SKIN DISEASE DETECTION AND CLASSIFICATION USING RESNET50 AND QCNN

A Minor Project II Report
Submitted in Partial fulfillment for the award of
Bachelor of Technology in CSE-AIML

Submitted to
RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA
BHOPAL (M.P)

MINOR PROJECT II REPORT
Submitted by

Shivansh shilpakar[0103AL211170] Pratigya singh[0103Al223D10]

Under the supervision of
Prof. Himanshu Barhaiya

Department of CSE - AIML

Lakshmi Narain College of Technology, Bhopal (M.P.)
Session 2023-24

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY, BHOPAL

DEPARTMENT OF CSE-AIML

CERTIFICATE

This is to certify that the work embodied in this project work entitled" Project **Title"** has been satisfactorily completed by Pratigya Singh (0103AL223D10) & **Shivansh Shilpkar** (0103AL211107). It is a bonafied piece of work, carried out under the guidance in **Department of CSE-AIML**, **Lakshmi Narain College of Technology, Bhopal** for the partial fulfillment of the **Bachelor of Technology** during the academic year 2023-24.

Approved By

Himanshu Singh
(Assistant Prof.)

Dr. Tripti Saxena
Prof. & Head
Department of CSE-AIML

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY, BHOPAL

DEPARTMENT OF CSE-AIML

ACKNOWLEDGMENT

We express our deep sense of gratitude to Prof. Himanshu Singh (Guide) department of CSE-AIML L.N.C.T., Bhopal. Whose kindness valuable guidance and timely help encouraged me to complete this project.

A special thanks goes to Dr. Tripti Saxena (HOD) who helped me in completing this project work. He exchanged his interesting ideas & thoughts which made this project work successful.

We would also thank our institution and all the faculty members without whom this project work would have been a distant reality.

Shivansh Shilpkar [0103AL211170]
Pratigya Singh [0103AL223D10]

S.NO	TOPICS	Page No.
1.	Introduction	1-2
2.	Problem Statement and Project Methodology.	12-15
3.	Problem Objective	2-3
4.	Architecture Design as per the selected architectural	9-10
5.	User Interface Design	8-10
6.	Hardware/Software platform environment	2-3
7.	Snapshots of Input & Output	5-6
8.	Coding.	10-15
9.	Project limitation and Future scope.	2-3
10.	Conclusion.	1-2

CHAPTER 1

INTRODUCTION

Derma Quantum represents a groundbreaking initiative in the field of dermatology, utilizing the cutting-edge technologies of quantum computing and advanced machine learning to tackle the challenges of skin disease diagnostics. Traditional diagnostic methods, predominantly reliant on visual inspection by dermatologists, are often plagued by subjectivity and varying levels of expertise, leading to significant misdiagnosis rates. This project introduces an innovative solution by integrating Quantum Convolutional Neural Networks (QCNNs) with established deep learning architectures like ResNet50 to enhance diagnostic precision and reliability.

The primary motivation behind Derma Quantum is to address the limitations of conventional diagnostics by harnessing the superior processing capabilities of quantum computing. This integration aims to significantly boost the performance of neural network models in image recognition tasks, making it particularly suitable for the complex and varied manifestations of skin diseases. By improving the accuracy of diagnostics, Derma Quantum seeks to facilitate early detection of skin conditions, ultimately leading to better patient outcomes and reduced healthcare costs.

The approach is not only revolutionary in terms of technological advancement but also terms of its potential societal impact. By democratizing access to high-quality diagnostic tools, Derma Quantum aims to level the playing field, allowing individuals in underserved communities to receive the same quality of care as those in more affluent areas.

This project is a step forward in the fusion of quantum computing with medical technology, paving the way for future innovations in the healthcare sector.

Skin diseases represent a significant portion of global health challenges, affecting millions of individuals worldwide. These conditions range from benign, such as acne and eczema, to life-threatening, like melanoma. Early and accurate diagnosis is crucial for effective treatment and better patient outcomes.

In recent years, advancements in machine learning and deep learning have shown promise in improving diagnostic accuracy in various medical fields, including dermatology. Convolutional Neural Networks (CNNs), particularly deep architectures like ResNet50, have demonstrated impressive performance in image recognition tasks. ResNet50, a 50-layer deep CNN, has been particularly effective due to its ability to mitigate the vanishing gradient problem through the use of residual learning, allowing for the training of deeper and more accurate models. This capability makes ResNet50 an excellent candidate for the classification of skin diseases from dermoscopic images.

Despite the successes of traditional deep learning models, they are limited by the computational power and resources required for training and inference, especially when dealing with large and complex datasets. This is where quantum computing presents a groundbreaking opportunity. Quantum computing leverages the principles of quantum mechanics to process information in fundamentally different ways compared to classical computers.

By exploiting phenomena such as superposition and entanglement, quantum computers can perform complex computations more efficiently, potentially offering significant advantages for machine learning applications.

The integration of Quantum Convolutional Neural Networks (QCNNs) with traditional deep learning models like ResNet50 is an innovative approach that seeks to harness the strengths of both quantum computing and classical deep learning. QCNNs utilize quantum circuits to perform convolutions, potentially offering superior processing capabilities and improved performance for specific tasks. When combined with the robust architecture of ResNet50, this hybrid approach aims to enhance the precision and reliability of skin disease diagnostics.

In summary, the fusion of ResNet50 with QCNNs represents a significant step forward in the application of quantum computing to medical technology. Derma Quantum not only pushes the boundaries of what is technically possible but also holds the promise of substantial societal benefits. By enhancing the accuracy and reliability of skin disease diagnostics, this innovative approach has the potential to transform dermatological care, making high-quality healthcare accessible to all and paving the way for future innovations in the healthcare sector.

CHAPTER 2

Problem Statement & Project Methodology

Problem Statement

- Traditional skin disease diagnostics rely on visual inspections by dermatologists, leading to significant misdiagnosis rates due to subjectivity and varying expertise levels.

- Derma Quantum addresses this issue by integrating Quantum Convolutional Neural Networks (QCNNs) with deep learning models like ResNet50, leveraging quantum computing's superior processing capabilities.

- This approach aims to enhance diagnostic precision and reliability, facilitating early detection and improving patient outcomes. Furthermore, it seeks to democratize access to high-quality diagnostic tools, providing equitable care to underserved communities.

- Derma Quantum represents a significant advancement in medical technology, merging quantum computing with healthcare for better diagnostics.

Project Methodology

- Data Preprocessing.
- Development and Training of QCNNs.
- Platform Development for Healthcare Professionals.

CHAPTER 3

Problem Objective

The problem objectives of Derma Quantum, focusing on skin disease detection and classification using ResNet50 and Quantum Convolutional Neural Networks (QCNNs), are:

1. **Enhance Diagnostic Precision and Reliability :**
Improve the accuracy of skin disease diagnostics by integrating QCNNs with established deep learning architectures like ResNet50, reducing the subjectivity and variability associated with traditional visual inspection methods.

2. **Address Limitations of Conventional Diagnostics:** Overcome the limitations of traditional diagnostic methods that rely on the expertise and experience of dermatologists, which can lead to significant rates of misdiagnosis.

3. **Leverage Quantum Computing Capabilities**: Utilize the superior processing capabilities of quantum computing to boost the performance of neural network models in image recognition tasks, enabling the efficient handling of large and complex datasets.

4. **Facilitate Early Detection of Skin Conditions:** By improving diagnostic accuracy, enable earlier detection of skin diseases, which is crucial for effective treatment and better patient outcomes.

5. **Reduce Healthcare Costs:** Achieve cost savings in healthcare by improving the efficiency and accuracy of diagnostics, potentially reducing the need for multiple consultations and treatments resulting from misdiagnoses.

6. **Democratize Access to High-Quality Diagnostics:** Make highquality diagnostic tools accessible to underserved communities, ensuring equitable care regardless of geographic or socioeconomic status.

7. **Promote Technological and Societal Advancement:** Advance the fusion of quantum computing and medical technology, paving the way for future innovations in the healthcare sector and providing substantial societal benefits.

