TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

MVQA:	Medical Visual Question Answering
ReLu:	Rectified Linear Unit
LSTM:	Long Short-Term Memory
RNN:	Recurrent Neural Network
EHR:	Electronic Health Records

ACKNOWLEDGMENT

We extend our sincere thanks to **Prof. S Ramachandram**, Vice-Chancellor, Anurag University, for his encouragement and constant coordination.

We would like to thank **Prof. Syeda Sameen Fatima**, Registrar of Anurag University, for her valuable suggestions and motivation to give our best while working on this project.

We take pleasure in thanking **Dr. A Mallikarjuna Reddy**, Associate Professor, Head Of Department, Department of Artificial Intelligence,Anurag University, for his incredible support and inspiration.

It is our privilege to express gratitude, and indebtedness to our coordinator **Dr. S. Krishna Anand**, Professor, Department of Artificial Intelligence, Anurag University, for his incredible support and constructive criticism which helped us do better.

We express our sincere gratitude to **Mr. M Hari Prasad**, Assistant Professor, Department of Artificial Intelligence, Anurag University, for his valuable insights, motivation and guidance for the successful completion of the work.

ABSTRACT

Medical Visual Question Answering (MVQA) is a rapidly evolving field that combines computer vision and natural language processing to answer questions related to medical images. MVQA aims to help healthcare professionals in making informed decisions by automating the analysis and interpretation of medical images. In this field, the input to the system is a medical image and a natural language question related to the image, and the output is a text-based answer to the question. The development of MVQA systems requires expertise in computer vision, natural language processing, and medical knowledge, and can have a significant impact on the field of medical imaging and diagnosis.

CHAPTER 1

INTRODUCTION

Medical Visual Question Answering (MVQA) is a field that combines computer vision and natural language processing to enable machines to automatically answer questions related to medical images. In recent years, MVQA has gained significant attention from researchers and practitionersin the healthcare industry due to its potential to revolutionize the way medical professionals accessand utilize medical imaging data. The main goal of MVQA is to enable machines to understand the content of medical images and answer questions about them in a natural language format. This is achieved by training computer models to recognize the features and patterns in medical images,as well as to comprehend the meaning of natural language queries. The models then use this information to generate accurate and informative responses to questions related to the medical image.

MVQA systems can be used in a wide range of medical applications, including clinical decision-making, medical education, and patient care. For instance, a physician can use an MVQA system to obtain information about an image in real-time, which can help them make accurate andtimely diagnoses. MVQA systems can also be used to educate medical students and to provide patients with information about their medical conditions. However, there are several challenges that need to be addressed in MVQA research.

One of the major challenges is the variability and complexity of medical images. Medical images can vary in size, shape, texture, and color, which makes it difficult for computer models to accurately identify and extract meaningful features fromthem. Additionally, medical terminology and jargon can also pose a challenge for MVQA systems,as they need to be trained to understand and use medical language in a contextually appropriate manner.

Despite these challenges, MVQA has the potential to significantly improve the efficiencyand accuracy of medical image analysis, which can ultimately lead to

better healthcare outcomes.As research in MVQA continues to evolve, we can expect to see further advancements in the fieldthat will make it an integral part of the healthcare industry.

CHAPTER 2

PROBLEM STATEMENT

Medical images are extremely complicated to comprehend for a person without expertise. The limited number of practitioners across the globe often face the issue of fatigue due to the high number of cases. This fatigue, physical and mental, can induce human-errors during the diagnosis.In such scenarios, having an additional opinion can be helpful in boosting the confidence of the decision-maker. Thus, it becomes crucial to have a reliable Visual Question Answering (VQA) system which can provide a "second opinion" on medical cases. However, most of the VQA systems that work today cater to real-world problems and are not specifically tailored for handlingmedical images. So, we propose an end-to-end deep learning-based algorithm, LSTM, for predicting the answer for the input query associated with the image. We cater to close-ended as well as open-ended type question-answer pairs. We conduct an extensive analysis to evaluate the performance of LSTM.

2.1 EXISTING METHODOLOGY:

There are several existing methodologies for medical visual question answering, which can be broadly classified into three categories:

Rule-based approaches: In this approach, domain-specific rules and heuristics are used to answerquestions based on medical images. For example, a rule-based approach might identify specific features in an X-ray image to diagnose a particular medical condition. While this approach can beeffective for simple questions, it is limited in its ability to handle complex questions that require a deeper understanding of the medical domain.

Machine learning approaches: In this approach, machine learning models are trained on a large dataset of medical images and corresponding questions and answers.

The models can then be used to answer new questions based on medical images. This approach has shown promising results inrecent years, with several studies reporting high accuracy rates. However, these models require large amounts of labeled data and can be limited by the quality and representativeness of the data.

Hybrid approaches: Hybrid approaches combine rule-based and machine learning techniques to improve the accuracy and robustness of medical visual question answering systems. For example, a system might use a rule-based approach to identify specific features in an X-ray image and thenuse a machine learning model to classify the image and provide a diagnosis. Hybrid approaches can leverage the strengths of both rule-based and machine learning approaches and can be more effective in handling a wide range of questions.

Overall, medical visual question answering is a rapidly evolving field, and new methodologies andtechniques are continually being developed and refined.

