Skin Cancer Detection Using AI on the Cloud

Client/Advisor:
Ashraf Gaffar

Senior design group:
SDMAY24-07

Team Members:
Anirudh Ambore, Mishari Alharbi, Ziyad Alqahtani,
Wonjun Choi, Evan Hanson, Abdelrahman Mohammed

Introduction

- Project Goal: Utilize AI and cloud computing to improve the detection of skin cancer.
- Primary Problem: Medical professionals' reliance on manual skin cancer diagnosis is inefficient and time-consuming.
- Ultimate Benefit: Increase the accuracy and speed of skin cancer detection, benefiting patients and healthcare providers.

Requirements & Constraints

- User Interface & Accessibility
  - Simple UI powered by web.
- AI Model for Accurate Diagnosis
  - Trained to classify cancerous/benign images.
- Cloud Deployment
  - Hosted on AWS/GCP.
  - Improves scalability, accessibility, and sustainability.
- Constraints
  - Specified level of accuracy.
  - Complying with data privacy regulations such as HIPAA.

Users & Operating Enviroment

- Users
  - Students
    - Benefiting from AI and cloud computing training that can be applied in various fields.
  - Medical Institutions
    - The AI model can assist healthcare professionals in skin cancer diagnosis by reducing misdiagnosis rates.
  - Patients
    - Faster and more accurate skin cancer detection can lead to early diagnosis and timely treatment.
- Operating Environment
  - Website applications will be expected to run on a standard browser.

Technical Details

- AI Model
  - Using Keras and ISIC database to implement and train the model.
  - The dataset is split into training, validation, and testing sets.
  - Images are resized to the same dimensions, and data augmentation is performed on training images.
- Cloud Architecture
  - GCP: Cloud Run runs the Flask app on the cloud with a default Compute Engine service. Cloud Storage stores the app resources and images.
  - AWS: A serverless architecture that leverages Lambda and EC2 to analyze images, store results in a DB with UUIDs, and retrieve them, effectively handling high traffic.

Testing

- Interface Testing: Testing the two main pages, home and result pages, using tools such as screenester.
- Integration Testing: Testing the communication between the different components of the system to ensure a proper flow.
- Security Testing: Validated defenses against common vulnerabilities by testing with random UUIDs, confirming that only authenticated requests retrieve data.
- Load Testing: Conducted using Locust to simulate 500 concurrent users, configuring the server for four processes and Auto Scaling for up to four EC2 instances, ensuring robust traffic handling.

Metrics (training on cloud)

Figure A. GCP _ after fine tuning

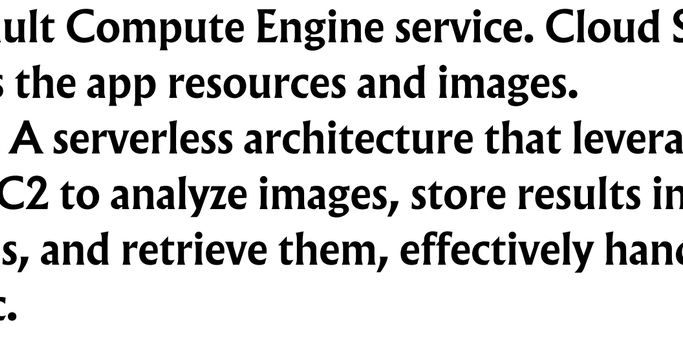

Figure B. AWS _ after fine tuning

Design Approach

Figure 1. Project Design

Figure 2. GCP Architecture

Figure 3. AWS Architecture