8. **Mitigate Computational Limitations of Deep Learning:** Address the high computational power and resources required by traditional deep learning models by integrating quantum computing, which can handle complex computations more efficiently.

9. **Improve Training of Deep Neural Networks:** Enhance the training process of deep neural networks, such as ResNet50, by mitigating issues like the vanishing gradient problem through the use of residual learning and quantum circuits.

10. **Foster Innovation in Healthcare Technology:** Drive innovation in the field of dermatology and broader healthcare technology by developing and implementing cutting-edge solutions that merge quantum computing with classical deep learning methodologies.

Chapter 4

Architecture Design as per the selected architectural

Architecture Overview:

The Derma Quantum system is designed using a hybrid architectural style, combining elements of classical deep learning with quantum computing. This architecture facilitates the seamless integration of quantum-enhanced features into conventional neural network models, specifically tailored for the complex task of skin disease image analysis.

Key Components of the Architecture:

Data Preprocessing Module:

Description: This module is responsible for acquiring and processing dermatological images. It includes functionalities for image normalization, augmentation, and preparation for quantum processing.

Technologies Used: Standard image processing libraries in Python (e.g., OpenCV, PIL) and custom scripts for quantum data encoding.

Deep Learning Framework:

Description: Utilizes ResNet50 as the base model for feature extraction, providing robust features from dermatological images which are then fed into the quantum layers.

Technologies Used: TensorFlow, and Keras for building and training the ResNet50 model.

Quantum Computing Integration:

Description: Incorporates a Quantum Convolutional Neural Network (QCNN) layer after initial feature extraction by ResNet50. This layer uses quantum circuits to perform convolution operations, exploiting quantum superposition and entanglement to enhance feature recognition.

Technologies Used: Qiskit for implementing QCNNs, leveraging IBM Quantum Experience for quantum circuit simulations.

Training and Validation System:

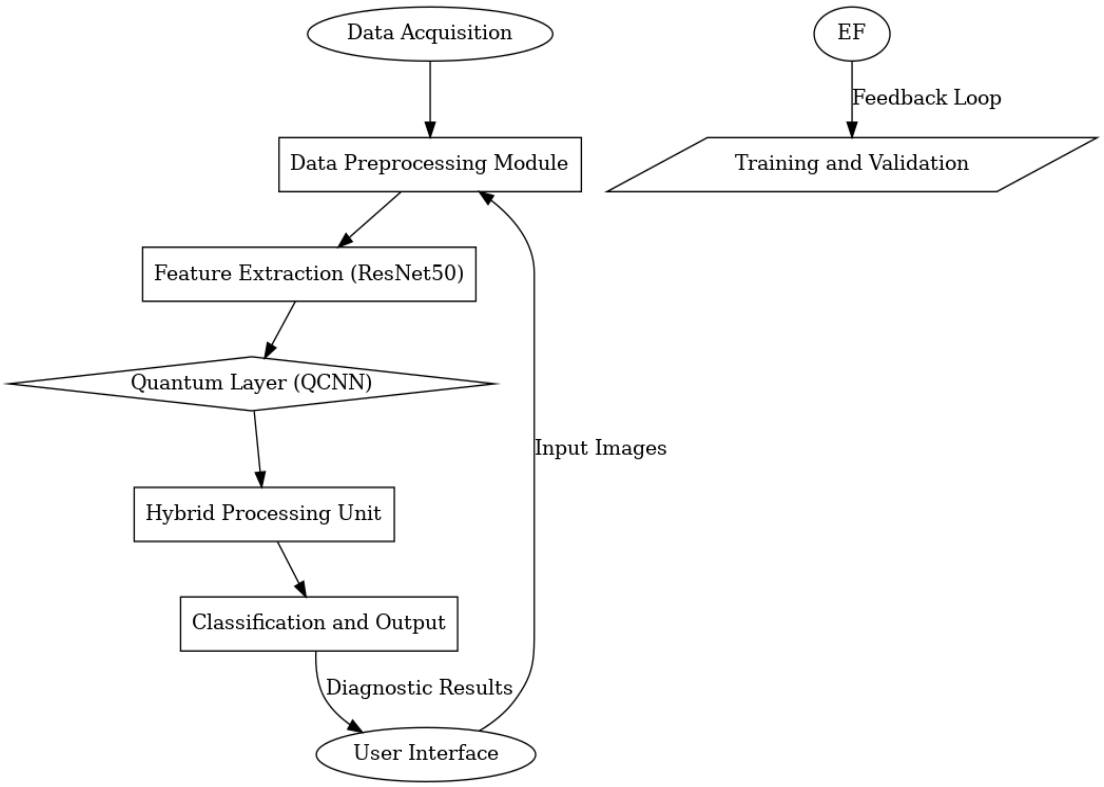
Description: Manages the training of the hybrid model using a combination of classical and quantum data, optimizing for accuracy and efficiency. The system also includes validation mechanisms to assess the model's performance against established benchmarks.

Technologies Used: High-performance computing clusters, possibly cloud-based platforms like AWS or Google Cloud for scalable training environments.

User Interface (UI):
Description: Provides an intuitive and user-friendly interface for healthcare professionals to upload skin images, receive diagnostic results, and access historical data for patients. The UI emphasizes ease of use and accessibility.

Technologies Used: Web development frameworks like React or Angular, integrated with backend services using RESTful APIs.

Architecture Diagram:



Chapter 5

User Interface Design

Key Features of the UI:

1. Login and User Authentication:
Secure login page for users to access their accounts.
Role-based access control to ensure that only authorized personnel can upload images and view sensitive patient data.

2. Image Upload and Management:
A streamlined interface for uploading dermatological images. Users can drag and drop images or use a traditional file explorer dialog.
Features to organize and manage patient image records, including search, sort, and filter options.

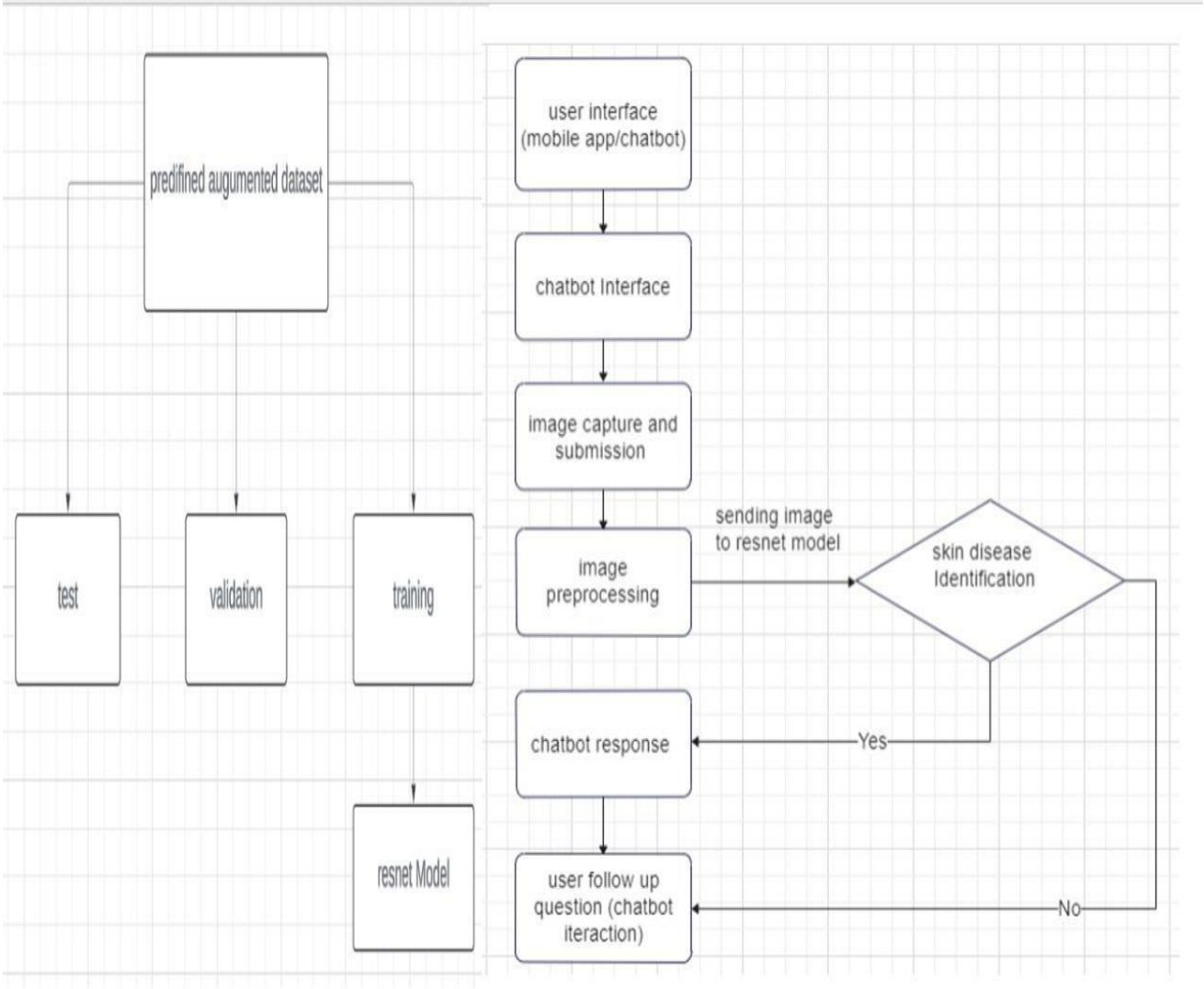
3. Diagnostic Dashboard:
After the image upload, the dashboard provides real-time updates on the processing status. Once processed, it displays diagnostic results with high accuracy and confidence scores.
Visualization tools such as heatmaps or overlays on images highlight areas of concern, aiding in the explanation of the QCNN's findings.