2.2 PROPOSED METHODOLOGY:

One proposed methodology for medical visual question answering using LSTM (Long Short-TermMemory) involves an LSTM model which is trained on a large dataset of medical images and corresponding questions and answers. The model is trained to learn the relationship between the medical images, questions, and answers. The LSTM model is fine-tuned on a smaller dataset of medical images and questions specific to a particular medical domain, such as radiology or dermatology. Fine-tuning helps to improve the accuracy and generalizability of the model for specific medical applications. Given a new medical image and question, the LSTM model predictsthe most likely answer based on the learned relationship between medical images, questions, and answers. The predicted answer is post-processed to ensure it is in a human-readable format and toprovide additional information, such as confidence scores or diagnostic recommendations.

One advantage of using LSTM models for medical visual question answering is their ability to handlethe sequential nature of both medical images and questions.

LSTMs can learn long-term dependencies between the different parts of a medical image and question, which can be critical for accurate diagnosis and treatment recommendations. However, LSTMs also require large amounts of training data and can be computationally expensive, which can limit their practical applicability.

CHAPTER 3

SPECIFICATION

3.1 SOFTWARE REQUIREMENTS

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Platform – In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries.

Operating system is one of the first requirements mentioned when defining system requirements (software). Software may not be compatible with different versions of same line of operating systems, although some measure of backward compatibility is often maintained. For example, most software designed for Microsoft Windows XP does not run on Microsoft Windows 98, although the converse is not always true. Similarly, software designed using newer features of Linux Kernel v2.6 generally does not run or compile properly (or at all) on Linux distributions using Kernel v2.2 or v2.4.

- Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

- Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

- Jupyter Notebook

The Jupyter Notebook is an open-source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programminglanguages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

- Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series. Pandas is a Python library used for working with data sets.

- Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython.

- NumPy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional arrayobjects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. It also discusses the various array functions, typesof indexing, etc.

3.2 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application isthe physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

Architecture

All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture. See also a list of common operatingsystems and their supporting architectures.

Processing power

The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing poweras the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored. This definition of power is often erroneous, as AMD Athlon and Intel Pentium CPUs at similar clock speed often have different throughput speeds. Intel Pentium CPUs have enjoyed a considerable degree of popularity,and are often mentioned in this category.

Memory

All software, when run, resides in the random access memory (RAM) of a computer. Memory requirements are defined after considering demands of the application, operating system,supporting software and files, and other running

processes. Optimal performance of other unrelated software running on a multi-tasking computer system is also considered when defining this requirement.

Secondary storage

Hard-disk requirements vary, depending on the size of software installation, temporary files created and maintained while installing or running the software, and possible use of swap space (if RAM is insufficient).

Display adapter

Software requiring a better than average computer graphics display, like graphics editors and high-end games, often define high-end display adapters in the systemrequirements.

Peripherals

Some software applications need to make extensive and/or special use of some peripherals, demanding the higher performance or functionality of such peripherals. Such peripherals include CD-ROM drives, keyboards, pointing devices, network devices, etc.

1. Operating System : Windows Only
2. Processor : i5 and above
3. Ram : 4GB and above
4. Hard Disk : 50 GB

CHAPTER 4

DATA EXTRACTION

4.1 COLLECTION OF DATASET

Initially, dataset for heart disease has been collected. After the collection, dataset is split into training data and testing data such that 70% of data is used for training while 30% of data is used for testing. The training data is used for prediction model learning and testing data is used forevaluating the prediction model. The dataset used for this project is Medical VQA which is extracted from Kaggle which consists of datasets helpful for data science and machine learning practitioners. The dataset consists of 3,064 images in the .h5 file format. Each h5 file contains 6 attributes that are used by the system.

4.2 ABOUT THE DATASET

Question Id: Id number of the question

Image Name: Path of the image

Image Organ: Type of Organ

Answer: Actual answer of the given question

Answer Type: Closed or Opened

Question: Different questions asked on different type of diseases related to the organ

CHAPTER 5

DATA PRE - PROCESSING

Data preprocessing is the process of transforming raw data into a clean, structured format that is suitable for analysis. It involves a series of steps to clean, transform, and prepare data for analysis.Effective data preprocessing is essential for accurate analysis, as raw data may contain errors, inconsistencies, and missing values that can lead to biased or incorrect results. Here are some of the key steps involved in data preprocessing for MedVQA:

Data Collection: The first step in MedVQA data preprocessing is collecting the medical images and their corresponding natural language questions. These images can come from various sources, such as electronic health records or medical databases.

Image Preprocessing: Once the images have been collected, they need to be preprocessed. This involves resizing the images to a standard size, normalizing the pixel values, and removing any artifacts or noise that may be present in the images.

Question Preprocessing: Natural language questions also need to be preprocessed before they can be used to train MedVQA models. This involves tokenizing the questions into words, removing stop words, and stemming the words to their base form.

Feature Extraction: Feature extraction is the process of extracting relevant features from the images and questions. For images, this may involve using techniques such as convolutional neuralnetworks to extract visual features. For questions, this may involve using techniques such as bag-of-words or word embeddings to extract semantic features.

Data integration: The extracted features from both the images and text data are combined to create a single dataset that can be used for machine learning models. This process involves ensuring that the features are compatible and that there is

no missing or inconsistent data.

Data Augmentation: Data augmentation is the process of artificially increasing the size of the dataset by generating new examples from the existing data. This can involve techniques such as rotating, flipping, or scaling the images, or paraphrasing the questions to create new variations.

Data Balancing: It is important to balance the dataset to ensure that the MedVQA model is not biased towards one class. This can be achieved by oversampling or undersampling the dataset, or by using more sophisticated techniques such as generating synthetic examples.

