

Skin Analysis and Recommendations System

BY

NG YONG SHEN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

© 2025 Ng Yong Shen. All rights reserved.

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr Kh'ng Xin Yi and my moderator, Encik Mohd Hafizul Afifi Bin Abdullah who has given me this bright opportunity to engage in the skin analysis and recommendation project which will involve integrated the deep learning model with the recommendation features into the mobile application. They provide me valuable directions and insight that played a significant role in helping me successfully complete this project. I would also like to express my heartfelt thanks to all my friends and my parents for their unwavering support and encouragement throughout this journey.

ABSTRACT

Perceived opacity and uneven performance of AI skin analysis tools drive users toward unguided, trial-and-error use of skincare products, amplifying ingredient conflicts and delaying improvement. To address this issue, this project develops a skin analysis and recommendation system that delivers a trusted skin analysis model alongside a content-based recommendation method. Firstly, the system provides real-time face detection using the Android library and produces results from the deployed model accompanied by accuracy metrics. This enables use across diverse environments and ensures reliable results through guidance on capture quality and low-confidence flags. To achieve robust skin analysis process, Several AI models, including Classic CNN, EfficientNetB0, ResNet50, and GPT assistant-based models, were trained and tested on datasets for acne severity level and skin type classification. Among these, the YOLOv8 model demonstrated superior performance, achieving testing accuracies of 76.2% for acne severity and 64.0% for skin type, outperforming all other models. The integration of GPT-4o introduces a GPT assistant-based approach that complements traditional deep learning by enhancing interpretability and flexibility in prediction tasks through concise rationales and uncertainty cues. Secondly, the system generates skincare products recommendations based on either the current analysis results or user-specified conditions through content-based filtering. The content-based filtering process applied predefined features rules, whereby acne severity level and skin type govern the selection of recommended product attributes. Users can then view detailed information about these products, with the system generating ingredient-based justifications to support informed decision-making. To enhance personalization, the system has been integrating LLM models, allowing users to select either the Gemini API or the OpenAI API, which provide additional insights and recommendations. Thirdly, selected products can finally be added to a personalized skincare routine, where the system performs AI analysis to evaluate overall compatibility of the routine with the user's skin condition. Fourthly, the system further aggregates skin analysis results to track progress trends for acne severity and skin type in overall, daily and monthly basis, where the system will perform descriptive analysis on progress trend to provide user with useful insight on their skin condition. This project successfully developed multiple module which are skin analysis, product recommendation, skincare routine management, and skin progress tracking to provide a seamless and reliable skin analysis and skincare recommendation experience while ensuring a user-centric and privacy-conscious mobile design.

Area of Study (Maximum 2): Artificial Intelligence, Mobile Application Development

Keywords (Maximum 5): Convolutional Neural Network (CNN), GPT Assistant-Based Integration, Skin Condition Classification, Content-Based Filtering, Mobile Application

TABLE OF CONTENTS

TITLE PAGE	I
COPYRIGHT STATEMENT	II
ACKNOWLEDGEMENTS	III
ABSTRACT	IV
TABLE OF CONTENTS	VI
LIST OF FIGURES	X
LIST OF TABLES	XX
LIST OF ABBREVIATIONS	XXI
CHAPTER 1 INTRODUCTION	1
1.1 Background Information	1
1.2.1 Problem Statement	1
1.2.2 Motivations	3
1.3 Research Objectives	4
1.3.1 To develop a skin analysis model for evaluating and determining users' skin conditions	4
1.3.2 To recommend skincare products suitable for users with various skin concerns	4
1.3.3 To develop a mobile application for skin analysis and recommendation system	4
1.4 Project Scope and Direction	5
1.5 Contributions	6
1.6 Report Organization	6
CHAPTER 2 LITERATURE REVIEW	8
2.1 Existing Mobile Apps for Skin Analysis and Recommendations	8
2.1.1 Artistry Virtual Beauty	8
2.1.2 SpotScan+	10
2.1.3 YouCam Makeup	12
2.1.4 Summary	13
2.2 Existing Works on Skin Analysis	14
2.3 Existing Works on Skincare Product Recommendation	18
CHAPTER 3 SYSTEM METHODOLOGY/ APPROACH	21
3.1 System Architecture Diagram	21
3.2 Use case diagram and Description	25
3.2.1 Register account	26

3.2.2 Login account	28
3.2.3 Sign Out Account	30
3.2.4 Perform skin analysis	31
3.2.5 Scan Face	32
3.2.6 Generate Skin Analysis Result	33
3.2.7 View Skin Progress	34
3.2.8 View Skin Analysis History	36
3.2.9 Input skin data	37
3.2.10 Get Skincare Recommendation	38
3.2.11 View product details	39
3.2.12 Manage Skincare Routine	40
3.2.13 Add Skincare to Routine	41
3.2.14 Create skincare routine	42
3.2.15 Edit skincare routine	43
3.2.16 Delete skincare routine	44
3.2.17 Manage Profile	45
3.2.18 Edit profile	46
3.3 Activity Diagram	47
3.3.1 User access	47
3.3.2 Perform skin analysis	48
3.3.3 Get Skincare Recommendation	49
3.3.4 View Skincare Product	50
3.3.5 Add skincare to routine	51
3.3.6 Modify Skincare Routine	52
3.3.7 Delete Skincare Routine	52
3.3.8 View Skincare Routine	53
3.3.9 Skin Progress Tracking	54
3.3.10 Manage Profile	55
3.4 Timeline diagram	56
CHAPTER 4 SYSTEM DESIGN	57
4.1 System Workflow Diagram	57
4.2 Model Classification Pipeline	60
4.2.1 Data Preparation	62

4.2.2 Data Preprocessing	62
4.2.3 Model Training	63
4.2.4 Model Evaluation	64
4.3 AI Models Workflow	66
4.3.1 Data Preparation for AI Models	66
4.3.2 Data Preprocessing for AI Models	74
4.3.3 Model Training for AI Models	77
4.3.4 Model Evaluation for AI Models	83
4.4 Recommendation Module	91
4.5 System Design Diagram	101
4.5.1 Flowchart	103
CHAPTER 5 SYSTEM IMPLEMENTATION	109
5.1 Hardware Setup	109
5.2 Software Setup	110
5.3 Setting and Configuration	113
5.4 Code Explanation	118
5.4.1 Authentication Module	118
5.4.2 Skin Analysis Module	120
5.4.3 Skincare Recommendation Module	128
5.4.4 Skincare Routine Module	132
5.4.5 Skin Progress Module	135
5.4.6 Manage Profile	137
5.5 System Operation (with Screenshot)	138
5.5.1 Authentication Module	138
5.5.2 Skin Analysis Module	142
5.5.3 Skincare Recommendation Module	145
5.5.4 Skincare Routine Module	150
5.5.5 Skin Progress Module	155
5.5.6 Manage Profile	158
5.6 Implementation Issues and Challenges	160
5.7 Concluding Remark	161
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION	162
6.1 Result Discussion for AI models	162

6.1.1 Classic CNN models	162
6.1.2 ResNet50 models	167
6.1.3 EfficientNetB0 model	171
6.1.4 YOLOv8 model	176
6.1.5 GPT Assistant-Based	181
6.1.6 Model Comparison	186
6.2 Testing Setup and Result	191
6.2.1 Authentication Module	191
6.2.2 Skin Analysis Module	199
6.2.3 Skincare Recommendation Module	201
6.2.4 Skincare Routine Module	206
6.2.5 Skin Progress Module	209
6.3 Project Challenges	212
6.4 Objectives Evaluation	213
6.5 Concluding Remark	214
CHAPTER 7 CONCLUSION AND RECOMMENDATIONS	214
7.1 Conclusion	215
7.2 Recommendation	216
REFERENCES	218
APPENDIXES	1
POSTERS	1

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1.1.1	skin analysis result	9
Figure 2.1.1.2	skin diary	9
Figure 2.1.3.1	Youcam makeup skin report	12
Figure 2.3.1	Product recommendation flowchart	18
Figure 2.3.2	Information submitted by users	18
Figure 2.3.3	Flowchart for Convolution Neural Network-based classifier model	19
Figure 3.1.1	Overall system architecture diagram	21
Figure 3.1.2	Skin Analysis and Recommendations system architecture diagram	23
Figure 3.2.1	Use case diagram	25
Figure 3.3.1.1	User access activity diagram	47
Figure 3.3.2.1	Perform skin analysis activity diagram	48
Figure 3.3.3.1	Get skincare recommendation activity diagram	49
Figure 3.3.4.1	View skincare product activity diagrama	50
Figure 3.3.5.1	Add skincare to routine activity diagram	51
Figure 3.3.6.1	Modify skincare routine activity diagram	52
Figure 3.3.7.1	delete skincare routine activity diagram	52
Figure 3.3.8.1	view skincare routine activity diagram	53
Figure 3.3.9.1	skin progress tracking activity diagram	54
Figure 3.3.10.1	manage profile activity diagram	55
Figure 3.4.1	Timelines for FYP1	56
Figure 3.4.2	Timelines for FYP2	56
Figure 4.1.1	System Workflow Diagram	57
Figure 4.2.1	Model Classification Pipeline	60
Figure 4.3.1.1	Directory of acne grading dataset	66
Figure 4.3.1.2	Code to calculate images in acne grading dataset	67
Figure 4.3.1.3	Total number of images in acne grading dataset	67
Figure 4.3.1.4	Directory of oily, dry and normal skin type dataset	68

Figure 4.3.1.5	Code to calculate images in oily, dry and normal skin types dataset	68
Figure 4.3.1.6	Total number of images in oily, dry and normal skin types dataset	68
Figure 4.3.1.7	Library used in data preparation	69
Figure 4.3.1.8	Dataset path for data preparation	69
Figure 4.3.1.9	Code to split the skin acne dataset	69
Figure 4.3.1.10	Code to split skin type dataset	70
Figure 4.3.1.11	Library used by GPT assistant-based in data preparation	71
Figure 4.3.1.12	Key to access Google Cloud service	71
Figure 4.3. 1.13	Variables used in skin acne data preparation	72
Figure 4.3. 1.14	Function to randomly select images	72
Figure 4.3. 1.15	Function to select all images	72
Figure 4.3. 1.16	Function to generate JSON list	73
Figure 4.3. 1.17	JSON list generated for training set	73
Figure 4.3. 1.18	JSON list generated for test set	73
Figure 4.3.2.1	Library import for data preprocessing	74
Figure 4.3.2.2	Variables declared	74
Figure 4.3.2.3	Code to perform image resizing and data augmentation	74
Figure 4.3.2.4	Load and preprocess image data from directory for model training	75
Figure 4.3.2.5	Library used in ResNet50 data preprocessing	76
Figure 4.3.2.6	Transformation pipelines in ResNet50 model	76
Figure 4.3.2.7	Dataset loading for ResNet50 model	76
Figure 4.3.3.1	Library used by classic CNN model in model training	77
	Code to build classic CNN model	78
Figure 4.3.3.2	Code to train classic CNN model	78
Figure 4.3.3.3	Load ResNet50 model	78
Figure 4.3.3.4	Array to store accuracy and loss	79
Figure 4.3.3.5	Training process for ResNet50 model	79
Figure 4.3.3.6	Validation process for ResNet50 model	79
Figure 4.3.3.7	Code to build EfficientNetB0 model	80
Figure 4.3.3.8	Code to start EfficientNetB0 learning process	80

Figure 4.3.3.9	Import Ultralytics library	81
Figure 4.3.3.10	Load pretrained model	81
Figure 4.3.3.11	Code to start model training process	81
Figure 4.3.3.12	API Key Declaration and Client Object Creation	81
Figure 4.3.3.13	Assistant used by skin acne dataset	82
Figure 4.3.3.14	Assistant used by skin type dataset	82
Figure 4.3.3.15	Store image information for skin acne dataset	82
Figure 4.3.3.16	Store image information for skin type dataset	82
Figure 4.3.3.17	Initializing thread and storing training messages	83
Figure 4.3.4.1	Libraries used in evaluation stage	83
Figure 4.3.4.2	Extraction of actual and predicted labels from test data	83
Figure 4.3.4.3	Confusion matrix for Classic CNN skin acne classification model across different acne severity levels	84
Figure 4.3.4.4	Confusion matrix for Classic CNN skin acne classification model across different acne severity levels	84
Figure 4.3.4.5	Classification report for Classic CNN skin acne classification model across different acne severity levels	84
Figure 4.3.4.6	library used in ResNet50 evaluation stage	84
Figure 4.3.4.7	Code to evaluate test set using ResNet50 model	84
Figure 4.3.4.8	Class name for skin acne dataset	85
Figure 4.3.4.9	Class name for skin type dataset	85
Figure 4.3.4.10	Confusion matrix for ResNet50 skin acne classification model across different acne severity levels	85
Figure 4.3.4.11	Confusion matrix for ResNet50 skin type classification model	85
Figure 4.3.4.12	Classification report for ResNet50 model	85
Figure 4.3.4.13	Code to evaluate the test data using EfficientNetB0 model	86
Figure 4.3.4.14	library used in EfficientNetB0 model evaluation stage	86
Figure 4.3.4.15	Confusion matrix for EfficientNetB0 skin acne classification model across different acne severity levels	86
Figure 4.3.4.16	Confusion matrix for EfficientNetB0 skin type classification model	86
Figure 4.3.4.17	Classification report for EfficientNetB0 model	86

Figure 4.3.4.18	Evaluate the test dataset using YOLOv8 model	87
Figure 4.3.4.19	Class name for skin acne dataset	87
Figure 4.3.4.20	library used in YOLOv8 model evaluation stage	87
Figure 4.3.4.21	Class name for skin type dataset	87
Figure 4.3.4.22	Confusion matrix for YOLOv8 skin acne classification model across different acne severity levels	88
Figure 4.3.4.23	Confusion matrix for YOLOv8 skin type classification model	88
Figure 4.3.4.24	Code to display precision, recall, and F1-score for each class	88
Figure 4.3.4.25	Sending Image and Prompt Message to GPT Assistant for Classification	89
Figure 4.3.4.26	Initiating GPT Assistant Run and Monitoring Processing Status	89
Figure 4.3.4.27	Retrieving and Storing Predicted Acne Levels from GPT Assistant Response	90
Figure 4.3.4.28	Function call to process test images using GPT assistant	90
Figure 4.4.1	library import recommendation module	91
Figure 4.4.2	code to import the csv file	91
Figure 4.4.3	content of csv file	91
Figure 4.4.4	code to remove extra spaces	92
Figure 4.4.5	code to replace the empty string with NaN	92
Figure 4.4.6	code to drop the NaN cells	92
Figure 4.4.7	code to check NaN cells	92
Figure 4.4.8	code to check unique type of skincare products	93
Figure 4.4.9	result of unique type of skincare	93
Figure 4.4.10	code that shows not_skincare array	93
Figure 4.4.11	code to remove the skincare under not_skincare array	93
Figure 4.4.12	skincare product type used	94
Figure 4.4.13	code to clean and standardize the data	94
Figure 4.4.14	code to perform MultiLabelBinarizer	95
Figure 4.4.15	code to remove acne trigger product	95
Figure 4.4.16	code to remove unused column	95
Figure 4.4.17	dictionary with feature rules for acne severity level	97

Figure 4.4.18	dictionary with feature rules for skin type	97
Figure 4.4.19	code for the filter_and_group function	98
Figure 4.4.20	code for Iterating Over Acne Levels and Skin Types to Display Filtered Product Groups	99
Figure 4.5.1	System design diagram of skin analysis and recommendations system	101
Figure 4.5.1.1	Flowchart for registration, login and main menu	103
Figure 4.5.1.2	Flowchart for skin analysis process	104
Figure 4.5.1.3	Flowchart for product recommendation process	105
Figure 4.5.1.4	Flowchart for skincare routine process	106
Figure 4.5.1.5	Flowchart for skin progress process	107
Figure 4.5.1.6	Flowchart for managing profile process	108
Figure 4.5.1.7	Flowchart for settings process	108
Figure 5.2.1	Python Programming Language	110
Figure 5.2.2	Jupyter Notebook	110
Figure 5.2.3	Java Programming Language	110
Figure 5.2.4	Firebase	110
Figure 5.2.5	Android Studio	111
Figure 5.2.6	GitHub	111
Figure 5.2.7	TensorFlow	111
Figure 5.2.8	Keras	111
Figure 5.2.9	Google Cloud	112
Figure 5.3.1	App-level build.gradle configuration for securely injecting Gemini and OpenAI API keys	113
Figure 5.3.2	App-level build.gradle dependencies configuration (upper half)	114
Figure 5.3.3	App-level build.gradle dependencies configuration (lower half)	115
Figure 5.3.4	Declared permissions in AndroidManifest.xml	115
Figure 5.3.5	Firebase project creation for the Skin Glow application	116
Figure 5.3.6	Configuration of SHA key and integration of google-services.json file in Firebase setup	116

Figure 5.3.7	API key generation in Google AI Studio for Gemini integration	117
Figure 5.3.8	API key generation in OpenAI platform for OpenAI integration	117
Figure 5.4.1.1	User Registration Function in Firebase	118
Figure 5.4.1.2	Email Verification Process in Firebase	119
Figure 5.4.1.3	sign in process in Firebase	119
Figure 5.4.2.1	Code Snippet of the toGrayscale Function for Face Scanning	120
Figure 5.4.2.2	Code Snippet of the Face Detection Process Using Android FaceDetector	121
Figure 5.4.2.3	Folder Path Showing the Trained YOLOv8 Model (best.pt)	122
Figure 5.4.2.4	Installing Required Dependencies in Google Colab for YOLOv8 to TensorFlow Lite Conversion	122
Figure 5.4.2.5	Uploading the best.pt File to Google Colab	122
Figure 5.4.2.6	Exporting the YOLOv8 Model to TensorFlow Lite Format	123
Figure 5.4.2.7	Successful Export of the YOLOv8 Model to TensorFlow Lite	123
Figure 5.4.2.8	Copying the Model Path to Download the .tflite File	123
Figure 5.4.2.9	YOLOv8 Models Stored in the Assets Folder of the Mobile Application	123
Figure 5.4.2.10	Code Snippet for Preprocessing Bitmap Before YOLOv8 Model Inference	124
Figure 5.4.2.11	Code Snippet for Detecting Skin Type Using YOLOv8 Models	125
Figure 5.4.2.12	Code Snippet for Uploading Images to Firebase Storage	126
Figure 5.4.2.13	Code Snippet for Storing Skin Analysis Results in Firestore	127
Figure 5.4.3.1	Code Snippet for Combining Acne Level and Skin Type Features	128
Figure 5.4.3.2	Mapping Acne Feature Rules for Content-Based Filtering	129
Figure 5.4.3.3	code snippet for retrieving filtered skincare products	130
Figure 5.4.3.4	code snippet for dynamic skincare product retrieval	131
Figure 5.4.4.1	code snippet for storing a new skincare routine in Firestore	132
Figure 5.4.4.2	code snippet of Try-catch block in checkForProductStepConflictWithVerification method	133

Figure 5.4.4.3	snippet code for validating product step conflicts in a skincare routine	134
Figure 5.4.5.1	snippet code for visualizing skin type progress over time	135
Figure 5.4.5.1	code snippet to update the accuracy chart with processed accuracy data	136
Figure 5.4.6.1	code snippet to update user profile	137
Figure 5.5.1.1	login page	138
Figure 5.5.1.2	register page	138
Figure 5.5.1.3	register details	139
Figure 5.5.1.4	login page after registered	139
Figure 5.5.1.5	email verification	140
Figure 5.5.1.6	email verification success	140
Figure 5.5.1.7	login details	141
Figure 5.5.1.8	main menu after successfully login	141
Figure 5.5.2.1	Face Detection and Capture Option	142
Figure 5.5.2.2	Image Processing After Capture	142
Figure 5.5.2.3	Skin Analysis Result Display	143
Figure 5.5.2.4	Acne Progress Chart	143
Figure 5.5.2.5	Updated User Status in Main Menu	144
Figure 5.5.3.1	Skincare Characteristics Page with No Previous Analysis	145
Figure 5.5.3.2	Skincare Characteristics Page with Current Analysis Result	145
Figure 5.5.3.3	Product Category Selection Page	146
Figure 5.5.3.4	Filtered Face Cleanser Products	146
Figure 5.5.3.5	Filtered Products by Name (“facial”)	147
Figure 5.5.3.6	Filtered Products by Brand (“face gym”)	147
Figure 5.5.3.7	Filter Options for Brand and Country	148
Figure 5.5.3.8	Filtered Products with No Results (Dermalogica, Australia)	148
Figure 5.5.3.9	Product Details (upper half)	149
Figure 5.5.3.10	product details(lower half)	149
Figure 5.5.3.11	Redirected Google Search Results	149
Figure 5.5.4.1	Skincare Routine Page with No Routine Created	150
Figure 5.5.4.2	Create Routine Page	150
Figure 5.5.4.3	Skincare Routine Page After Routine Creation	151

Figure 5.5.4.4	Viewing a Selected Routine	151
Figure 5.5.4.5	Editing Routine Name and Description	152
Figure 5.5.4.6	Updated Routine After Saving Changes	152
Figure 5.5.4.7	Adding a Product to a Routine	153
Figure 5.5.4.8	Routine Details Displaying Added Product	153
Figure 5.5.4.9	AI Routine Analysis Result (upper half)	154
Figure 5.5.4.10	AI Routine Analysis Result(lower half)	154
Figure 5.5.5.1	Acne progress page	155
Figure 5.5.5.2	Skin type page	155
Figure 5.5.5.3	Skin Acne Severity Graph (Overall Period)	156
Figure 5.5.5.4	AI Descriptive Analysis with Recommendations	156
Figure 5.5.5.5	Daily Skin Acne Severity Graph	157
Figure 5.5.5.6	Skin Type Graph	157
Figure 5.5.6.1	Edit Profile Page (Before Changes)	158
Figure 5.5.6.2	Edit Profile Page (After Changes)	158
Figure 5.5.6.3	Updated User Profile Displayed in Main Menu	159
Figure 6.1.1.1	Training and validation accuracy and loss across epochs for classic CNN skin acne classification model	162
Figure 6.1.1.2	Confusion matrix for classic CNN skin acne classification model	163
Figure 6.1.1.3	Training and validation accuracy and loss across epochs for classic CNN skin type model	165
Figure 6.1.1.4	Confusion matrix for classic CNN skin type classification	166
Figure 6.1.2.1	Training and validation accuracy and loss across epochs for ResNet50 skin acne classification model across different severity levels	167
Figure 6.1.2.2	Confusion matrix for ResNet50 skin acne classification model across different severity levels	168
Figure 6.1.2.3	Training and validation accuracy and loss across epochs for ResNet50 skin type classification model	169
Figure 6.1.2.4	Confusion matrix for ResNet50 skin type classification model	170

Figure 6.1.3.1	Training and validation accuracy and loss across epochs for EfficientNetB0 skin acne classification model across different severity levels	172
Figure 6.1.3.2	Confusion matrix for EfficientNetB0 skin acne classification across different severity levels	173
Figure 6.1.3.3	Training and validation accuracy and loss across epochs for EfficientNetB0 skin type classification model	174
Figure 6.1.3.4	Confusion matrix for EfficientNetB0 skin type classification model	175
Figure 6.1.4.1	Accuracy of YOLOv8 skin acne classification on training and validation sets across different severity levels	176
Figure 6.1.4.2	Accuracy of YOLOv8 skin acne classification model on testing set across different severity levels	177
Figure 6.1.4.3	Confusion matrix for YOLOv8 skin acne classification model across different severity levels	178
Figure 6.1.4.4	Accuracy of YOLOv8 skin type classification model on training and validation sets	179
Figure 6.1.4.5	Accuracy of YOLOv8 skin type classification model on test set	179
Figure 6.1.4.6	Confusion matrix for YOLOv8 skin type classification	180
Figure 6.1.5.1	Sample result for GPT assistant-based skin acne classification across different severity levels	181
Figure 6.1.5.2	Confusion matrix for GPT assistant-based skin acne classification across different severity levels	182
Figure 6.1.5.3	Sample result for GPT assistant-based skin type classification	183
Figure 6.1.5.4	Confusion matrix for GPT assistant-based skin type classification	184
Figure 6.2.1.1	Registration process test case 1	191
Figure 6.2.1.2	Registration process test case 2	192
Figure 6.2.1.3	Registration process test case 3	193
Figure 6.2.1.4	Registration process test case 4	194
Figure 6.2.1.5	Login process test case 1	195

Figure 6.2.1.6	Login process test case 2	196
Figure 6.2.1.7	Reset password process test case 1	197
Figure 6.2.1.8	Reset password process test case 2	198
Figure 6.2.2.1	Skin analysis process test case 1	199
Figure 6.2.2.2	Skin analysis process test case 2	200
Figure 6.2.3.1	Skincare recommendation process test case 1	202
Figure 6.2.3.2	Skincare recommendation process test case 2	203
Figure 6.2.3.3	Skincare recommendation process test case 3	205
Figure 6.2.4.1	Skincare routine test case 1	207
Figure 6.2.4.2	Skincare routine test case 2	208
Figure 6.2.5.1	Skin progress test case 1	209
Figure 6.2.5.2	Skin progress test case 2	211

LIST OF TABLES

Table Number	Title	Page
Table 2.1.4.1	Summary of existing mobile applications	13
Table 2.2.1	Comparison of testing and training results in skin condition	14
Table 2.2.2	Comparison of testing and training result in facial pore	15
Table 2.2.3	Skin type classification report	15
Table 2.2.4	Validation accuracy for each model	16
Table 2.2.5	Result for YOLOv5, YOLOv8 and Detectron	17
Table 2.2.6	Comparison on the mean average precision of models	17
Table 2.3.1	Dataset before and after preprocessing process	19
Table 2.3.2	Percentage of skincare products relevant to users	20
Table 4.2.3.1	hyperparameter used in model training	64
Table 4.2.4.1	Sample of confusion matrix	65
Table 4.4.1	product features with its binary value representation	96
Table 4.4.2	Skincare Product Distribution by Acne Level and Skin Type	100
Table 5.1	Specifications of laptop	109
Table 5.2	Specifications of mobile devices	109
Table 6.1.1.1	classification report for Classic CNN skin acne model	164
Table 6.1.1.2	classification report for Classic CNN skin type model	166
Table 6.1.2.1	classification report for ResNet50 skin acne classification	169
Table 6.1.2.2	classification report for ResNet50 skin type classification	171
Table 6.1.3.1	classification report for EfficientNetB0 skin acne model	174
Table 6.1.3.2	classification report for EfficientNetB0 skin type classification	176
Table 6.1.4.1	Precision, recall and F1-score for YOLOv8 skin acne classification	178
Table 6.1.4.2	Precision, recall and F1-score for YOLOv8 skin type classification	181
Table 6.1.5.1	classification report for GPT assistant-based skin acne	183
Table 6.1.5.2	classification report for GPT assistant-based skin type	185
Table 6.1.6.1	Training and validation accuracy comparison between models for skin acne	186

Table 6.1.6.2	Testing accuracy comparison between models for skin acne	187
Table 6.1.6.3	Testing accuracy of YOLOv8 and GPT Assistant-Based for skin acne	187
Table 6.1.6.4	YOLOv8 model confidence level for skin acne	188
Table 6.1.6.5	Training and validation accuracy comparison between models for skin type	189
Table 6.1.6.6	Testing accuracy comparison between models for skin type	189
Table 6.1.6.7	Testing accuracy of YOLOv8 and GPT Assistant-Based for skin type	190
Table 6.1.6.8	YOLOv8 model confidence level for skin type	190
Table 6.2.1.1	Registration process test case 1	191
Table 6.2.1.2	Registration process test case 2	192
Table 6.2.1.3	Registration process test case 3	193
Table 6.2.1.4	Registration process test case 4	194
Table 6.2.1.5	Login process test case 1	195
Table 6.2.1.6	Login process test case 2	196
Table 6.2.1.7	Reset password process test case 1	197
Table 6.2.1.8	Reset password process test case 2	198
Table 6.2.2.1	Skin analysis process test case 1	199
Table 6.2.2.2	Skin analysis process test case 2	200
Table 6.2.3.1	Skincare recommendation process test case 1	201
Table 6.2.3.2	Skincare recommendation process test case 2	203
Table 6.2.3.3	Skincare recommendation process test case 3	204
Table 6.2.4.1	Skincare routine test case 1	206
Table 6.2.4.2	Skincare routine test case 2	208
Table 6.2.5.1	skin progress test case 1	209
Table 6.2.5.2	Skin Progress test case 2	210

LIST OF ABBREVIATIONS

LLM	Large Language Model
CNN	Convolutional Neural Network
RESNET50	Residual Network
MTCNN	Multi-Task Cascaded Convolutional Neural Network
R-CNN	Regional-Based Convolutional Neural Network
AI	Artificial Intelligence
GPT	Generative Pre-Trained Transformers
API	Application Programming Interface

Chapter 1

Introduction

In this chapter, we provide a brief introduction, project statement, motivation, research objectives, project scope and direction, contributions, and the organization of the report.