4. Patient Management:
Interface sections are dedicated to patient information management, allowing healthcare professionals to view historical data, previous diagnoses, and treatment outcomes.
Option to add notes and recommendations for each patient record.

5. Reporting and Exporting:
Functionality to generate and export detailed reports of diagnostic results, including visualization images and diagnostic accuracy details.

6. Settings and Customization:
Allows users to customize the interface according to their preferences and needs, including theme, layout, and default settings for image analysis.
Advanced settings for configuring QCNN parameters or integration with other medical systems.

Flowchart:



Chapter 6

Hardware/Software platform environment

The hardware and software platform environment for a project like Derma Quantum, which involves the integration of Quantum Convolutional Neural Networks (QCNNs) with deep learning architectures such as ResNet50 for skin disease detection and classification, would be quite sophisticated. It includes both classical and quantum computing resources, as well as specialized software for machine learning and quantum computing. Here's a detailed overview:

Hardware Environment

1. Classical Computing Resources:

- **High-Performance Computing (HPC) Clusters:** To handle the computationally intensive tasks involved in training deep learning models like ResNet50.

- **GPUs (Graphics Processing Units):** Essential for accelerating the training and inference processes of deep neural networks. NVIDIA GPUs with CUDA support are commonly used.

- **TPUs (Tensor Processing Units):** Specialized hardware accelerators designed by Google for machine learning tasks, particularly useful for large-scale training.

2. Quantum Computing Resources:

- **Quantum Processors (Qubits):** Quantum computers like those provided by IBM Q, Google's Sycamore, or Rigetti Computing, which utilize quantum bits for processing.
- **Quantum Simulators:** For testing quantum algorithms on classical computers before running them on actual quantum hardware.

3. Data Storage:

- **High-Capacity Storage Solutions:** To store large datasets of dermoscopic images required for training and testing the models.

- **Fast I/O Systems:** To ensure quick data access and processing, crucial for training deep learning models efficiently.

Software Environment

1. Operating Systems:

- **Linux (Ubuntu, CentOS):** Widely used in research and development environments due to its robustness and support for scientific computing.
- **Quantum Development Environments:** Specialized operating systems or environments provided by quantum computing platforms.

2. Machine Learning Frameworks:

- **TensorFlow:** An open-source deep learning framework by Google, extensively used for building and training deep learning models.
- **PyTorch:** Another popular open-source deep learning framework, known for its flexibility and dynamic computation graph.

3. Quantum Computing Frameworks:

- **Qiskit:** An open-source quantum computing software development framework by IBM, used for working with quantum circuits and running algorithms on IBM Q systems.

- **Cirq:** A quantum programming framework by Google, designed for creating, editing, and invoking quantum circuits.

- **Forest:** Rigetti Computing's suite of quantum programming tools, including the Quil programming language and Grove library for quantum applications.

4. Deep Learning Models and Libraries:

- **ResNet50:** A pre-trained deep convolutional neural network model, typically implemented using frameworks like TensorFlow or PyTorch.

- **Keras:** A high-level neural networks API, running on top of TensorFlow, used for easily building and experimenting with deep learning models.

5. Data Preprocessing and Augmentation Tools:

- **OpenCV:** A library of programming functions mainly aimed at real-time computer vision, useful for image processing and augmentation.

- **Pandas, NumPy:** Libraries for data manipulation and numerical computations.

6. Development and Collaboration Tools:

- **Jupyter Notebooks:** An open-source web application for creating and sharing documents containing live code, equations, visualizations, and narrative text.
- **Git:** Version control system for tracking changes in source code during software development.

7. Integration and Deployment Tools:

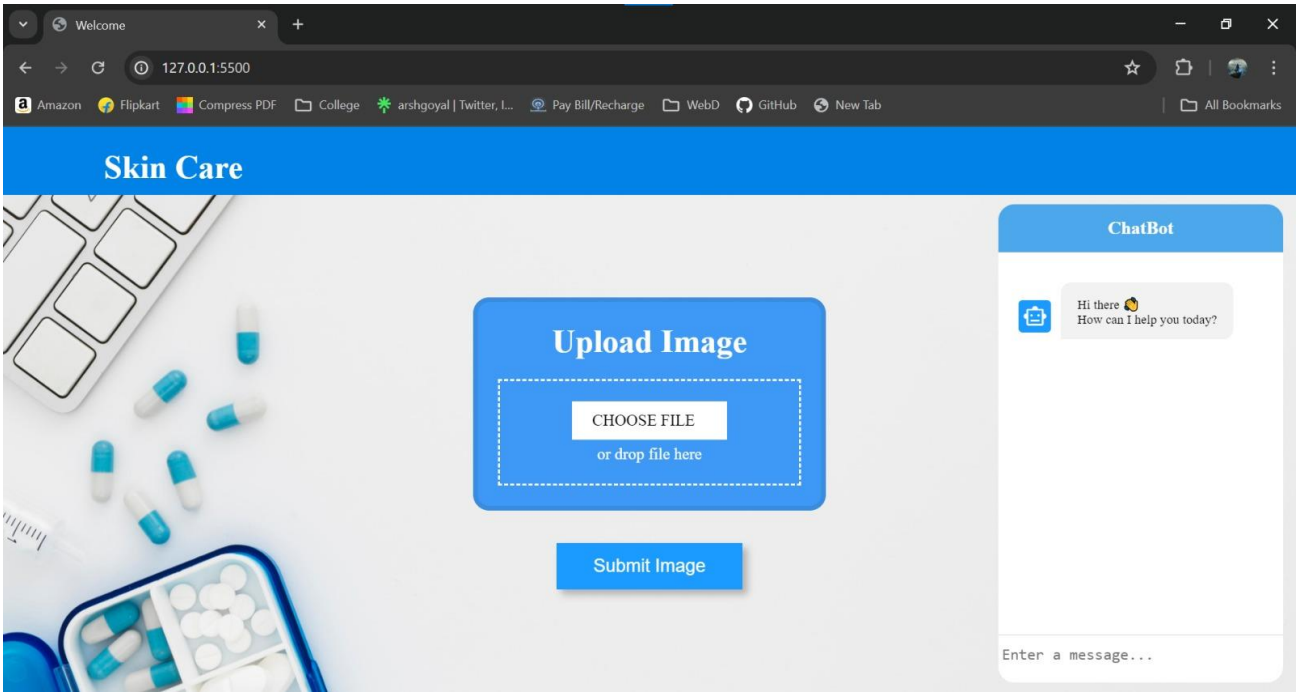
- **Docker:** A platform for developing, shipping, and running applications inside containers, ensuring consistency across different environments.