CHAPTER 6

MODEL DEVELOPMENT

6.1 DEEP LEARNING

Deep learning has shown great promise in medical visual question answering, which involves answering questions based on medical images such as X-rays, CT scans, and MRIs. With the increasing availability of large medical datasets and advances in deep learning algorithms, medical visual question answering has become an important application of deep learning in healthcare.

In medical visual question answering, deep learning models are trained on large datasets of medical images and corresponding text-based descriptions, such as radiology reports. The models learn toextract features from the medical images and map them to the corresponding text-based descriptions, allowing them to answer questions related to the medical images.

Medical visual question answering has the potential to assist healthcare professionals in making accurate and efficient diagnoses. For example, a deep learning model could be trained to answer questions such as "What is the diagnosis?" or "What is the treatment plan?" based on a medical image. Such models can also help to reduce the workload of healthcare professionals, allowing them to focus on more critical tasks.

However, it is important to note that deep learning models for medical visual question answeringare still in the early stages of development and require large amounts of high-quality data for training. Additionally, the interpretability of these models remains a challenge, as it can be difficultto understand how the models arrived at their answers. Nonetheless, with continued research and development, deep learning has the potential to revolutionize medical visual question answering and improve healthcare outcomes.

6.2 ALGORITHMS

6.2.1 LONG SHORT-TERM MEMORY (LSTM) :

Long Short-Term Memory (LSTM) is a type of artificial neural network architecture that is designed to handle the problem of vanishing gradients in standard Recurrent Neural Networks (RNNs). LSTM networks were introduced by Hochreiter and Schmidhuber in 1997 and have since been widely used in various fields of research, including natural language processing, speech recognition, and computer vision.

The key difference between an LSTM and a traditional RNN is that the LSTM has a memory cellthat can maintain its state over time. This memory cell acts as a conveyor belt, where informationcan be written to, read from, or forgotten. Additionally, LSTM has three gating mechanisms: the input gate, forget gate, and output gate. These gates control the flow of information into and out of the memory cell and allow the network to selectively remember or forget information over time.

The input gate regulates the amount of information that is written to the memory cell. It takes as input the current input to the LSTM, and the previous output and state of the LSTM, and computesa new input that is filtered and stored in the memory cell.

The forget gate controls the amount of information that is discarded from the memory cell. It takes as input the same information as the input gate, but computes a new value that determines which information in the memory cell should be forgotten.

The output gate regulates the amount of information that is read from the memory cell. It takes asinput the current input to the LSTM, and the previous output and state of the LSTM, and computesa new output that is based on the information stored in the memory cell.

LSTMs have shown to be highly effective in many applications, such as speech recognition, machine translation, and image captioning. One of the key advantages of LSTMs is their ability tolearn long-term dependencies in time-series data. Because LSTMs can selectively remember or forget information over time, they are able to process sequences of inputs with variable lengths and maintain a long-term memory of past events.

One common approach to using LSTM networks in medical VQA is to use a "joint embedding" model, where the question and image are both embedded in a common feature space. The LSTM network then processes these embeddings and generates an answer based on the joint representation. This approach has been shown to be effective in a number of medical VQA tasks,such as identifying anatomical structures in medical images and predicting disease severity.

Another approach is to use attention mechanisms, which allow the LSTM network to focus on specific regions of the medical image that are relevant to answering the question. For example, if the question asks about a specific anatomical structure, the attention mechanism can highlight therelevant region of the medical image and allow the LSTM network to generate a more accurate answer.

Overall, LSTM networks have shown great promise in medical VQA tasks, and are likely to become an increasingly important tool in the medical field as more medical data becomes available for analysis.

6.2.2 ACTIVATION FUNCTIONS:

Activation functions are a critical component of artificial neural networks, as they determine the output of a node or neuron in a network. They introduce non-linearity into the network, allowing it to model complex relationships between inputs and outputs. There are several activation functions commonly used in neural networks, each with their own strengths and weaknesses.

- **Tanh:**

  The hyperbolic tangent (tanh) activation function is a popular activation function used in artificial neural networks. Similar to the sigmoid function, it is a non-linear function that maps input values to output values between -1 and 1. However, unlike the sigmoid function, the tanh function has a stronger gradient, making it less prone to the vanishing gradient problem in deep neural networks. In the context of medical visual question answering, the tanh activation function can play an important role in improving the accuracy of the model. Medical images are often complex and require a deep neural network architecture to accurately extract features and information from the images. The tanh activation function can help mitigate the vanishing gradient problem and provide a stronger gradient signal for the model to learn from. Furthermore, the tanh activation function can help the model better handle the scale and range of the features in the medicalimages.

  Medical images often have a wide range of intensity values, and the tanh functioncan help normalize the features and bring them within a more manageable range for the neural network. Overall, the tanh activation function can be a useful tool in improving theaccuracy and performance of medical visual question answering models. Its ability to mitigate the vanishing gradient problem and better handle the scale and range of features in medical images makes it a valuable asset in the field of medical image analysis.

- **ReLu:**

  The Rectified Linear Unit (ReLU) activation function is a popular choice for deep neural networks, including those used in medical visual question answering. It is defined as f(x)= max(0, x), meaning that any input value less than 0 is set to 0, and any value greater thanor equal to 0 is passed through unchanged. This creates a non-linear relationship between the input and output of the neuron, allowing for complex representations to be learned by the neural network. One of the main advantages of the ReLU activation function is its computational efficiency. Unlike other activation functions such as the sigmoid and tanh functions, the ReLU function involves a simple thresholding operation that can be computed quickly and efficiently.