1.1 Background Information

In recent years, technological advancements have transformed numerous sectors of industries, including healthcare and personal care. AI has become increasingly popular worldwide, significantly improving people's quality of life. AI can perform tasks that normally require human intelligence. AI algorithms have been applied in dermatology to predict and make decisions that assist dermatologists in diagnosing skin conditions using deep learning models. CNN model is among the most popular models used in skin condition analysis due to their effectiveness in handling images and their high accuracy [1].

As people place more emphasis on their physical appearance, skin issues have become a major concern. Skin problems can significantly impact a person's confidence and social interactions. However, many people struggle to trust the results provided by AI models due to a lack of understanding, leading to doubts about the accuracy of these models in identifying individual skin conditions. Consequently, individuals often rely on trial and error to find suitable skincare products [2]. This trial-and-error approach can potentially worsen their skin condition. Therefore, an AI-driven skin analysis solution is essential for accurately analysing skin conditions and providing personalized skincare recommendations tailored to individual needs, ensuring effective improvement of skin health.

1.2 Problem Statement and Motivation

1.2.1 Problem Statement

With the advent of technology and social media, physical appearance has become more important than ever, as people often make judgments based on facial appearance during first impressions. This has made facial care a higher priority in many individuals' lives. As a result, people are increasingly attentive to their appearance, especially their face to boost self-confidence and foster better social relationships. According to a survey in

CHAPTER 1

[3], 75.8% of a sample of 582 students aged 16 to 24 reported being affected by acne, which significantly impacted their quality of life and confidence. However, several major challenges still exist, often causing frustration and helplessness for individuals striving to achieve clear and healthy skin.

Although the use of technology in analyzing skin conditions has become increasingly popular, users often do not trust the accuracy of these systems. This is because the results can be precise only when individuals capture their facial images under conditions that meet the system's specific requirements. However, when these conditions are not met, the results may lack accuracy. As noted in [4], environmental factors like lighting during image capture can greatly impact the model's performance, making it harder for the system to produce consistent and reliable results. Therefore, individuals are required to meet specific system conditions to ensure robust and accurate analysis experience.

Moreover, most people tend to purchase skincare products physically without any professional guidance. This can potentially lead to errors, as the products chosen may not be suitable for their skin type. Using the wrong products may worsen their skin condition over time [5]. Without proper guidance and recommendations that accurately assess their skin condition and identify suitable products, individuals may experience irritation or breakouts due to the ineffectiveness or harshness of the products. Therefore, proper skincare must be supported by personalized recommendations and expert guidance, which are essential for helping individuals improve their skin health and achieve the desired results.

In addition, many people choose to visit skin specialists who are experienced in the field of dermatology at clinics or hospitals. However, these professionals are often overwhelmed with consultation requests, resulting in long waiting times for appointments. As a result, individuals may seek alternative methods to address their skin problems while waiting. During this period, some may turn to home remedies, such as harsh exfoliation techniques or DIY treatments found online. Unfortunately, these unverified methods can worsen their skin condition, leading to increased irritation or even scarring.

1.2.2 Motivations

Several critical issues have been identified, which motivate the development of a system designed to benefit society. These issues include the limited accuracy of current skin analysis models, the lack of personalized product recommendations, and the long waiting times required to consult with professional dermatologists.

Firstly, the accuracy of the skin analysis model is crucial in providing users with precise results, as it directly affects the effectiveness of the entire skin analysis process. Inaccurate analysis may lead to incorrect product recommendations, reducing the system's reliability. Therefore, this project will evaluate three different skin analysis models to determine the most suitable one that can be implemented to achieve the highest accuracy in experiments. A highly accurate model with low error rates will not only boost users' confidence in using the system but also ensure that the results are reliable and effective for individuals.

Besides, the product recommendations provided by many existing applications or platforms are often limited, as they typically promote only their own brand's products. This restricts users to a narrow selection and serves the business interests of the company rather than the users' needs. In contrast, the proposed system will offer a wide range of skincare products that are either certified by dermatologists or come from reputable brands such as Kiehl's and Fresh. This ensures that users can explore and choose products from different brands that best match their skin conditions and types. It also allows flexibility based on users' budgets, as skincare products can range from affordable to premium prices.

In addition, the development of a skin analysis mobile application can potentially address the issue of long waiting times for dermatology consultations. Users can analyze their skin conditions directly through the app, allowing them to take proactive steps to improve their skin without needing immediate professional appointments. The mobile application will also allow users to store their skin images in a database, enabling them to track and compare their skin condition over time. This feature helps users visually assess their progress, supported by a scoring system that quantifies improvements. Moreover, user feedback can be collected through the app to

continuously enhance the model's accuracy and refine product recommendations, ensuring a better and more personalized experience. Ultimately, this application can benefit not just individual users, but anyone struggling with skin conditions such as acne.

1.3 Research Objectives

There are three main objectives that this project aims to achieve:

1.3.1 To develop a skin analysis model for evaluating and determining users' skin conditions

This objective will focus on developing the core functionality of the system, specifically the skin analysis model. The model will leverage image processing and deep learning techniques to assess users' skin conditions and assign a score to reflect their skin health. The study will compare the accuracy of several deep learning models, including classic CNN, EfficientNetB0, ResNet50, YOLOv8, and image classification integrated with the OpenAI API. By identifying the most accurate model, this approach will ensure that users' skin conditions are accurately assessed by the system.

1.3.2 To recommend skincare products suitable for users with various skin concerns

This objective will focus on recommending skincare products that are suitable for users after they scan their skin. Personalized skincare products will be suggested based on the results of the skin analysis, ensuring that the products are tailored to the users' specific skin conditions. By recommending appropriate products, the system will help improve the users' skin effectively while minimizing the risk of side effects.

1.3.3 To develop a mobile application for skin analysis and recommendation system

This objective will focus on developing a user-friendly mobile application that provides users with seamless experience and an intuitive interface. The deep learning model and skincare product recommendation module will be integrated into the mobile application. Users will be able to track their skin improvement over time as their analysis results will be stored as past results within the application. By storing these results, users can easily compare their current skin condition with previous ones,

allowing them to monitor progress and make more informed decisions about their skincare routine.

1.4 Project Scope and Direction

In this project, users will be able to use the mobile application to perform skin analysis actions. Before accessing the mobile application, users are required to log in if they have an existing account. If not, they will need to register an account to access the system. Additionally, users can choose to log in using their Google or Facebook accounts.

After users access the system, they can perform a skin analysis, which requires them to open their camera and position their face for scanning. Once users scan their face, the image will be sent to the backend server, which stores the user's skin information. This is because the skin analysis model is hosted on the backend server rather than on the mobile device, to save storage space. After analyzing the user's skin condition, the result will be displayed, showing details such as fine lines, pores, and redness of the skin.

Additionally, users can view skincare products recommended based on their skin conditions. These products come from different brands and are selected to best suit the user's skin at that moment. The system will also display a graph showing the percentage of improvement after users have used the recommended products. Users can compare the current results with previous ones to observe the differences. They will also be allowed to provide feedback for each session, ensuring the system can be improved further in the future.

Moreover, the results will be stored on the user's mobile device, and a backup copy will be sent to the server. The data storing the user's skin analysis results will be deleted after six months.

1.5 Contributions

This project aims to provide significant benefits to people in Malaysia, particularly individuals dealing with acne-related skin problems. Most existing mobile applications do not specifically address skin problems faced by Malaysians. Therefore, this project is designed to accurately analyze the skin of Malaysians, ensuring that their skin issues are clearly identified. With a focus on high accuracy in the skin analysis model, the project will ensure that the products recommended to individuals are suited to their specific skin conditions, thereby reducing the likelihood of selecting unsuitable products.

Additionally, the product recommendations will offer a wide range of skincare products from different brands, including dermatologist-tested and popular brands in the market, such as Fresh and SkinCeuticals. This will provide users with the flexibility to choose from various brands, rather than limiting their options to a single brand. Dermatologist-tested and popular products are less likely to worsen skin problems, as they have been tested by other patients or users before. As a result, users can be more confident that the recommended products will effectively improve their skin.

In addition, developing a mobile application for the skin analysis model and product recommendations will further enhance the system's flexibility and convenience for users. The mobile app will allow users to analyze their skin conditions and receive product recommendations more quickly compared to a web-based skin analysis system. It will also enable users to monitor their skin condition more frequently, as the mobile app can send daily notifications reminding them to update their skin images. This feature will encourage users to be more proactive in improving their skin health.

1.6 Report Organization

The report is structured into seven chapters which are Introduction, Literature Review, System Methodology or Approach, System Design, System Implementation, System Evaluation and Discussion, and Conclusion and Recommendations.

Chapter One introduces the project by providing background information, the problem statement, motivation, research objectives, project scope, and contributions. Chapter

CHAPTER 1

Two reviews existing mobile applications and prior research related to skin analysis and skincare product recommendations. Chapter Three discusses the system methodology, which consists of the system architecture diagram, use case diagram and descriptions, activity diagram, and timeline diagram.

Chapter Four explains the system design, including the system workflow diagram, model classification pipeline which covering data preparation, preprocessing, training, and evaluation, AI model workflow, recommendation module, and system design flowchart. Chapter Five presents the system implementation, covering hardware and software setup, settings and configurations, code explanations, system operations, and implementation issues and challenges. Chapter Six focuses on system evaluation and discussion, where the results of the AI models, testing setup, outcomes, project challenges, and evaluation of objectives are discussed. Finally, Chapter Seven provides the conclusion and recommendations, summarizing the overall work completed and offering suggestions for future improvements.

Chapter 2

Literature Review

In this chapter, we will review the features of existing mobile applications that provide skin analysis and product recommendation features. We will also examine the skin analysis models used to evaluate the skin conditions of users, as well as the product recommendation models that offer personalized skincare routines based on individual skin conditions.

2.1 Existing Mobile Apps for Skin Analysis and Recommendations

In this section, we will review various mobile applications that incorporate skin analysis features and product recommendations. This review will help us gain insights into how these applications analyse skin conditions and recommend products, providing a better understanding of current trends in mobile applications for skincare.

2.1.1 Artistry Virtual Beauty

Artistry Virtual Beauty is a mobile application developed by Amway, a multinational company specializing in health, beauty, home care, and skincare products. The application allows customers to personalize their skincare routines based on their skin condition. Key features of the app include a built-in skin analysis tool, a skin diary for tracking changes over time, and a product recommendation system tailored to users' individual skin needs.

Strength

The application features a built-in skin analysis tool that allows users to scan their face using the mobile device's camera. After scanning, the system displays the user's skin condition, highlighting aspects such as spots, wrinkles, texture, and dark circles, with marks assigned to each feature. The system then combines these factors to determine an overall skin health score and also estimates the user's skin age. Figure 2.1.1.1 shows the results of the skin analysis within the application.

The system also includes a skin diary feature that tracks the user's skin health after following the skincare routine. It displays both the skin age determined by the analysis and the user's real age. Every skin analysis score is recorded in the skin diary, allowing

CHAPTER 2

users to compare their skin condition over time. Users can view summaries of their skin condition over periods such as 7 days, 30 days, 90 days, or 150 days. The skin diary also presents an overall summary in graph form, helping users monitor their skin's progress. If the user's skin condition worsens, it may indicate issues with the skincare products being used, suggesting that the products may not be suitable for the user or are ineffective in improving their skin.



Figure 2.1.1.1 skin analysis result[6]



Figure 2.1.1.2 skin diary[6]

Weaknesses

Artistry Virtual Beauty includes a product recommendation feature that activates after users scan their skin and view their skin diary on their mobile devices. Based on a detailed analysis of the user's skin condition, products from the Artistry brand will be recommended. These personalized recommendations ensure that users receive skincare products that are best suited to their specific needs.

Recommendation

The product recommendation feature can be enhanced by offering users a wider selection of personalized skincare products from various brands. This approach ensures that users can choose products that are better suited to their specific skin conditions, increasing the chances of improving their skin health quickly. Since each user has unique skin needs, providing a diverse range of brands will increase the likelihood of finding the most suitable product for their skin.

2.1.2 SpotScan+

La Roche-Posay AI Skin Analysis Tool is a platform developed by La Roche-Posay, a well-known dermatologist-recommended skincare brand from France. This platform offers several features aimed at helping users improve their skincare routine.

Strengths

This platform offers skin analysis services developed in collaboration with leading dermatologists, making it one of the most trustworthy tools available. It has been trained using approximately 6,000 images from various skin conditions, ensuring high accuracy in analysing skin. The algorithm used for skin analysis boasts an accuracy rate of 90%, as stated in [7]. Additionally, the platform's accuracy has been validated through a clinical study involving 50 patients suffering from acne, under the supervision of three expert dermatologists [8].

Besides, it provides built-in skin analysis features that can scan the users' faces. Users are required to take photos from the center, left, and right sides of their face. The system then processes the images and identifies skin conditions such as pimples, blackheads, and pigmented marks. It assigns a grade based on the severity of the skin condition—the higher the score, the more severe the imperfections [7].

After the skin analysis process, the system requires users to input their skin type (oily, dry, or combination) and age before recommending products. This is because each individual has different skin conditions and ages. By combining this information with the results of the skin analysis, the system will be able to suggest products that are more

CHAPTER 2

suited to their skin condition, ensuring that users can improve their skin condition effectively.

Weaknesses

The requirement for proper face positioning is quite strict because users must ensure that their face is correctly aligned with the camera so that the image can be successfully captured from three different angles: the center, left, and right sides of the face. The application does not provide guidance or a demonstration on how users should position their face for the camera. This can cause users to struggle with adjusting their face, leading to frustration and a time-consuming process.

In addition, a significant limitation of this platform is that the results of the skin analysis are not stored in the database after the user completes the skin analysis process. This means the analysis results will be permanently deleted once the user leaves the page. This limitation prevents the user from monitoring their skin conditions by keeping track of their skin over time. Therefore, this limitation reduces the effectiveness of the platform for users [8].

Recommendation

The system requires users to adjust their face to a very specific position to take the photo. To address this issue, the system should guide users to ensure they know how to position their face correctly when capturing the photo for the skin analysis process. Additionally, the system should store the analysis results in a database, allowing users to retrieve previous results in the future and compare improvements after using the skincare products recommended by the system.

2.1.3 YouCam Makeup

Youcam makeup is one of the mobile applications which is owned by Prefect Corporation. The application was built-in variety of features such as ai hairstyle, photo makeup, skin diagnostics and others. We will mainly focus on the features which related to skin diagnostics.

Strengths

Users can utilize skin diagnostics features to scan their face using their mobile devices. This feature analyzes users' skin for ten different concerns, including spots, radiance, wrinkles, moisture, pores, texture, dark circles, eye bags, redness, and oiliness. After the user takes a photo of their face, the application will analyze and evaluate the skin scores based on their conditions. It then calculates an average skin score, as shown in Figure 2.1.3.1.

The application also offers a skin diary feature that records users' skin conditions each time they take a photo for analysis. Users can check their average skin scores and track the frequency of their skin analyses for more accurate tracking over time. This feature helps users monitor their skin condition and observe changes over time.

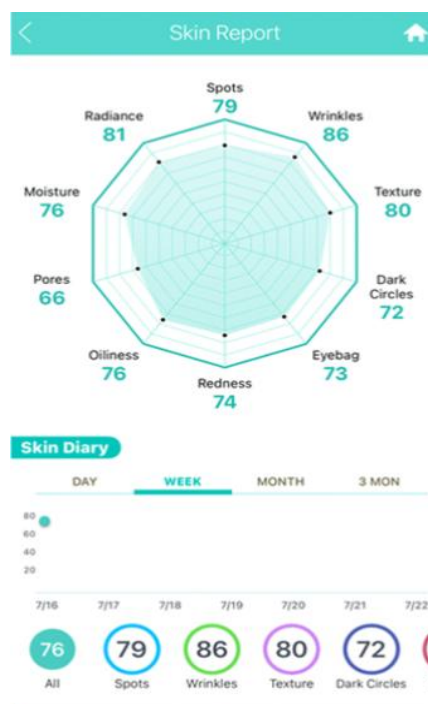


Figure 2.1.3.1 Youcam makeup skin report[9]

2.1.4 Summary

Table 2.1.4.1 Summary of existing mobile applications.

	Artistry Virtual Beauty	La Roche Posay AI Skin Analysis Tool	Youcam Makeup	Proposed system
Classification of skin type	No	No	No	Yes
Analysis of skin condition	Yes	Yes	Yes	Yes. Classify acne severity level
Skin analysis history	Yes	No	Yes	Yes.
Product recommendation range	Limited	Limited	No	Wide range. Products will from different brands
Availability of mobile application	Yes. Applicable for both Android and iOS	No.	Yes. Applicable for both Android and iOS	Yes. Applicable for Android
Save the analysis result	Yes	No.	Yes	Yes

2.2 Existing Works on Skin Analysis

Deep learning and machine learning models can both be used to analyse the skin conditions of individuals. However, the accuracy of these models varies significantly. In this section, we will review these models, identify those with higher accuracy rates, and discuss their limitations.

As mentioned in [10], studies conducted in the skin analysis area used the MTCNN model in the image processing process to filter the background of the image. The CNN model was then used in the skin analysis process, producing an accuracy of 82% using a dataset of 80 skin images, which were categorized into oily and dry skin. However, they found that the model was not efficient due to the dataset size being too small, and the accuracy of the results was affected by the skin conditions, particularly when the skin was slightly oily during image capture. The author in [11] also concluded that the model was not effective in classifying skin types from the images, achieving only a 78.6% accuracy rate. However, it was effective in detecting acne on the skin, achieving a successful 98.9% accuracy in acne detection.

The authors in [12] proposed different CNN models to investigate the highest accuracy CNN model in their paper. They proposed two CNN models to compare with the classic CNN model, LeNet-5, for classifying skin conditions as good, bad, or makeup, using 300 images with 20 extracted 51 x 68 patch images. The first model uses a 7-layer architecture, while the second model uses a 10-layer architecture, and the LeNet-5 model has only 6 layers. They concluded that Model 2 achieved the highest accuracy rate compared to the other two models, but Model 1 had the lowest loss rate, meaning the probability of it classifying images incorrectly was the lowest. The test and training results for Models 1, 2, and 3 are shown in Table 2.2.1.

Table 2.2.1 Comparison of testing and training results in skin condition[12]

Testing	<i>Model 1 (2C2P3F)</i>	<i>Model 2 (3C3P4F)</i>	<i>Model 3 (3C2P1F)</i>
<i>Accuracy rate</i>	0.87	0.93	0.86
<i>Loss rate</i>	0.12	0.59	-

The author [13] from another study also concluded that the model using the same architecture as Model 2 in Figure 2.2.1 achieved the highest performance in identifying pores on the skin, with a 91% accuracy rate, compared to the other two models are the 7-layer CNN model and the LeNet-5 model, which achieved 87% and 84% accuracy rates, respectively, as shown in Table 2.2.2. The study also highlighted that more facial images are required to address more challenging facial skin problems.

Table 2.2.2 Comparison of testing and training result in facial pore [13]

Testing	<i>Model A</i> (2C2P2F)	<i>Model B</i> (2C2P3F)	<i>Model C</i> (3C3P4F)
<i>Accuracy rate</i>	0.84	0.87	0.91

In addition, a study from [14] used the ResNet50 model to classify skin conditions into six categories: acne, carcinoma, eczema, keratosis, milia, and rosacea, using a dataset consisting of 2,394 images. This study highlighted that the ResNet50 model was able to achieve 92% validation accuracy. Another study in [15] also focused on analyzing skin type using a dataset consisting of 3,502 images with the ResNet50 model. The results showed that the model achieved 85% accuracy on the test set, as shown in Figure 2.2.1.

	precision	recall	f1-score	support
0	0.92	0.79	0.85	97
1	0.79	0.91	0.85	116
2	0.89	0.84	0.87	103
accuracy			0.85	316
macro avg	0.87	0.85	0.85	316
weighted avg	0.86	0.85	0.85	316

Figure 2.2.1 Skin type classification report [15]

In addition, the study in [16] applied models such as EfficientNetB0, HRNET, DenseNet, and ResNet50 for classifying skin images. The dataset consisted of 7,155 images used to classify skin conditions into six categories: oily, hyperpigmentation, acne, redness, blackheads, and normal. The author concluded that EfficientNetB0 achieved the highest accuracy among all models, with 99% training accuracy and 92% testing accuracy. This was followed by the ResNet50 model, which achieved the same training accuracy but slightly lower testing accuracy, at 91%. Another study in [17]

used models such as ResNet50, MobileNet, and EfficientNetB0 to train the model and test the validation accuracy on a dataset consisting of 3,556 images to analyze skin conditions. They concluded that EfficientNetB0 achieved the highest accuracy at 99%, compared to the ResNet50 and MobileNet models, as shown in Table 2.2.3.

Table 2.2.3: Validation accuracy for each model [27]

	Accuracy	Precision	Recall	F1-Score
ResNet50	0.93	0.93	0.93	0.93
MobileNet	0.94	0.94	0.94	0.94
EfficientB0	0.99	0.99	0.99	0.98

In addition, the authors in [18] also used YOLOv5 models to analyze skin conditions by using different numbers of epochs and batch sizes. The YOLOv5 model achieved better accuracy when they used a batch size of 16 and 100 epochs in the experiment, as this provided the optimum result in terms of precision, recall, and mAP50. However, they found that the result could be further optimized if the dataset of images were larger, and that a larger batch size could increase the training speed but would require higher GPU performance.

The authors in [19] conducted an experiment comparing the YOLOv5, YOLOv8, and Detectron models, which are deep learning models. In the experiment, they used 1,457 images and generated additional images using StyleGAN Ada to expand the dataset to 4,000 images, which helped resolve issues related to insufficient dataset size. They concluded that YOLOv8 achieved a higher accuracy rate compared to the other models, with its precision (80.2%) being higher than that of YOLOv5. However, its recall was lower than YOLOv5's, which meant that it missed some true positive data. The results of the three models are shown in Table 2.2.3.

Table 2.2.3 Result for YOLOv5, YOLOv8 and Detectron[15]

Model Name	mAP@0.5 : 0.95	Precision	Recall	F1Score
YOLOv5	73.5%	76.1%	68.1%	71.88%
YOLOv8	73.6%	80.2%	65.3%	71.99%
Detectron2	37.7%	42.1%	43.6%	42.84%

Moreover, the authors in [20] conducted an experiment involving a Mask R-CNN model to process the skin images before performing the skin analysis. This was combined with YOLOv3 and YOLOv4 using the same dataset to test the accuracy of the combination of models used for analysing the skin. The experiment concluded that combining the Mask R-CNN image processing model with YOLOv4 significantly improved the accuracy of the results when compared to using only one model in the training process. However, the dataset required must be large enough, although it needed a smaller dataset compared to models that did not use the Mask R-CNN method. The results of the combination of Mask R-CNN, YOLOv3, and YOLOv4 are shown in Table 2.2.4.

Table 2.2.4: Comparison on the mean average precision of models[16]

Method\Training	500	1000	1500
YOLOv3	54.52	50.01	55.68
YOLOv4	58.74	56.98	56.29
Mask R-CNN + YOLOv3 (100 images for training)	50.03	47.43	56.53
Mask R-CNN + YOLOv3 (250 images for training)	50.38	46.54	53.70
Mask R-CNN + YOLOv3 (500 images for training)	55.02	52.39	58.13
Mask R-CNN + YOLOv4 (100 images for training)	55.97	57.64	53.68
Mask R-CNN + YOLOv4 (250 images for training)	55.10	56.48	53.19
Mask R-CNN + YOLOv4 (500 images for training)	57.73	60.38	59.75

2.3 Existing Works on Skincare Product Recommendation

Few studies have been conducted on product recommendation in skin analysis systems. However, it plays a crucial role in providing personalized skincare product solutions for users based on their skin conditions.

As mentioned in [21], the skincare product dataset comes from certified and branded sources such as Sephora and Myntra Beauty Section. These products are recommended to individuals who use the system to capture their skin images. After the analysis process, the product recommendation system utilizes collaborative filtering. Users input data extracted from the image they take, allowing the system to recommend products that are suitable for their skin type and skin tone based on their skin and product vectors. The system performs cosine similarity to identify the products best suited for users. Figure 2.3.1 shows the flowchart of the product recommendation system, and Figure 2.3.2 displays the information submitted by users to track the skincare products suitable for their skin.

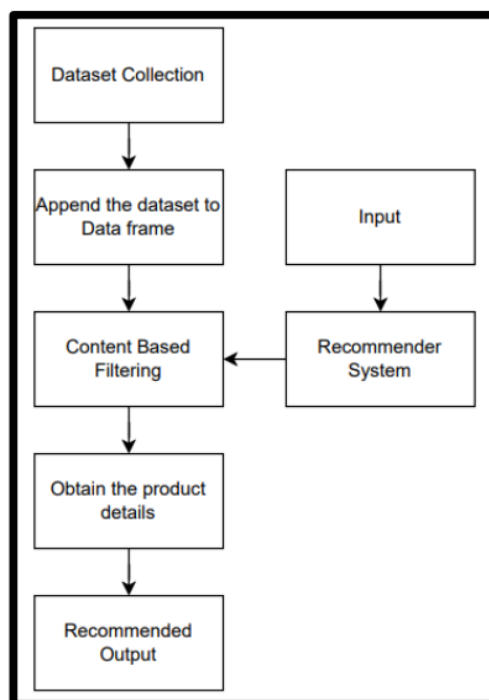


Figure 2.3.1: Product recommendation flowchart [21]

The form, titled 'Results', is used for user input. It includes a 'Skin' dropdown menu currently set to '2'. Below this, there are radio button options for 'Type' (All, Normal, Oily, Dry) and 'Acne' (Low, Moderate, Severe). A section titled 'Specify other skin concerns' contains checkboxes for various conditions: Sensitive, Winkles, Pore, Blackheads, Blemishes, Eye bags, Fine lines, Redness, Pigmentation, Whiteheads, Dark circles, and Dark spots. A 'SUBMIT' button is located at the bottom of the form.

Figure 2.3.2: Information submitted by users [21]

A previous study from [22] used a combination of the CNN model with product recommendations, called the Convolutional Neural Network-based classifier. In this model, the product dataset includes various attributes such as product ingredients and product names. After the users capture their skin image and perform the analysis process, the system filters out the products that are suitable for their skin condition. The system then identifies their skin type and recommends products based on that skin type. The model successfully achieved an accuracy of 99% after combining CNN with product recommendations. The flowchart of the model is shown in Figure 2.3.3.

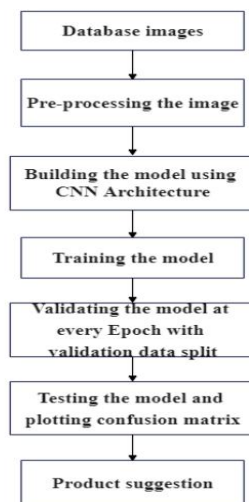


Figure 2.3.3 Flowchart for Convolution Neural Network-based classifier model [22]

In the previous study by [23], collaborative filtering techniques were divided into two different approaches: model-based and memory-based. To compare the accuracy of both techniques, they used three different types of datasets: transactions, products, and users. After the preprocessing process, the transaction data remained the same, but the number of products and users was reduced by filtering out irrelevant data and removing outliers. Table 2.3.1 displays the pre-processed data in the dataset.

Table 2.3.1 Dataset before and after preprocessing process [23]

Dataset	
<i>Before Preprocessing</i>	<i>After Preprocessing</i>
290.060 Transactions	290.060 Transactions
95.468 Products	40.640 Products
50.000 Users	26.672 Users

CHAPTER 2

After preprocessing, the study split the dataset into three datasets with different numbers of transactions. The users were allowed to choose three items from the transaction history, and five products were recommended to them. The average number of relevant products from different models is shown in Table 2.3.2. It is evident that the model-based recommendation is more accurate compared to the memory-based recommendation.

Table 2.3.2 Percentage of skincare products relevant to users [23]

Method	Dataset 1	Dataset 2	Dataset 3	Average
Memory-based	18,71	18,71	14,20	17,20
Model-based	17,42	20,65	16,13	18,07

In short, different models are implemented in the skin analysis system. Based on the findings of the skin analysis, the system will then apply the collaborative filtering technique to recommend skincare products to users. In this project, we will compare CNN, YOLOv8, EfficientNetB0, ResNet50, and the integrated OpenAI API with GPT Assistant to identify the most suitable model for implementing the skin analysis system and ensuring the highest possible accuracy. Afterward, products will be recommended to users based on the results of the skin analysis process. Users will also be able to retrieve their previous skin analysis results and compare them to track the improvement in their skin over time.