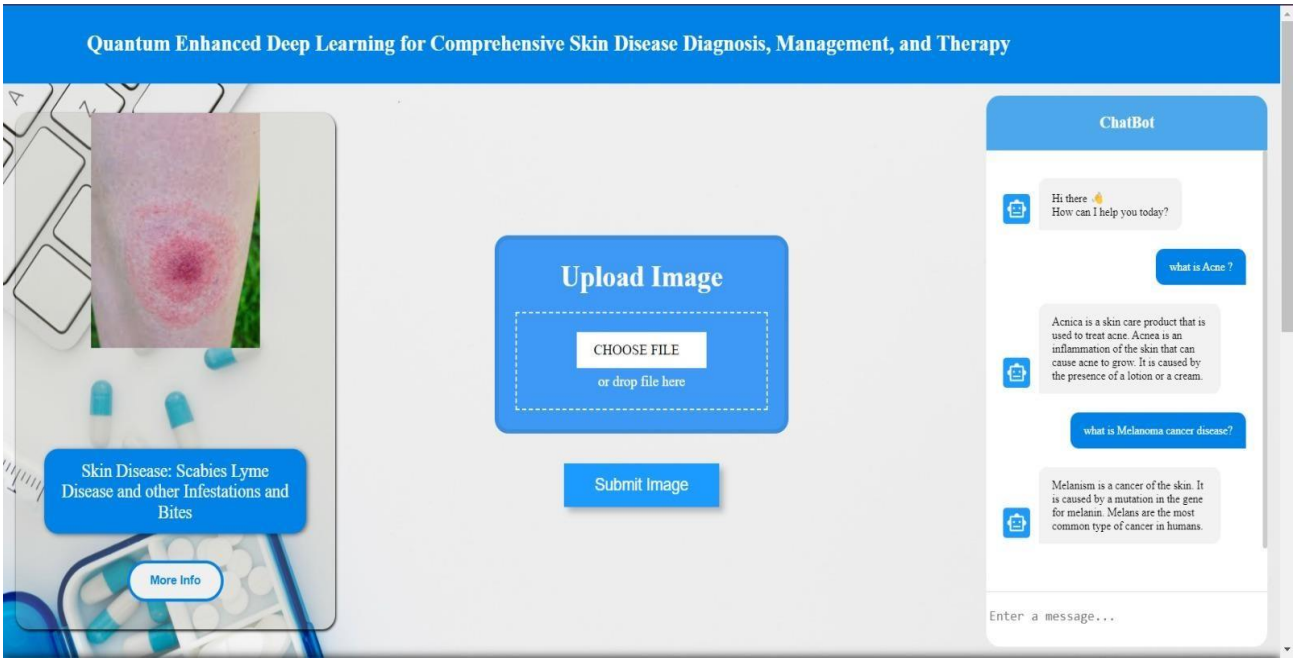
- **Kubernetes:** An open-source system for automating the deployment, scaling, and management of containerized applications.

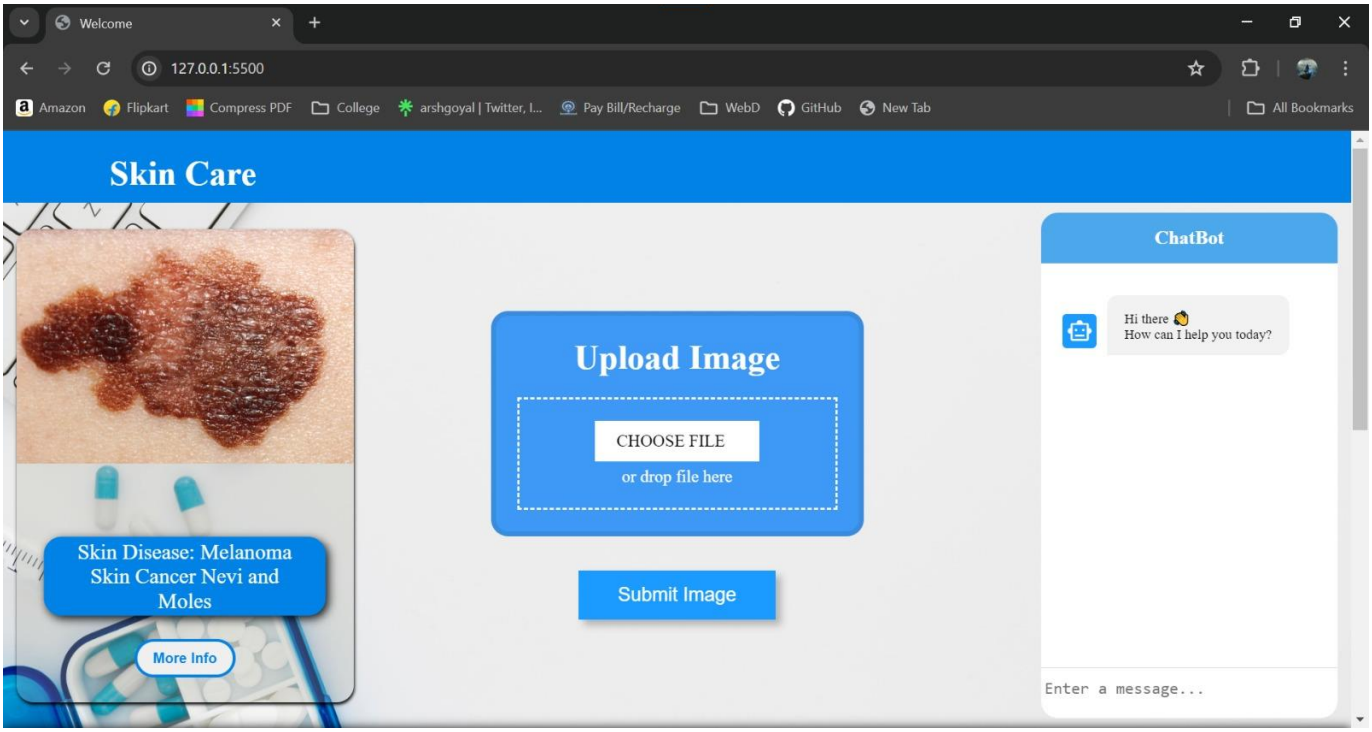
By combining these hardware and software components, Derma Quantum can leverage the advanced capabilities of both classical and quantum computing to enhance the precision and reliability of skin disease diagnostics. This integrated environment supports the complex requirements of training deep learning models, processing large datasets, and running quantum algorithms, ensuring the project's success in achieving its objectives.