  This makes it well-suited for large-scale medical imageanalysis tasks that involve processing a large number of images and features. In addition to its computational efficiency, the ReLU activation function has been shown to be effective in deep neural networks, particularly in medical image analysis tasks. By allowing the neural network to learn non-linear relationships between the input and output, the ReLU function can improve the accuracy and performance of medical visual question answeringmodels.

- **Softmax:**

  The softmax activation function is commonly used in neural networks, especially in classification tasks. It is used to transform the output of a neural network into a probability distribution over the output classes, which can be useful in many applications, including medical visual question answering. In medical visual question answering, the softmax function is typically used as the final layer of the neural network to produce the probability distribution over possible answers to a given question about a medical image. The softmax function takes the output values from the previous layer of the neural network and normalizes them into a probability distribution, where the sum of the probabilities across all possible answers is equal to 1. The highest probability output is considered as the predicted answer to the question.

  One advantage of using the softmax function in medical visual question answering is that it can handle multiple possible answers to a question. In many cases, a medical image may have more than one possible diagnosis or treatment option, and the softmax function can provide a probability distribution over these options, allowing the model to account for uncertainty in its predictions. Furthermore, the softmax function can help to balance the network's outputs by preventing any one output from dominating the others. This can be particularly useful in medical visual question answering, where different diagnoses or treatments may have similar probabilities.

CHAPTER 7

RESULTS

After performing the deep learning approach for training and testing the accuracy of the LSTM isbetter compared to other algorithms. Calculating accuracy for medical visual question answering using LSTM algorithm involves comparing the predicted answer generated by the model to the actual answer to the question. The accuracy can be calculated as follows:

- Collect a dataset of medical images and their corresponding questions and answers.
- Preprocess the data, including resizing and normalizing the images and converting the text to numerical values using techniques such as tokenization and word embedding.
- Train the LSTM model on the preprocessed data.
- Use the trained model to generate answers for a set of test data.
- Compare the predicted answers to the actual answers for the test data.
- Calculate the accuracy by dividing the number of correctly predicted answers by the totalnumber of test cases.
- For example, if the model generates the correct answer for 80 out of 100 test cases, theaccuracy would be 80%.

It's important to note that accuracy alone may not be a sufficient metric for evaluating the performance of a medical visual question answering system. Other metrics such as precision, recall, and F1 score may also be relevant depending on the specific use case and requirements. Additionally, it may be necessary to evaluate the system's performance on a diverse range of medical images and questions to ensure robustness and generalization.

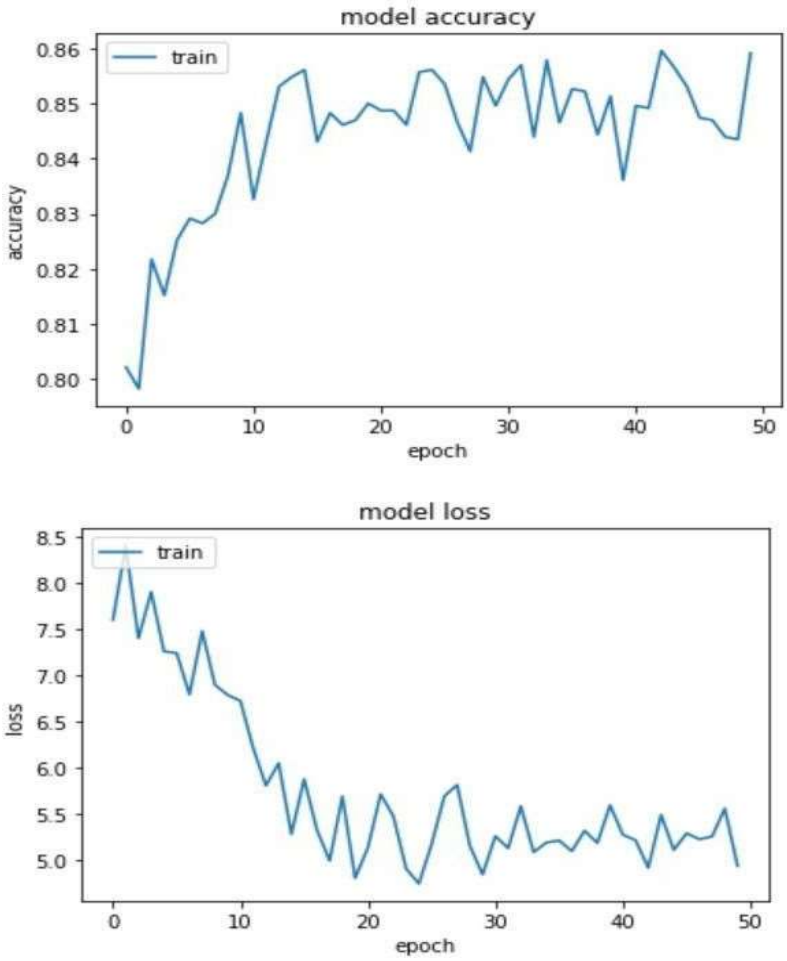

Fig 7.1 Accuracy and Loss Line graph

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

8.1 CONCLUSION:

In conclusion, medical visual question answering using LSTM algorithm is a promising approach to improving medical image analysis and diagnosis. With the ability to accurately extractfeatures and information from medical images and answer complex questions based on that information, LSTM-based models have the potential to revolutionize medical diagnosis and treatment.

