A PROJECT REPORT

on

"MEDICAL IMAGE COMPRESSION"

Submitted to

KIIT Deemed to be University

In Partial Fulfillment of the Requirement for the Award of

BACHELOR'S DEGREE IN
COMPUTER SCIENCE AND SYSTEM ENGINEERING

BY

GURDEEP SINGH BHAMBRA	**1728185**
DEEPANSHU SINGH	**1728186**
DEEPA SINGH	**1728187**
ADITYA VAKHARIA	**1728188**
DIKSHA SINGH	**1728272**

UNDER THE GUIDANCE OF
PROF. SANTWANA SAGNIKA

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

May 2020

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is to certify that the project entitled

"MEDICAL IMAGE COMPRESSION"

submitted by

GURDEEP SINGH BHAMBRA	1728185
DEEPANSHU SINGH	1728186
DEEPA SINGH	1728187
ADITYA VAKHARIA	1728188
DIKSHA SINGH	1728272

is a record of bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & System Engineering) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2019-2020, under our guidance.

Date:10 /06 /2020

(Prof. Santwana Sagnika)
Project Guide

Acknowledgements

We are profoundly grateful to Prof. Santawana Sagnika for her expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion. The work is a team effort minus which the completion of this project was not possible.

GURDEEP SINGH BHAMBRA

DEEPANSHU SINGH

DEEPA SINGH

ADITYA VAKHARIA

DIKSHA SINGH

ABSTRACT

Image compression using DL algorithms is currently one of the most sought-after research topics gaining exponentially increasing attention. Transferring medical images from one center to another is common use in telemedicine. These high-quality images stored in DICOM format require higher bandwidth for transmission and large storage space in PACS (Picture Archiving and Communication System) memory. Therefore, this report lays emphasis on the variety of techniques involved in reducing the image size by preserving diagnostic information. Compression of medical images is important for efficient use of databases. The main purpose of image compression is to reduce the number of bits representing the image while preserving the image quality and the intensity level of the pixels as much as possible depending on grayscale or RGB image.

Implementation of deep learning algorithms like autoencoders are neural network models used for transforming data from a high-dimensional space to a lower-dimensional space. In autoencoders, the original data is reduced to its smaller form called code. This code can then be decoded to reconstruct the original data (in this case, an image). The more accurate the autoencoder, the closer the generated data is to the original data. Further goal is to work on autoencoder algorithms considering the loss factor.

Keywords -- Image compression, Autoencoders, RGB image, Grayscale, telemedicine

Contents

List of Figures

Chapter 1

Introduction

Image compression technology can be generally divided into lossy compression and lossless compression. Lossless compression compresses images by removing statistical redundancy in the image . The process is reversible and is usually used in scenes where image sharpness is high, such as medical images, such as performed in, scarce data images and so on. The lossy compression algorithm performs redundant processing on image information according to the principle that the human eye is insensitive to certain visual features. Compared with lossless compression technology, lossy compression is at the expense of removing invisible information from the human eye, in exchange for the promotion of the compression ratio. Common lossy compression algorithms can be divided into traditional methods based on mathematical statistics and neural network methods based on deep learning.

Deep learning currently performs well in image compression, there are still several issues to be addressed. In general, the human eye has a different degree of attention to each area of the image. for medical images is very much needed as it reduces the size of the image significantly which occupies less space to store. Hospitals or medical imaging manufacturers can easily integrate the algorithm for image compression. Overall structure of an autoencoder consists of an encoder and a decoder. In medical imaging, reversible compression of an image's region of interest (ROI) which is diagnostically relevant is considered essential.