Chapter 3

System Methodology/Approach

3.1 System Architecture Diagram

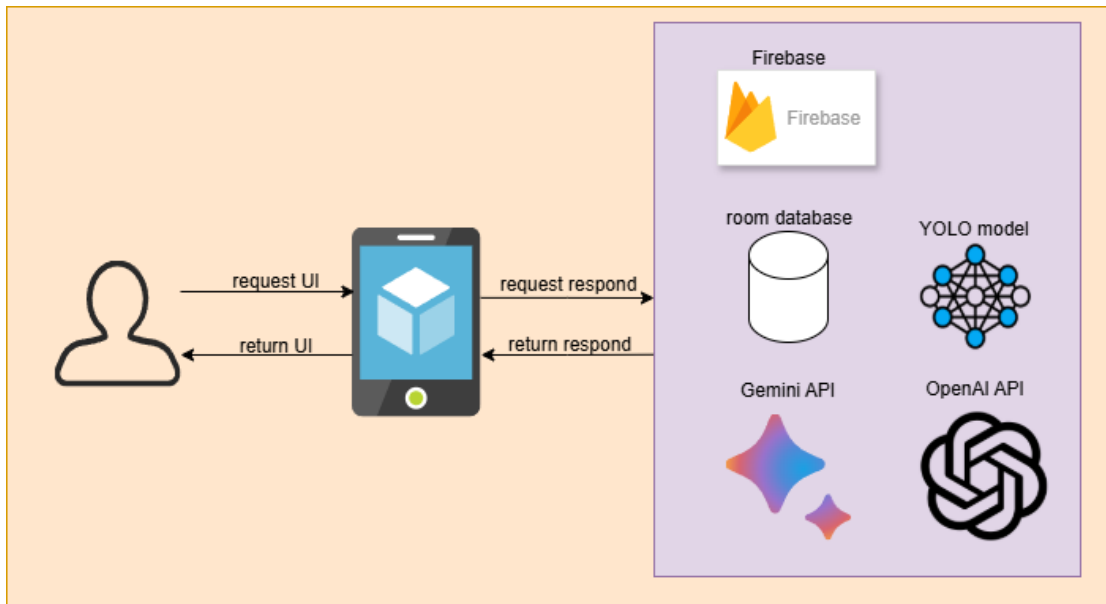


Figure 3.1.1 Overall system architecture diagram

Figure 3.1.1 shows the overall system architecture diagram, which is designed and developed for the skin analysis and recommendation system to provide seamless integration of functionalities for users. These functionalities include skin analysis, product recommendations, skincare routine management, and skin progress tracking.

The user interacts with the mobile application interface, which serves as the primary medium for executing tasks such as performing skin analysis, viewing skincare product recommendations, tracking skin progress, and managing skincare routines. The interface is mainly used to capture the behaviour of the user, where the user is required to perform actions such as capturing skin images or requesting product recommendations. To ensure that user interactions remain intuitive and responsive, the application integrates multiple backend services that process requests and return the relevant results. Firebase acts as the main backend cloud service, providing authentication, user account management, and synchronization of data across devices. This allows users to enjoy a seamless experience while ensuring that user-related data is securely stored in the database. Complementing this, the Room Database is used locally on the user's device to store skincare products and enable quick retrieval of data

from the skincare dataset. This design ensures that users can efficiently access product information even when internet connectivity is limited or unavailable. Since the skincare dataset will be frequently queried for recommendations and routine management, maintaining a local Room Database reduces the need for repeated calls to the Firebase server and significantly improves response time.

In addition, the YOLO model is deployed within the mobile application to process classification requests, determining the user's acne severity level and skin type, and returning accurate results directly to the application. To further enhance the system's recommendation capabilities, users are given the flexibility to select their preferred LLM models either the Gemini API or the OpenAI API. The chosen LLM supports product suggestions, skincare routine validation, and progress tracking to ensure that skincare product combinations stored in a routine are suitable for the user's acne severity and skin type. Each recommended product can also be validated through the selected LLM to confirm its suitability. For skin progress tracking, the selected LLM will generate descriptive analyses of the skin condition of user over time, enabling effective monitoring of improvements and providing personalized insights.

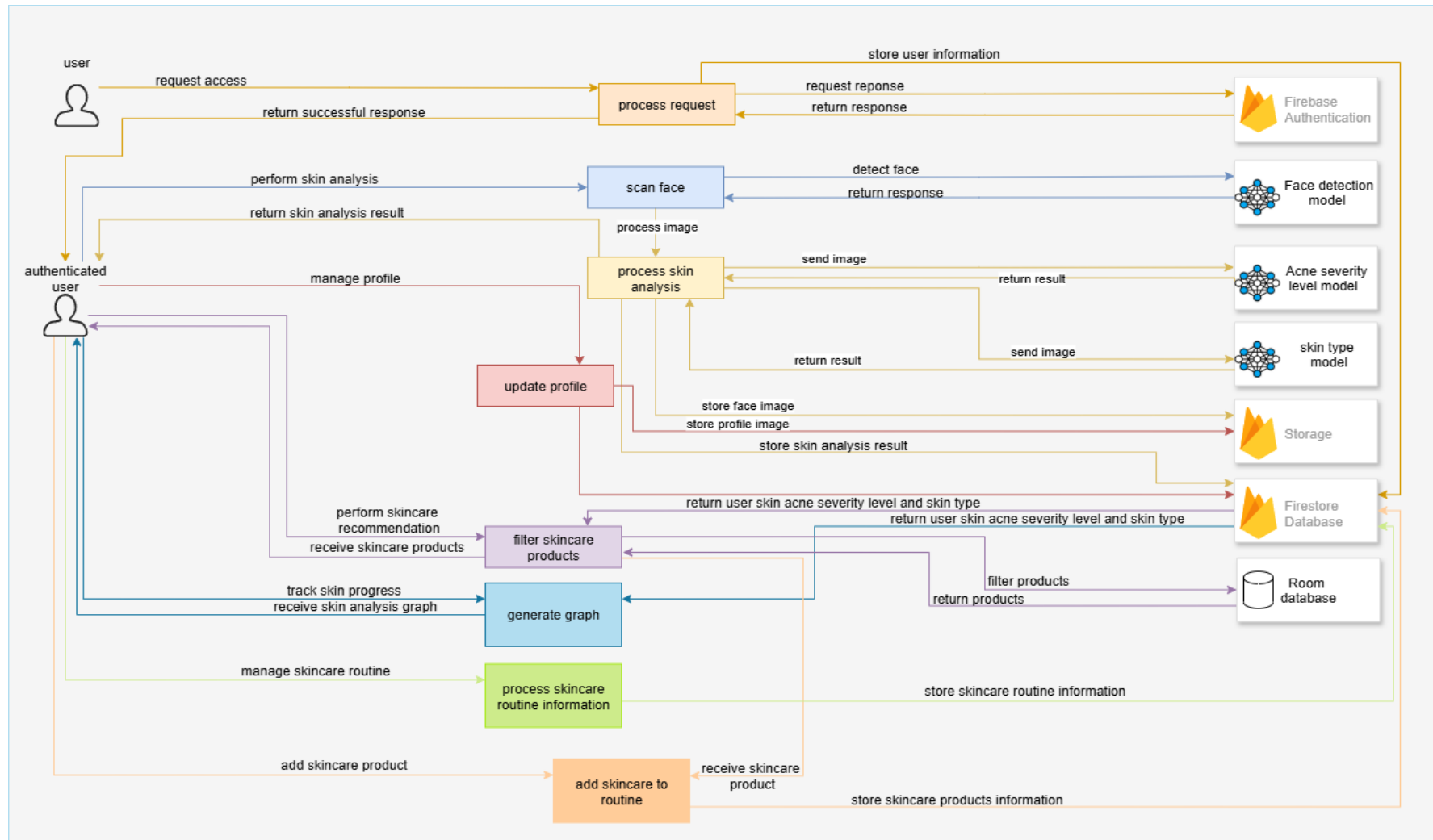


Figure 3.1.2 Skin Analysis and Recommendations system architecture diagram

CHAPTER 3

The mobile application system architecture, as shown in Figure 3.1.2 will begin when a user requests access to the mobile application. This request is handled through Firebase Authentication, which validates the credentials and securely stores the user's account information. Once authentication is successful, the user becomes an authenticated user and can access all the core features of the system, such as skin analysis, skincare recommendations, skin progress, and skincare routine tracking.

When performing skin analysis, the authenticated user initiates real-time face detection through the Android Library. The system detects the face within the frame and captures it. The captured image is then processed and analysed by two deep learning models: the acne severity model, which classifies acne levels into predefined categories, and the skin type model, which identifies the user's skin type. The processed image is stored in Firebase Storage, while the analysis results are saved in the Firestore Database. These results are also immediately returned to the user through the application interface, ensuring real-time feedback.

The system also allows the users to track their skin progress over time. Previous skin analysis results are retrieved from the database, and a graph is generated to visualize changes in acne severity levels and skin type progression. This helps users monitor improvements or worsening conditions across different periods. Additionally, the system also allows the user to manage their profile, where profile images and personal details can be updated and stored securely.

For skincare product recommendations, the system filters available products based on the skin analysis results. Product data is stored locally in the Room Database, ensuring efficient filtering and retrieval. The system cross-references acne severity level and skin type with product features to return the most relevant skincare products to the user. Additionally, filtering attributes such as acne severity level, skin type, and product category are stored in the Firestore Database. Furthermore, the system provides a skincare routine feature, where users can add selected products to their personalized routine, which is also stored in the Firestore Database. This routine information helps users follow a structured skincare schedule.

3.2 Use case diagram and Description

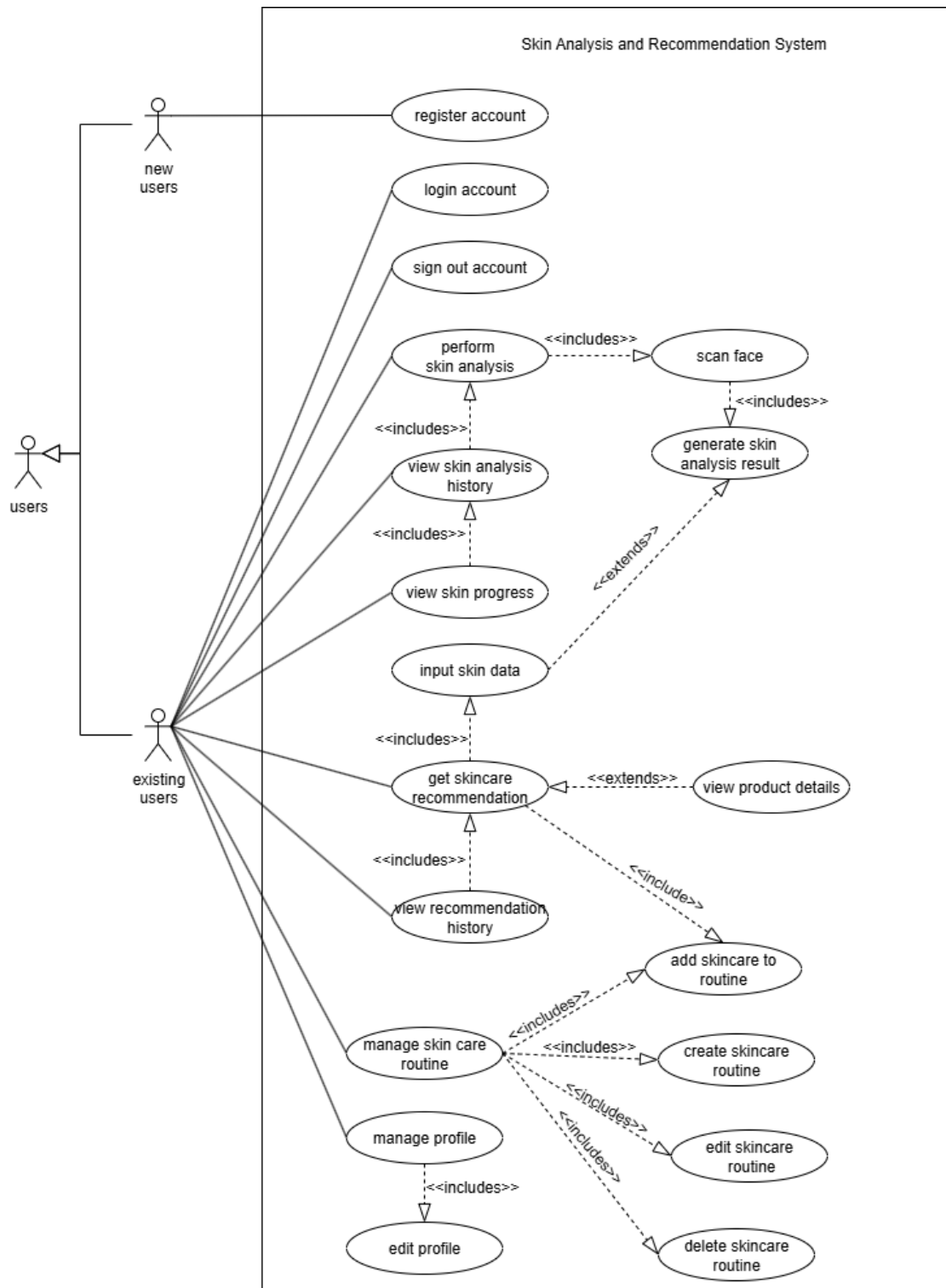


Figure 3.2.1 Use case diagram

3.2.1 Register account

Use Case Name: Register account	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to register an account		
Brief Description: This use case will describe the registration process done by the new user.		
Trigger: New users want to register to access the system		
Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The new user clicked the register button to register account 2. The user fills in their information. 3. The user clicks on register to register as a user 4. The system validates the details in the registration form and store in database. 5. The system directs the user to the home page.		
SubFlows: - S-1: Fill in their information 1. The user fills in full name 2. The user fills in email address 3. The user fills in passwords 4. The user fills in confirm password		

Alternate/Exceptional Flows:

S-1, 1a: The system will display “full name is required”

S-1, 2a: The system will display “email address is required”

S-1, 2b: The system will display “please enter a valid email address”

S-1, 3a: The system will display “password is required”

S-1, 4a: The system will display “please confirm your password”

S-1, 4b: The system will display “password must contain at least 8 characters with letters, numbers, and symbols (@\$!%*#?&@;)”

S-1, 4a: The system will display “password doesn’t match”

3.2.2 Login account

Use Case Name: Login account	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to login to an account		
Brief Description: This use case will describe the login process done by the new user.		
Trigger: New users want to login to access the system Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The existing user fills in their details to log in to the system. 2. The user clicks on the log in button to log in to the system. 3. The system will validate the details entered by the user. 4. The system will redirect the user to the main menu page.		
SubFlows: - S-1: Fill in login details 1. The user fills in an email address. 2. The user fills in passwords.		

Alternate/Exceptional Flows:

S-1, 1a: The system will display “email address is required”.

S-1, 1b: The system will display “please enter a valid email address”.

S-1, 2a: The system will display “password is required”.

S-1, 2b: The system will display “password must contain at least 8 characters with letters, numbers, and symbols (@\$!%*#?&@;)”.

3.2.3 Sign Out Account

Use Case Name: Login account	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to sign out account		
Brief Description: This use case will describe the sign out process done by user		
Trigger: New users want to sign out the system		
Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The new user clicked the settings icon. 2. The system will redirect user to settings page. 3. The user will click on the sign-out button. 4. The system will process the user and redirect user to login page.		
SubFlows: -		
Alternate/Exceptional Flows: -		

3.2.4 Perform skin analysis

Use Case Name: Perform skin analysis	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to perform skin analysis		
Brief Description: This use case will describe the performing skin analysis process		
Trigger: Existing users want to perform skin analysis		
Type: External		
Relationships: Association: User Include: scan face Extend: - Generalization: -		
Normal Flow of Events: 1. The user will log in to the system 2. The user will click on the skin analysis button to perform skin analysis in the main menu Execute scan face use case		
SubFlows: -		
Alternate/Exceptional Flows: -		

3.2.5 Scan Face

Use Case Name: Scan face	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to scan their face		
Brief Description: This use case will describe the process of scanning user face		
Trigger: User wants to scan their face to perform skin analysis		
Type: External		
Relationships: Association: User Include: generate skin analysis result Extend: - Generalization: -		
Normal Flow of Events: 1. The user adjusts their face to make sure it exists on the frame 2. The system provides feedback based on the adjustment of face 3. The user clicks on the capture button if their face is within the frame Execute generating skin analysis result use case		
SubFlows: S-1: Capture their face <ul style="list-style-type: none">The user can choose to retake the image by clicking the retake image button		
Alternate/Exceptional Flows: S-2,1a: face detected. Please center your face at the center of frame S-2,2a: no face detected.		

3.2.6 Generate Skin Analysis Result

Use Case Name: Generate skin analysis result	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to generate skin analysis result		
Brief Description: This use case will describe the generate skin analysis result process		
Trigger: Existing user want to generate skin analysis result process		
Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The system will process the image captured. 2. The system will get the result from the YOLOv8 model. 3. The system will display the result to the user.		
SubFlows: - S-1: Process Image Captured 1. The system sends the image to the acne severity YOLOv8 model to determine the acne severity level of the user’s face. 2. The system sends the image to the skin type YOLOv8 model to determine the skin type of the user’s face. 3. The model returns the results once the prediction is complete.		
Alternate/Exceptional Flows: -		

3.2.7 View Skin Progress

Use Case Name: view skin progress	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to view skin progress		
Brief Description: This use case will describe the view skin progress		
Trigger: Existing users want to view skin progress		
Type: External		
Relationships: Association: User Include: view skin analysis history Extend: - Generalization: -		
Normal Flow of Events: <div>1. The existing user clicks on skin progress button.</div> <div>2. The system will redirect the user to skin progress page.<div>If the user wants to view the skin analysis history<div>Execute view skin analysis history</div></div></div> <div>3. The user can view their skin progress in graph form.<div>The user will first redirect to view skin acne severity progress:<div>S-1: View skin acne severity progress is performed</div>The user can choose to view skin type progress:<div>S-2: View skin type progress is performed</div></div></div>		
SubFlows: - S-1: View skin acne severity progress <div>1. The system computes the user’s skin history to generate the results and accuracy graphs for acne severity level.</div> <div>2. The system performs descriptive analysis using the Gemini API.</div> S-2: View skin type progress <div>1. The system computes the user’s skin history to generate the results and accuracy graphs for skin type.</div>		

CHAPTER 3

2. The system performs descriptive analysis using the Gemini API.
Alternate/Exceptional Flows: -

3.2.8 View Skin Analysis History

Use Case Name: view skin analysis history	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to view skin analysis history		
Brief Description: This use case will describe the process of viewing skin analysis history		
Trigger: Existing users want to view skin analysis history		
Type: External		
Relationships: Association: User Include: perform skin analysis Extend: - Generalization: -		
Normal Flow of Events: 1. The system will retrieve the skin analysis history from database. 2. The system will display skin analysis history. 3. The user will not view any history if not perform skin analysis. If user wants to perform skin analysis: Execute perform skin analysis use case.		
SubFlows:		
Alternate/Exceptional Flows: -		

3.2.9 Input skin data

Use Case Name: Input skin data	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to input skin data		
Brief Description: This use case will describe the input skin data process		
Trigger: Existing users want to input skin data Type: External		
Relationships: Association: User Include: - Extend: Generate skin analysis result Generalization: -		
Normal Flow of Events: 1. The existing user clicks on products recommendation main menu. 2. The system will redirect user to select the skin acne severity level and skin type If user performed skin analysis: S-1: Use current result If user does not perform skin analysis: S-2: Use customize result		
SubFlows: - S-1: Use current result 1. The user can use their current latest skin analysis result with acne severity level and skin type. 2. The system will automatically fill in their skin acne severity level and skin type 3. The user will need to click on “confirm” button. S-2: Use customize result 1. The user needs to select the skin acne severity level and skin type on their own. 2. The user needs to click on “confirm” button.		
Alternate/Exceptional Flows:		

3.2.10 Get Skincare Recommendation

Use Case Name: Get skincare recommendation	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to get skincare recommendation		
Brief Description: This use case will describe the get skin care recommendation process		
Trigger: Existing users want to get skincare recommendation		
Type: External		
Relationships: Association: User Include: input skin data Extend: view product details Generalization: -		
Normal Flow of Events: 1. The user needs to input their skin data Execute input skin data use case 2. The system will record the skin data inputted by users. 3. The user needs to select the product category. 4. The system will filter out the product which meets the product category and user skin data. 5. The user clicks on each product to get the read the product details. Execute view product detail use case. 6. The user clicks on “add to routine” on each product to add the product into the skincare routine. Execute add skincare to routine use case.		
SubFlows: -		
Alternate/Exceptional Flows: -		

3.2.11 View product details

Use Case Name: view product details	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to view product details		
Brief Description: This use case will describe the process of viewing product details		
Trigger: New users want to view product details Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The system renders skincare product details from the room database. 2. The system will use skincare product details to get AI recommendation. 2. The system displays skincare product details and AI recommendations to user.		
SubFlows: -		
Alternate/Exceptional Flows: -		

3.2.12 Manage Skincare Routine

Use Case Name: manage skincare routine	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to manage skincare routine		
Brief Description: This use case will describe the manage skincare routine process		
Trigger: New users want to manage skincare routine		
Type: External		
Relationships: Association: User Include: add skincare to routine, create skincare routine, edit skincare routine, delete skincare routine Extend: - Generalization: -		
Normal Flow of Events: 1. The user clicks on skincare routine button in main menu. 2. The system redirects the user to skincare routine pages. 3. The user can choose to “add new routine” Execute create new routine use case 4. The user can choose to modify existing routine Execute edit skincare routine use case 5. The user can choose to delete existing routine Execute delete skincare routine use case 6. The user can view the skin care analysis history to add skincare products Execute add skincare to routine use case		
SubFlows: -		
Alternate/Exceptional Flows: -		

3.2.13 Add Skincare to Routine

Use Case Name: add skincare to routine	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – add skincare to routine		
Brief Description: This use case will describe the process of adding skincare to routine		
Trigger: New users want to add skincare to routine		
Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The system will allow users to add skincare products to the routine. If the user has an existing routine: S-1: Add into routine S-2: Create a new routine If the user does not have routine: S-2: Create a new routine		
SubFlows: S-1: Add into routine 1. The user selects an existing routine to add the skincare product selected. S-2: Create a new routine 1. The user creates a new routine Execute create new routine 2. The user selects created routine to add the skincare product selected.		
Alternate/Exceptional Flows: -		

3.2.14 Create skincare routine

Use Case Name: create skincare routine	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to create skincare routine		
Brief Description: This use case will describe the process of creating skincare routine		
Trigger: New users want to create skincare routine		
Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The user fills in the routine details to create a new skincare routine. 2. The user clicks on “create routine” button to create the routine.		
SubFlows: S-1: Routine details 1. The user fills in the routine name 2. The user fills in the description of routine		
Alternate/Exceptional Flows: S-1,1a: The system displays “routine name is required”.		

3.2.15 Edit skincare routine

Use Case Name: Edit skincare routine	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to edit skincare routine		
Brief Description: This use case will describe the process of editing skincare routine.		
Trigger: New users want to edit skincare routine		
Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The system will display current routine details to allow users to make changes. 2. The user modifies the routine details. 3. The system validates the routine details and saves the routine details into database.		
SubFlows: S-1: Routine details 1. The user changes the routine name. 2. The user changes the routine description.		
Alternate/Exceptional Flows: S-1,1a: The system will display “the routine name cannot be empty”.		

3.2.16 Delete skincare routine

Use Case Name: Delete skincare routine	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to delete skincare routine		
Brief Description: This use case will describe process of deleting skincare routine		
Trigger: New users want to delete skincare routine		
Type: External		
Relationships: Association: User Include: - Extend: - Generalization: -		
Normal Flow of Events: 1. The system will display delete routine dialog for users. 2. The user clicks on “delete” button to delete routine.		
SubFlows: -		
Alternate/Exceptional Flows:		

3.2.17 Manage Profile

Use Case Name: manage profile	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to manage their profile		
Brief Description: This use case will describe the process on managing their profile		
Trigger: Existing users want to manage their profile		
Type: External		
Relationships: Association: User Include: edit profile Extend: - Generalization: -		
Normal Flow of Events: 1. The existing user clicks on profile to edit profile 2. The system will redirect the user to edit profile Execute edit profile user case		
SubFlows: -		
Alternate/Exceptional Flows:		

3.2.18 Edit profile

Use Case Name: edit profile	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detail, Essential	
Stakeholders and Interests: User – want to edit their profile		
Brief Description: This use case will describe the process on editing their profile		
Trigger: Existing users want to edit their profile		
Type: External		
Relationships: Association: User Include: edit profile Extend: - Generalization: -		
Normal Flow of Events: 1. The user can edit their profile details and change their profile photo. 2. The user can choose to “save changes” or “cancel” to discard any changes		
SubFlows: - S-1: Save changes <div><div>1.</div><div>The system will start to process the changes made by users.</div></div> <div><div>2.</div><div>The system will redirect user back to the main menu.</div></div> S-2: Cancel <div><div>1.</div><div>The system will discard the changes made by users.</div></div> <div><div>2.</div><div>The system will redirect user back to the main menu.</div></div>		
Alternate/Exceptional Flows: -		