However, it's important to note that the development of accurate and robust LSTM models for medical visual question answering requires a significant amount of training data, expert knowledge, and computational resources. Furthermore, the interpretability of LSTM-based modelscan be a challenge, as it can be difficult to understand how the model arrived at a given answer.

Despite these challenges, the potential benefits of LSTM-based medical visual question answeringsystems are clear. By improving the accuracy and speed of medical diagnosis and treatment, thesesystems have the potential to save lives and improve the quality of care for patients around the world. As research in this field continues to advance, we can expect to see exciting newdevelopments and innovations in medical image analysis and diagnosis using LSTM-based algorithms.

8.2 FUTURE SCOPE:

The field of medical visual question answering using deep learning models such as LSTM holds great promise for the future of medical diagnosis and treatment. There are several potentialareas for future research and development in this field, including:

Development of more accurate and efficient models: While LSTM-based models have shown promising results for medical visual question answering, there is still

room for improvement in terms of accuracy and efficiency. Future research could focus on developing new deep learning architectures or refining existing models to achieve even higher levels of accuracy and speed.

Integration with electronic health records (EHRs): The integration of medical visual question answering systems with electronic health records (EHRs) could greatly improve the efficiency and accuracy of medical diagnosis and treatment. By extracting information from medical images andanswering questions based on that information, these systems could provide clinicians with valuable insights and improve patient outcomes.

Application in remote and underserved areas: Medical visual question answering systems could also be particularly valuable in remote and underserved areas where access to trained medical professionals and resources is limited. By providing accurate and timely diagnosis and treatment recommendations, these systems could help improve health outcomes and reduce healthcare disparities.

Integration with other medical technologies: Medical visual question answering systems could also be integrated with other medical technologies such as wearable devices or robotics to further improve diagnosis and treatment. For example, a medical robot could use a visual question answering system to interpret medical images and assist with surgical procedures.

Overall, the future of medical visual question answering holds great promise for improving the accuracy and efficiency of medical diagnosis and treatment. As research in this field continues toadvance, we can expect to see exciting new developments and innovations that have the potentialto save lives and improve the quality of care for patients around the world.