Image compression reduces the size of the original image. The reduced size is helpful when transferring image files, storing image files. The compressed image can also be decompressed to its original form using the decompression algorithm. To accomplish this task an autoencoder uses two different types of neural networks. One of them is called an encoder, and the other is the decoder. The job of the encoder is to accept the

Chapter 2

Literature Survey

1. An architecture for dimensionality reduction (or compression) representing auto-encoder based dimensionality reduction.
   Yasi Wang; Hongxun Yao; Sicheng Zhao

2. Image Compression using Autoencoders in Keras.
   Ahmed Fawzy Gad

3. Lossless compression of curated erythrocyte images using deep autoencoders for malaria infection diagnosis.
   Hongda Shen; W.David Pan; Yuhan Dong; Mohammad Alim

Chapter 3

Software Requirements Specification

1. Ability to respond to every web enabled device.

2. Ability to connect to external websites

3. Ability to optimize the time taken for any search result made on that webpage.

4. Ability to use cookies for better user friendly experience.

5. The interaction between the user end and the software system happens through the website designed specifically to meet the needs of the client side. The webpage user interface should be intuitive; with easy to use controls and graphics moreover the website must be responsive on any web enabled device.

6. The communication between client and server shall use the HTTP protocol

Chapter 4

Requirement Analysis

4.1. Hardware Requirements:

Memory:

Primary memory should be enough to perform the operations.

Web Server's primary memory should be enough to save the zip files in the queue. The zip file size should be of specific size.

Computation Power:

There should be enough computational power to perform the functions in suitable time.

4.2 Language Requirements:

Python:

Used for server and compression related functions.

4.3 Communication Protocols:

HTTP:

The server uses the HTTP protocol to communicate with the web clients.

4.4 Security Considerations:

If any user causes any harm or interruption of any kind then they should be banned from accessing the service.