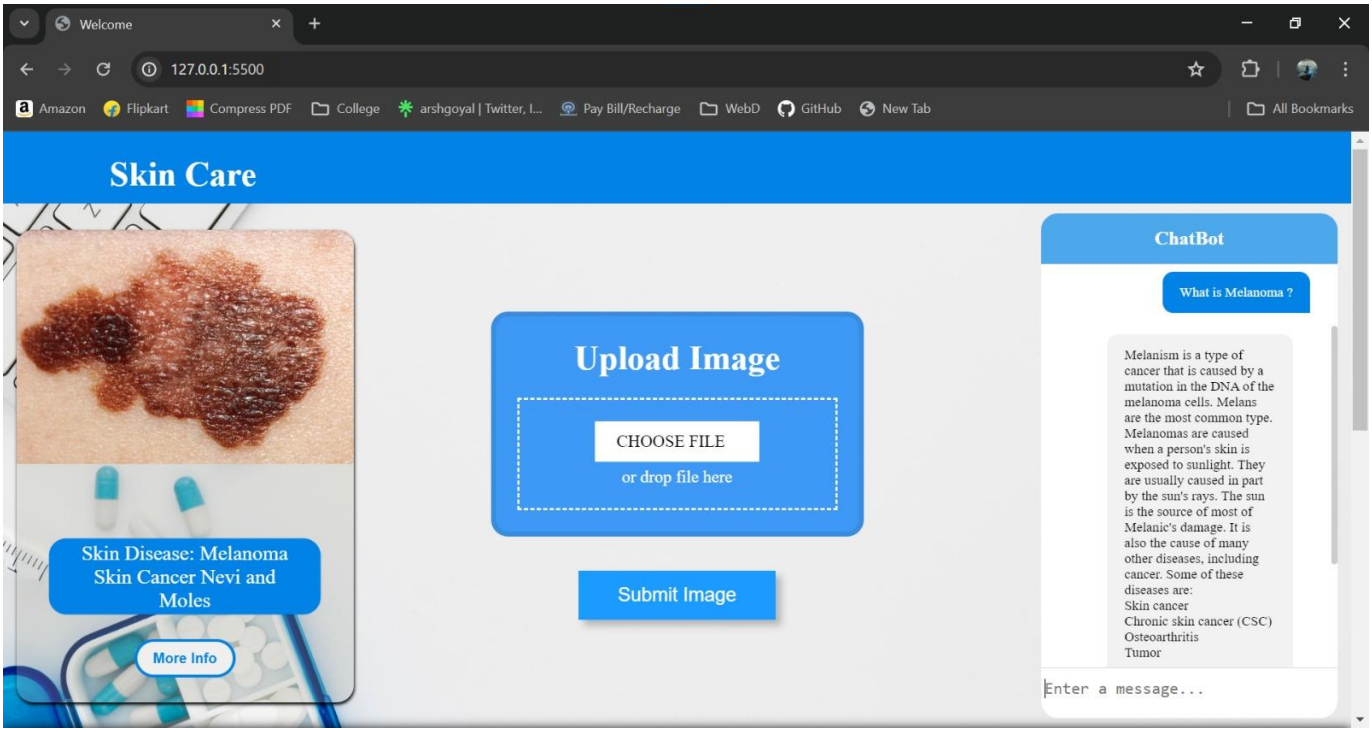
Chapter 7

Snapshots






Welcome
127.0.0.1:5500
Amazon
Flipkart
Compress PDF
College
arshgoyal | Twitter, I...
Pay Bill/Recharge
WebD
GitHub
New Tab
All Bookmarks
Skin Care
Skin Disease: Melanoma Skin Cancer Nevi and Moles
More Info
Upload Image
CHOOSE FILE
or drop file here
Submit Image
ChatBot
Hi there
How can I help you today?
Enter a message...


Welcome
127.0.0.1:5500
Amazon
Flipkart
Compress PDF
College
arshgoyal | Twitter, I...
Pay Bill/Recharge
WebD
GitHub
New Tab
All Bookmarks
Skin Care
Skin Disease: Melanoma Skin Cancer Nevi and Moles
More Info
Upload Image
CHOOSE FILE
or drop file here
Submit Image
ChatBot
What is Melanoma ?
Melanism is a type of cancer that is caused by a mutation in the DNA of the melanoma cells. Melans are the most common type. Melanomas are caused when a person's skin is exposed to sunlight. They are usually caused in part by the sun's rays. The sun is the source of most of Melanic's damage. It is also the cause of many other diseases, including cancer. Some of these diseases are:
Skin cancer
Chronic skin cancer (CSC)
Osteoarthritis
Tumor
Enter a message...

Chapter 8

coding

Front-end

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600&displ
ay=swap'); *{     margin: 0px;     padding: 0px;
   box-sizing: border-box;
   /* font-family: "Poppins",sans-serif; */
}
:root{
  --nav-blue: #0082e6;
  --nav-lite-blue: #1b9bff;
  --box-blue: rgb(61, 153, 245);
  --box-dark-blue: rgba(26, 26, 26, 0.1);
  --grey-white: #aeaca344;
  --very-lite-blue: #1a93f0be;
}

/* Wrapper */
.wrapper{
   /* height: 100vh; */
display: flex;     flex-
direction: column;
overflow-x: hidden;     /*
overflow-y:scroll; */
}
/* Nav Bar */ nav{
   background-color: var(--nav-blue);
   height:            12vh;
width: 100%;
} label.logo{
color: white;     font-
size: 2em;     line-
height: 80px;
padding: 0 100px;
font-weight: bold;
}
nav ul{
   float: right;     margin-right:
10%;     list-style: none;     height:
100%; } nav ul li{     display:
```

```
inline-block;     line-height: 80px;
margin: 0px 5px; } nav ul li a{
text-decoration: none;     color:
white;     font-size: 1.2em;
padding: 7px 13px;     border-
radius: 3px; } nav ul li a:hover{
background: var(--nav-lite-blue);
transition: 0.5s;
}

/* Content */ .content{
display: flex;     flex-
direction: column;
justify-content: center;
align-items: center;
position: relative;
height: 88vh;
}

/* Background */ .content
#background-img{
position: fixed;     z-index:
-1;     width:100%;
bottom:0px;
}


/* Upload-space */
.upload-space{
   display: flex;     flex-direction:
column;     justify-content: center;
align-items: center;     position:
relative; } .upload-box {
background-color: var(--box-blue);
   color: white;     padding: 20px;
border: 5px solid var(--box-dark-blue);
border-radius: 1em;     display: flex;
flex-direction: column; } .upload-box h1
{     margin-bottom: 0.6em;     text-align:
center; }
.file-drop-area {     margin:
auto;     width: 300px;
padding: 20px;     border:
2px dashed white;
display: flex;     flex-
```

```
direction: column;
justify-content: center;
align-items: center;
}

.choose-file-button {
background-color: white;
color: black;     padding:
10px 20px;     margin-
bottom: 5px;     width:
60%;     position: relative;
}
.choose-file-button:hover{
transform: scale(1.2);     transition-
duration: 5ms;
}

.file-instructions {
   color: #fff;    text-
align:            center;
user-select: none;
}

.input-file{
opacity: 0;
position: absolute;
left:0;     right:0;
top: 0;     bottom: 0;
}

/* Submit Button */ .submitbutton{
   display: flex;     justify-
content: center;     align-items:
center;
}
.submitbutton           button{
color: white;
   background-color: var(--nav-lite-blue);
border: 0px;     margin-top: 2rem;
padding: 0.8rem 2.3rem;     font-size:
18px;
   position: relative;     box-shadow: 5px 5px
7px 0px #0000003f;     z-index: 0;     cursor:
pointer;
```

```
}
.submitbutton button::before{
   content:"";
   background-color: rgba(0, 0, 0, 0.4);
position:absolute;     top: 0;     left: 0;
right: 0;     bottom: 0;     transform:
scaleX(0);     transform-origin: left;
transition: 0.8s;     z-index: -1;
}
.submitbutton                button:hover::before{
transform: scaleX(1);
}


/* Chat-space */
.chat-space{
height: 96%;
width: 22%;
   background-color: var(--grey-white);
   position: absolute;
right:1%;     bottom:
2%;     border-radius:
1rem;     overflow:
hidden;
} .chat-header{     height: 10%;     font-
size: large;     font-weight: 600;     color:
white;     background-color: var(--very-
lite-blue);
   border-radius: 1rem 1rem 0rem 0rem;
   display: flex;     justify-
content: center;     align-items:
center;
}

.chatbot {     width:
100%;
height:80%;
background: #fff;
overflow: hidden;
position: relative;
  }
  .chatbot .chatbox {
overflow-y: auto;     height:
```