CHAPTER 9

APPENDIX

Jupyter Notebook Code:

```
import warnings,gensim
warnings.filterwarnings('ignore')
model_path = 'GoogleNews-vectors-negative300.bin.gz'     # Path where the model is stored
model_w2v = gensim.models.KeyedVectors.load_word2vec_format(model_path, binary=True)           # Loading the model using genism

import json
import h5py
import numpy as np
import copy
from random import shuffle, seed
import sys
import os.path
import argparse
import glob
import numpy as np
import scipy.io
import pdb
import string
import h5py
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
```

```
import gensim
import json
import re
import cv2
import matplotlib.pyplot as plt



def extract_feat(doc):
    feat = []
    for word in doc:
        try:
            feat.append(model_w2v[word])
        except:
            pass
    return feat

def tokenize(sentence):
    return [i for i in re.split(r"([-.\"',:? !\$#@~()*&\^%;\[\]/\\\+<>\n=])", sentence) if i!="
and i!=' ' and i!='\n'];

def prepro_question(imgs, method):
    # preprocess all the question
    print('example processed tokens:')

    '''
    Input: The question from trainset.json
    Performs tokenization and lowering of the question
    Output: Embedded version of the question
```

```
    Note that: We have still not padded the questions, just for information each of the word
will be a 300 Dimensional Vector,
    hence, the word vector (which will be obtained after padding) will be a (21,300)
Dimensional
    '''
    for i,img in enumerate(imgs):
        s = img['question'].lower()
        if method == 'nltk':
            txt = word_tokenize(str(s).lower())
        else:
            txt = tokenize(s)
        img['processed_tokens'] = txt
        if i < 10: print(txt)
        if i % 1000 == 0:
            sys.stdout.write("processing %d/%d (%.2f%% done)   \r" %  (i, len(img),
i*100.0/len(imgs)) )
            sys.stdout.flush()
    return imgs

def get_top_answers(imgs, num_ans):

    """
    Print the questions and returns the time, one answer is repeated
    """
    counts = {}
    for img in imgs:
        try:
            ans = img['answer'].lower()  # If the string is a number, it would result into error
```

```
        except :
            ans = str(img['answer'])
        counts[ans] = counts.get(ans, 0) + 1

    cw = sorted([(count,w) for w,count in counts.items()], reverse=True)
    print('top answer and their counts:')
    print('\n'.join(map(str,cw[:20])))

    vocab = []
    for i in range(min(num_ans,len(cw))):
        vocab.append(cw[i][1])

    return vocab[:num_ans]


def filter_question(imgs, atoi):
    new_imgs = []
    for i, img in enumerate(imgs):
            new_imgs.append(img)

    print('question number reduce from %d to %d '%(len(imgs), len(new_imgs)))
    return new_imgs

manualMap = { 'none': '0', 'zero': '0', 'one': '1', 'two': '2', 'three':
                '3', 'four': '4', 'five': '5', 'six': '6', 'seven': '7',
                'eight': '8', 'nine': '9', 'ten': '10' }

imgs_train = json.load(open('trainset.json' , 'r'))      # Unnnecesarily, i have used the same
file two times
```

```
num_ans = 1000
top_ans = get_top_answers(imgs_train, num_ans)
atoi = {w:i for i,w in enumerate(top_ans)}                # Word : Count
itoa = {i:w for i,w in enumerate(top_ans)}                # Count : Word
feat_dim = 300                                            # 300 Dimensional Vector
imgs_data_train = json.load(open('trainset.json' , 'r'))  # trainset.json
num_ans = 10    # Even 1 should work fine, but I had taken reference from COCO
dataset, and hence, 10 (10 represents the top 10 answers to a picture)
method = 'nltk'
max_length = 21                    # Max Length of the question
dir_path = "QA"    # The path where we will be storing .h5 file
N = len(imgs_data_train)

image_path = 'VQA_RAD Image Folder'

def save_data():


    for i,img in enumerate(imgs_data_train):
        #print('X' , img['ques_id'])
        img_path = image_path+img['image_name']


        s = img['question']
        print(i,s)   # Print the number and the question
        if method == 'nltk':
            try:
                txt = word_tokenize(str(s).lower())
            except :
                txt = str(s)
```

```
            else:
                txt = tokenize(s)

            img['processed_tokens'] = txt
            question_id = img['qid']
            feat = np.array(extract_feat(img['processed_tokens']))
            label_arrays = np.zeros((1, max_length, feat_dim), dtype='float32')
            label_length = min(max_length, len(feat)) # record the length of this sequence
            label_arrays[0, :label_length, :] = feat
            try:
               ans_arrays = atoi[img['answer'].lower()]
            except :
               ans_arrays = atoi[str(img['answer'])]

            f = h5py.File(os.path.join( dir_path , str(question_id) + '.h5'), "w")
            f.create_dataset("ques_train", dtype='float32', data=label_arrays)
            f.create_dataset("answers", dtype='uint32', data=ans_arrays)
            f.close()
        return

data = save_data()

import json
import h5py
import numpy as np
import copy
from random import shuffle, seed
import sys
import os.path
```

```
import argparse
import glob
import numpy as np
import scipy.io
import pdb
import string
import h5py
import nltk
from nltk.tokenize import word_tokenize
import gensim
import json
import re
import cv2
import tensorflow as tf
import numpy as np
import keras
import pandas as pd
import os
import matplotlib.pyplot as plt
dropout_rate = 0.4
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


def extract_feat(doc):
    feat = []
    for word in doc:
        try:
            feat.append(model_w2v[word])
```

```
        except:
            pass
    return feat

def tokenize(sentence):
    return [i for i in re.split(r"([-.\"',:? !\$#@~()*&\^%;\[\]/\\\+<>\n=])", sentence) if i!="
and i!=' ' and i!='\n'];

def prepro_question(imgs, method):

    # preprocess all the question
    print('example processed tokens:')
    for i,img in enumerate(imgs):
        s = img['question'].lower()
        if method == 'nltk':
            txt = word_tokenize(str(s).lower())
        else:
            txt = tokenize(s)
        img['processed_tokens'] = txt
        if i < 10: print(txt)
        if i % 1000 == 0:
            sys.stdout.write("processing %d/%d (%.2f%% done)   \r" %  (i, len(img),
i*100.0/len(imgs)) )
            sys.stdout.flush()
    return imgs

def get_top_answers(imgs, num_ans):
    counts = {}
    for img in imgs:
```

```
        try:
            ans = img['answer'].lower()
        except :
            ans = str(img['answer'])
        counts[ans] = counts.get(ans, 0) + 1 # Frequency count

    cw = sorted([(count,w) for w,count in counts.items()], reverse=True)
    # print('top answer and their counts:')
    # print('\n'.join(map(str,cw[:20])))

    vocab = []
    for i in range(min(num_ans,len(cw))):
        vocab.append(cw[i][1])

    return vocab[:num_ans]


def filter_question(imgs, atoi):
    new_imgs = []
    for i, img in enumerate(imgs):
            new_imgs.append(img)

    print('question number reduce from %d to %d '%(len(imgs), len(new_imgs)))
    return new_imgs

def image_layer(input_shape):
    base_model = tf.keras.applications.VGG16(input_shape=input_shape,
include_top=False,weights='imagenet')
    base_model.trainable = False # Do not train it
```

```
    x = base_model.layers[-2].output  # Shape would be (28*28*512)
    x = tf.reshape(x , [-1,x.shape[2]*x.shape[1] , x.shape[3]]) # Shape would be
(1,784,512)
    x = tf.keras.layers.Dense(1024)                    # This step can be found out in the
slides, that after feature extraction, they are connecting a dense layer, slide - 6 (Transform
into a same size vector)
    return x

def vgg_preprocessing(model,image):
    return model(image)

def load_data():

        images = []
        questions = []
        answers = []
        ids = []

        #print(start,end)
        #arrs = np.random.randint(0,len(imgs_data_train),batch)
        #data = [imgs_data_train[i] for i in arrs]

        data = imgs_data_train   # trainset.json
        model = image_layer(input_shape = (448,448,3)) # Making VGG16 Model
        for i,img in enumerate(data):

            img_path = img['image_name']  # Image Name
            question_id = img['qid']      # Question id
```

```
        #label_arrays = np.zeros((1, max_length, feat_dim), dtype='float32') #