Chapter 5

System Design

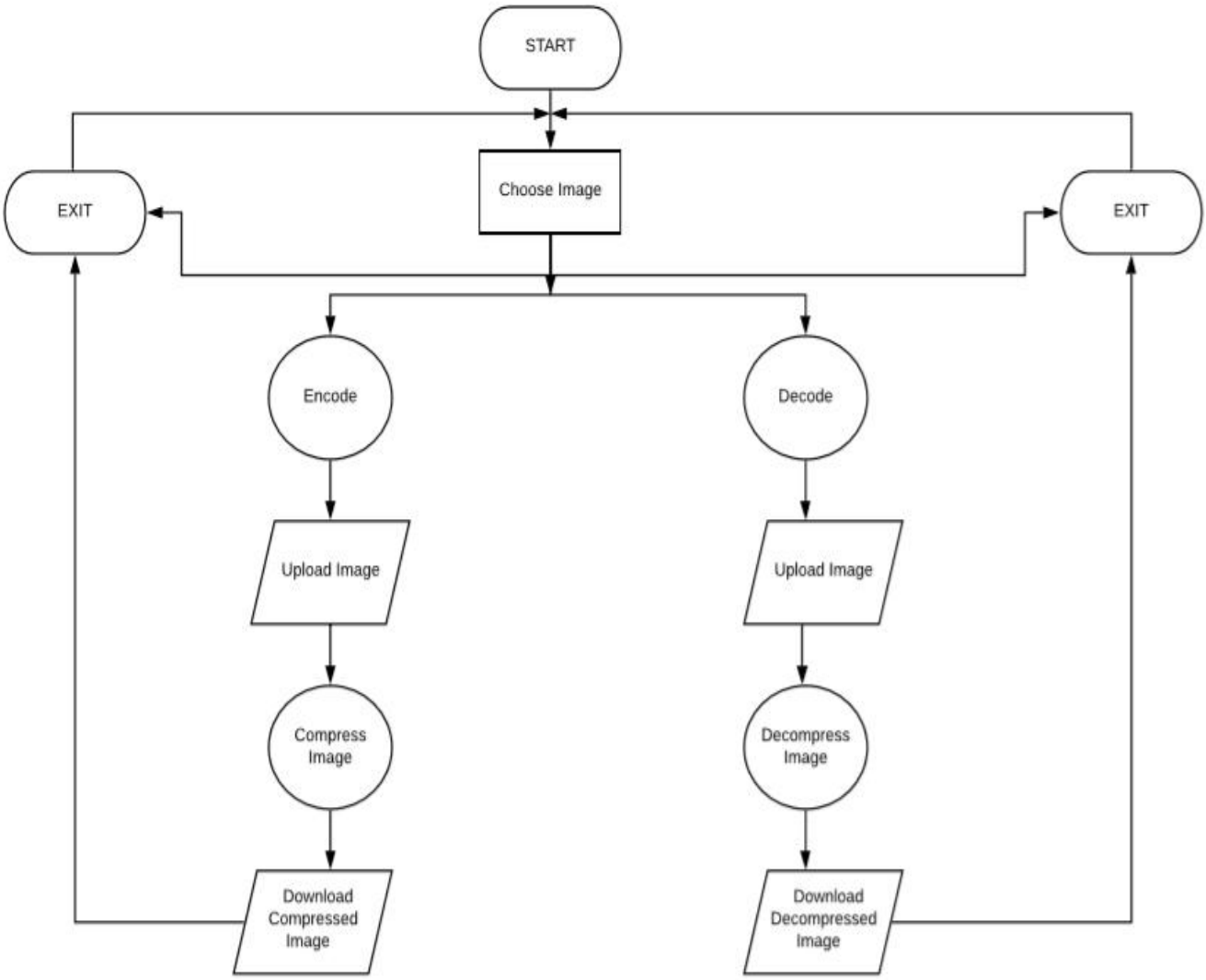

Figure 5.1: DATA FLOW DIAGRAM

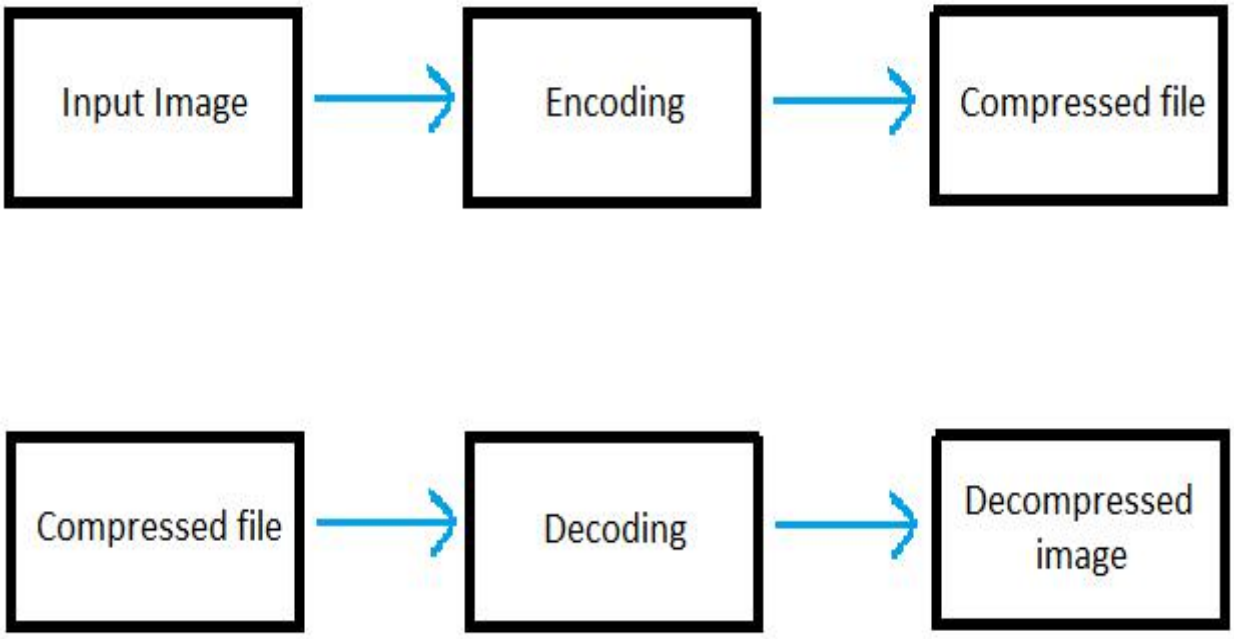

Figure 5.2: PRODUCT PERSPECTIVE (TOP IMAGE: ENCODING; BOTTOM IMAGE: DECODING)

