505998
Cost after iteration 600: 0.445100
Cost after iteration 700: 0.375407
Cost after iteration 800: 0.401864
Cost after iteration 900: 0.319662
Cost after iteration 1000: 0.309285
Cost after iteration 1100: 0.544967
Cost after iteration 1200: 0.184671
Cost after iteration 1300: 0.118333
Cost after iteration 1400: 0.099025
```



```
► pred_train = predict(train_x, train_y, parameters)
```

```
↳ Accuracy: 0.6555023923444976
```

```
► pred_test = predict(test_x, test_y, parameters)
```

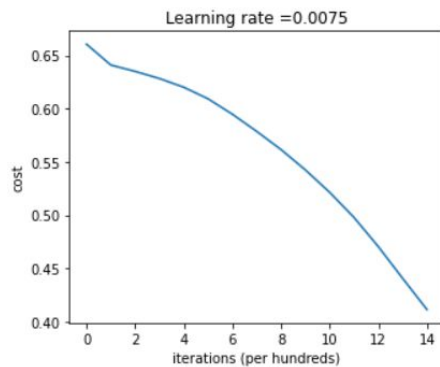
```
Accuracy: 0.295454545454547
```

Figure 33 : Prediction results for 32 X 32 images

- 16X16

```
parameters = L_layer_model(train_x, train_y, layers_dims, num_iterations=1500, print_cost = True)
```

```
Cost after iteration 0: 0.660536
Cost after iteration 100: 0.640941
Cost after iteration 200: 0.634995
Cost after iteration 300: 0.628282
Cost after iteration 400: 0.620103
Cost after iteration 500: 0.609220
Cost after iteration 600: 0.594757
Cost after iteration 700: 0.578621
Cost after iteration 800: 0.561586
Cost after iteration 900: 0.542488
Cost after iteration 1000: 0.521423
Cost after iteration 1100: 0.497841
Cost after iteration 1200: 0.470489
Cost after iteration 1300: 0.440691
Cost after iteration 1400: 0.411411
```



```
[385] pred_train = predict(train_x, train_y, parameters)
```

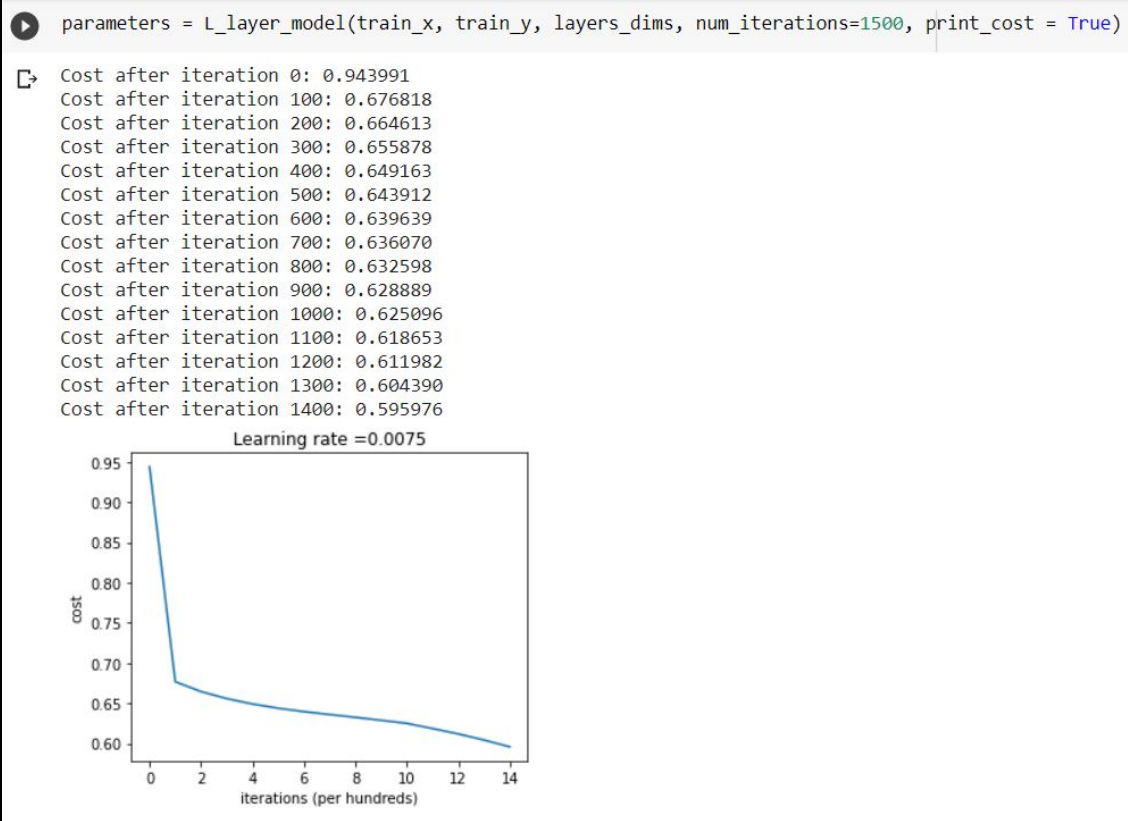
```
Accuracy: 0.8421052631578947
```