3.3 Activity Diagram

3.3.1 User access

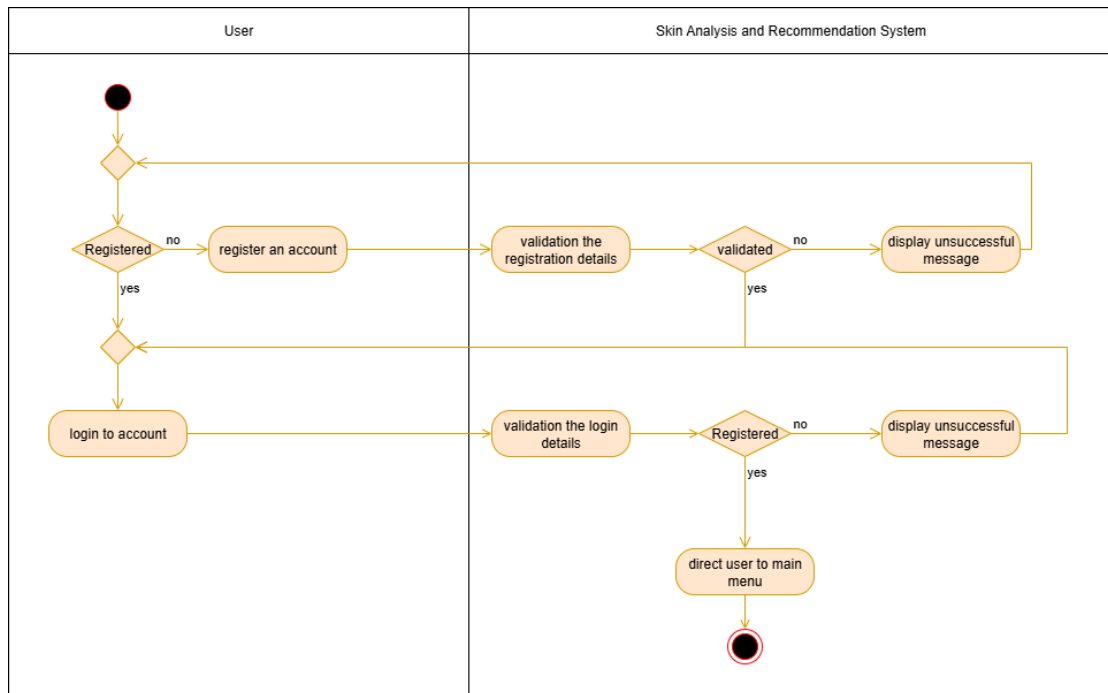


Figure 3.3.1.1 User access activity diagram

3.3.2 Perform skin analysis

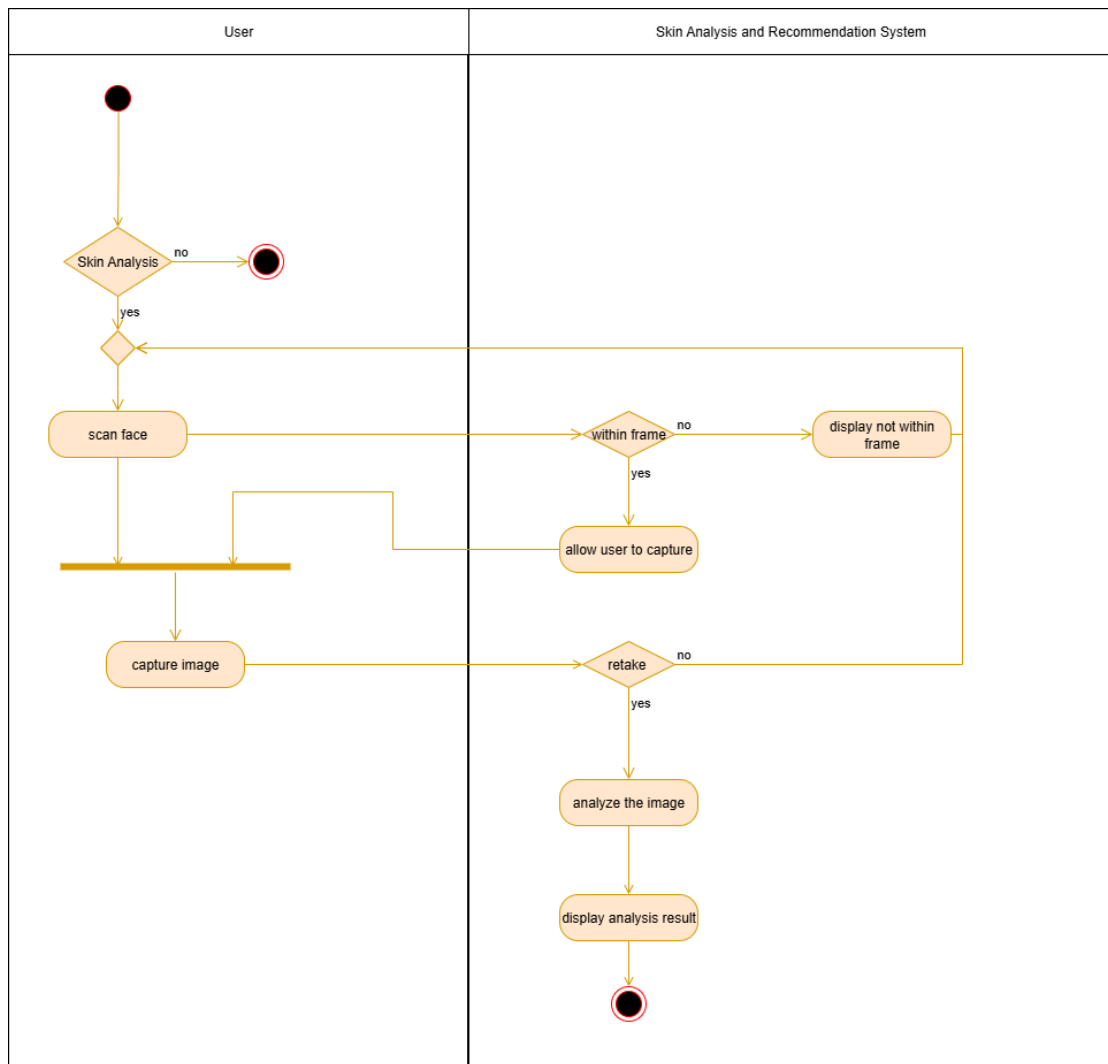


Figure 3.3.2.1 Perform skin analysis activity diagram

3.3.3 Get Skincare Recommendation

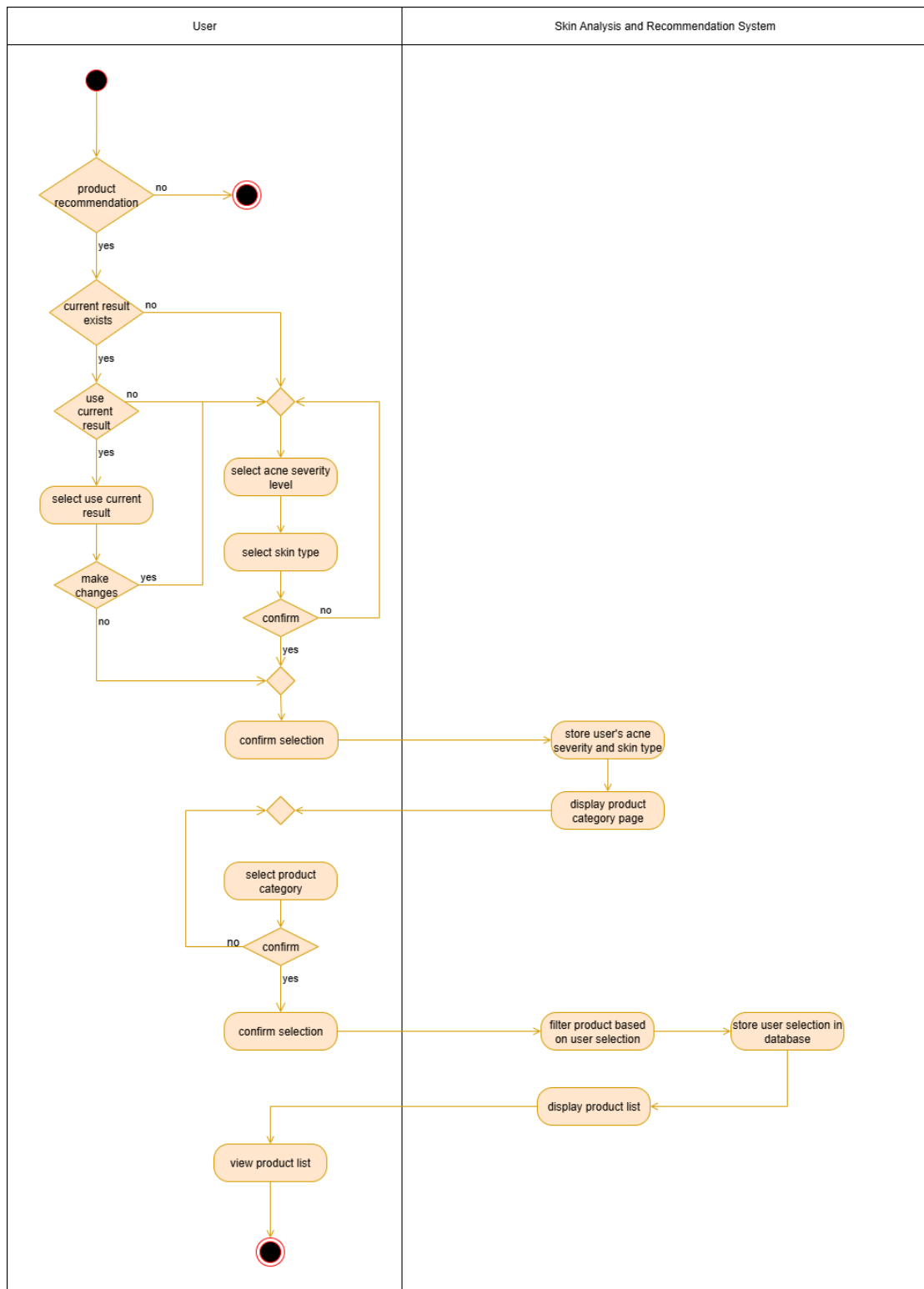


Figure 3.3.3.1 Get skincare recommendation activity diagram

3.3.4 View Skincare Product

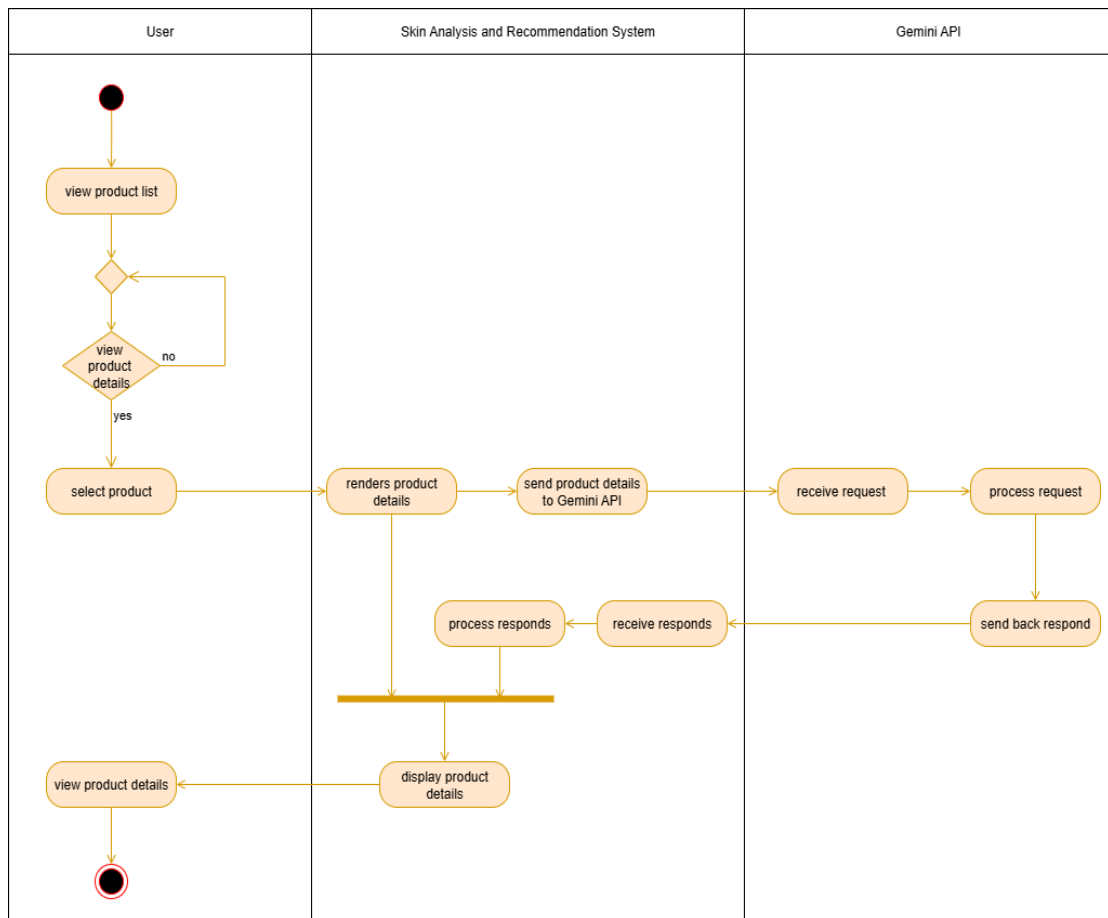


Figure 3.3.4.1 View skincare product activity diagrama

3.3.5 Add skincare to routine

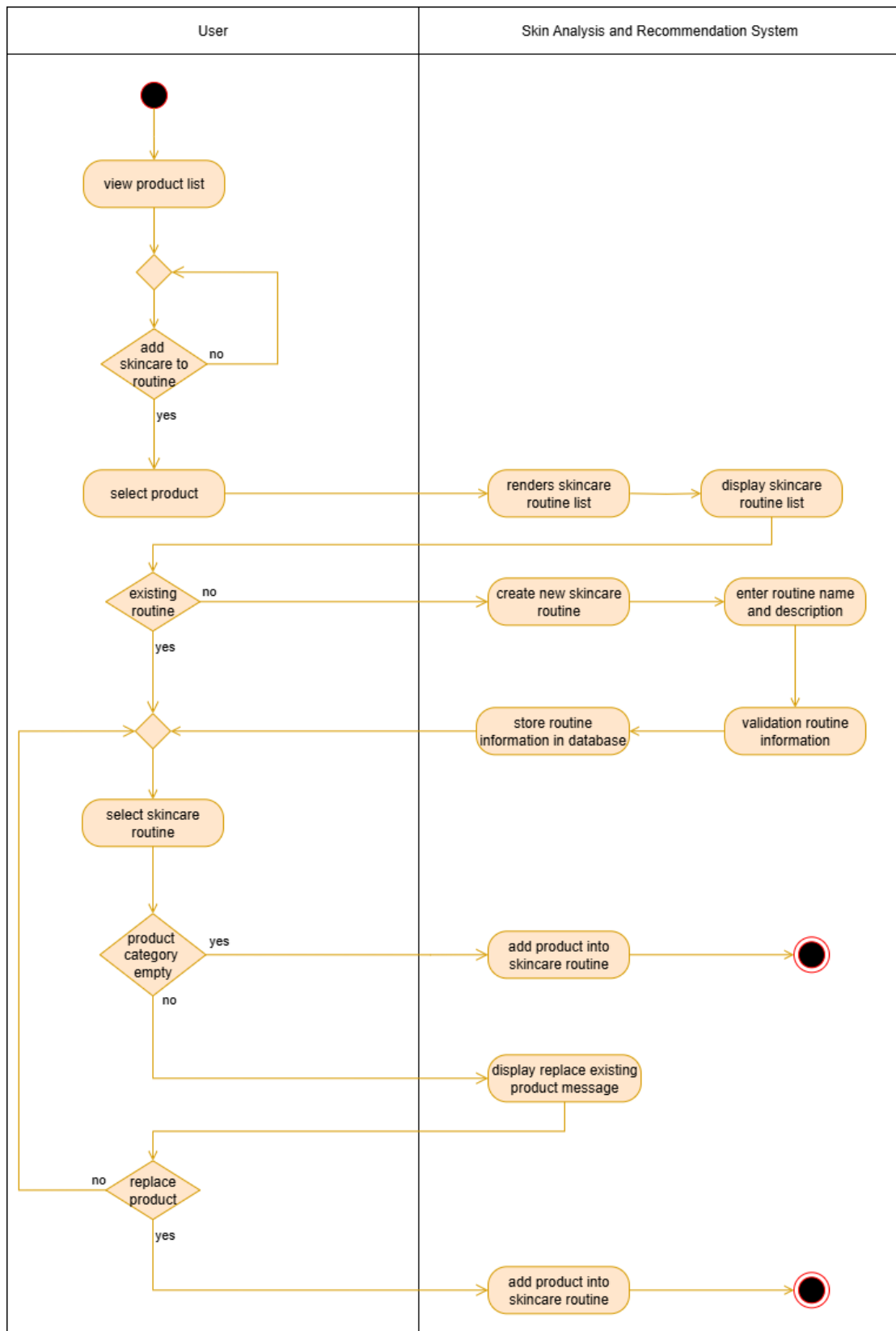


Figure 3.3.5.1 Add skincare to routine activity diagram

3.3.6 Modify Skincare Routine

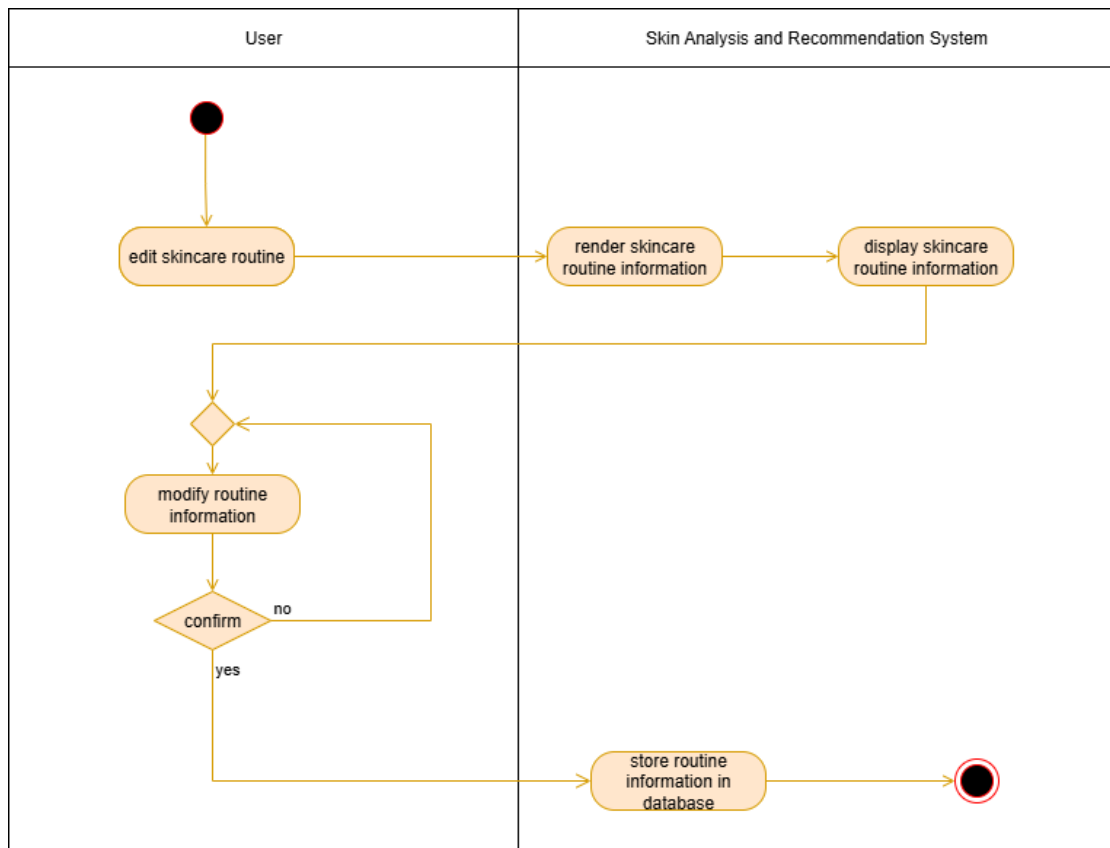


Figure 3.3.6.1 Modify skincare routine activity diagram

3.3.7 Delete Skincare Routine

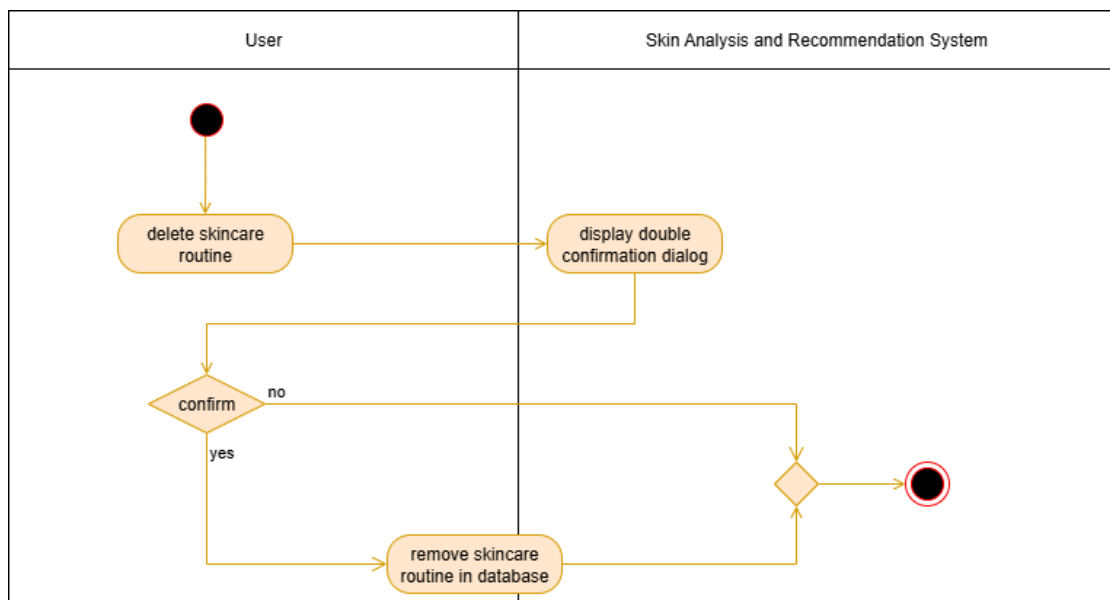


Figure 3.3.7.1 delete skincare routine activity diagram

3.3.8 View Skincare Routine

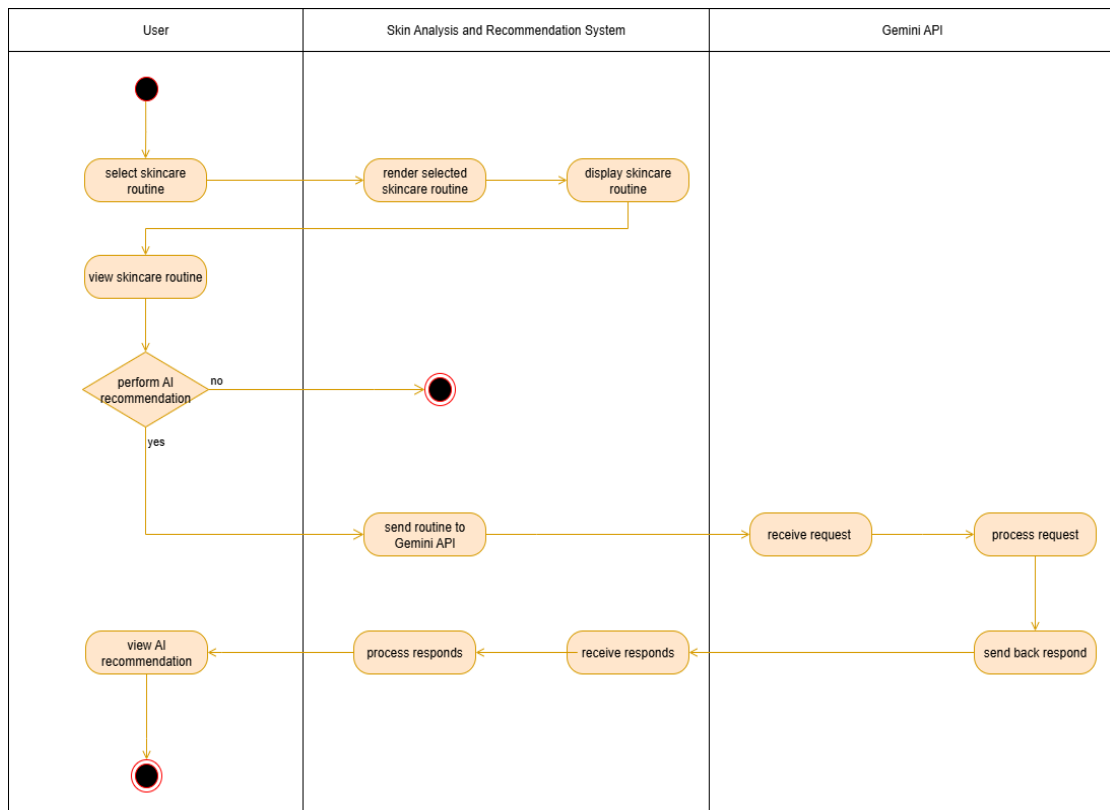


Figure 3.3.8.1 view skincare routine activity diagram

3.3.9 Skin Progress Tracking

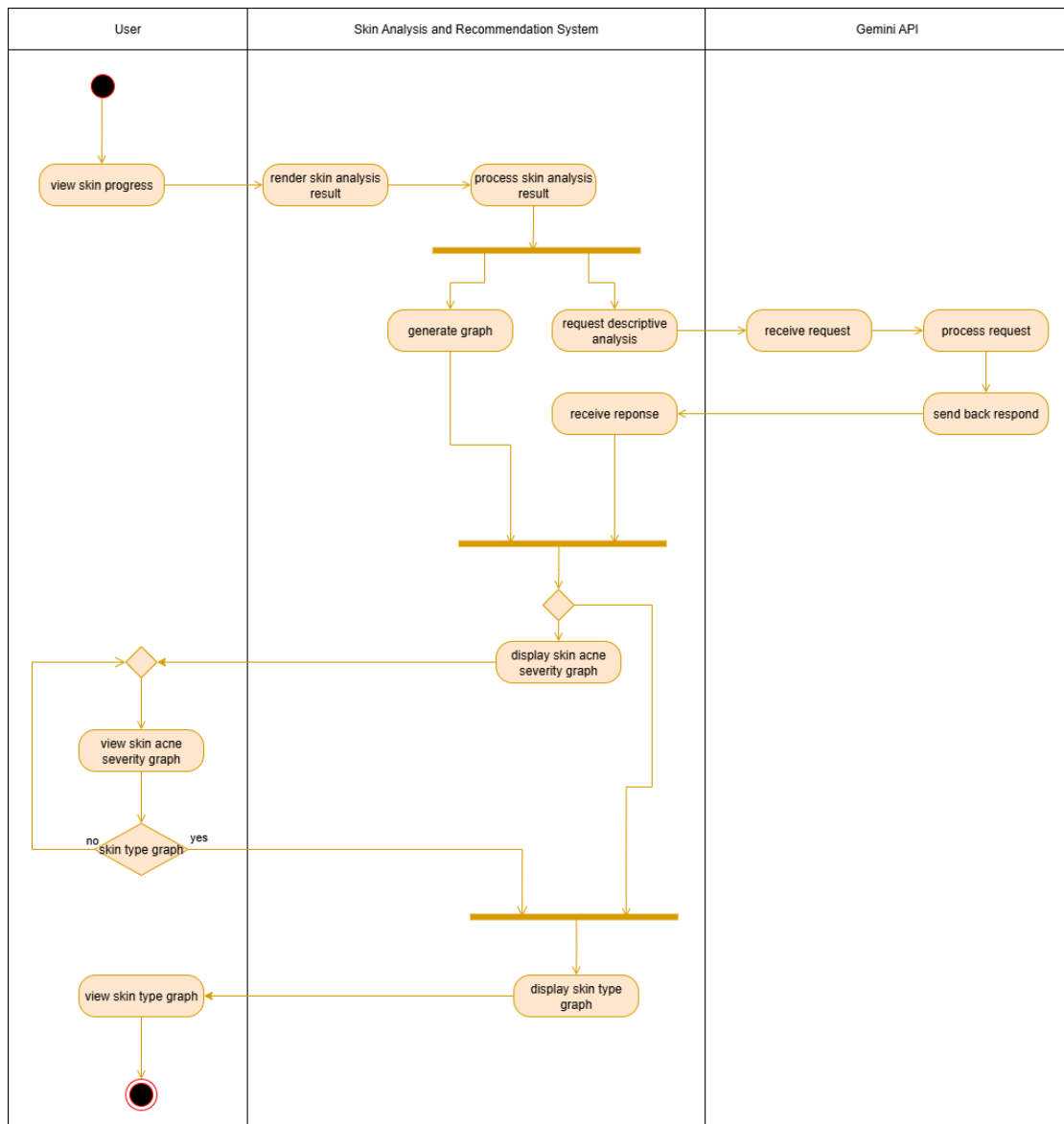


Figure 3.3.9.1 skin progress tracking activity diagram

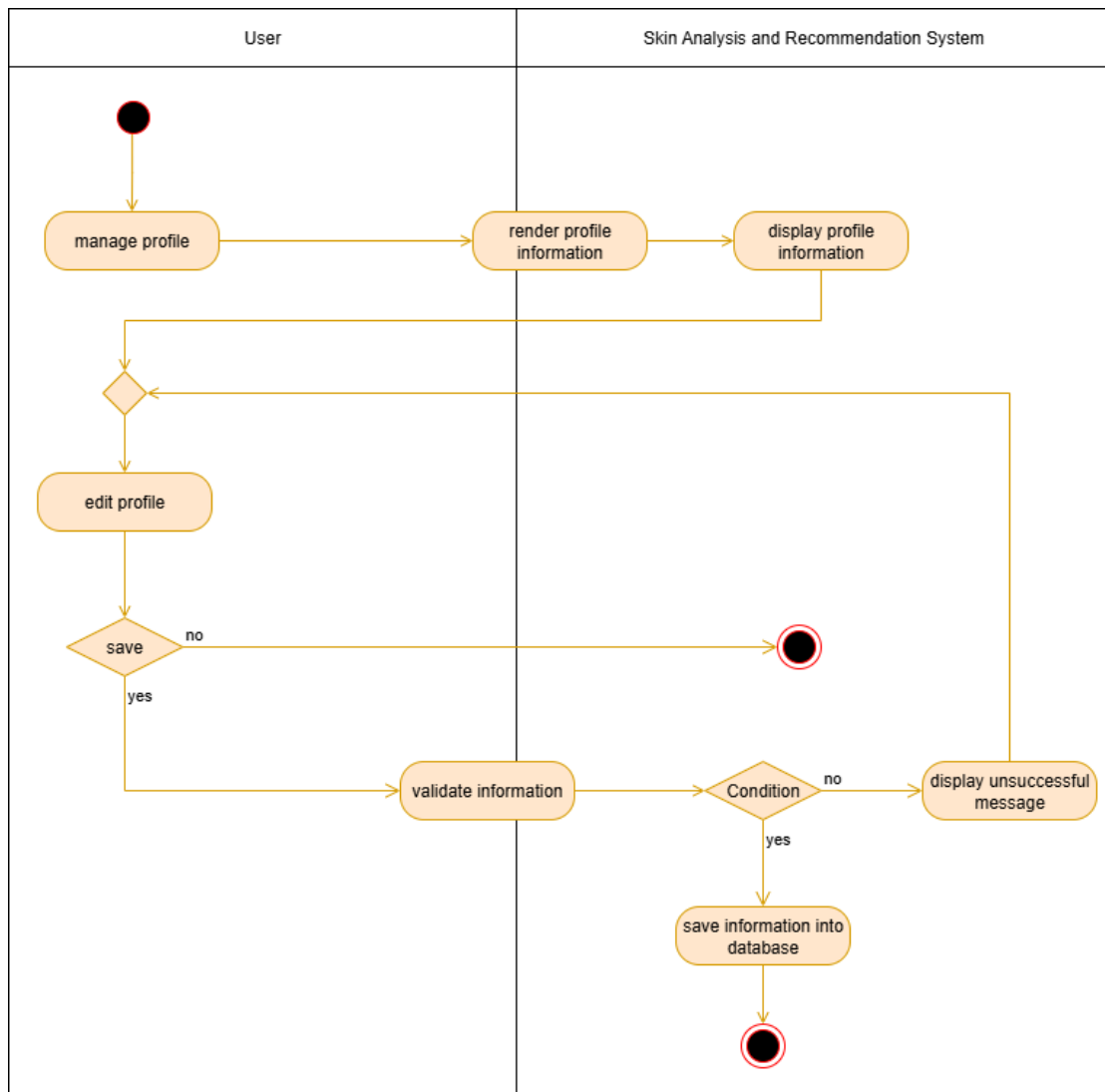
3.3.10 Manage Profile

Figure 3.3.10.1 manage profile activity diagram

3.4 Timeline diagram

In FYP1, the focus will be on achieving the first objective, which involves comparing different models and completing the report for FYP1. The timeline for FYP1 is shown in Figure 3.4.1.