Somethings are taken directly from

        with h5py.File(os.path.join(dir_path,str(question_id) + '.h5'),'r') as hf:
            question = hf['.']['ques_train'][()] # Embedded question
            answer = hf['.']['answers'][()]      # Embedded answer

        image = cv2.imread(os.path.join('VQA_RAD Image Folder',img_path) ,
cv2.IMREAD_COLOR) # Reading the image
        image = cv2.cvtColor(image , cv2.COLOR_BGR2RGB)
        image = cv2.resize(image , (448,448)) # Reshape

        #image = vgg_preprocessing(model,image)
        # Apply VGG16 Preprocessings

        images.append(image)
        questions.append(np.array(question))
        answers.append(np.array(answer))
        ids.append(question_id)
        if i%100==0:
          print("Processed =>",i,' which is',round(100*i/len(data),2),'%')

    questions = np.reshape(np.array(questions) , [-1,max_length,feat_dim])
    return (np.array(images) , questions ,np.array(answers) , np.array(ids))

imgs_train = json.load(open('trainset.json' , 'r'))
num_ans = 1000
top_ans = get_top_answers(imgs_train, num_ans )
atoi = {w:i for i,w in enumerate(top_ans)}
```

```
itoa = {i:w for i,w in enumerate(top_ans)}
feat_dim = 300
imgs_data_train = json.load(open('trainset.json' , 'r'))
num_ans = 10
method = 'nltk'
max_length = 21
dir_path = "QA"
N = len(imgs_data_train)

import tensorflow as tf
import numpy as np
import keras,h5py
import pandas as pd
import os
import json
from tqdm import tqdm
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from keras.applications.vgg16 import VGG16
import keras.activations
import keras.backend as kbe
from keras.callbacks import EarlyStopping
import tensorflow.keras.layers as layers
from keras.layers import Activation, Add, Concatenate, Conv1D, Dense, Dropout,
Embedding, Softmax
from keras.layers import Input, GlobalMaxPooling1D, Lambda, Multiply, RepeatVector,
Reshape
from keras.layers import BatchNormalization
```

```
from keras.models import Model
from keras.regularizers import l2
import pickle
from pprint import pprint

''' These parameters are for some of the previous attempts, so ignore it, the main part is :
datagen = load_data(), and I don't want to remove all of these,
because indeed there can be some ideas coming out from these lines of code'''

embed_size = 300
q_len = 21
height = 224
width = 224
lstm_units = 256
attention_dim = 512
num_output = 1000
max_questions = 3064

batch_size = 32
lr = 0.001
articles = ['a', 'an', 'the']
manualMap = { 'none': '0', 'zero': '00 'one': '1', 'two': '2', 'three':
                '3', 'four': '4', 'five': '5', 'six': '6', 'seven': '7',
                'eight': '8', 'nine': '9', 'ten': '10' }


datagen = load_data()   # Load the Data

images,questions,answers,ids =datagen[0],datagen[1],datagen[2],datagen[3]
```

```
print("Images have a size of:",images.shape)
print("Questions have a size of:",questions.shape)
print("Answers have a size of:",answers.shape)
print("Ids have a size of:",ids.shape)
dir_path = r'QA' # The directory where the .h5 file for each entry is saved
m = 100
for i in range(images.shape[0]):
    ans_array = answers[i]
    image_array = images[i]
    quest_array = questions[i]
    question_id = ids[i]
    f = h5py.File(os.path.join( dir_path , str(question_id) + '.h5'), "w") # Loading the 'h5
file
    f.create_dataset("ques_train", dtype='float32', data=quest_array) # Question
Embedding
    f.create_dataset("image_vector", dtype='float32', data=image_array) # Image
Embedding (Not preprocessed)
    f.create_dataset("answers", dtype='uint32', data=ans_array)      # Answers in embedded
form
    f.close()
    if i%m ==0:
        print("Processed =>", i,' total percentage =>', round(100*i/images.shape[0],2),' %')
print("Your processing has been done")


import os,h5py
# Image Model
import tensorflow as tf
import numpy as np
from tensorflow.keras import Model
```

```
import warnings
warnings.filterwarnings('ignore')

''' Preprocessing with the VGG 16 Model '''

model = tf.keras.applications.VGG16(include_top=False,weights='imagenet',
      input_shape=(448,448,3))
#print("The Last layer")
last_layer = model.layers[-1].output   # Last layer has an output layer of (14,14,512)
model = Model(model.input,last_layer)
model.trainable = False
# print(model.summary())
def extract_feature(image):
   ''' Preprocessing with VGG Netowrk'''
   image = model(image)
   return image  # Shape is (196,512)

''' The below model will convert (196,512) to (21,300) (i.e same as the dimension of
word embedding) '''

dimen_red = tf.keras.Sequential()  # Use for converting (196,512) -> (21,300)
dimen_red.add(tf.keras.layers.Conv2D(300,kernel_size=(1,1),input_shape= (14,14,512)))
dimen_red.add(tf.keras.layers.Reshape((196,300)))
dimen_red.add(tf.keras.layers.Permute((2,1)))  # Reshaping about the axis, useful for
applying the dense network
dimen_red.add(tf.keras.layers.Dense(21))
dimen_red.add(tf.keras.layers.Permute((2,1)))  # Reshaping about the axis, useful for
applying the dense network
```

```
train_dir = r'QA/' # Containing .h5 file
images = []
ans = []
ques = []
count = 0
content = os.listdir(train_dir)[:2300]   # The GPU Memory became full after this, hence
had to take just these much samples :(
length = len(content)
for i in content:
    # Reading the data
    file = h5py.File(train_dir+i)
    images.append(np.array(file['.']['image_vector'][()]))
    ans.append(np.array(file['.']['answers'][()]))
    ques.append(np.array(file['.']['ques_train'][()]))
    count+=1
    if count%100 == 0:
      print("The count is:",count,"and the percentage proportion
is:",round(100*count/length,2),'%')
images = tf.convert_to_tensor(np.array(images))   # For the GPU purpose
ans = tf.convert_to_tensor(np.array(ans))
ques = tf.convert_to_tensor(np.array(ques))


l = []
length = images.shape[0]
for i,j in enumerate(images):
  l.append(dimen_red(j))     # Making it to the same shape as that of question embedding
  if i%100 ==0:
    print("The count is:",i,"and the percentage proportion is:",round(100*i/length,2),'%')
images = tf.convert_to_tensor(np.array(l))
```

```
images = tf.reshape(images,[length,21,300])
img = images  #Tensor containing images
que = ques  # Tensor containing question vector
img = img/255.0 # Normalizing

que.shape,img.shape

ques = tf.keras.layers.Input((21,300))  # Input Model (for ques)
images = tf.keras.layers.Input((21,300)) # Input Model (for images)

from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Embedding, LSTM, GlobalMaxPooling1D,
SpatialDropout1D,Flatten,Concatenate

''' Imagica is for the preprocessing of the image part'''
imagica = Dense(512,activation='tanh')(images)
imagica = Dense(512,activation='tanh')(images)


''' quesa is for the ques layer, which means preprocessing of the question layer'''
quesa = LSTM(512, dropout = 0.3,return_sequences = True,input_shape =
(21,300))(ques)
quesa = Dense(512, activation = 'relu')(quesa)
quesa = Dropout(0.3)(quesa)
quesa = Dense(512, activation = 'relu')(quesa)
quesa = Dropout(0.3)(quesa)
```

```
''' Concatenating both image and the question layer'''
quesa = Concatenate()([quesa,imagica])
quesa = Flatten()(quesa)
out = tf.keras.layers.Dense(476,activation='softmax')(quesa) # Final output has 476
different categories, you can check by finding length of uniquue answers :)

model.compile(optimizer=tf.keras.optimizers.Adam(lr = 0.01),loss
='sparse_categorical_crossentropy',metrics=['accuracy'])

answers = tf.keras.utils.to_categorical(ans)
answers.shape

model.summary()

history=model.fit([img,que],ans,epochs = 50,batch_size=32,verbose=1)

model.save('VQA_Model')

prediction = tf.argmax(model.predict([img,que]),axis=1).numpy()
count=0
for i in range(len(ans)):
    if(prediction[i]==ans[i]):
        count=count+1
acc=(count/len(ans))*100
acc

import json
x = open('trainset.json','r')
train = json.load(x)
```

```
train[0]


import matplotlib.pyplot as plt
# summarize history for accuracy
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


import pickle
x = open('cache/trainval_label2ans.pkl','rb')
a2lab = pickle.load(x)
import cv2
img_path = 'VQA_RAD Image Folder/'
import matplotlib.pyplot as plt
%matplotlib inline
for i in range(760,770,2):
  plt.figure(figsize=(10,8))
```

```
image = cv2.imread(img_path+ train[i]['image_name'])
plt.imshow(image)
value = "Actual: "+str(train[i]['answer'])+' predicted value: '+ str(a2lab[prediction[i]])
plt.axis('off')
print("The Question is:",train[i]['question'])
print("The answer is:",value)
print("*"*50)
plt.show()
```