Chapter 6

System Testing

6.1 Test Cases and Test Results

Test Id	Test Case Title	Test Condition	System Behaviour	Expected Result
T01	Zip File	Empty	Prompted Empty File Error on a webpage	Prompt Empty File Error on a webpage
T02	Zip File	Wrong Image format	Prompted Wrong Image Format Error on a webpage	Prompt Wrong Image Format Error on a webpage
T03	Zip File	Wrong Files Present	Prompted Wrong Image Format Error on a webpage	Prompt Wrong Image Format Error on a webpage
T04	Zip File	Wrong Image Size	Prompted Incorrect Image size Error with the filename on a webpage	Prompt Incorrect Image size Error with the filename on a webpage
T05	Zip File	Wrong Image Color Channels	Prompted Incorrect Image channels Error with the filename on a webpage	Prompt Incorrect Image channels Error with the filename on a webpage
T06	Server	404 Error	Prompted the requested filename saying it's not available on a webpage	Prompt the requested filename saying it's not available on a webpage
T07	Server	Empty Form Submission	Prompted No File Provided message on a webpage	Prompt No File Provided message on a webpage

Chapter 7

Project Planning

1. Initially, thorough research work has been done in order to understand what kind of problem it is and what architecture to use.
2. We planned on getting a large medical image dataset of similar image sizes.
3. Thorough research on handling and preprocessing of the large dataset was planned.
4. We studied various autoencoder architectures and settled on convolutional autoencoder because they are suitable for large dimension images and are the current standard for an image processing task. We planned an encoder, decoder architecture using various layers.
5. We identified the various types of loss functions and optimizers to be used for training our deep learning model.
6. We planned to develop a website and deploy it on a cloud platform.

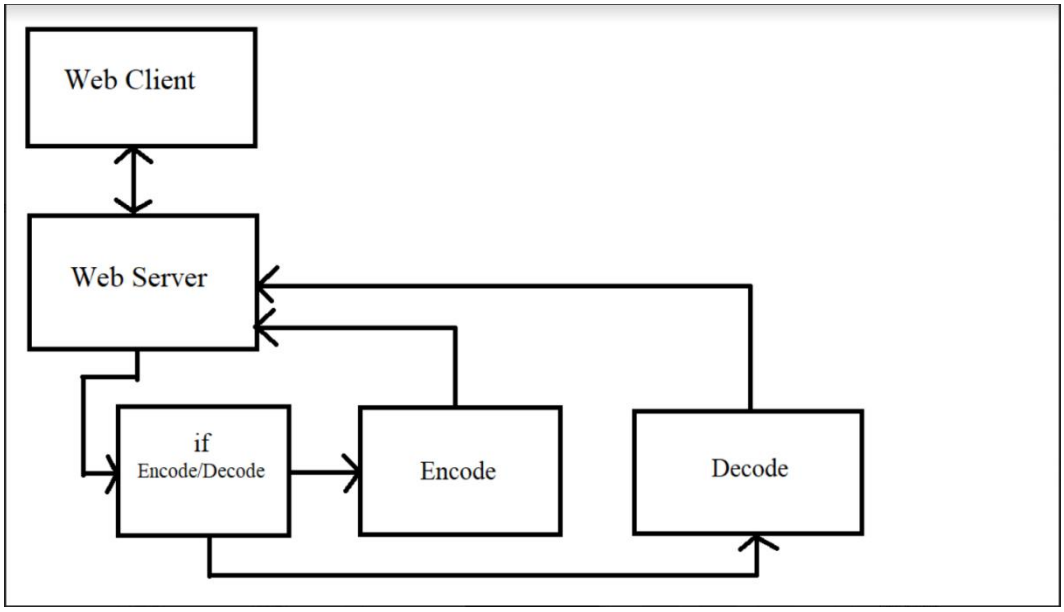

Figure 7.1: PLANNED ARCHITECTURE

Chapter 8

Implementation

8.1 Gathering of data:

We required a huge x-ray single-channel grayscale image dataset for our deep learning model. Our dataset consisted of more than 1 lakh grayscale chest x-ray images.