		FYP1												
Task ID	Task Name	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Revise the proposal													
2	Discuss with supervisor on FYP1													
4	Performance Analysis of CNN and YOLO Models													
5	Performance Analysis of Resnet and EfficientNet Models													
6	Compare performance of all models													
7	Complete system design and overview													
8	Complete implementation issues and challenges													
9	Complete preliminary work in report													
10	Review the report													
11	Report submission													
12	Result presentation													

Figure 3.4.1 Timelines for FYP1

In FYP2, the focus of the project will shift to two additional objectives which are recommending skincare products for users and developing a model for scanning user skin to recommend suitable skincare products.

		FYP2												
Task ID	Task Name	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Discuss with supervisor on FYP2													
2	Perform cleaning on skincare products dataset													
3	Performance analysis on skincare products dataset													
4	Design layout of mobile application													
5	Develop the layout of mobile application													
6	Implement the model into mobile application													
7	Perform system testing													
8	Complete the report													
9	Review the report													
10	Report submission													
11	Presentation													

Figure 3.4.2 Timelines for FYP2

Chapter 4

System Design

4.1 System Workflow Diagram

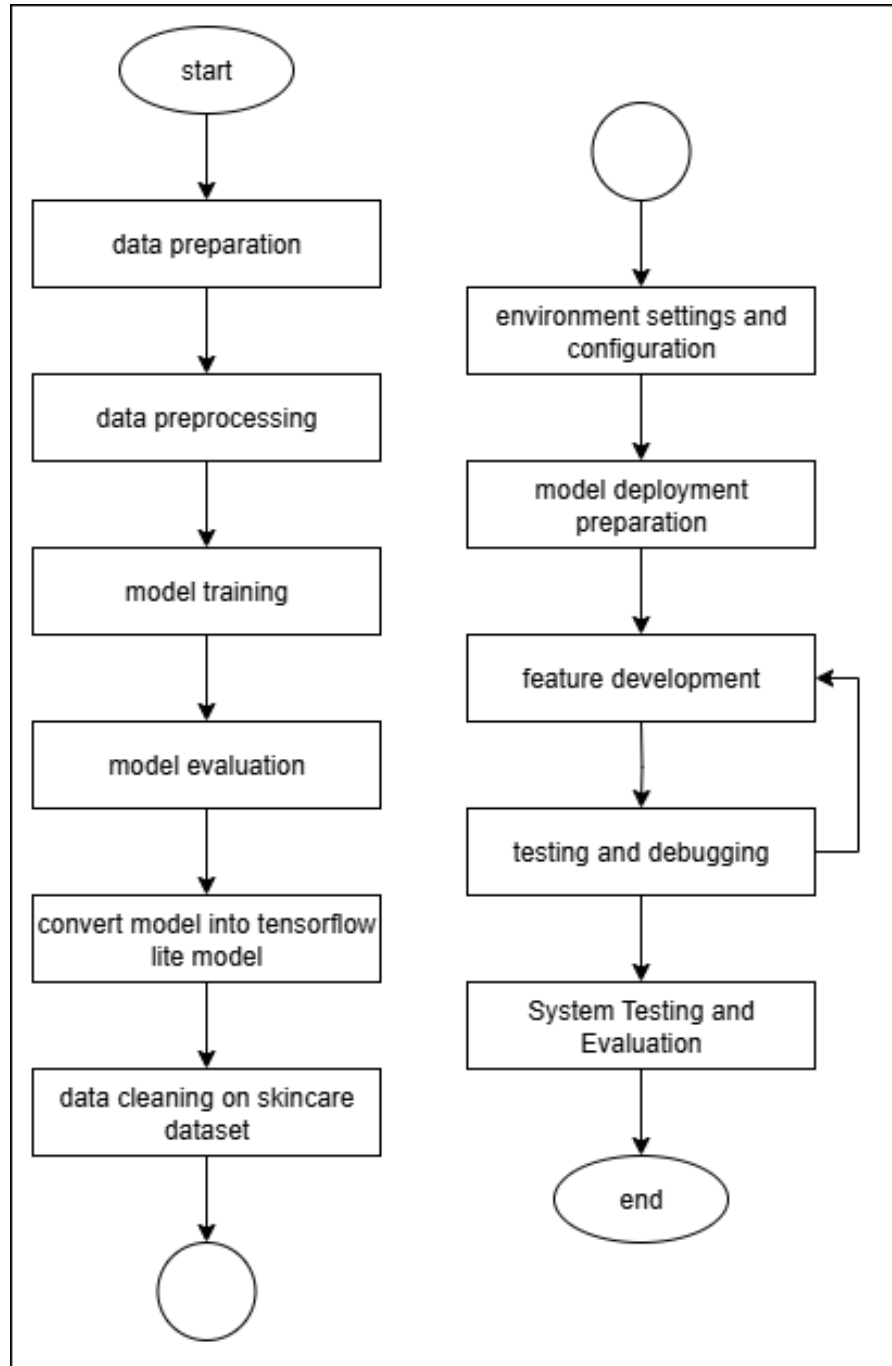


Figure 4.1.1 System Workflow Diagram

In Figure 4.1.1, the system workflow diagram illustrates the overall process of developing the skin analysis and recommendation system. The workflow begins with the data preparation phase, where relevant datasets are collected and organized. This phase not only involves model development but also involves preparing the skincare dataset used in the product recommendation module, enabling the system to suggest suitable products to users. The next step is data preprocessing, which includes cleaning, resizing, normalizing, and augmenting the dataset to ensure its suitability for training. Subsequently, five AI models which are Classic CNN, EfficientNetB0, ResNet50, YOLOv8, and the GPT assistant-based model are trained and tested to identify the best-performing approach. Model evaluation is then conducted using a test set, with metrics such as accuracy, precision, recall, and confusion matrices employed to assess effectiveness. Finally, the best-performing model is selected and converted into a TensorFlow Lite model to optimize performance for deployment on the mobile application.

Before proceeding with mobile application development, a data cleaning process is applied to the skincare dataset to ensure its suitability for system integration. This process involves removing empty cells, separating combined cells, correcting inconsistent values, and eliminating duplicate records. Additionally, irrelevant or noisy entries are filtered out to improve the overall quality of the dataset. This step is crucial, as a clean and reliable dataset directly contributes to the accuracy of the model and the overall performance of the mobile application.

After completing the data cleaning process, the workflow continues with environment settings and configuration, where the development environment is configured for mobile application development, backend server services, and AI API tools such as the Gemini API. This is followed by model deployment preparation, which ensures that the trained model is properly integrated into the mobile application. Next, the process moves to feature development, where core functionalities such as skin analysis, product recommendations, skincare routine management, and progress tracking are implemented. During feature development, continuous testing and debugging are performed to ensure that both the mobile application and the integrated model function correctly without errors. Finally, system testing and evaluation are conducted to

CHAPTER 4

validate the complete application in terms of usability, performance, and overall reliability.

Through this workflow, the AI model and mobile application are seamlessly integrated, resulting in a complete and effective skincare analysis and recommendation system.

4.2 Model Classification Pipeline

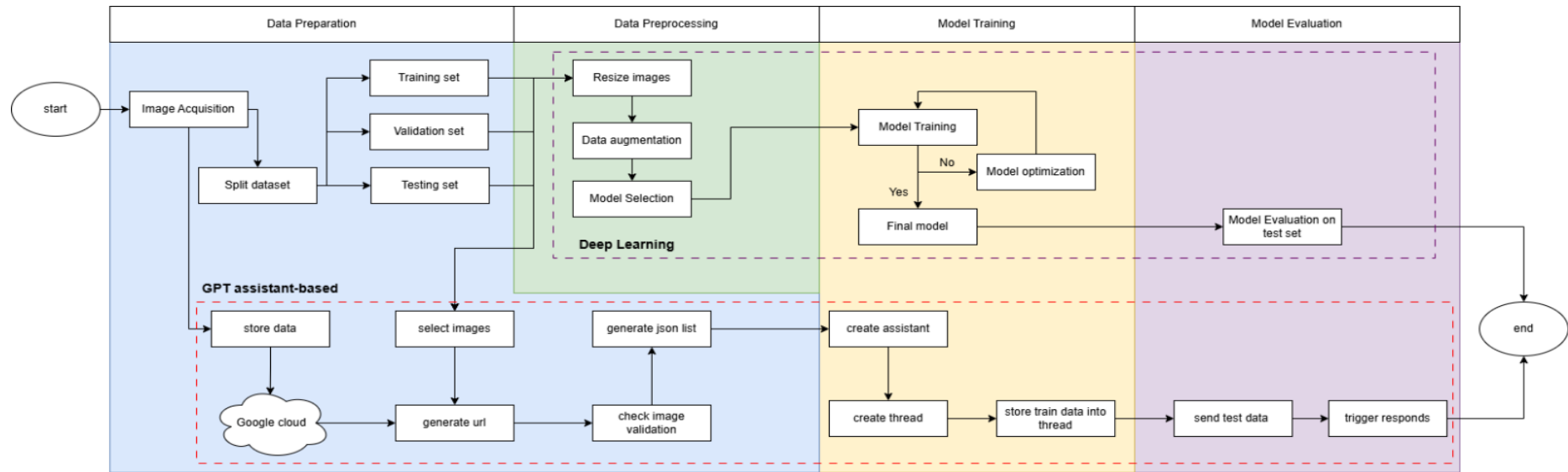


Figure 4.2.1 Model Classification Pipeline

Figure 4.2(a) shows that the model classification pipeline of both deep learning models and GPT assistant-based model. It will separate into four main stages which are data preparation, data preprocessing, model training and model evaluation.

In the data preparation stage, the datasets which are collected are required to build the model. Once all the images are collected, the images will then split into three sets of data which are training, validation and testing set. For GPT assistant-based models, the images are uploaded to Google Cloud Storage to reduce the usage of tokens when triggering responses via the OpenAI API. A set of random images is selected from both the training and testing sets. The names of the selected images are used to retrieve the corresponding Google Cloud links, and an additional check is performed to ensure that all selected images are available in the cloud. Then, a JSON list is generated to store the URLs of the images with their respective labels.

In the data preprocessing stage, deep learning classification models will involved several steps, including resizing the images across all three datasets. Additionally, data augmentation is performed on the training dataset to enhance the model's ability to generalize.

In the model training stage, various deep learning models which are the classic CNN, YOLOv8, EfficientNetB0, and ResNet50 models are selected for training using the training and validation datasets. The accuracy of each model is recorded. Optimization techniques, includes adjusting the learning rate, applying dropout, and standardizing the inputs, are applied as needed to improve performance. If the model demonstrates no signs of overfitting or underfitting, meaning the accuracies of both the training and validation datasets are well-balanced. Then, it will be considered the final model. For GPT assistant-based model, it will involve creating the GPT assistant and initializing the thread. The labelled training images are stored as messages within the thread to allow the assistant to learn the context of the images along with their respective labels.

In the model evaluation stage, the final model is tested using the unseen testing dataset to assess its performance. This ensures the model is accurate and robust, avoiding scenarios where the model performs well during training but poorly during evaluation.

Finally, the results of all models are compared to determine the most suitable one. For, GPT assistant-based model, the images selected from the testing set are sent to the assistant for analysis. The assistant analyses each of the testing images and predicts its corresponding level based on the training context provided.

4.2.1 Data Preparation

In data preparation stage, images were collected from two Kaggle datasets which are an acne grading dataset by R. Lathiya [24] and a skin type dataset by Alwi [25]. The acne grading dataset contains 999 images categorized into three severity levels which are level 0, level 1, and level 2. The skin type dataset includes 672 images classified into dry, normal, and oily skin types.

Besides, both will split into three sets of data which are training, validation and testing sets with portions of 75%, 15% and 10% for both deep learning pipeline and GPT assistant-based pipeline [26]. For GPT assistant-based pipeline, there will be an extra step which is the images collected will store the entire dataset in Google Cloud Storage without splitting it. It will then retrieve the images from local storage to obtain the image names. After obtaining the image names, the pipeline will generate corresponding URLs based on the location of the dataset in Google Cloud Storage, allowing access to the images. Once the URLs are generated, the pipeline will move on to the data preprocessing stage.

4.2.2 Data Preprocessing

After splitting the images into three datasets, they will undergo preprocessing, which includes resizing them to a uniform shape. Data augmentation will be applied to the training set of each dataset to increase its diversity.

For deep learning approach, four models have been used which are Classic CNN, ResNet50, EfficientNetB0, and YOLOv8 model. Among these models, the Classic CNN serves as a baseline to evaluate the performance of the other models. It consists of three convolutional layers with a max pooling layer. The purpose of using this model is to investigate whether the classic CNN model can handle the data effectively when compared with the other pretrained models used in this project. For the YOLOv8 model,

YOLOv8n, also known as YOLOv8 nano, was chosen due to its lightweight and fast architecture, making it suitable for tasks that require fast computation. Its design allows for fast processing speeds with minimal GPU usage. Additionally, we will incorporate YOLOv8n-cls, a variant fine-tuned for image classification, was used to enhance classification accuracy and overall performance [27].

The EfficientNetB0 model was selected due to its balance between accuracy and computational efficiency. As a lightweight and powerful pretrained model, it can achieve high accuracy with fewer parameters. In this project, transfer learning is used by freezing the base layers. This means that the features learned in these base layers remain, and only the newly added top layers will train on the dataset. This approach helps the model learn faster and more efficiently without retraining the entire model [28].

For the ResNet50 model, this model is another pretrained model that is well-suited for image classification [29]. One of the key advantages of this model is that it retains its powerful learned features while allowing customization of only the top layer to suit the project's needs. This includes adjustments to metrics such as the dropout rate and learning rate. Modifications such as adjusting the dropout rate and learning rate are applied to tailor the model to the specific needs of this task.

4.2.3 Model Training

In the training stage, three optimizers are defined: dropout, learning rate, and epochs. Dropout involves randomly disabling neurons during the training process to reduce overfitting [30]. This technique helps speed up and optimize the learning process while improving the model's performance on unseen data. Additionally, the learning rate is a crucial hyperparameter that regulates how the model adapts to errors at the end of each epoch [31]. The number of times the entire training and validation set are processed by the model is referred to as epochs [32]. Table 3.3.5(a) shows the dropout, learning rate, and epochs used for the classic CNN model, ResNet50 model, and EfficientNetB0 model. The epochs for the YOLOv8 model will be set as 20.

Table 4.2.3.1 hyperparameter used in model training

Dropout	0.3
Learning rate	0.001
Epochs	20

The GPT assistant-based approach will also be utilized in this project. This method takes advantage of the capabilities of natural language processing and advanced reasoning provided by large language models to complement traditional deep learning techniques.

For the GPT assistant-based classification, the GPT-4o model was chosen due to its balance of performance, efficiency, and lower cost compared to other OpenAI models. The assistant integrates with threads, which act as “sessions” that store the messages sent to the assistant. This setup allows multiple messages to be stored in a single thread, preventing the token limit from being exceeded in each conversation and improving overall prediction efficiency [33]. In addition, 25 images are randomly selected from the training set and stored in the thread for GPT assistant-based model learning purposes.

4.2.4 Model Evaluation

Model evaluation is the process of assessing how well a machine learning or deep learning model performs on unseen data. In the deep learning pipeline, the test set is fed into the trained model to evaluate its accuracy and reliability. During the evaluation, various metrics such as accuracy, precision, recall, F1-score, and confusion matrix are used to analyze the model's performance. The confusion matrix helps to understand how well the model handles different data categories. The sample of the confusion matrix is shown in Table 4.2.4(a) [34].

For the GPT assistant-based model, the test data is sent to the assistant, triggering a response to identify the class of the images based on the labelled images sent during the model training process. The results will be used to build a confusion matrix showing the true predictions and false predictions for each class. From the confusion matrix,

precision, recall, and F1-score are computed, offering insights into the model's performance and its ability to identify images and reveal weaknesses.

Table 4.2.4.1 Sample of confusion matrix [34]

	Predicted	Predicted
Actual	True Positive (TP)	False Negative (FN)
Actual	False Positive (FP)	True Negative (TN)

The formula of accuracy, precision, recall and F1-score are follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (1)$$

$$\text{Precision} = \frac{\text{TP} + \text{FP}}{\text{TP}} \quad (2)$$

$$\text{Recall} = \frac{\text{TP} + \text{FN}}{\text{TP}} \quad (3)$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} + \text{Recall}}{\text{Precision} \times \text{Recall}} \quad (4)$$

where True Positives (TP) refer to images that are correctly predicted to belong to their actual class, while True Negatives (TN) are images correctly identified as not belonging to a class they do not actually belong to. False Positives (FP) occurs when images are wrongly predicted to belong to a class they do not belong to, and False Negatives (FN) happens when images are incorrectly classified as belonging to a different class instead of their correct one.

4.3 AI Models Workflow

4.3.1 Data Preparation for AI Models

Data Preparation for CNN, EfficientNetB0, ResNet50 and YOLOv8 model

The first dataset, contributed by an author R. Lathiya [24], is known as the acne grading dataset. This dataset consists of 999 images, separated into three levels of acne skin which are level 0, level 1, and level 2. The images are stored in the dataset directory and separated into three folders called Level_0, Level_1, and Level_2, as shown in Figure 4.3.1.1. Then, a new Python notebook will be created under Jupyter Notebook in .ipynb format.

The total number of images will be computed by first importing the necessary libraries and declaring a variable called `dataset_path` to access the dataset directory. Two additional variables, `total_image_count` and `level_counts`, will be declared to store the total number of images in the dataset and the number of images in each level, respectively. Then, using a for loop, each level folder in the dataset will be iterated over to calculate the number of images. The code is shown in Figure 4.3.1.2, and the total number of images along with the images in each level are displayed in Figure 4.3.1.3.

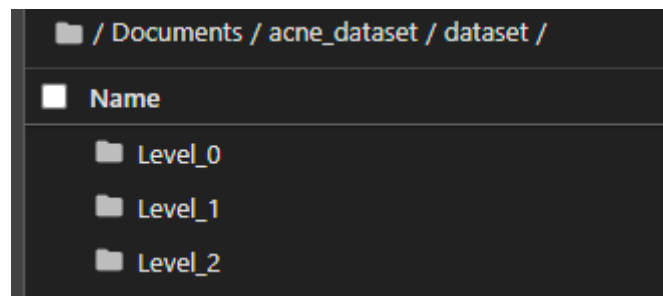


Figure 4.3.1.1 Directory of acne grading dataset

```

import os

dataset_path = "dataset"

total_image_count = 0
level_counts = {}

for category in ["Level_0", "Level_1", "Level_2"]:
    category_path = os.path.join(dataset_path, category)

    images = [f for f in os.listdir(category_path) if f.lower().endswith((".jpg", ".png", ".jpeg"))]

    level_counts[category] = len(images)
    total_image_count += len(images)

print("📁 Image count per level:")
for level, count in level_counts.items():
    print(f"  - {level}: {count} images")

print(f"\n📁 Total images before splitting: {total_image_count}")

```

Figure 4.3.1.2 Code to calculate images in acne grading dataset

```

📁 Image count per level:
  - Level_0: 387 images
  - Level_1: 473 images
  - Level_2: 139 images

📁 Total images before splitting: 999

```

Figure 4.3.1.3 Total number of images in acne grading dataset

Besides, the second dataset was contributed by another author, Alwi [25] and is known as the oily, dry, and normal skin types dataset. This dataset consists of 672 images, separated into three skin condition types: dry, normal, and oily. The images are stored in the dataset directory and separated into three different folders called Dry, Normal, and Oily, as shown in 4.3.1.4. Then, a new Python notebook will be created under Jupyter Notebook in .ipynb format.

The total number of images will be computed by importing the necessary libraries and declaring the variable `dataset_path` to access the dataset directory. Two additional variables, `total_image_count` and `skin_type_counts`, will be declared to store the total number of images in the dataset and the total number of images in each skin type, respectively. Then, using a for loop, each skin type folder in the dataset will be iterated over to calculate the number of images. The code is shown in 4.3.1.5. The total number of images and the total number of images in each skin type are displayed in 4.3.1.6.

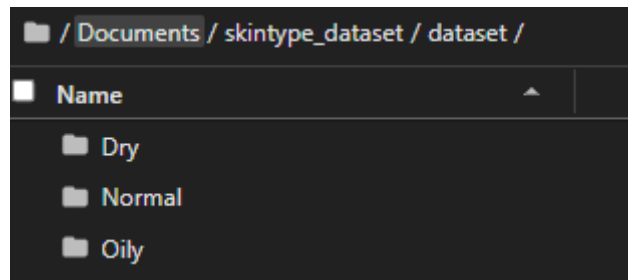


Figure 4.3.1.4 Directory of oily, dry and normal skin type dataset

```
import os

dataset_path = "dataset"

total_image_count = 0
skin_type_counts = {}

for skin_type in ["Oily", "Normal", "Dry"]:
    skin_type_path = os.path.join(dataset_path, skin_type)

    images = [f for f in os.listdir(skin_type_path) if f.lower().endswith((".jpg", ".png", ".jpeg"))]

    skin_type_counts[skin_type] = len(images)
    total_image_count += len(images)

print("📁 Image count per skin type:")
for skin_type, count in skin_type_counts.items():
    print(f"  - {skin_type}: {count} images")

print(f"📁 Total images before splitting: {total_image_count}")
```

Figure 4.3.1.5 Code to calculate images in oily, dry and normal skin types dataset

```
📁 Image count per skin type:
- Oily: 201 images
- Normal: 267 images
- Dry: 204 images

📁 Total images before splitting: 672
```

Figure 4.3.1.6 Total number of images in oily, dry and normal skin types dataset

The images in each dataset will be divided into three categories: training, validation, and testing. For each dataset, the images will be organized into 75% for the training set, 15% for the validation set, and 10% for the testing set. First, the necessary libraries required to perform data preprocessing on both datasets will be imported, as shown in 4.3.8. Then, the `train_path`, `test_path`, and `val_path` variables will be used to store the paths for the training, testing, and validation set images, respectively.

```
import os
import shutil
import random
```

Figure 4.3.1.7 library used in data preparation

```
train_path = "dataset/train"
test_path = "dataset/test"
val_path = "dataset/val"
```

Figure 4.3.1.8 Dataset path for data preparation

New directories will be created using `os.makedirs` if the directories to store images do not already exist. Then, it will read every image file from the corresponding folder with extensions `.jpg`, `.png`, or `.jpeg` for each category. The images will be randomly shuffled to ensure diversity in the datasets. The shuffled images will be organized into three subsets: 75% for training, 15% for validation, and 10% for testing. The images will be copied into their respective directories—`train_path`, `test_path`, and `val_path`—for each category using `shutil.copy`. This structured approach ensures the dataset is properly prepared for model training and evaluation. The categories for the skin acne dataset are `Level_0`, `Level_1`, and `Level_2`, as shown in Figure 4.3.1.9. For the skin type dataset, the categories are `Dry`, `Normal`, and `Oily` for skin type classification, as shown in Figure 4.3.1.10.

```
for folder in [train_path, test_path, val_path]:
    os.makedirs(folder, exist_ok=True)

# Iterate through Level_0, Level_1, Level_2
for category in ["Level_0", "Level_1", "Level_2"]:
    category_path = os.path.join(dataset_path, category)

    images = [f for f in os.listdir(category_path) if f.lower().endswith((".jpg", ".png", ".jpeg"))]

    random.shuffle(images)

    # Split dataset: 75% train, 15% validation, 10% test
    train_split = int(0.75 * len(images))
    remaining_images = images[train_split:] # Remaining 25%

    # Split remaining into 15% validation, 10% test
    val_split = int(0.60 * len(remaining_images))
    val_images = remaining_images[:val_split]
    test_images = remaining_images[val_split:]

    for split_path in [train_path, test_path, val_path]:
        os.makedirs(os.path.join(split_path, category), exist_ok=True)

    # Copy images to respective folders
    for img in images[:train_split]:
        shutil.copy(os.path.join(category_path, img), os.path.join(train_path, category, img))
    for img in val_images:
        shutil.copy(os.path.join(category_path, img), os.path.join(val_path, category, img))
    for img in test_images:
        shutil.copy(os.path.join(category_path, img), os.path.join(test_path, category, img))
```

Figure 4.3.1.9 Code to split the skin acne dataset

```

for folder in [train_path, test_path, val_path]:
    os.makedirs(folder, exist_ok=True)

# Iterate through Level_0, Level_1, Level_2
for category in ["Oily", "Normal", "Dry"]:
    category_path = os.path.join(dataset_path, category)

    images = [f for f in os.listdir(category_path) if f.lower().endswith((".jpg", ".png", ".jpeg"))]

    random.shuffle(images)

    # Split dataset: 75% train, 15% validation, 10% test
    train_split = int(0.75 * len(images))
    remaining_images = images[train_split:] # Remaining 25%

    # Split remaining into 15% validation, 10% test
    val_split = int(0.60 * len(remaining_images))
    val_images = remaining_images[:val_split]
    test_images = remaining_images[val_split:]

    for split_path in [train_path, test_path, val_path]:
        os.makedirs(os.path.join(split_path, category), exist_ok=True)

    # Copy images to respective folders
    for img in images[:train_split]:
        shutil.copy(os.path.join(category_path, img), os.path.join(train_path, category, img))
    for img in val_images:
        shutil.copy(os.path.join(category_path, img), os.path.join(val_path, category, img))
    for img in test_images:
        shutil.copy(os.path.join(category_path, img), os.path.join(test_path, category, img))

```

Figure 4.3.1.10 Code to split skin type dataset

The dataset is split into 75% training, 15% validation, and 10% testing. For the skin acne dataset, there are 748 training images, 150 validation images, and 101 testing images, totalling 999 images across Level 0, Level 1, and Level 2, as shown in Table 4.3.1.1. For the skin type dataset, there are 503 training images, 100 validation images, and 69 testing images, totalling 672 images categorized as Dry, Normal, and Oily, as shown in Table 4.3.1.2.

Table 4.3.1.1 Image count per level in skin acne dataset

	Training set (75%)	Validation set (15%)	Test set (10%)
Level 0	290	58	39
Level 1	354	71	48
Level 2	104	21	14
Total	748	150	101

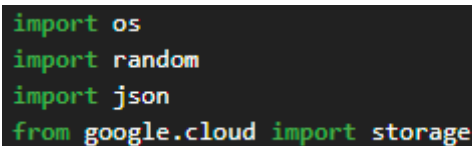
Table 4.3.1.2 Image count per skin type in skin type dataset

	Training Set (75%)	Validation Set (15%)	Test Set (10%)
Dry	153	30	21
Normal	200	40	27
Oily	150	30	21
Total	503	100	69

Data Preparation for GPT Assistant-Based

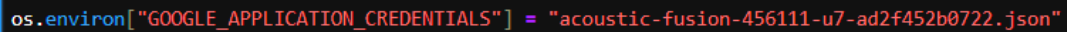
In data preparation for the GPT Assistant-Based model, there is an additional process compared to other models like classic CNN, EfficientNetB0, ResNet50, and YOLOv8. This is because the GPT Assistant-Based model will store the images in Google Cloud Storage to ensure easier retrieval when interacting with the GPT model, and to reduce the tokens spent on each conversation.