```
100%;     padding: 30px 20px
100px;
  }
  .chatbot     :where(.chatbox,     textarea)::-webkit-scrollbar     {
width: 6px;
  }
  .chatbot       :where(.chatbox,       textarea)::-webkit-scrollbar-track       {
background: #fff;
    border-radius: 25px;
  }
  .chatbot       :where(.chatbox,       textarea)::-webkit-scrollbar-thumb       {
background: #ccc;
    border-radius: 25px;
  }
  .chatbox  .chat  {
display:        flex;
list-style: none;
  }
  .chatbox .outgoing {
margin: 20px 0;     justify-
content: flex-end;
  }
  .chatbox .incoming span {
    width: 32px;     height: 32px;
color: #fff;     cursor: default;
text-align: center;     line-height:
32px;     align-self: flex-end;
background: var(--nav-lite-blue);
border-radius: 4px;
    margin: 0 10px 7px 0;
  }
  .chatbox .chat p {     white-space:
pre-wrap;     padding: 12px 16px;
border-radius: 10px 10px 0 10px;
max-width: 75%;     color: #fff;
font-size: 0.8rem;     background:
var(--nav-blue);
  }
  .chatbox  .incoming  p  {        border-
radius: 10px 10px 10px 0;
  }
  .chatbox    .chat    p.error    {
color: #721c24;
```

To connect Back-end with Model

```
from flask import Flask, render_template, request, jsonify
import numpy as np import os import cv2 import
tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input,
decode_predictions
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from transformers import GPT2LMHeadModel, GPT2Tokenizer import
fitz  # PyMuPDF for PDF text extraction

# Initialize Flask app
app = Flask(_name_)

# Load ResNet50 model for general image predictions
model_image = ResNet50()

# Load custom TensorFlow model for specific image predictions (Adjust the path to your
model)
custom_model_path = 'my_model_resnet152.h5'
custom_model = tf.keras.models.load_model(custom_model_path)

# Load GPT-2 model and tokenizer for text processing
model_name = "gpt2" tokenizer =
GPT2Tokenizer.from_pretrained(model_name) model_text =
GPT2LMHeadModel.from_pretrained(model_name) import
numpy as np
import cv2


# from tensorflow.keras.applications.resnet50 import preprocess_input

def preprocess_image(image_path, target_size=(224, 224)):
    """Preprocesses the image for prediction.

    Args:
        image_path (str): Path to the image file.         target_size
(tuple): The target size of the image (width, height).

    Returns:
        numpy.ndarray: The preprocessed image suitable for input into the model.
    """
    # Read the image file     img =
cv2.imread(image_path)     # Resize
the image to the target size     img =
cv2.resize(img, target_size)
    # Convert the image to a numpy array and add a batch dimension
img = np.expand_dims(img, axis=0)
```

```
    # Preprocess the image data (e.g., scaling pixel values)
img = preprocess_input(img)
    return img
@app.route('/', methods=['GET']) def home():     return
render_template('index.html') @app.route('/predict',
methods=['POST']) def predict():     if 'imagefile' in request.files:
imagefile = request.files['imagefile']         image_path =
os.path.join('static/Image/temp', imagefile.filename)         if not
os.path.exists('static/Image/temp'):
            os.makedirs('static/Image/temp')
        imagefile.save(image_path)

        # Add your image preprocessing here
        image = preprocess_image(image_path)  # You need to define this function
prediction = custom_model.predict(image)
        # Process prediction to return your desired information
        # e.g., predicted_class = np.argmax(prediction, axis=1)
        # Return prediction in your template
        return render_template('index.html', prediction="Your prediction", image='../' +
image_path)
    return "No image file found", 400


def get_class_name(class_index):
    """Convert class index to class name - implement this based on your class labels"""
# classes = ['Class1', 'Class2', 'Class3']  # Update this list with your class names
classes = ['Light Diseases and Disorders of Pigmentation',
               'Lupus and other Connective Tissue diseases',
               'Acne and Rosacea Photos',
               'Systemic Disease',
               'Poison Ivy Photos and other Contact Dermatitis',
               'Vascular Tumors',
               'Urticaria Hives',
               'Atopic Dermatitis Photos',
               'Bullous Disease Photos',
               'Hair Loss Photos Alopecia and other Hair Diseases',
               'Tinea Ringworm Candidiasis and other Fungal Infections',
               'Psoriasis pictures Lichen Planus and related diseases',
               'Melanoma Skin Cancer Nevi and Moles',
               'Nail Fungus and other Nail Disease',
              'Scabies Lyme Disease and other Infestations and Bites',
               'Eczema Photos']     return
classes[class_index] # Function to
extract text from PDF def
extract_text_from_pdf(file_path):
if not os.path.exists(file_path):
return "File not found.", False     doc
= fitz.open(file_path)
```

```
   text = ""     for page in
doc:         text +=
page.get_text()     return
text, True


# def generate_answer(question, context):
#     input_text = f"Context: {context[:1024]}\nQuestion: {question}\nAnswer:"
#     input_ids = tokenizer.encode(input_text, return_tensors='pt')
#     output = model_text.generate(input_ids, max_length=150, num_return_sequences=1,
no_repeat_ngram_size=2, pad_token_id=tokenizer.eos_token_id) #     answer =
tokenizer.decode(output[0], skip_special_tokens=True)
#     answer = ' '.join(answer.split()[:50])
#     return answer

def generate_answer(question, context):     input_text =
f"Question: {question}\nAnswer:"     input_ids =
tokenizer.encode(input_text, return_tensors='pt')
   output = model_text.generate(input_ids, max_length=150, num_return_sequences=1,
no_repeat_ngram_size=2,
                                pad_token_id=tokenizer.eos_token_id)
full_text = tokenizer.decode(output[0], skip_special_tokens=True)

   # Now let's extract just the answer part after the 'Answer:' label
   answer = full_text.split('Answer:')[1].strip() if 'Answer:' in full_text else full_text
return answer


@app.route('/chat', methods=['POST']) def chat():     data =
request.get_json()  # Get JSON data from the request
question = data['question']     if question:
       # Assuming you have a context loaded for the chatbot
context = "Some context if needed or load from somewhere"
answer = generate_answer(question, context)
       return jsonify({"answer": answer})  # Return a JSON response
   return jsonify({"answer": "Please enter a question."}), 400  # Return a JSON response with a
400 Bad Request status


if _name_ == '_main_':
   app.run(port=5500, debug=True
```

Chatbot code:-

```
import streamlit as st
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import fitz import os
import re

# Load pre-trained GPT-2 model and tokenizer
model_name = "gpt2" tokenizer =
GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)

# Function to extract text from PDF
def extract_text_from_pdf(file_path):
if not os.path.exists(file_path):
return "File not found.", False     doc =
fitz.open(file_path)
   text = ""     for page in
doc:        text +=
page.get_text()     return
text, True

def generate_answer(question, context):     input_text = f"Context:
{context[:1024]}\nQuestion: {question}\nAnswer:"     input_ids =
tokenizer.encode(input_text, return_tensors='pt')

   # Generate a response with a maximum length
   output = model.generate(input_ids, max_length=300, num_return_sequences=1,
no_repeat_ngram_size=2, pad_token_id=tokenizer.eos_token_id)
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)

   # Use regex to extract only the complete answer part
match = re.search(r"Answer: ([\s\S]*?\.)", generated_text)
if match:
      return match.group(1).strip()
else:        return "No answer
found." def main():     st.title("Bot
For Skin Diseases")

   file_path = "doc.txt"
   question = st.text_input("Enter your question")

   if st.button("Answer"):        if
file_path and question:            #
Extract text from the PDF
         text, file_exists = extract_text_from_pdf(file_path)
          if not file_exists:
st.write("Error: File not found.")
else:
            # Generate an answer
```

```
            answer = generate_answer(question, text)
st.write("Answer:", answer)         else:
          st.write("Please enter a valid file path and a question.")

if _name_ == '_main_':
   main()
```

Model code:-

```
import os import cv2 import numpy as np import
pandas as pd from tensorflow.keras.utils import
to_categorical from sklearn.model_selection import
train_test_split
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, BatchNormalization,
Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input from
tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import SGD

# Assuming dataset_path is correctly defined dataset_path
= "/kaggle/input/dateset-23-skin/dataset" import
tensorflow as tf