```
[386] pred_test = predict(test_x, test_y, parameters)
```

```
Accuracy: 0.5454545454545454
```

Figure 34 : Prediction results for 16 X 16 images

- 8X8



```
[418] pred_train = predict(train_x, train_y, parameters)
```

Accuracy: 0.722488038277512

```
[419] pred_test = predict(test_x, test_y, parameters)
```

Accuracy: 0.5

Figure 35 : Prediction results for 8 X 8 images

Comparison of neural network prediction on image dataset downscaled to respective dimensions		
Dimensions(in pixels)	Accuracy on Training Set	Accuracy on Test Set
64 X 64	98%	93%
50 X 50	100%	84%
32 X 32	65%	29%
16 X 16	84%	54%
8 X 8	72%	50%

Table 3: Comparison of neural network prediction on image dataset downscaled to respective dimensions

4.5 Analysis

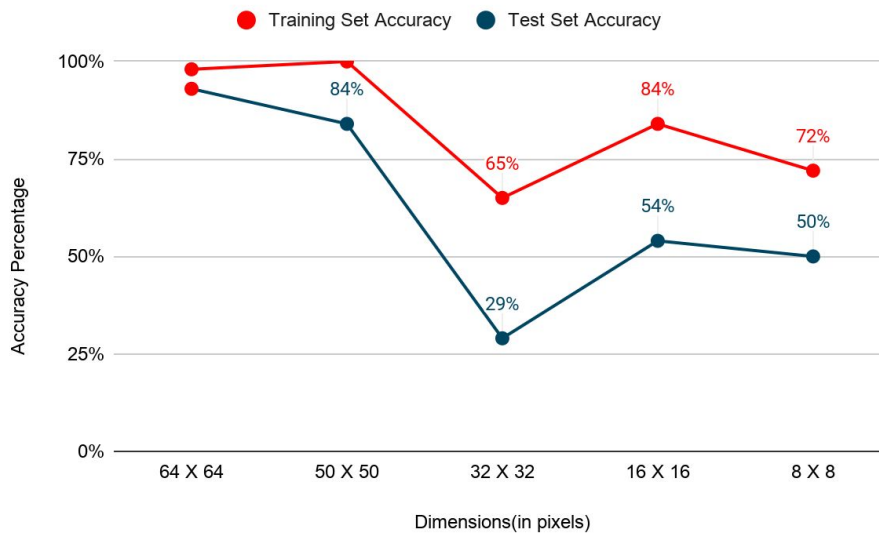


Figure 36 : Analysis of the test set and training set predictions on different image sizes

It was observed that the prediction accuracy on test set decreases uniformly as the size of the images decrease. However it was observed that for the specific image downscaling to 32X32 results in exceptionally low accuracy and seems like an outlier in the results.

5. Future Work And It's Specific Applications

- The same model can be further extended to predict results for images encrypted in various other formats.
- The neural network model can also be modified appropriately so as to enable it to predict the image data with considerable accuracy
- This project has various applications in the domain of privacy preservation such as:
 - Processing confidential information in the domain of finance
 - Working with medical images without revealing them in their true form
 - Sharing confidential marketing information by companies to their various clients
- Further work would include trying and testing the neural network model on various other encryption methods besides downscaling.

6. CONCLUSION

The growing concern over recent years to preserve the privacy of sensitive information, has increased the demand for cryptographic techniques suitable for addressing privacy-related issues in data-driven models. Besides, the growing interest in Machine Learning As a Service (MLAS), where a marketplace of predictors is available on a pay-per-use basis, requires attention to the security and privacy of different models. Different methods can be used to protect the data. These methods are promising in terms of throughput, latency, and accuracy, but they require some way to establish trust between the cloud and the data owner. The provider also needs to guarantee the safety of the keys, and the safety of the data against attackers while it is stored in the cloud.

REFERENCES

1. <https://www.coursera.org/learn/neural-networks-deep-learning>
2. <https://eprint.iacr.org/2014/331.pdf>
3. <https://dl.acm.org/doi/pdf/10.1145/3411501.3419432>
4. <https://link.springer.com/article/10.1186/s13640-018-0358-7>
5. <https://www.hindawi.com/journals/cmmm/2020/3910250/>
6. <http://proceedings.mlr.press/v48/gilad-bachrach16.pdf>
7. https://openaccess.thecvf.com/content_CVPRW_2019/papers/CV-COPS/Nandakumar_Towards_Deep_Neural_Network_Training_on_Encrypted_Data_CVPRW_2019_paper.pdf
8. <https://medium.com/dropoutlabs/encrypted-deep-learning-training-and-predictions-with-tf-encrypted-keras-557193284f44>
9. <http://www.lichao.work/files/2019-C-ICDCS.pdf>
10. <https://ieeexplore.ieee.org/document/8885038>
11. <https://arxiv.org/abs/1412.6181>
12. <https://arxiv.org/abs/1911.11377>
13. <https://www.hindawi.com/journals/cmmm/2020/3910250/>
14. <https://www.sciencedirect.com/science/article/abs/pii/S0957417414007325>