First, we need to import the necessary libraries, as shown in Figure 4.3.11 and then set the environment variable 'GOOGLE_APPLICATION_CREDENTIALS' to the .json file that contains the service account credentials, which authenticate the user when accessing Google Cloud services, as shown in Figure 4.3.12.



```
import os
import random
import json
from google.cloud import storage
```

Figure 4.3.11 library used by GPT assistant-based in data preparation



```
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = "acoustic-fusion-456111-u7-ad2f452b0722.json"
```

Figure 4.3.12 key to access google cloud service

After setting up the environment, the paths for the locally stored training and test sets will be declared. Next, define the base_url for the Google Cloud bucket created in the Google Cloud Console and specify the bucket_name. For the skin acne dataset, bucket_name will be dataset_skinacne, while for the skin type dataset, it will be dataset_skinType. We will also define train_images_per_level to store the number of training images for each level. If needed, we can define test_images_per_level to specify the number of test images to be used. Figure 4.3.1.13 shows the code used in the skin acne dataset preparation.

```

train_path = "./dataset/train/"
test_path = "./dataset/test/" # Path to your test set
base_url = "https://storage.googleapis.com/dataset_skinacne/"
bucket_name = "dataset_skinacne"
train_images_per_level = 5
test_images_per_level = 3

```

Figure 4.3.1.13 variables used in skin acne data preparation

After defining the required variables, a function called `random_select_images` will be defined to select a specific number of images from the local folder. It will take two parameters: `dataset_path` and `num_images_per_level`. For example, if `train_path` and 3 are passed as arguments into the function, it will randomly select 3 images from each level in the `train_path` and return them to the function call. Figure 4.3.1.14 shows the structure of the function.

```

def random_select_images(dataset_path, num_images_per_level):
    selected_images = {}
    for level in os.listdir(dataset_path):
        level_folder = os.path.join(dataset_path, level)
        if os.path.isdir(level_folder):
            images = [f for f in os.listdir(level_folder) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
            if images:
                selected_images[level] = random.sample(images, min(num_images_per_level, len(images)))
    return selected_images

```

Figure 4.3.1.14 function to random select images

Another function called `get_all_images_from_test_set` is defined to collect all the images from the test set by using `test_set_path` as a parameter. Figure 4.3.1.15 shows the structure of the function.

```

def get_all_images_from_test_set(test_set_path):
    all_images = {}
    for level in os.listdir(test_set_path):
        level_folder = os.path.join(test_set_path, level)
        if os.path.isdir(level_folder):
            images = [f for f in os.listdir(level_folder) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
            if images:
                all_images[level] = images
    return all_images

```

Figure 4.3.1.15 function to select all images

After collecting all images from local storage, the last function is used to generate JSON list consists of URLs of images with their corresponding levels. This function called

`generate_image_links_from_cloud`, will interact with the specified Google Cloud Storage bucket to construct valid URLs based on the provided base URL and selected images. Each entry in the JSON list includes the `image_url` and its associated level. Figure 4.3.1.16 illustrates the structure of this function.

```
def generate_image_links_from_cloud(base_url, bucket_name, selected_images):
    client = storage.Client()
    bucket = client.bucket(bucket_name)
    json_links = []
    all_blobs = [blob.name for blob in bucket.list_blobs()]
    all_blobs_lower = [b.lower() for b in all_blobs]

    for level, images in selected_images.items():
        for image in images:
            blob_name = f"{level}/{image}"
            if blob_name.lower() in all_blobs_lower:
                image_url = f"{base_url}{level}/{image}"
                json_links.append({
                    "image_url": image_url,
                    "level": level
                })
            else:
                print(f"X Not found in bucket: {blob_name}")
    return json_links
```

Figure 4.3.1.16 function to generate JSON list

After declaring all the required functions, the images from the training set will be selected randomly and stored in `train_images`. Then, `train_images` will be used to generate the image URLs and store them in `train_images_data`, as shown in Figure 4.3.1.17. It will also store all the test images in `test_images`, generate their URLs, and save them into `test_images_data`, as shown in Figure 4.3.1.18.

```
train_images = random_select_images(train_path, train_images_per_level)
train_images_data = generate_image_links_from_cloud(base_url, bucket_name, train_images)
train_images_data
```

Figure 4.3.1.17 json list generate for training set

```
test_images = get_all_images_from_test_set(test_path)
test_images_data = generate_image_links_from_cloud(base_url, bucket_name, all_test_images)
test_images_data
```

Figure 4.3.1.18 json list generate for test set

4.3.2 Data Preprocessing for AI Models

Data Preprocessing for classic CNN model and EfficientNetB0 model

After splitting the images into the training set, validation set, and testing set, we will proceed to preprocess the images. First, we will import the libraries required for data preprocessing for the classic CNN model and EfficientNetB0 model, as shown in Figure 4.3.2.1. Then, we will define the `image_size`, which will be used to resize the images to the same dimensions to fit the model. Next, we will define the `batch_size` for the model, as shown in Figure 4.3.2.2.

```
import numpy as np
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Figure 4.3.2.1 library import for data preprocessing

```
image_size=(224,224)
batch_size = 32
```

Figure 4.3.2.2 variables declared

Besides, we will define a function called `preprocess_image` to resize the image to 224×224 using the interpolation method called `cv2.INTER_CUBIC`, which smooths and sharpens the image to produce better results. Then, we will use the `ImageDataGenerator` class imported from the `tensorflow.keras.preprocessing.image` library to call the `preprocess_image` function and perform data augmentation. The training set will undergo data augmentation with a rotation range of 20, zoom range of 0.2, and random horizontal flipping. The data augmentation will be applied randomly to each image during training to increase dataset diversity. Meanwhile, the validation and test sets will only use the `preprocess_image` function without data augmentation. Figure 4.3.2.3 shows the code that performs image resizing and data augmentation.

```
def preprocess_image(img):
    img = np.array(img, dtype=np.float32)
    img = cv2.resize(img, image_size, interpolation=cv2.INTER_CUBIC)
    return img

# Define ImageDataGenerator (without additional augmentation)
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_image,
                                    rotation_range=20,
                                    zoom_range=0.2,
                                    horizontal_flip=True)

valid_datagen = ImageDataGenerator(preprocessing_function=preprocess_image)
test_datagen = ImageDataGenerator(preprocessing_function=preprocess_image)
```

Figure 4.3.2.3 Code to perform image resizing and data augmentation

Finally, we will load images from the training, validation, and testing directories using the `flow_from_directory` function, which is part of Keras' image data preprocessing utilities. For the datasets which are training, validation, and test sets—the images will be read from the specified directory, resized to a specific size defined by `image_size`, and grouped into batches of size `batch_size`. The `class_mode='categorical'` argument specifies that the labels will be one-hot encoded, which is suitable for multi-class classification tasks.

```
train_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

valid_data = valid_datagen.flow_from_directory(
    valid_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)
```

Figure 4.3.2.4 Load and preprocess image data from directory for model training

Data Preprocessing for ResNet50 model

For the ResNet50 model, the necessary libraries will be imported as shown in Figure 4.3.2.5. Then, two transformation pipelines will be defined: one for the training set and another for the validation and testing sets. Both pipelines will resize the images to 224 x 224. The training pipeline will also perform data augmentation, which includes

horizontal flipping, random rotation by 20 degrees, and random zooming. Figure 4.3.2.6 shows the two transformation pipelines for the ResNet50 model.

```
import torch
from torch.utils.data import DataLoader
from torchvision import datasets
import torchvision.transforms as v2
```

Figure 4.3.2.5 library used in ResNet50 data preprocessing

```
transform = v2.Compose([
    v2.ToImage(),
    v2.RandomResizedCrop(size=(224, 224), antialias=True),
    v2.RandomHorizontalFlip(p=0.5),
    v2.RandomRotation(degrees=20),
    v2.RandomAffine(degrees=0, scale=(0.8, 1.2)),
    v2.ToDtype(torch.float32, scale=True),
])

transform2 = v2.Compose([
    v2.ToImage(),
    v2.RandomResizedCrop(size=(224, 224), antialias=True),
    v2.ToDtype(torch.float32, scale=True),
])
```

Figure 4.3.2.6 transformation pipelines in ResNet50 model

In Figure 4.3.2.7, the training, validation, and test sets will be loaded using "ImageFolder". Each dataset will apply the transformation pipeline, which will resize the images for all three datasets and apply data augmentation only to the training set. Then, a "DataLoader" will be created for batching, shuffling, and loading data in parallel.

```
# Load dataset
base_path = './dataset'
train_dataset = datasets.ImageFolder(root=f'{base_path}/train/', transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=4)

val_dataset = datasets.ImageFolder(root=f'{base_path}/val/', transform=transform)
val_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=4)

test_dataset = datasets.ImageFolder(root=f'{base_path}/test/', transform=transform)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=4)

# Get number of classes
num_classes = len(train_dataset.classes)
```

Figure 4.3.2.7 dataset loading for ResNet50 model

4.3.3 Model Training for AI Models

Model Training for Classic CNN model

In model training for the classic CNN model, the necessary libraries will be imported as shown in Figure 4.3.3.1. The classic CNN model consists of three layers of convolutional blocks. The first layer includes 32 filters with a size of 3×3 and uses 'same' padding to maintain the input size. It is followed by a ReLU activation function and a 2×2 max-pooling layer to reduce the spatial size. The next two convolutional layers increase the number of filters to 64 and 128, respectively.

After the three layers of convolutional blocks, the model will use a Flatten layer to convert the multidimensional output into one dimension. This will be passed into a fully connected Dense layer with 128 neurons and a "ReLU" activation function. It will then be followed by a Dropout rate of 0.3, and the final output layer will be a Dense layer with 3 neurons, representing the three classes for this model.

The model is compiled using the Adam optimizer with a learning rate of 0.001 and the categorical cross-entropy loss function, which is ideal for handling multi-class classification problems with one-hot encoded labels. The structure of the model is illustrated in Figure 4.3.3.2.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

Figure 4.3.3.1 library used by classic cnn model in model training

```

model = Sequential([
    # First Convolution Block
    Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(image_size[0], image_size[1], 3)),
    MaxPooling2D((2, 2)),

    # Second Convolution Block
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    MaxPooling2D((2, 2)),

    # Third Convolution Block
    Conv2D(128, (3, 3), padding='same', activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()

```

Figure 4.3.3.2 code to build classic CNN model

The model will then be trained using the `fit()` method, which will perform training on the `train_data` with the `validation_data` for validation purposes. The number of epochs for the model training process is defined as 20. Figure 4.3.3.3 shows the process of running the model training for the classic CNN model.

```

history = model.fit(train_data,
                    validation_data=valid_data,
                    epochs=20)

```

Figure 4.3.3.3 code to train classic CNN model

Model Training for ResNet50 model

In the model training stage for the ResNet50 model, a pretrained ResNet50 model will be loaded from the PyTorch model hub. The final fully connected layer, `model.fc`, will be replaced with a custom classifier by setting the dropout rate to 0.3, as shown in Figure 4.3.3.4.

```

model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', weights='ResNet50_Weights.DEFAULT')
num_classes = 3 # Change according to your dataset
model.fc = nn.Sequential(
    nn.Dropout(p=0.3), # Prevent overfitting
    nn.Linear(model.fc.in_features, num_classes)
)

```

Figure 4.3.3.4 load ResNet50 model

After loading and modifying the model, the `train_accuracies`, `val_accuracies`, `train_losses`, and `val_losses` arrays will be declared to store the data during the training process, as shown in Figure 4.3.3.5.

```
train_accuracies = []
val_accuracies = []
train_losses = []
val_losses = []
```

Figure 4.3.3.5 array to store accuracy and loss

Before starting the training process, the loss function for multiclass classification is defined and stored in `criterion`, while the optimizer is defined with a learning rate of 0.001 and stored in `optimizer`. The number of epochs is defined as 20 and stored in `epochs`.

During the training process, the loss is calculated in `criterion(output, labels)` to compare the predicted and actual labels. Then, the model weights are updated using `optimizer.step()`, as the model weights help the model learn from its mistakes and improve in the next iteration, as shown in Figure 4.3.3.6. In the validation process, the model's performance is evaluated on unseen data, but the model weights are not updated, as shown in Figure 4.3.3.7.

```
# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 20
for epoch in range(epochs):
    # ----- Training Phase -----
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    train_accuracy = 100 * correct / total
    train_loss = running_loss / len(train_loader)
```

Figure 4.3.3.6 training process for ResNet50 model

```
# ----- Validation Phase -----
model.eval()
val_running_loss = 0.0
val_correct = 0
val_total = 0

with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)

        val_running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        val_correct += (predicted == labels).sum().item()
        val_total += labels.size(0)

val_accuracy = 100 * val_correct / val_total
val_loss = val_running_loss / len(val_loader)

# Store for visualization
train_accuracies.append(train_accuracy)
val_accuracies.append(val_accuracy)
train_losses.append(train_loss)
val_losses.append(val_loss)
```

Figure 4.3.3.7 validation process for ResNet50 model

Model Training for EfficientNetB0 model

In the model training stage for EfficientNetB0, EfficientNetB0 without the top layer is loaded, and the input size of the images is specified. Then, the base model layers are frozen, and only the custom layers are trained. The top layer is then defined and added to the model. It begins with a Global Average Pooling layer that reduces the spatial dimensions, followed by two dropout layers, each with a rate of 0.3, and a dense layer with 256 neurons and "ReLU" activation added between the dropout layers. The final output layer uses a SoftMax activation function and provides output class probabilities. The completed model is then optimized using the Adam optimizer with a learning rate of 0.001 and the categorical cross-entropy loss function, which is suitable for multi-class problems where labels are one-hot encoded. Accuracy is used as the evaluation metric inside the model.

```
# Load EfficientNetB0 without top layers
base_model = EfficientNetB0(weights="imagenet", include_top=False, input_shape=(image_size[0], image_size[1], 3))

# Freeze base model layers
base_model.trainable = False

# Add custom classifier on top
x = GlobalAveragePooling2D()(base_model.output)
x = Dropout(0.3)(x)
x = Dense(256, activation="relu")(x)
x = Dropout(0.3)(x)
output_layer = Dense(train_data.num_classes, activation="softmax")(x) # Multi-class classification

# Create final model
model = Model(inputs=base_model.input, outputs=output_layer)

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss="categorical_crossentropy",
              metrics=["accuracy"])

model.summary()
```

Figure 4.3.3.8 code to build EfficientNetB0 model

The model will now start training using the training data and validating with the validation data. The number of epochs is set to 20 for the EfficientNetB0 model.

```
history = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=20
)
```

Figure 4.3.3.9 code to start EfficientNetB0 learning process

Model Training for YoLov8 model

When we start training the dataset using the YOLOv8 model, we will need to import the YOLO class from the ultralytics library, as shown in Figure 4.3.3.10. Then, we will need to load the pretrained YOLO model into a variable called model, as shown in Figure 4.3.3.11.

```
from ultralytics import YOLO
```

Figure 4.3.3.10 import ultralytics library

```
model = YOLO("yolov8n-cls.pt")
```

Figure 4.3.3.11 load pretrained model

After loading the model, we will start training the model by calling the train() method. Inside the train() method, we will declare the path of the folder that contains the dataset, named "dataset", set the number of epochs to 20, and specify the image size as 225. The declaration of the train() method is shown in Figure 4.3.3.12.

```
model.train(data="dataset", epochs=20, imgsz=224)
```

Figure 4.3.3.12 code to start model training process

Model Training for GPT Assistant-Based

In model training for GPT Assistant-Based, the necessary library should be imported to create the client object by passing on an API key. This OpenAI API key is used to authenticate and authorize users to access the OpenAI API platform. Once access is granted, users can interact with GPT to perform various tasks. Figure 4.3.3.13 shows that the API key has been declared and saved into the client object.

```
import openai
from openai import OpenAI
client = openai.Client(api_key="sk-proj-h8-VMBNzwn8hAqz8BNZvurcu8Pj18Qq_sHSPJMGk77rwdclFA8_x_0gdy83s4a02v8jQT381skF3x_8FQ1owS117Xg0ddcl-K2y3GV7-Pxt1pw_es18885kcTVC10Vvg21DaFwY5aa-62ZwCKIA")
```

Figure 4.3.3.13 API Key Declaration and Client Object Creation

After storing the image URLs and their respective conditions into the messages list, a new thread will be created using the OpenAI client object, which serves as a container that stores a series of messages. Then, it will loop through the messages, and each message will be stored in the thread, which will include a `thread_id` referring to a single session, as shown in Figure 4.3.3.14.

```
assistant = client.beta.assistants.create(
    name="Image Level Classifier",
    instructions="You classify images into Level_0, Level_1, or Level_2 based on training examples.",
    model="gpt-4o",
)
```

Figure 4.3.3.14 Assistant used by skin acne dataset

```
assistant = client.beta.assistants.create(
    name="Image Level Classifier",
    instructions="You classify images into Dry, Normal, or Oily based on training examples.",
    model="gpt-4o",
)
```

Figure 4.3.3.15 Assistant used by skin type dataset

An empty list called `messages` will be used to store the formatted text messages. It will loop through `train_images_data`, which stores the JSON list consisting of the image URLs and their respective conditions. The condition of the images refers to the skin acne level for the skin acne dataset and the skin type for the skin type dataset, as shown in Figure 4.3.3.16 and Figure 4.3.3.17.

```
messages = [] # initialize as a list, not a string
for image in train_images_data:
    msg = f"This image has URL: {image['image_url']} and its acne level is {image['level']}."
    messages.append(msg)
```

Figure 4.3.3.16 store image information for skin acne dataset

```
messages = [] # initialize as a list, not a string
for image in train_images_data:
    msg = f"This image has URL: {image['image_url']} and its skin type is {image['level']}."
    messages.append(msg)
```

Figure 4.3.3.17 store image information for skin type dataset

After storing the image URLs and their respective conditions into the messages list, a new thread will be created using the OpenAI client object, which serves as a container that stores a series of messages. Then, it will loop through the messages, and each

message will be stored in the thread, which will include a `thread_id` referring to a single session, as shown in Figure 4.3.3.17.

```
thread = client.beta.threads.create()
for item in messages:
    client.beta.threads.messages.create(
        thread_id=thread.id,
        role="user",
        content=item
    )
```

Figure 4.3.3.17 initializing thread and storing training messages

4.3.4 Model Evaluation for AI Models

Model Evaluation for Classic CNN model

In the model evaluation for the classic CNN model, the necessary libraries will be imported as shown in Figure 4.3.4.1. Then, `y_true` will retrieve the actual class labels of the `test_data`, while `y_pred` will store the predicted class labels for each image by running the trained CNN model on the `test_data`, as shown in Figure 4.3.4.2.

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

Figure 4.3.4.1 libraries used in evaluation stage

```
y_true = test_data.classes
y_pred = np.argmax(model.predict(test_data), axis=1)
```

Figure 4.3.4.2 Extraction of actual and predicted labels from test data

After obtaining `y_true` and `y_pred`, a confusion matrix will be generated to visualize the performance of the models across different skin conditions, such as skin acne levels for the skin acne dataset and skin types for the skin type dataset. Figure 4.3.4.3 and Figure 4.3.4.4 show the code that generates the confusion matrix. Then, a classification report will also be generated to provide a detailed performance summary, including precision, recall, and F1-score. Figure 4.3.4.5 shows the code to generate the classification report.

```
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for classic CNN Skin Acne Classification")
plt.show()
```

Figure 4.3.4.3 Confusion matrix for Classic CNN skin acne classification

```
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for classic CNN Skin Type Classification")
plt.show()
```

Figure 4.3.4.4 Confusion matrix for Classic CNN model skin type classification

```
print(classification_report(y_true, y_pred, target_names=class_labels))
```

Figure 4.3.4.5 Classification report for classic CNN model

Model Evaluation for ResNet50 model

In the model evaluation for the ResNet50 model, the necessary libraries are imported as shown in Figure 4.3.4.6. Before displaying the confusion matrix, the class_names for each dataset will be declared. Two arrays, y_true and y_pred, will be declared to store the data used for generating the confusion matrix. The model will now perform evaluations on the test set and store the results into y_true and y_pred, as shown in Figure 4.3.4.7.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
```

Figure 4.3.4.6 library used in ResNet50 evaluation stage

```
model.eval()
y_true = []
y_pred = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)

        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())
```

Figure 4.3.4.7 code to evaluate test set using ResNet model

From Figure 4.3.4.8, the class_names for the skin acne dataset are defined as Level_0, Level_1, and Level_2. Meanwhile, the class_names for the skin type dataset are defined as Dry, Normal, and Oily, as shown in Figure 4.3.4.9.

```
class_names = ['Level_0', 'Level_1', 'Level_2']
```

Figure 4.3.4.8 class name for skin acne dataset

```
class_names = ['Dry', 'Normal', 'Oily']
```

Figure 4.3.4.9 class name for skin type dataset

The confusion matrix is visualized for both the skin acne classification model and the skin type classification model, as shown in Figure 4.3.4.10 and Figure 4.3.4.11 respectively. Finally, the classification report is generated for both models, as shown in Figure 4.3.4.12.

```
# Generate confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="g", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for ResNet50 Skin Acne Classification")
plt.show()
```

Figure 4.3.4.10 Confusion matrix for ResNet50 skin acne classification

```
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="g", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for ResNet50 Skin Type Classification")
plt.show()
```

Figure 4.3.4.11 Confusion matrix for ResNet50 skin type classification

```
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=class_names, digits=3))
```

Figure 4.3.4.12 Classification report for ResNet50

Model Evaluation for EfficientNetB0 model

In the model evaluation for the EfficientNetB0 model, the required libraries will be imported, as shown in Figure 4.3.4.13. The model predicts class probabilities for the test dataset using model.predict, and the highest probability class for each sample is

selected using `np.argmax` to obtain the final predicted class labels, `y_pred`. The true class labels are retrieved from `test_data.classes` and stored in `y_true`. Using the computed information from `y_pred` and `y_true`, a confusion matrix is created using `confusion_matrix(y_true, y_pred)`, as shown in Figure 4.3.4.14.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
```

Figure 4.3.4.13 library used in EfficientNetB0 model evaluation stage

```
y_pred_probs = model.predict(test_data, batch_size=32, verbose=1)
y_pred = np.argmax(y_pred_probs, axis=1)

y_true = test_data.classes

cm = confusion_matrix(y_true, y_pred)
```

Figure 4.3.4.14 code to evaluate the test data using EfficientNetB0

The confusion matrix is visualized for both the skin acne classification model and the skin type classification model, as shown in Figure 4.3.4.15 and Figure 4.3.4.16, respectively. Finally, the classification report is generated for both models, as shown in Figure 4.3.4.17.

```
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="g", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for EfficientNet-B0 Skin Acne Classification")
plt.tight_layout()
plt.show()
```

Figure 4.3.4.15 Confusion matrix for EfficientNetB0 skin acne classification

```
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="g", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for EfficientNet-B0 Skin Type Classification")
plt.tight_layout()
plt.show()
```

Figure 4.3.4.16 Confusion matrix for EfficientNetB0 skin type classification

```
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=class_names, digits=3))
```

Figure 4.3.4.17 classification report for EfficientNetB0

Model Evaluation for YoLov8 model

In model evaluation for the YOLOv8 model, the necessary libraries are imported to create the evaluation graph for the model, as shown in Figure 4.3.4.18. The trained model is then evaluated on the test dataset by specifying the dataset directory and setting verbose=True to ensure that the model displays evaluation metrics such as accuracy and loss. The code for testing the model on the test data is shown in Figure 4.3.4.19.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Figure 4.3.4.18 library used in YOLOv8 model evaluation stage

```
results = model.val(split="test", verbose=True)
```

Figure 4.3.4.19 evaluate the test dataset

Before displaying the confusion matrix, the class_names for each dataset will be declared. From Figure 4.3.4.20, the class_names for the skin acne dataset are defined as Level_0, Level_1, and Level_2. Meanwhile, the class_names for the skin type dataset are defined as Dry, Normal, and Oily, as shown in Figure 4.3.4.21.

```
class_names = ['Level_0', 'Level_1', 'Level_2']
```

Figure 4.3.4.20 class name for skin acne dataset

```
class_names = ['Dry', 'Normal', 'Oily']
```

Figure 4.3.4.21 class name for skin type dataset

The confusion matrix is visualized for both the skin acne classification model and the skin type classification model, as shown in Figure 4.3.4.22 and Figure 4.3.4.23 respectively. Finally, the precision, recall, and F1-score will be displayed for both models, as shown in Figure 4.3.4.24.

```

cm = np.array(results.confusion_matrix.matrix).T # Transpose it!

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="g", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for YOLOv8 skin acne Classification")
plt.tight_layout()
plt.show()

```

Figure 4.3.4.22 Confusion matrix for YOLOv8 skin acne classification

```

cm = np.array(results.confusion_matrix.matrix).T # Transpose it!

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="g", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix for YOLOv8 Skin Type Classification")
plt.tight_layout()
plt.show()

```

Figure 4.4.4.23 Confusion matrix for YOLOv8 skin type classification

```

for i in range(cm.shape[0]):
    TP = cm[i, i]
    FP = cm[:, i].sum() - TP
    FN = cm[i, :].sum() - TP

    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    f1 = 2 * (precision * recall) / (precision + recall)

    # Print results for the current class
    print(f"Class: {class_names[i]}")
    print(f" Precision: {precision:.3f}")
    print(f" Recall: {recall:.3f}")
    print(f" F1-Score: {f1:.3f}")
    print("-" * 30)

```

Figure 4.4.4.24 code to display precision, recall and F1-score for each class

Model Evaluation for GPT Assistant-Based

After storing all the messages into the thread created during model training, the next step is to create a function called `send_testing_images_to_chatgpt`. This function will consist of a few main parts, as shown in Figure 4.3.4.25 Figure 4.3.4.26 and Figure 4.3.4.27. The function will loop through each test image, with each iteration referring to the image as "image".

CHAPTER 4

In Figure 4.3.4.25, the content that instructs the GPT assistant to predict on the test image will be created in the existing thread. It will consist of text that asks the assistant to classify the image into Level 0, Level 1, or Level 2 for the skin acne dataset. For the skin type dataset, the instruction will be to classify the image into Dry, Normal, or Oily.

```
client.beta.threads.messages.create(
    thread_id=thread_id,
    role="user",
    content=[
        {
            "type": "text",
            "text": "Analyze this image and respond ONLY with Level_0, Level_1, or Level_2."
        },
        {
            "type": "image_url",
            "image_url": {
                "url": image['image_url']
            }
        }
    ]
)
```

Figure 4.3.4.25 Sending Image and Prompt Message to GPT Assistant for Classification

In Figure 4.3.4.26, a new run is initiated with the assistant to process the previously sent message. Then, it will enter an infinite while loop that continuously checks the status of the run. The loop will break once the run is either completed, failed, or cancelled. This ensures that the process will not proceed to the next step until a valid response or termination condition is reached.

```
run = client.beta.threads.runs.create(
    thread_id=thread_id,
    assistant_id=assistant_id,
    instructions="Analyze the image and respond ONLY with Level_0, Level_1, or Level_2."
)

# 3. Wait for completion
while True:
    run_status = client.beta.threads.runs.retrieve(
        thread_id=thread_id,
        run_id=run.id
    )

    if run_status.status == "completed":
        break
    elif run_status.status in ["failed", "cancelled"]:
        print(f"Run failed for image {image['image_url']}")
        predicted_levels.append(None)
        continue
    time.sleep(1) # Avoid rate limiting
```

Figure 4.3.4.26 Initiating GPT Assistant Run and Monitoring Processing Status

In Figure 4.3.4.27, the most recent message will be retrieved from the thread, which contains the result of the prediction from the assistant. The message will be extracted, and the predicted value will be stored in the predicted_levels list. The predicted and actual values for the respective image will then be printed.

```

messages = client.beta.threads.messages.list(
    thread_id=thread_id,
    order="desc",
    limit=1
)

if messages.data:
    response = messages.data[0].content[0].text.value
    predicted_level = response.strip()
    predicted_levels.append(predicted_level)
    print(f"Image: {image['image_url']} | Predicted: {predicted_level} | Actual: {image['level']}")
else:
    predicted_levels.append(None)

```

Figure 4.3.4.27 Retrieving and Storing Predicted Acne Levels from GPT Assistant Response

In Figure 4.3.4.28, the function call will pass the `test_images_data` in JSON list format, the thread ID, the assistant ID created during the training stage, and the token limit. The function will process each test image, return a list of predicted values, and store them inside the `predicted_levels` variable.

```

predicted_levels = send_testing_images_to_chatgpt(
    testing_data=test_images_data,
    thread_id=thread.id,
    assistant_id=assistant.id,
    token_limit=200
)

```

Figure 4.3.4.28 Function call to process test images using GPT assistant

4.4 Recommendation Module

In the recommendation module, content-filtering recommendation approach will be used by filtering skincare products that match skin of user based on the skin care products after using features. The dataset used is from kaggle.com which was contributed by an author, K. R. Hasan [34] which is collected from the website called skinsort.com. We will first proceed with preprocessing the data which will involve the library known as pandas as shown in Figure 4.4.1.

```
import pandas as pd
```

Figure 4.4.1 library import recommendation module

We will then be using the read_csv commands from the pandas library to read the csv file and store it inside the variable called df as shown in Figure 4.4.2.

```
df = pd.read_csv("datasheet.csv")
```

Figure 4.4.2 code to import the csv file

We will then execute df to view the rows and columns inside the dataset. The dataset consists of 19049 skincare products. There are six columns that exist inside the dataset which are brand, name, type, country, ingredients, and afterUse. The Figure 4.4.3 shows the content of the skincare dataset imported from the csv file.

	brand	name	type	country	Ingredients	afterUse
0	The Ordinary	Glycolic Acid 7% Toning Solution	Toner	Canada	Water,Glycolic Acid,Rosa Damascena Flower Wate...	Good For Oily Skin,Skin Texture,Reduces Large ...
1	La Roche-Posay	Toleriane Hydrating Gentle Face Cleanser	Face Cleanser	France	Water,Glycerin,Pentaerythrityl Tetraethylhexan...	Good For Oily Skin,Redness Reducing,Reduces It...
2	The Ordinary	Niacinamide 10% + Zinc 1%	Facial Treatment	Canada	Water,Niacinamide,Pentylene Glycol,Zinc PCA,DL...	Good For Oily Skin,Redness Reducing,Acne Fight...
3	Youth To The People	Superfood Antioxidant Cleanser	Face Cleanser	United States	Water,Cocamidopropyl Hydroxysultaine,Sodium Co...	Redness Reducing,Reduces Irritation,Skin Textu...
4	COSRX	Low pH Good Morning Gel Cleanser	Face Cleanser	South Korea	Water,Cocamidopropyl Betaine,Sodium Lauroyl Me...	Good For Oily Skin,Reduces Irritation,Reduces ...
...
19045	CeraVe	Hydrating Facial Cleanser	Face Cleanser	Canada	Water,Glycerin,Cetearyl Alcohol,Peg-40 Stearat...	Redness Reducing,Anti-Aging,Scar Healing,Bright...
19046	Beauty of Joseon	Ginseng Essence Water	Essence	South Korea	Panax Ginseng Root Water,Butylene Glycol,Glyce...	Good For Oily Skin,Redness Reducing,Reduces It...
19047	CeraVe	PM Facial Moisturizing Lotion	Night Moisturizer	Canada	Water,Glycerin,Caprylic/Capric Triglyceride,Ni...	Good For Oily Skin,Redness Reducing,Anti-Aging...
19048	The Ordinary	AHA 30% + BHA 2% Peeling Solution	Facial Treatment	Canada	Glycolic Acid,Water,Aloe Barbadensis Leaf Wate...	Good For Oily Skin,Reduces Irritation,Skin Tex...
19049	COSRX	Advanced Snail 96 Mucin Power Essence	Essence	South Korea	Snail Secretion Filtrate,Betaine,Butylene Glyc...	Redness Reducing

Figure 4.4.3 content of csv file

After reading the file, we will proceed to remove the extra spaces from string data, but we will keep non-string values unchanged as shown in Figure 4.4.4.