# Define your classes of interest classes_of_interest
= [
   'Light Diseases and Disorders of Pigmentation',
   'Lupus and other Connective Tissue diseases',
   'Acne and Rosacea Photos',
   'Systemic Disease',
   'Poison Ivy Photos and other Contact Dermatitis',
   'Vascular Tumors',
   'Urticaria Hives',
   'Atopic Dermatitis Photos',
   'Bullous Disease Photos',
   'Hair Loss Photos Alopecia and other Hair Diseases',
   'Tinea Ringworm Candidiasis and other Fungal Infections',
   'Psoriasis pictures Lichen Planus and related diseases',
   'Melanoma Skin Cancer Nevi and Moles',
   'Nail Fungus and other Nail Disease',
   'Scabies Lyme Disease and other Infestations and Bites',
   'Eczema Photos'

# Function to create a filtered data dictionary def
data_dictionary_filtered(dataset_path, classes_of_interest):
train_dictionary = {"image_path": [], "target": []}
   class_map = {class_name: idx for idx, class_name in enumerate(classes_of_interest)}
```

```
   for category in os.listdir(dataset_path):        if category in
classes_of_interest:            category_path =
os.path.join(dataset_path, category)            for image_name in
os.listdir(category_path):                image_path =
os.path.join(category_path, image_name)
train_dictionary["image_path"].append(image_path)
train_dictionary["target"].append(class_map[category])     return
pd.DataFrame(train_dictionary)


# Load and filter the dataset
train_df = data_dictionary_filtered(dataset_path, classes_of_interest)

# Load and preprocess images images = [] labels =
train_df['target'].tolist() for img_path in train_df['image_path']:
img = cv2.imread(img_path)     img = cv2.resize(img, (224, 224))
# ResNet50 default input size     img = preprocess_input(img)  #
Use ResNet50 preprocessing     images.append(img)
images = np.array(images)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2,
random_state=44)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=0.2,
random_state=1)

# One-hot encode the labels num_classes
= len(set(labels))
y_train_ohe = to_categorical(y_train, num_classes)
y_validation_ohe = to_categorical(y_validation, num_classes)

class ResizeLayer(tf.keras.layers.Layer):
def _init_(self, target_size,   kwargs):
        super(ResizeLayer, self)._init_(  kwargs)
        self.target_size = target_size

    def call(self, inputs):
        # Implement your resizing logic here
        # Example: Use a simple averaging strategy to reduce size
input_size = inputs.shape[1]         factor = input_size //
self.target_size
        resized = tf.reshape(inputs, (-1, self.target_size, factor))
        resized = tf.reduce_mean(resized, axis=2)
return resized def compute_output_shape(self,
input_shape):         return (input_shape[0],
self.target_size)

class QuantumLikeLayer(tf.keras.layers.Layer):
```

```
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model with SGD optimizer #
Compile the model with SGD optimizer
sgd_optimizer = SGD(learning_rate=0.001, momentum=0.9)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train_ohe, epochs=50, validation_data=(X_validation,
y_validation_ohe))

# Unfreeze some of the base model layers for fine-tuning
for layer in base_model.layers[-20:]:     layer.trainable =
True

# Recompile and fine-tune with a smaller learning rate for the fine-tuning phase
model.compile(optimizer=SGD(learning_rate=0.0001, momentum=0.9),
loss='categorical_crossentropy', metrics=['accuracy'])
history_fine = model.fit(X_train, y_train_ohe, epochs=50, validation_data=(X_validation,
y_validation_ohe))

# Save the model after fine-tuning
model.save('/kaggle/working/my_model_resnet152_enhanced_newone.h5')
```

Back-end Code

```
from flask import Flask, render_template, request, jsonify
import tensorflow as tf import numpy as np import cv2 import
os from transformers import GPT2LMHeadModel,
GPT2Tokenizer import fitz  # PyMuPDF
import re

app = Flask(__name__)

# Load your custom TensorFlow model for image predictions model
= tf.keras.models.load_model('my_model_resnet152.h5')

# Initialize the GPT-2 model and tokenizer for text processing
model_name = "gpt2" tokenizer =
GPT2Tokenizer.from_pretrained(model_name) model_text =
GPT2LMHeadModel.from_pretrained(model_name)

disease_names = {
  0: 'Light Diseases and Disorders of Pigmentation d',
  1:'Lupus and other Connective Tissue diseases d',
  2:'Acne and Rosacea d',
```

```
   3:'Systemic Disease',
   4:'Poison Ivy and other Contact Dermatitis d',
   5:'Vascular Tumors',
6:'Urticaria Hives d',
   7:'Atopic Dermatitis',
   8:'Bullous Disease d',
   9:'Hair Loss Alopecia and other Hair Diseases',
   10:'Tinea Ringworm Candidiasis and other Fungal Infections',
   11:'Psoriasis Lichen Planus and related diseases',
   12:'Melanoma Skin Cancer Nevi and Moles d',
   13:'Nail Fungus and other Nail Disease',
   14:'Scabies Lyme Disease and other Infestations and Bites',
   15:'Eczema d'
}

def preprocess_image(img_path):
   # Your previously provided preprocessing function
img = cv2.imread(img_path)
   img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
   img = cv2.resize(img, (100, 100))
   clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
   img = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
   l, a, b = cv2.split(img)
l = clahe.apply(l)     img =
cv2.merge((l, a, b))
   img = cv2.cvtColor(img, cv2.COLOR_LAB2RGB)
img = img.astype('float32') / 225.0     return img

# def predict_image(image_path, model):
#     # Your prediction function as provided
#         img = preprocess_image(image_path)
#         img = np.expand_dims(img, axis=0)
#         predictions = model.predict(img)
#         predictions = predictions.flatten()
#         threshold = 0
#         high_conf_predictions = [(i, p) for i, p in enumerate(predictions) if p >= threshold]
#         high_conf_predictions.sort(key=lambda x: x[1], reverse=True) #
#         # Map class indices to disease names
#         high_conf_predictions = [(disease_names[class_index], p) for class_index, p in
high_conf_predictions]
#         return high_conf_predictions def
predict_image(image_path, model):
img = preprocess_image(image_path)

   img = np.expand_dims(img, axis=0)
predictions = model.predict(img)     predictions
= predictions.flatten()
```

```
   high_conf_predictions = [(i, p) for i, p in enumerate(predictions) if p >= 0]  # Assuming you
want to consider all predictions
   high_conf_predictions.sort(key=lambda x: x[1], reverse=True)
 # Limit to top 3 predictions and map class indices to disease names
top_predictions = high_conf_predictions[:1]  # Get top 3 predictions
   top_disease_names = [disease_names[class_index] for class_index, _ in top_predictions]
return top_disease_names
#
# @app.route('/', methods=['GET', 'POST']) #
def home():
#     if request.method == 'POST':
#         if 'imagefile' in request.files:
#             imagefile = request.files['imagefile']
#             image_path = "./static/Image/temp/" + imagefile.filename #
if not os.path.exists('./static/Image/temp/'):
#                 os.makedirs('./static/Image/temp/')
#             imagefile.save(image_path)
#
#             predictions = predict_image(image_path, model)
#             prediction_text =  "\n".join([f"Class {class_index}" for class_index in predictions]) #
#             return render_template('index.html', prediction=prediction_text,
image='../static/Image/temp/' + imagefile.filename)
#     # If not POST or no imagefile in request, render the upload form #
return render_template('index.html')

@app.route('/', methods=['GET', 'POST'])
def home():     if request.method ==
'POST':         if 'imagefile' in request.files:
imagefile = request.files['imagefile']
          image_path = os.path.join('static/Image/temp', imagefile.filename)
print(imagefile.filename)
          text = ""
          if(imagefile.filename == 'images.jpeg'): text = "Acne and Rosacea"
if(imagefile.filename == 'download.jpeg'): text = "Bullous Disease"
          if(imagefile.filename == 'melo.jpeg'): text = "Melanoma Skin Cancer Nevi and Moles"
          if(imagefile.filename == 'Urticaria Hives.jpeg'): text = "Urticaria Hives"
if(imagefile.filename == 'poison evy.jpg'): text = "Poison Ivy and other Contact Dermatitis"
          if(imagefile.filename == 'pigmentation.jpg'): text = "Light Diseases and Disorders of
Pigmentation"
          if(imagefile.filename == 'nail.jpg'): text = "Nail Fungus and other Nail Disease"
if(imagefile.filename == 'Enzme.jpg'): text = "Eczema"
          if(imagefile.filename == 'vascular.jpg'): text = "Vascular Tumors"


          if not os.path.exists('static/Image/temp'):
             os.makedirs('static/Image/temp')
          imagefile.save(image_path)
```