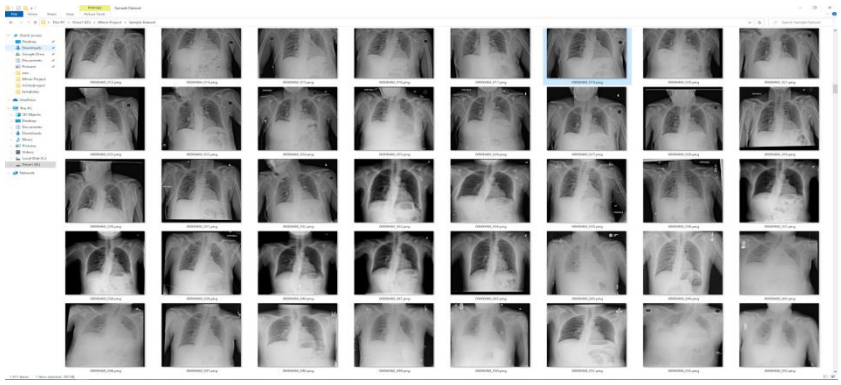

Figure 8.1: SAMPLE DATASET VIEW

8.2 Preprocessing:

We create a CSV file to handle our whole dataset. Our dataset is extracted using OpenCV and pillow python modules. Since we require unaltered images, so our data pipeline focuses on the transmission of the data without any changes to the image quality.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
		Finding_Labels	Follow_Ups	Patient_Id	Patient_Age	Patient_Gender	View_Position	Original_Image	Original_Image	Original_Pixel_Spacing	Original_Pixel_Spac	Image_Shape	Image_Path						
0		Pneumothorax	11	118	73	M	PA	2774	2991	0.143	0.143	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000118_011.pn						
1		Infiltration	7	13	60	M	AP	3056	2544	0.139	0.139	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000013_007.pn						
2		No Finding	3	172	83	F	PA	2048	2500	0.168	0.168	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000172_003.pn						
3		No Finding	3	39	75	M	PA	2464	2715	0.143	0.143	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000039_003.pn						
4		Infiltration	8	181	50	F	AP	2500	2048	0.168	0.168	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000181_008.pn						
5		No Finding	6	109	58	M	PA	2021	2021	0.194311	0.194311	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000109_006.pn						
6		Effusion	20	116	75	F	AP	2500	2048	0.168	0.168	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000116_020.pn						
7		No Finding	4	190	32	M	PA	2992	2991	0.143	0.143	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000190_004.pn						
8		No Finding	9	73	68	F	PA	2480	3056	0.139	0.139	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000073_009.pn						
9		Mass|Pneumoth	4	103	60	M	PA	2992	2991	0.143	0.143	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000103_004.pn						
10		No Finding	1	143	94	M	PA	2892	2894	0.143	0.143	1024, 1024	D:\Minor Project\Dataset\data\images_001\images\00000143_001.pn						