```
df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
```

Figure 4.4.4 code to remove extra spaces

We replaced empty strings such as "" and "N/A" in the dataset with NaN using the predefined pd.NA function from the pandas library, as shown in Figure 4.4.5. After that, we proceeded to drop the rows containing NaN values, as shown in Figure 4.4.6.

```
# Replace "N/A" and empty strings with NaN
df.replace(["", "N/A"], pd.NA, inplace=True)
```

Figure 4.4.5 code to replace the empty string with NaN

```
# Drop rows with any NaN (i.e., empty) cells
df.dropna(inplace=True)
```

Figure 4.4.6 code to drop the NaN cells

Then, we double-checked to ensure that there were no rows containing empty cells, as shown in Figure 4.4.7.

```
# Check if there are any missing (empty) values
if df.isna().any().any():
    print("⚠️ There are still empty cells in the CSV.")
else:
    print("✅ No empty cells found in the CSV.")
```

Figure 4.4.7 code to check NaN cells

Then, we checked the types of skincare products in the database, as shown in Figure 4.4.8. The products in the dataset include toner, face cleanser, facial treatment, serum, general moisturizer, sunscreen, exfoliator, face makeup, bath & body, makeup

remover, day moisturizer, other haircare, shampoo, fragrance, hand care, conditioner, lip moisturizer, eye moisturizer, eye makeup, tanning, wet mask, emulsion, makeup applicator, night moisturizer, sheet mask, lip makeup, oil, essence, cheek makeup, overnight mask, nail care, lip mask, eye mask, other, tool, and false eyelash, as shown in Figure 4.4.9.

```
df['type'].unique()
```

Figure 4.4.8 code to check unique type of skincare products

```
array(['Toner', 'Face Cleanser', 'Facial Treatment', 'Serum',  
      'General Moisturizer', 'Sunscreen', 'Exfoliator', 'Face Makeup',  
      'Bath & Body', 'Makeup Remover', 'Day Moisturizer',  
      'Other Haircare', 'Shampoo', 'Fragrance', 'Hand Care',  
      'Conditioner', 'Lip Moisturizer', 'Eye Moisturizer', 'Eye Makeup',  
      'Tanning', 'Wet Mask', 'Emulsion', 'Makeup Applicator',  
      'Night Moisturizer', 'Sheet Mask', 'Lip Makeup', 'Oil', 'Essence',  
      'Cheek Makeup', 'Overnight Mask', 'Nail Care', 'Lip Mask',  
      'Eye Mask', 'Other', 'Tool', 'False Eyelash'], dtype=object)
```

Figure 4.4.9 result of unique type of skincare

Since some of the products are not used in our system, which mainly focuses on recommending skincare products, we stored those unused product types in the `not_skincare` array, as shown in Figure 4.4.10. We then filtered out the products listed in the `not_skincare` array from the dataset, as shown in Figure 4.4.11.

```
not_skincare = [  
    'Facial Treatment', 'Face Makeup', 'Bath & Body', 'Makeup Remover',  
    'Other Haircare', 'Shampoo', 'Fragrance', 'Hand Care', 'Conditioner',  
    'Eye Makeup', 'Tanning', 'Makeup Applicator', 'Lip Makeup',  
    'Cheek Makeup', 'Nail Care', 'Other', 'Tool', 'False Eyelash',  
    'Eye Mask', 'Lip Mask', 'Overnight Mask', 'Wet Mask',  
    'Sheet Mask', 'Eye Moisturizer', 'Lip Moisturizer', 'Oil']
```

Figure 4.4.10 code that shows `not_skincare` array

```
df = df[~df['type'].isin(not_skincare)]  
df
```

Figure 4.4.11 code to remove the skincare under `not_skincare` array

After cleaning the products listed in the `not_skincare` array, we executed the code `df['type'].unique()` again, and the results are shown in Figure 4.4.12.

```
array(['Toner', 'Face Cleanser', 'Serum', 'General Moisturizer',
      'Sunscreen', 'Exfoliator', 'Day Moisturizer', 'Emulsion',
      'Night Moisturizer', 'Essence'], dtype=object)
```

Figure 4.4.12 skincare product type used

We created a function called `clear_and_standardize`, which is used to split the data in a sentence by commas into a list. The function then trims extra spaces, converts the text into a title case for consistency, and ignores any blank entries. Duplicate values are also removed while preserving the original order. Finally, we apply this function to the existing `afterUse` column and create a new column named `afterUse_cleaned`. Figure 4.4.13 shows the function of `clear_and_standardize` to clean and standardize the after use effects.

```
# Clean and standardize the 'afterUse' column
def clear_and_standardize(cell):
    if pd.isna(cell):
        return []
    # Split by comma, strip whitespace, remove duplicates
    effects = [effect.strip().title() for effect in cell.split(',') if effect.strip()]
    return list(dict.fromkeys(effects)) # Removes duplicates while preserving order

# Apply the function
df['afterUse_cleaned'] = df['afterUse'].apply(clear_and_standardize)
```

Figure 4.4.13 code to clean and standardize the data

We then imported the `MultiLabelBinarizer` from `scikit-learn` to transform the cleaned `afterUse_cleaned` column, which contains lists of effects, into a binary format where each effect becomes its own column with values of 1 (present) or 0 (absent). The result is stored in a `DataFrame`, with column names representing different effects and rows aligned with the original `DataFrame` index and stored in a variable called `after_use_mlb`. This `DataFrame` is then merged back into the main `DataFrame` `df`, representing each effect as an individual column alongside the original data. Figure

4.4.14 shows the application of the MultiLabelBinarizer on the after-use effects column.

```
# Use multi-label binarizer to expand the effects
from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer()
after_use_mlb = pd.DataFrame(mlb.fit_transform(df['afterUse_cleaned']),
                             columns=mlb.classes_,
                             index=df.index)

# Merge back into main DataFrame
df = pd.concat([df, after_use_mlb], axis=1)
```

Figure 4.4.14 code to perform MultiLabelBinarizer

After applying the MultiLabelBinarizer on the after-use effect column, we removed products marked as acne triggers and dropped the acne trigger column, as shown in Figure 4.4.15. Then, the afteruse and afteruse_cleaned columns were also dropped, as they are not used in the system, as shown in Figure 4.4.16.

```
df = df[df["Acne Trigger"] != 1]
df.drop(columns=['Acne Trigger'], inplace=True)
```

Figure 4.4.15 code to remove acne trigger product

```
df.drop(columns=['afterUse'], inplace=True)
df.drop(columns=['afterUse_cleaned'], inplace=True)
```

Figure 4.4.16 code to remove unused column

Table 4.4.1 product features with its binary value representation

Product Features	Binary Value Representation	
	0 (Now present)	1 (Present)
Acne Fighting	Cannot fight acne	Can fight acne
Anti-Aging	Cannot reduce aging signs	Can reduce aging signs

Brightening	Cannot brighten skin	Can brighten skin
Dark Spots	Cannot reduce dark spots	Can reduce dark spots
Drying	Does not cause skin dryness	May cause skin dryness
Good For Oily Skin	Not suitable for oily skin	Suitable for oily skin
Hydrating	Does not provide hydration	Provides hydration
Irritating	Does not irritate skin	May irritate skin
May Worsen Oily Skin	Does not worsen oily skin	May worsen oily skin
Redness Reducing	Cannot reduce redness	Can reduce redness
Reduces Irritation	Cannot reduce irritation	Can reduce irritation
Reduces Large Pores	Cannot reduce large pores	Can reduce large pores
Rosacea	Not suitable for rosacea	Beneficial for rosacea
Scar Healing	Cannot help healing scars	Can help healing scars
Skin Texture	Cannot improve skin texture	Can improve skin texture

We will declare the features associated with the acne severity with the skin type as shown in the Figure 4.4.17. The dictionary defines which skincare features are recommended for users depending on their acne severity level. The keys represent acne severity levels which are 0 means mild, 1 means moderate, and 2 means severe. Each level contains its feature rules with values 1 means recommended or 0 means not recommended. This can ensure that the system will filter out the product based on the feature rules for each acne severity level. The feature rules for each acne severity level as shown below:

- Level 0: Products with Brightening and Anti-Aging are suitable.
- Level 1: Products with Acne Fighting and Reduces Irritation properties are suitable.
- Level 2: Products with Acne Fighting, Scar Healing, and Reduces Irritation are suitable.

```

acne_features = {
  0: {
    'Brightening': 1,
    'Anti-Aging': 1
  },
  1: {
    'Acne Fighting': 1,
    'Reduces Irritation': 1
  },
  2: {
    'Acne Fighting': 1,
    'Scar Healing': 1,
    'Reduces Irritation': 1
  }
}

```

Figure 4.4.17 dictionary with feature rules for acne severity level

Besides, the dictionary in Figure 4.4.18 defines recommended features based on the user's skin type. The keys represent skin types which are dry, normal and oily. Each skin type maps to features marked with 1 means recommended or 0 means not recommended. This can ensure that the system will filter out the product based on the feature rules for each skin type. The feature rules for each skin type as shown below:

- Dry skin: Products that are Hydrating and Redness Reducing are recommended.
- Normal skin: Products that are Hydrating are suitable.
- Oily skin: Products that are Good for Oily Skin and Reduces Large Pores are recommended, while those that May Worsen Oily Skin are flagged as not recommended.

```

skin_features = {
  'dry': {
    'Hydrating': 1,
    'Redness Reducing': 1,
  },
  'normal': {
    'Hydrating': 1,
  },
  'oily': {
    'Good For Oily Skin': 1,
    'Reduces Large Pores': 1,
    'May Worsen Oily Skin': 0
  }
}

```

Figure 4.4.18 dictionary with feature rules for skin type

After we define the feature rules for skin acne severity and skin type, we will define a function called `filter_and_group` to filter the skincare products based on each combination of acne severity level and skin type as shown in Figure 4.4.19.

```
def filter_and_group(df, acne_level, skin_type):
    # Collect acne rules (both 1 and 0)
    acne_rules = acne_features[acne_level]
    skin_rules = skin_features[skin_type]

    # Merge rules together
    rules = {**acne_rules, **skin_rules}

    # Start with all products
    filtered = df.copy()

    # Apply each rule one by one
    for feature, required_value in rules.items():
        filtered = filtered.loc[filtered[feature] == required_value]

    # Group the remaining products by type
    grouped = filtered.groupby("type")

    return grouped
```

Figure 4.4.19 code for the `filter_and_group` function

The code defines two lists which are `skin_types` (`['dry', 'normal', 'oily']`) and `skin_acne_severity` (`[0, 1, 2]`). It then iterates through all combinations of acne severity levels and skin types. For each combination, it calls the function `filter_and_group(df, level, skin)` to filter products based on the specified rules. If no products match the criteria, the system outputs “No matching products.” Otherwise, it groups the results by product type and prints the product type followed by the names of all matching products. Figure 4.4.20 shows the code to filter all products that are related to each combination of skin type and skin acne severity.

```

skin_types = ['dry', 'normal', 'oily']
skin_acne_severity = [0, 1, 2]

for level in skin_acne_severity:  # ✅ Loop directly over the list
    for skin in skin_types:
        print(f"\n✅ Acne Level: {level}, Skin Type: {skin}")
        grouped = filter_and_group(df, level, skin)

        if grouped.ngroups == 0:
            print("No matching products.")
        else:
            for product_type, group in grouped:
                print(f"{product_type}:")
                for _, row in group.iterrows():
                    print(f"    - {row['name']}")

```

Figure 4.4.20 code for Iterating Over Acne Levels and Skin Types to Display Filtered Product Groups

Table 4.4.2 presents the distribution of skincare products across different combinations of acne severity levels and skin types for each product category.

Table 4.4.2 Skincare Product Distribution by Acne Level and Skin Type

Acne Level	Acne Level 0			Acne Level 1			Acne Level 2		
Product type \ Skin Type	Dry	Normal	Oily	Dry	Normal	Oily	Dry	Normal	Oily
Day Moisturizer	6	10	12	2	4	5	2	3	2
Emulsion	–	–	1	–	–	–	–	–	–
Essence	13	17	19	5	5	10	3	3	5
Exfoliator	8	18	145	3	3	41	1	1	10
Face Cleanser	13	30	105	9	10	49	4	4	20
General Moisturizer	47	82	63	16	17	33	14	15	20
Night Moisturizer	2	4	22	1	2	6	–	1	5
Serum	115	141	386	43	46	118	26	29	76
Sunscreen	7	20	15	1	1	5	1	1	4
Toner	38	52	140	17	19	52	10	11	21

4.5 System Design Diagram

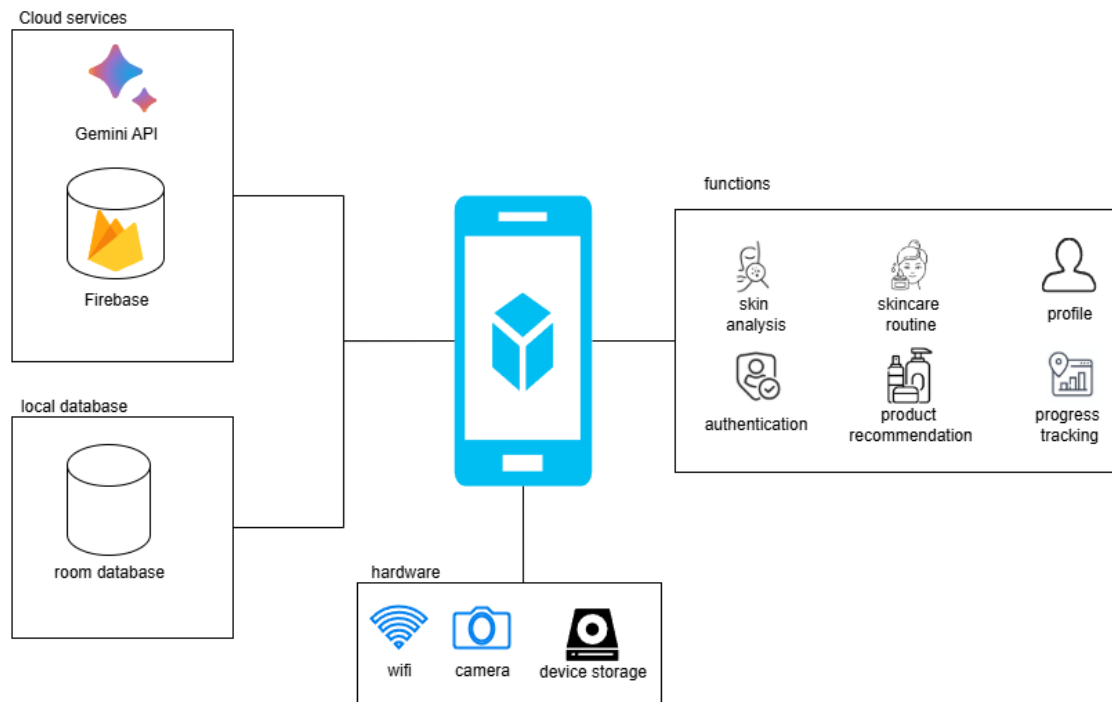


Figure 4.5.1 System design diagram of skin analysis and recommendations system

In the skin analysis and recommendation system, the mobile application integrates cloud services such as Gemini API and Firebase. The Gemini API serves as the intelligence layer of the system, providing AI-driven insights to enhance the user experience. It will use to generate the product recommendations, routine recommendations and descriptive analysis for the user of mobile application. For the product recommendations, the users can receive the suggestion for each skincare products that match their skin acne severity level and skin type. For routine recommendations, the users will use the skincare routine created and receive the recommendation from the Gemini API so that they can know that the suitability of combination of skincare products based on their current skin condition and history. Then, for the descriptive analysis, the Gemini API will analyse the past skin progress data to provide a meaningful summaries and insight on the user improvement trends. Therefore, Gemini API will act as a suggestion module, helping users make better-informed decisions by offering data-driven recommendations and explanations. This reduces the reliance on trial-and-error in skincare selection.

CHAPTER 4

Besides, firebase will use as the cloud database to store the information for each user including their skin analysis result, skincare routine, profile information, and history of skin analysis and product recommendations. Then, the room database will use as local database to store the skincare products dataset. The skincare products dataset will use in product recommendation module to provide the recommendations to user based on their acne severity and skin type.

For the functionality within the mobile application, users are required to authenticate themselves. New users must register for an account, while existing users can log in directly to access the application. Once authenticated, the system allows users to perform skin analysis and receive product recommendations. After performing a skin analysis, users can also track their skin progress to monitor their condition over time. In addition, they can add skincare products to their skincare routine, enabling them to refer to the routine and observe improvements in their skin from time to time. Users can also manage their profile information, including profile image, full name, age, and gender.

The user is required to use Wi-Fi to access the mobile application, as the system relies on cloud services that need an internet connection. In addition, a local database is stored in the device storage, meaning the user must utilize their local storage. Furthermore, the user must use their device camera to perform skin analysis, since this feature requires camera access to enable real-time face detection for analyzing acne severity level and skin type.

4.5.1 Flowchart

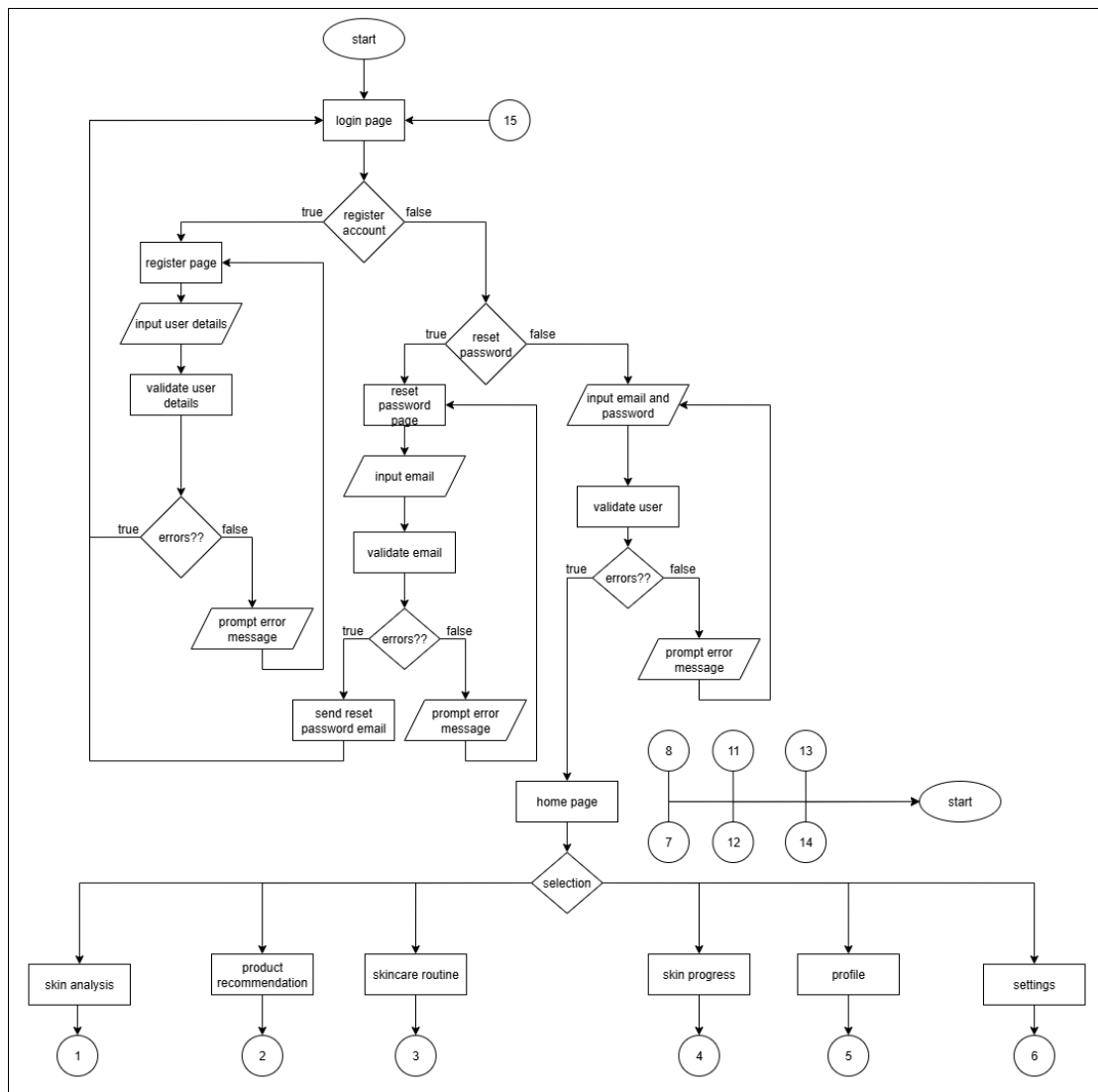


Figure 4.5.1.1 Flowchart for registration, login and main menu

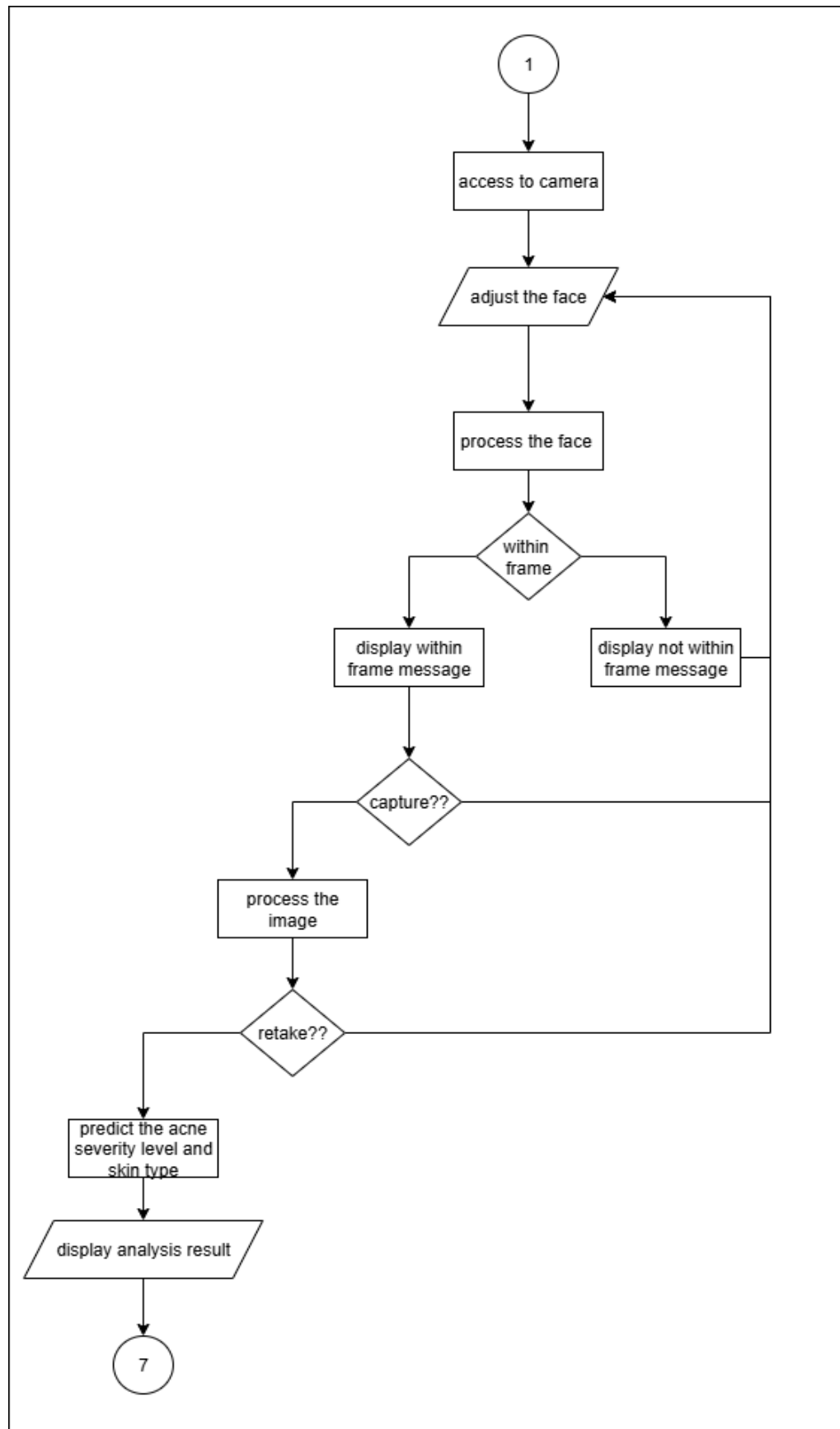


Figure 4.5.1.2 Flowchart for skin analysis process

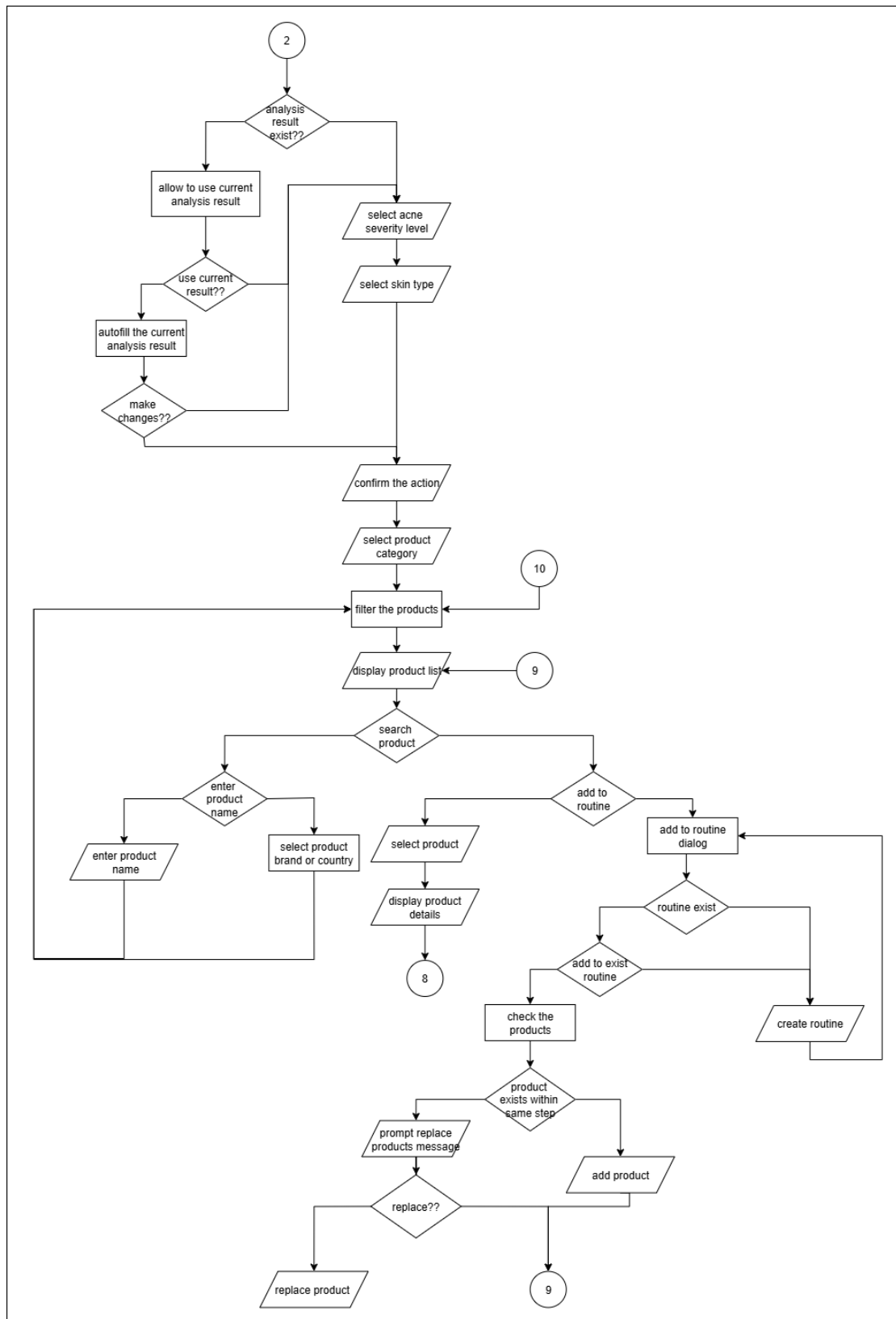


Figure 4.5.1.3 Flowchart for product recommendation process

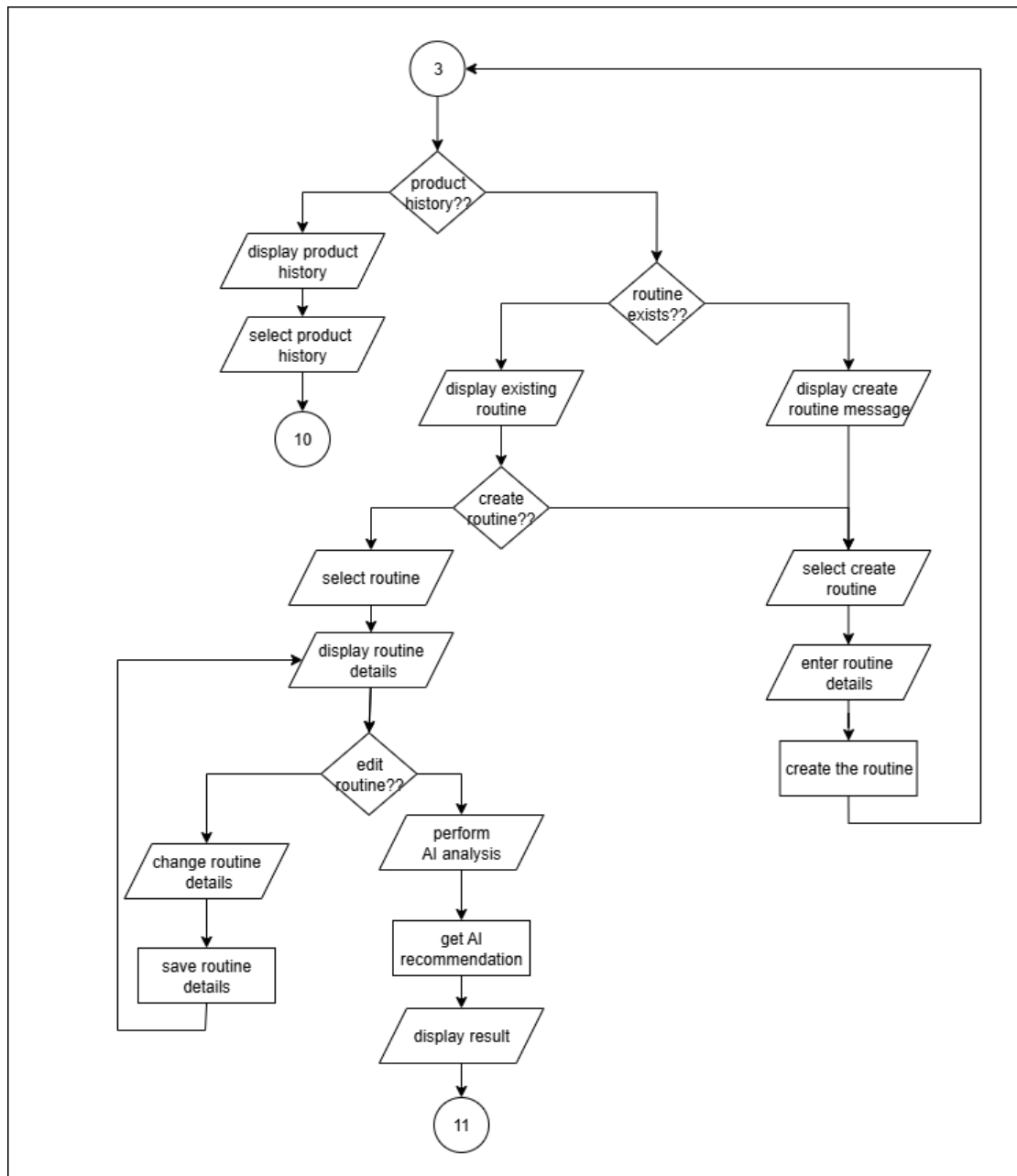


Figure 4.5.1.4 Flowchart for skincare routine process

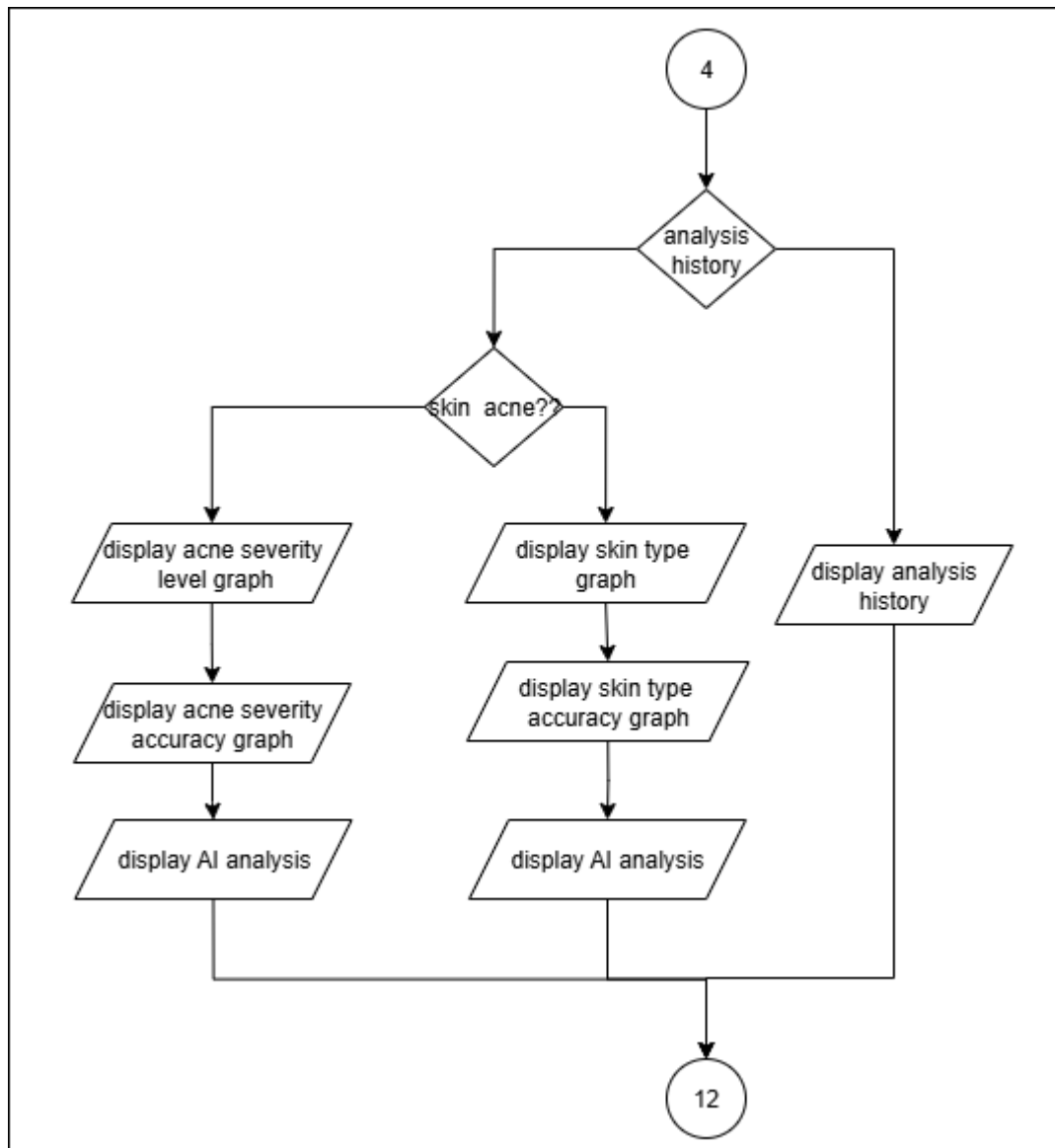


Figure 4.5.1.5 Flowchart for skin progress process

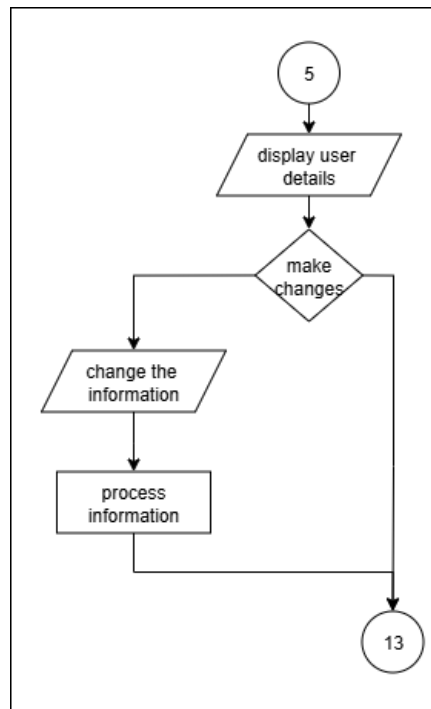


Figure 4.5.1.6 Flowchart for managing profile process

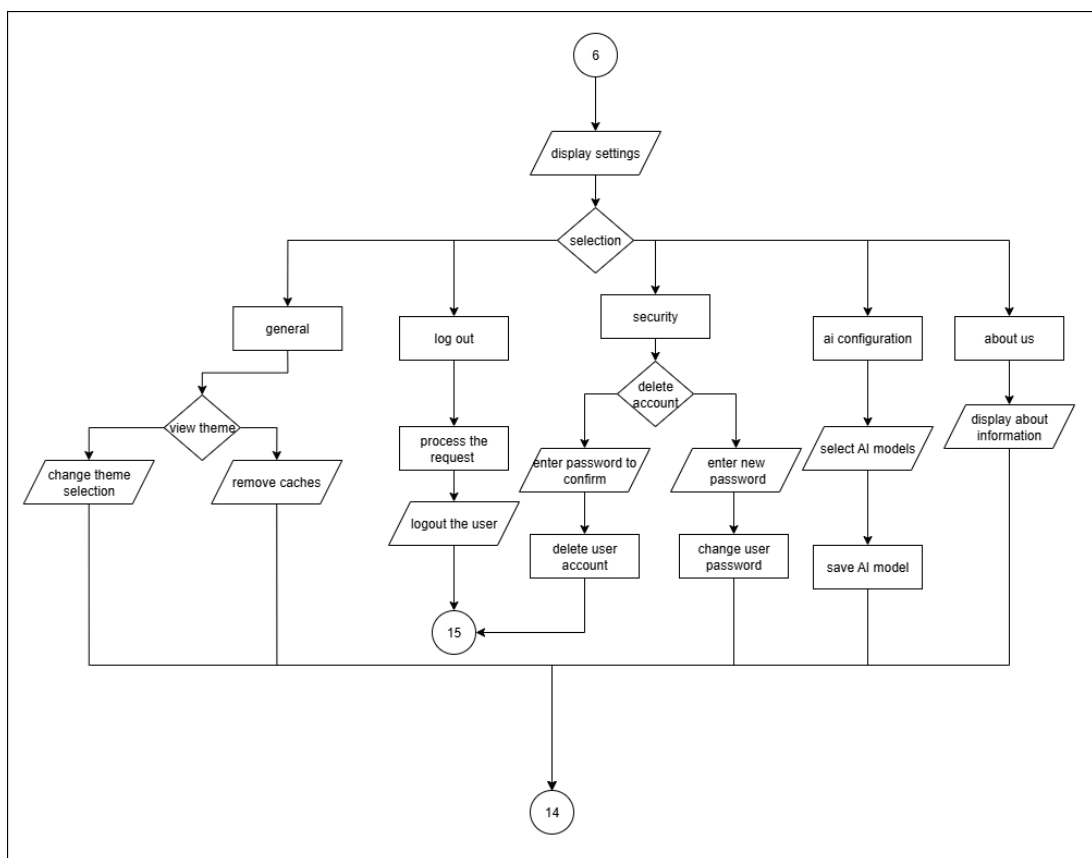


Figure 4.5.1.7 Flowchart for settings process

Chapter 5

System Implementation

5.1 Hardware Setup

The hardware involved in this project includes a laptop and a mobile device. The laptop is used for development tasks such as training the model, managing cloud services, and developing mobile applications. Meanwhile, the mobile device is used to install and test the developed mobile application. This helps ensure that the application runs smoothly on mobile devices and is free of errors in real-world usage scenarios. The experiments were conducted on a laptop with the specifications outlined in Table I to ensure sufficient computational capability.

Table 5.1 Specifications of laptop

Description	Specifications
Model	Acer Nitro AN515-45
Processor	AMD Ryzen 7 5800H with Radeon Graphics
Operating System	Windows 11
Graphic	NVIDIA GeForce GTX 1650
Memory	16GB DDR4 RAM
Storage	512GB SATA SSD

Table 5.2 Specifications of mobile devices

Description	Specifications
Model	Samsung S25 ultra
Processor	Snapdragon® 8 Elite
Operating System	Android 15
Memory	12GB RAM
Storage	256GB SSD

5.2 Software Setup

In this project, the tools involved include Android Studio, Firebase, python, Keras, TensorFlow, java, GitHub and google cloud.

Python will be used to develop the standard CNN model, EfficientNetB0 model, ResNet50 model, YOLOv8 model, and the image classification model integrated with the GPT assistant. All classification models will run locally on the computer using Jupyter Notebook, which allows coding in the Python programming language.



Figure 5.2.1 Python Programming
Language



Figure 5.2.2 Jupyter Notebook

The Java programming language will be used to design the layout and structure of the mobile application for the Android platform. Besides, Firebase will serve as the database for data storage and management. It will be used to store user information, including analysis results and product recommendations. Firebase also supports user authentication and management, allowing users to sign in or register in the mobile application using their social media accounts such as Facebook or Google. Additionally, Firebase can be used for image storage, temporarily holding images taken by users before they are sent to the cloud for further processing.



Figure 5.2.3 Firebase



Figure 5.2.4 Java Programming
Language

Additionally, Android Studio is an integrated development environment (IDE) designed for creating Android mobile applications, offering robust tools for coding, designing, and debugging. Developers can use a drag-and-drop interface to design the mobile application layout with UI components. Furthermore, Android Studio enables seamless integration with GitHub, a version control platform, allowing developers to track code changes and efficiently manage updates.



Figure 5.2.5 Android Studio



Figure 5.2.6 GitHub

Google developed TensorFlow, an open-source deep learning framework that provides flexibility for building machine learning and deep learning models. Users can create models using either computational graphs or eager execution. These models can run on CPUs, GPUs, or TPUs across a wide range of platforms, from portable devices to supercomputers.

Keras is a user-friendly, high-level API integrated with TensorFlow that is used for defining and training models. It simplifies the model training process by providing an intuitive interface, making it easier for developers to create and manage deep learning models.



Figure 5.2.7 TensorFlow



Figure 5.2.8 Keras

Google Cloud is a platform that allows users to store images in Google Cloud Storage by creating storage buckets. Images can be uploaded to these buckets and are accessible via automatically generated URL links, which enable direct access to the images. This

provides greater flexibility for users by allowing seamless access and integration of image resources across various applications. In this project, Google Cloud Storage is utilized to support the OpenAI API assistant-based classification by linking image URLs during both training and evaluation phases.



Figure 5.2.9 Google Cloud

5.3 Setting and Configuration

First, Figure 5.3.1 shows the Android configuration section in the app-level build.gradle file. In this step, the build.gradle is configured to include the API keys for both Gemini and OpenAI, which are stored in the local.properties file. These properties are then loaded and retrieved during the build process. Finally, the retrieved keys are stored in the BuildConfig.java file, which is automatically generated when the project is built, allowing the keys to be accessed securely within the application code.