CHAPTER 10

REFERENCES

[1] Q. Wu, D. Teney, P. Wang, C. Shen, A. Dick, and A. van den Hengel, ‘‘**Visual questionanswering: A survey of methods and datasets**’’ Comput. Vis. Image Understand., vol. 163, pp. 21–40, Oct. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/ S1077314217300772

[2] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh,‘‘**VQA: Visualquestion answering**’’ in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 2425–2433.

[3] D. Gurari, Q. Li, A. J. Stangl, A. Guo, C. Lin, K. Grauman, J. Luo, and J. P. Bigham, ‘‘**VizWizgrand challenge: Answering visual questions from blind people**’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 3608–3617.

[4] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, ‘‘**Embodied questionanswering**’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 2054–2063.

[5] A. Das, S. Kottur, K. Gupta, A. Singh, D. Yadav, J. M. Moura, D. Parikh, and D. Batra, ‘‘**Visualdialog**’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 326–335.

[6] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, ‘‘**Making the v in VQA matter: Elevating the role of image understanding in visual question answering**’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 6904–6913.

[7] J. P. Bigham, S. White, T. Yeh, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller,A. Tatarowicz, and B. White, ‘‘**VizWiz: Nearly real-time answers to visual questions**’’ in Proc. 23nd Annu. ACM Symp. User Interface Softw. Technol. UIST, 2010, pp. 333–342, doi: 10. 1145/1866029.1866080.

[8] D. Gurari and K. Grauman, ‘‘**CrowdVerge: Predicting if people will agree on the answer to a visual question**’’ in Proc. CHI Conf. Hum. Factors Comput. Syst., May 2017, pp. 3511–3522, doi:10.1145/3025453.3025781.

[9] N. Bhattacharya, Q. Li, and D. Gurari, ‘‘**Why does a visual question have different answers?**’’in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 4271–4280.

[10]F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh, ‘‘**Quality controlin crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions**’’ACM Comput. Surv., vol. 51, no. 1, pp. 1–40, Apr. 2018, doi: 10.1145/314814