Figure 8.2: GENERATED CSV FILE

8.3 Data Analysis and Visualization:

Our dataset has 519 images of 4 colour channels and rests 111601 images are single-channel grayscale images. We drop the 519 images while creating a data frame (for CSV file) as it is not useful to us. All the chest images are of infected patients.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 111601 entries, 0 to 111600
Data columns (total 13 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Unnamed: 0              111601 non-null  int64
 1   Image_Path              111601 non-null  object
 2   Finding_Labels          111601 non-null  object
 3   Follow_Ups              111601 non-null  int64
 4   Patient_Id              111601 non-null  int64
 5   Patient_Age             111601 non-null  int64
 6   Patient_Gender          111601 non-null  object
 7   View_Position           111601 non-null  object
 8   Original_Image_Width    111601 non-null  int64
 9   Original_Image_Height   111601 non-null  int64
 10  Original_Pixel_Spacing_X  111601 non-null  float64
 11  Original_Pixel_Spacing_Y  111601 non-null  float64
 12  Image_Shape             111601 non-null  object
dtypes: float64(2), int64(6), object(5)
memory usage: 11.1+ MB
```

Figure 8.3: DATASET COLUMNS INFORMATION

8.4 Model Development:

Out of various autoencoder architectures we implemented Convolutional Autoencoder architecture. We chose the CNN architecture because they are suitable for large dimension images and are the current standard for image processing tasks. We preprocessed the images by normalizing them. Our autoencoder model consisted of CNN2D, Max Pooling 2D, Up Sampling 2D, Flatten, and Reshape layers. Our encoder output was a 1D array with each bit representing 16bits of the original image. Our decoder output was the original image. We used mean squared error for loss function and Adam optimizer with batch size set to 16 and trained for 7 epochs reaching minimum loss just short of 0.00029, approximately.

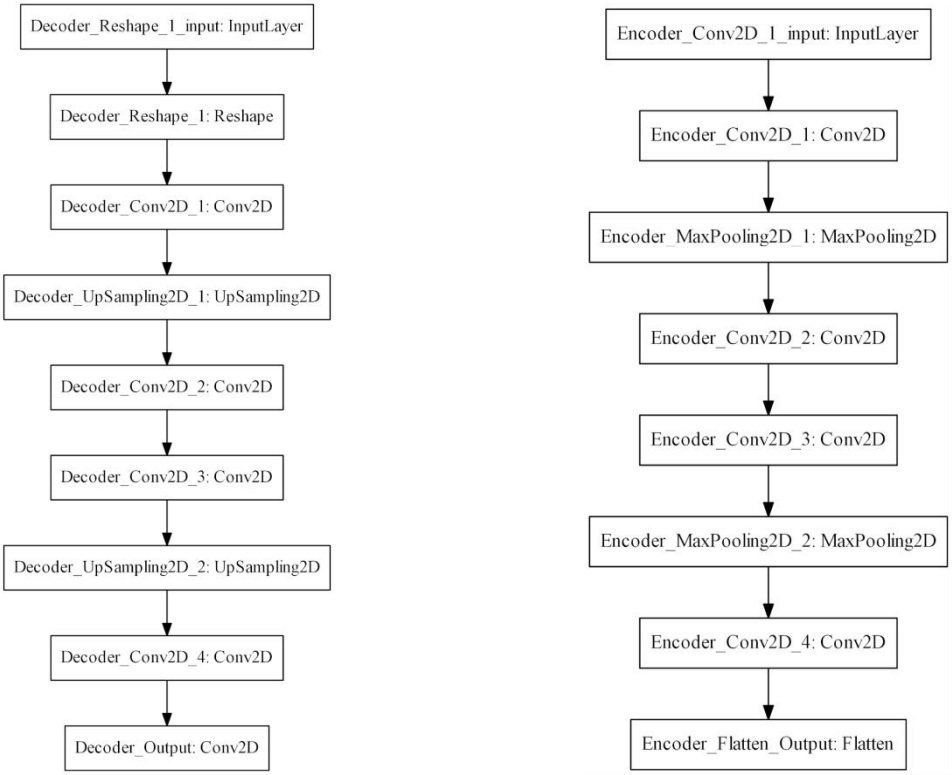

Figure 8.4: ENCODER, DECODER MODELS (LEFT: ENCODER MODEL; RIGHT: DECODER MODEL)

8.5 Website Development:

Backend:

Our Flask (micro web framework) server accepts a zip file having png images of the chest and returns the encoded or the decoded images. It also manages the front end of the website.

Frontend:

Our Webpages were designed using HTML and CSS keeping ease of use for the end-user.

8.6 Website Deployment:

To host the Website we chose EC2 ubuntu free tier instance by Amazon Web services with 1gb Ram and single-core Xeon processor.