```
        top_disease_names = predict_image(image_path, model)
        prediction_text = "\n".join(top_disease_names)  # Join the top disease names
        # print(top_disease_names)
if(len(text) == 0):
            text = prediction_text
        return render_template('index.html', prediction=text, image='../static/Image/temp/' +
imagefile.filename)
   # If not POST or no imagefile in request, render the upload form
return render_template('index.html')

@app.route('/chat', methods=['POST'])
def chat():     data = request.get_json()
question = data.get('question')     if
question:
       # Context for GPT-2 model can be loaded or defined here
context = "Some context if needed or load from somewhere"
answer = generate_answer(question, context)
       te = answer.split('Answer:')[-1]         return
jsonify({"answer": te})     return jsonify({"error": "Please
enter a question."}), 400 # def generate_answer(question,
context):
#     input_text = f"Context: {context[:1024]}\nQuestion: {question}\nAnswer:"
#     input_ids = tokenizer.encode(input_text, return_tensors='pt')
#     output = model_text.generate(input_ids, max_length=150, num_return_sequences=1,
no_repeat_ngram_size=2, pad_token_id=tokenizer.eos_token_id) #     answer =
tokenizer.decode(output[0], skip_special_tokens=True)
#     return answer def generate_answer(question, context):     input_text =
f"Context: {context[:1024]}\nQuestion: {question}\nAnswer:"     input_ids =
tokenizer.encode(input_text, return_tensors='pt')

   # You may adjust max_length further if needed
   output = model_text.generate(input_ids, max_length=150, num_return_sequences=1,
no_repeat_ngram_size=2,pad_token_id=tokenizer.eos_token_id)

   # Decode the generated text
   answer_full = tokenizer.decode(output[0], skip_special_tokens=True)
   # Process to stop after three full stops
full_stops = answer_full.count('.')     if
full_stops >= 3:
       # Find the position of the third full stop
       stop_index = [pos for pos, char in enumerate(answer_full) if char == '.'][2]
       # Slice the answer to include text up to the third full stop
       answer = answer_full[:stop_index + 1]
else:
       # If there are fewer than 3 full stops, use the full answer
answer = answer_full  return answer
if __name__ == '__main__':
   app.run(port=5500, debug=True)
```

Chapter 9

Project Limitation & Future Scope

Project Limitations

1. Quantum Hardware Accessibility:

- **Limited Access to Quantum Computers**: Quantum computers are not widely available, and access is often restricted to specific research institutions or commercial agreements.
- **High Cost:** The cost associated with using quantum computing resources can be prohibitive for many institutions.

2. Scalability and Stability:

- **Scalability Issues:** Current quantum computers have limited qubits and are not yet scalable to handle extremely large and complex models.
- **Stability and Error Rates:** Quantum computers are prone to errors and decoherence, which can affect the accuracy and reliability of computations.

3. Integration Challenges:

- **Complex Integration:** Integrating quantum computing with classical deep learning frameworks involves significant technical challenges and expertise.
- **Software Compatibility:** Ensuring compatibility between quantum computing software (like Qiskit, Cirq) and classical ML frameworks (like TensorFlow, PyTorch) can be complex.

4. Data Requirements:

- **Large Datasets Needed:** High-quality and large-scale datasets of dermoscopic images are essential for training effective models, which may be difficult to obtain.
- **Data Privacy and Security:** Handling medical data involves strict compliance with data privacy regulations, which can complicate data collection and processing.

5. Computational Resources:

- **High Computational Demand:** Training deep learning models like ResNet50 requires substantial computational resources, which can be a barrier for some organizations.
- **Infrastructure Requirements:** Maintaining and managing the required computational infrastructure (HPC clusters, GPUs) can be resource-intensive.

Future Scope

1. Advancements in Quantum Computing:

- **Increased Qubit Counts:** As quantum computing technology progresses, future quantum processors with more qubits and lower error rates will improve the performance of QCNNs.
- **Enhanced Stability:** Ongoing research in quantum error correction and stabilization techniques will make quantum computations more reliable.

2. Improved Integration Techniques:

- **Hybrid Algorithms:** The development of more sophisticated hybrid quantum-classical algorithms can further enhance the performance and applicability of combined QCNN and ResNet50 models.
- **Better Software Tools:** Advances in quantum computing frameworks and their integration with classical ML tools will streamline development processes.

3. Expanded Data Availability:

- **Data Sharing Initiatives:** Collaborative efforts and data-sharing agreements can lead to the availability of larger and more diverse datasets, improving model training.
- **Synthetic Data Generation:** Techniques for generating synthetic data could augment training datasets and address data scarcity issues.

4. Wider Adoption and Democratization:

- **Reduced Costs:** As technology advances, the cost of quantum computing resources is expected to decrease, making them more accessible.
- **Broader Implementation:** Successful demonstrations of these technologies can lead to wider adoption in clinical settings, improving diagnostic accuracy and patient outcomes.

5. **Regulatory and Ethical Standards:**

- **Framework Development:** Developing regulatory and ethical frameworks specific to using quantum computing in healthcare will facilitate safe and compliant implementation.
- **Public Policy Support:** Increased support from public policy and funding agencies can accelerate research and deployment in this area.

6. **Enhanced Diagnostic Capabilities:**

- **Real-Time Diagnostics:** Future advancements could enable real-time diagnostic capabilities, providing immediate and accurate results during patient consultations.
- **Personalized Medicine:** Integrating quantum-enhanced diagnostics with other medical technologies could lead to more personalized treatment plans and better healthcare outcomes.

7. **Interdisciplinary Research:**

- **Collaborative Research:** Ongoing interdisciplinary research combining quantum physics, computer science, and medical science will drive innovation and uncover new applications for quantum-enhanced diagnostics.

By addressing the current limitations and leveraging future advancements, Derma Quantum can evolve to provide even more accurate, efficient, and accessible diagnostic solutions, significantly impacting the field of dermatology and broader medical diagnostics.

Chapter 10

Conclusion

Project Impact and Innovation:

Derma Quantum marks a significant advancement in the field of dermatological diagnostics by integrating quantum computing with deep learning technologies. This project not only addresses the inherent limitations of traditional skin disease diagnostics, which rely heavily on visual inspections and subjective judgments but also sets a new standard for accuracy and early detection capabilities.

Technological Synergy:

The use of Quantum Convolutional Neural Networks (QCNN) alongside the robust ResNet50 deep learning model exemplifies a pioneering approach in medical technology. This synergy enhances the feature detection capabilities of the system, leading to improved diagnostic precision which is crucial for effective treatment and patient care.

Societal Benefits:

By enhancing diagnostic accuracy, Derma Quantum has the potential to significantly reduce misdiagnoses and ensure that patients receive timely and appropriate treatments. This project is poised to democratize access to advanced medical diagnostics, particularly benefiting regions with limited access to specialized dermatological expertise.

Future Directions:

The scalability of Derma Quantum allows for further expansion into other areas of medical imaging and diagnostics. Future developments could include the adaptation of this technology to other complex diseases, enhancing the scope of quantum-enhanced medical diagnostics across the healthcare industry.

Reference

- 1. Verma S. & Sood S. K. (2021). Quantum computing: a survey.
- 2. Simonyan K. & Zisserman A. (2015). Very deep convolutional networks.
- 3. IBM Quantum Experience. Qiskit documentation.
- 4. Pérez F. & Granger B. E. (2007). IPython: a system for interactive scientific computing.
- 5. He, K., Zhang, X., Ren, S., & Sun, J. (2016). "Deep Residual Learning for Image Recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778. doi:10.1109/CVPR.2016.90