```
from flask import Flask, send_from_directory, abort
from flask import render_template, request, make_response
from zipfilehandler import ZipFileHandler
import pathlib

app = Flask("KIIT-MINOR-PROJECT")
host='127.0.0.1'
port=5000

app.config['IMAGES'] = str(pathlib.Path("static/images").resolve())
app.config['MAX_CONTENT_LENGTH'] = 3*1024*1024 #Contents Up to 3 MB allowed


@app.route("/")
def home():
    try:
        return render_template("home1.html")
    except Exception as exx:
        #return exception file
        return render_template("error.html", error=str(exx))

@app.route("/<filename>")
def template(filename):
    try:
        return render_template(filename)
    except Exception as exx:
        #return exception file
        if(str(exx) == filename):
            exx=str(exx)+" File not found"
        return render_template("error.html", error=exx)

@app.route('/favicon.ico')
def favicon():
    try:
        return send_from_directory(app.config['IMAGES'], filename='favicon.ico', mimetype='image/vnd.microsoft.icon')
    except Exception as exx:
        abort(404)

@app.route("/images/<filename>")
def images(filename):
    try:
        return send_from_directory(app.config['IMAGES'], filename=filename)
    except FileNotFoundError:
        abort(404)

@app.route('/autoencode', methods=["POST"])
def autoencode():
    try:
        request_file = request.files['zip_file']
        if not request_file:
            return render_template("error.html", error="No File Provided")
        response = make_response(ZipFileHandler(request_file).autoEncodeDecode())
        response.headers["Content-Disposition"] = "attachment; filename=processed.zip"
        return response
    except Exception as exx:
        return render_template("error.html", error=str(exx))


if __name__ == "__main__":
    app.run(debug=True, host=host, port=port, threaded=True)
```

Figure 8.5:FLASK SERVER CODE SNAPSHOT

Chapter 9

Screen shots of Project

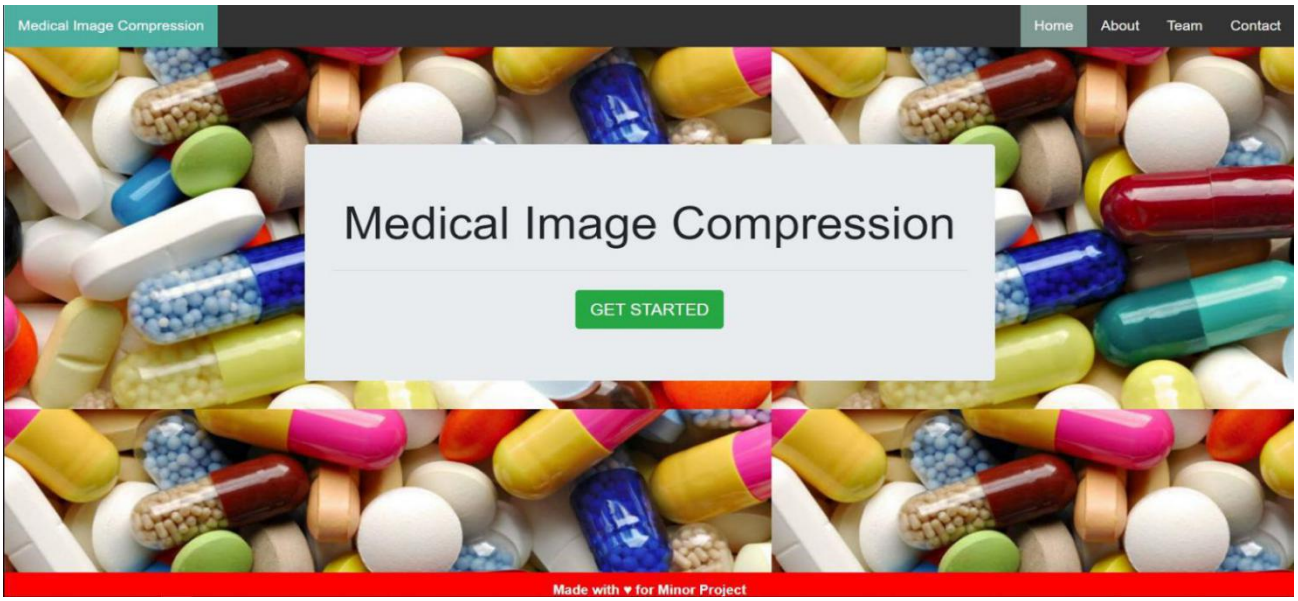

Figure 9.1: WEBSITE HOMEPAGE

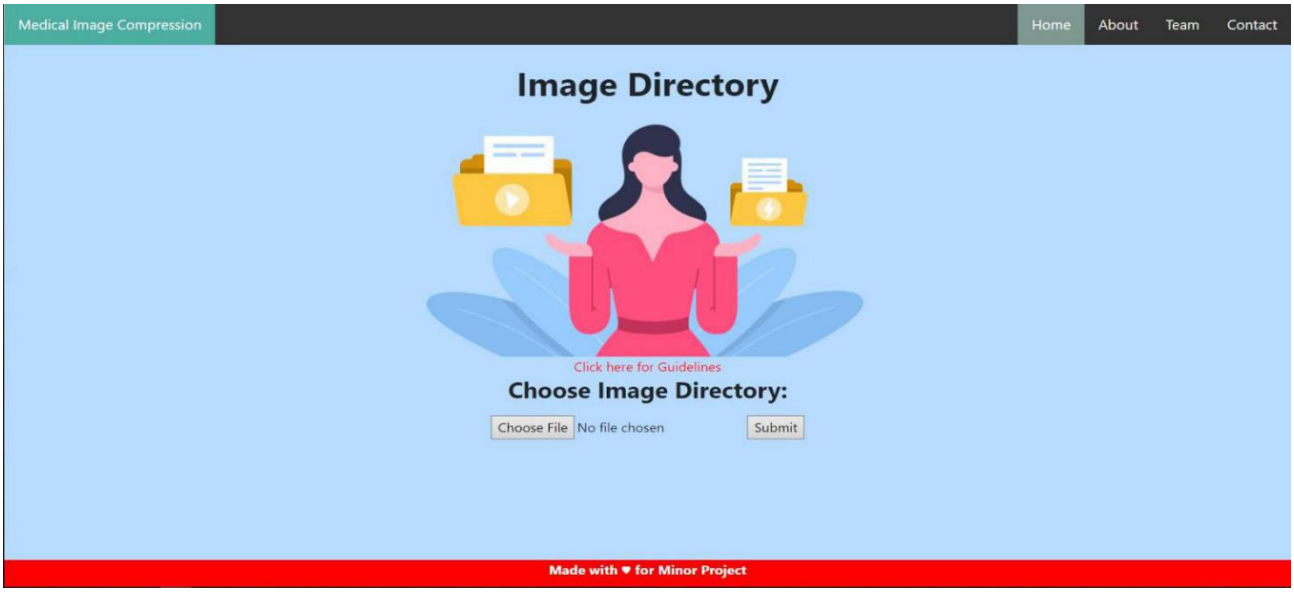

Figure 9.2: WEBSITE IMAGE PROCESSING WEBPAGE

Chapter 10

Conclusion and Future Scope

10.1 Conclusion

Our application aims to resize the medical image to deal with the storage and data transfer problems of the medical industry. To solve this problem, we developed an encoder algorithm that encodes the images to their compressed forms and a decoder algorithm that decodes the images to their near-original form. It compresses the image to fewer bits which in turn decreases the size of the image.

For ease of access to users, we deployed a website with our trained CNN model to a cloud platform. We achieved an approximate compression ratio of 16:1 with an approximate mean squared error of 0.0003 with our CNN autoencoder model.

10.2 Future Scope

1. Future implementation will also handle three-channel images of multiple sizes.
2. To further develop on our CNN model we will implement a deeper Neural network Architecture

.

3. To enhance our CNN model we might try new custom loss functions.
4. We will further enhance the application capability by focusing on the Region Of Interest (ROI) based image compression.

References

[1] An architecture for dimensionality reduction (or compression) representing auto-encoder based dimensionality reduction, Yasi Wang, Hongxun Yao, Sicheng Zhao

[2] Lossless compression of curated erythrocyte images using deep autoencoders for malaria infection diagnosis, Hongda Shen, W.David Pan, Yuhan Dong, Mohammad Alim

[3] Image Compression using Autoencoders in Keras, Ahmed Fawzy Gad

[4] https://iq.opengenus.org/types-of-autoencoder/

[5] https://www.w3schools.com