```
android {
    namespace 'com.skingslow.app'
    compileSdk 34

    defaultConfig {
        applicationId "com.skingslow.app"
        minSdk 24
        targetSdk 34
        versionCode 1
        versionName "1.0"
        multiDexEnabled true

        // Securely add API key - prioritize local.properties, fallback to gradle.properties
        def localProperties = new Properties()
        def localPropertiesFile :File = rootProject.file('local.properties')
        if (localPropertiesFile.exists()) {
            localPropertiesFile.withInputStream { inputStream -> localProperties.load(it) }
        }

        def geminiApiKey = localProperties.getProperty('GEMINI_API_KEY') ?: project.findProperty('GEMINI_API_KEY') ?: ''
        def openaiApiKey = localProperties.getProperty('OPENAI_API_KEY') ?: project.findProperty('OPENAI_API_KEY') ?: ''
        buildConfigField "String", "GEMINI_API_KEY", "\"${geminiApiKey}\""
        buildConfigField "String", "OPENAI_API_KEY", "\"${openaiApiKey}\""
        buildConfigField "String", "GEMINI_MODEL_NAME", "\"gemini-1.5-flash\""
    }
}
```

Figure 5.3.1: App-level build.gradle configuration for securely injecting Gemini and OpenAI API keys

The dependencies required for this project are implemented as shown in Figure 5.3.2 and Figure 5.3.3. A range of libraries are included to support the system's core features. AndroidX components such as Core, AppCompat, Material Design, and ConstraintLayout are used to create a clean and responsive interface. Firebase modules, including Authentication, Firestore, and Storage, handle secure user login, data storage in the cloud, and media management, while Google Sign-In provides a simple login option. Glide is applied for fast and reliable image loading, and ViewPager2 enables smooth navigation across different sections of the app. CameraX, together with Guava, supports real-time camera functions like image capture and video recording, which are essential for skin analysis. TensorFlow Lite libraries allow the deployment of lightweight AI models such as YOLO, which classify acne severity and skin type directly on the device. Room is used as the local database to manage offline data, while OpenCSV is included for handling skincare product datasets in CSV format. For

intelligent recommendations, the system connects with large language models via the Gemini API or the OpenAI API, with OkHttp managing secure API communication. Lastly, MPAndroidChart is integrated to present skin progress trends through graphs, helping users track changes and improvements over time.

```
dependencies {
    implementation 'androidx.core:core:1.13.0'
    implementation 'androidx.appcompat:appcompat:1.7.0'
    implementation 'com.google.android.material:material:1.12.0'
    implementation 'com.google.firebase:firebase-storage:21.0.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.2.1'
    // Firebase dependencies
    implementation 'com.google.firebase:firebase-auth:23.2.1'
    implementation 'com.google.firebase:firebase-firestore:25.1.4'

    // Google Sign-In
    implementation 'com.google.android.gms:play-services-auth:21.2.0'

    // Image loading library
    implementation 'com.github.bumptech.glide:glide:4.16.0'

    // ViewPager2 for tab layout
    implementation 'androidx.viewpager2:viewpager2:1.1.0'

    // CameraX dependencies
    implementation 'androidx.camera:camera-core:1.3.4'
    implementation 'androidx.camera:camera-camera2:1.3.4'
    implementation 'androidx.camera:camera-lifecycle:1.3.4'
    implementation 'androidx.camera:camera-video:1.3.4'
    implementation 'androidx.camera:camera-view:1.3.4'
    implementation 'androidx.camera:camera-extensions:1.3.4'

    // Guava for ListenableFuture (required by CameraX)
    implementation 'com.google.guava:guava:32.1.3-android'

    // TensorFlow Lite dependencies for YOLO face detection
    implementation 'org.tensorflow:tensorflow-lite:2.13.0'
    implementation 'org.tensorflow:tensorflow-lite-support:0.4.4'
    implementation 'org.tensorflow:tensorflow-lite-metadata:0.4.4'
```

Figure 5.3.2: App-level build.gradle dependencies configuration (upper half)

```

// Room Database dependencies
implementation 'androidx.room:room-runtime:2.6.1'
annotationProcessor 'androidx.room:room-compiler:2.6.1'
implementation 'androidx.room:room-rxjava2:2.6.1'

// CSV parsing library
implementation 'com.opencsv:opencsv:5.9'

// Google Generative AI (Gemini)
implementation 'com.google.ai.client.generativeai:generativeai:0.7.0'

// HTTP client for OpenAI API calls
implementation 'com.squareup.okhttp3:okhttp:4.12.0'

// MPAndroidChart for graph visualization
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'

```

Figure 5.3.3: App-level build.gradle dependencies configuration (lower half)

Inside the AndroidManifest.xml, as shown in Figure 5.3.4, this project includes the necessary Android permissions. These permissions cover internet access to communicate with external services such as Firebase, permission to send notifications, and the ability to check network status to ensure the mobile app is connected to the internet. In addition, the project requires permissions to read and write files, access images from storage, and use the camera for real-time detection.

```

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

```

Figure 5.3.4: Declared permissions in AndroidManifest.xml

After completing the initial settings in Android Studio, the next step is to set up Firebase for this project. A new Firebase project named Skin Glow is created, as illustrated in the example shown in Figure 5.3.5.

× Create a project

Let's start with a name for your project [?]

Project name

skin-glow

skin-glow-a7a3a

Already have a Google Cloud project?
[Add Firebase to Google Cloud project](#)

Continue

Figure 5.3.5: Firebase project creation for the Skin Glow application

In Figure 5.3.6, the SHA key configuration of the Android app is shown. For this step, the SHA-1 or SHA-256 key must be generated and added to the Firebase project. After that, the `google-services.json` file can be downloaded from Firebase and placed into the Android project directory, following the setup instructions provided by Firebase.

Android apps

com.skinglow.app

SDK setup and configuration

Need to reconfigure the Firebase SDKs for your app? Revisit the SDK setup instructions or just download the configuration file containing keys and identifiers for your app.

[See SDK instructions](#) [google-services.json](#)

App ID [?]

App nickname

Add a nickname [?]

Package name

com.skinglow.app

SHA certificate fingerprints [?]

Type [?]

	SHA-1
	SHA-256
	SHA-256

[Add fingerprint](#)

[Remove this app](#)

Figure 5.3.6: Configuration of SHA key and integration of `google-services.json` file in Firebase setup

CHAPTER 5

The Gemini API key and the OpenAI API key can be generated from their respective API platforms. For Gemini, the key is created in Google AI Studio by clicking on the Get API Key button located at the bottom-left of the dashboard. The user can then generate a new key by selecting the Create API Key option at the top-right corner. Once the key is successfully created, it is displayed as shown in Figure 5.3.7. Similarly, the OpenAI API key is generated through the OpenAI API platform. The user simply needs to click on Create New Secret Key to generate a new key, which can then be viewed after creation, as illustrated in Figure 5.3.8.



Figure 5.3.7: API key generation in Google AI Studio for Gemini integration

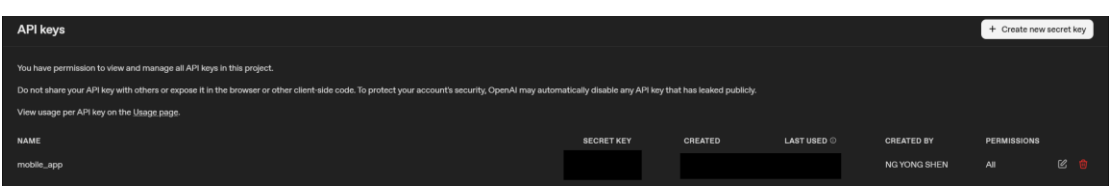


Figure 5.3.8: API key generation in OpenAI platform for OpenAI integration

5.4 Code Explanation

5.4.1 Authentication Module

The `registerWithEmailPasswordAndName` method is used to register a user account with an email, password, and full name, as shown in Figure 5.4.1.1. It begins by creating an account in Firebase using the user's email and password. Once the account is created, the system updates the profile with the full name provided during registration. A verification email is then sent to the user's email address to confirm their account, as shown in Figure 5.4.1.2. This step ensures that the email address registered in the mobile application is valid and authenticated. After that, the user's details, including their unique ID, full name, and email, are stored in the Firestore database for future use. To maintain security, the system signs the user out immediately after registration until the email verification is completed. In addition, the function handles errors such as issues with sending the verification email or saving the user's data, and it provides clear error messages, so the user understands what went wrong.

```
public void registerWithEmailPasswordAndName(String email, String password, String fullName,
AuthCallback callback) {
    firebaseAuth.createUserWithEmailAndPassword(email, password)
        .addOnSuccessListener(authResult -> {
            FirebaseUser firebaseUser = authResult.getUser();
            if (firebaseUser != null) {
                // Update the user's display name
                UserProfileChangeRequest profileUpdates = new UserProfileChangeRequest.Builder()
                    .setDisplayName(fullName)
                    .build();

                firebaseUser.updateProfile(profileUpdates)
                    .addOnSuccessListener(aVoid -> {...})
                    .addOnFailureListener(exception -> {
                        String errorMessage = "Failed to update user profile: "
                            + getErrorMessage(exception);
                        AuthResult result = new AuthResult( success: false, errorMessage, exception);
                        callback.onFailure(result);
                    });
            } else {
                AuthResult result = new AuthResult( success: false, message: "Registration failed: User creation error");
                callback.onFailure(result);
            }
        })
        .addOnFailureListener(exception -> {
            String errorMessage = getErrorMessage(exception);
            AuthResult result = new AuthResult( success: false, errorMessage, exception);
            callback.onFailure(result);
        });
}
```

Figure 5.4.1.1 User Registration Function in Firebase


```

firebaseUser.updateProfile(profileUpdates)
    .addOnSuccessListener(aVoid -> {
        // Send email verification
        firebaseUser.sendEmailVerification()
        .addOnSuccessListener(emailVoid -> {
            // Store user data in Firestore using UserRepository
            userRepository.storeUser(firebaseUser.getId(), email, fullName,
                // yong shen
                new UserRepository.UserCallback() {
                    // yong shen
                    @Override
                    public void onSuccess(String message) {
                        // Sign out the user after successful registration and
                        // data storage
                        firebaseAuth.signOut();

                        AuthResult result = new AuthResult(succes true,
                            message: "Account created successfully! Please check your email to verify your account, then sign in.");
                        callback.onSuccess(result);
                    }
                    // yong shen
                    @Override
                    public void onFailure(String error) {
                        // Even if Firestore storage fails, we should sign out
                        // the user
                        firebaseAuth.signOut();

                        String errorMessage = "Account created and verification email sent, but failed to store user data: "
                            + error;
                        AuthResult result = new AuthResult(succes false, errorMessage);
                        callback.onFailure(result);
                    }
                });
        })
        .addOnFailureListener(emailException -> {
            // Even if email verification fails, we should sign out the user
            firebaseAuth.signOut();

            String errorMessage = "Account created but failed to send verification email."

```

Figure 5.4.1.2 Email Verification Process in Firebase

Besides, another method is used to allow a user to sign in with their email and password as shown in Figure 5.4.1.3. It first attempts to authenticate the user through Firebase. If the login is successful and the account exists, the system checks whether the user's email address has been verified. If the email is verified, the system converts the Firebase user into a custom user object and returns a success message. If the email is not verified, the system immediately signs the user out and prompts them to verify their email before signing in. If the login attempt fails because the user is not found or an error occurs, the system provides a clear error message, so the user knows what went wrong.

```

public void signInWithEmailAndPassword(String email, String password, AuthCallback callback) {
    firebaseAuth.signInWithEmailAndPassword(email, password)
        .addOnSuccessListener(authResult -> {
            FirebaseUser firebaseUser = authResult.getUser();
            if (firebaseUser != null) {
                // Check if email is verified
                if (firebaseUser.isEmailVerified()) {
                    User user = convertFirebaseUserToUser(firebaseUser);
                    AuthResult result = new AuthResult(succes true, message: "Login successful", user);
                    callback.onSuccess(result);
                } else {
                    // Sign out unverified user
                    firebaseAuth.signOut();
                    AuthResult result = new AuthResult(succes false,
                        message: "Please verify your email address before signing in. Check your inbox for the verification email.");
                    callback.onFailure(result);
                }
            } else {
                AuthResult result = new AuthResult(succes false, message: "Login failed: User not found");
                callback.onFailure(result);
            }
        })
        .addOnFailureListener(exception -> {
            String errorMessage = getErrorMessage(exception);
            AuthResult result = new AuthResult(succes false, errorMessage, exception);
            callback.onFailure(result);
        });
}

```

Figure 5.4.1.3 sign in process in Firebase

5.4.2 Skin Analysis Module

The `toGrayscale` method is used in the face scanning process, where it first converts real-time captured images from RGB color format into grayscale. It begins by retrieving the width and height of the bitmap, which represents the colored image, and then creates an empty bitmap in grayscale format. The function extracts all pixel values from the original image and loops through each pixel to separate its red, green, and blue components. Using these values, it calculates a single grayscale value, which is then applied equally to the red, green, and blue channels of each pixel, effectively removing color while preserving important image details. After converting the RGB values into grayscale, the function updates the new bitmap with the grayscale values and returns the processed grayscale image. The code snippet of this function is shown in Figure 5.4.2.1.

```
private Bitmap toGrayscale(Bitmap bitmap) {
    try {
        int width = bitmap.getWidth();
        int height = bitmap.getHeight();

        Bitmap grayscaleBitmap = Bitmap.createBitmap(width, height, Bitmap.Config.RGB_565);

        int[] pixels = new int[width * height];
        bitmap.getPixels(pixels, 0, width, 0, 0, width, height);

        for (int i = 0; i < pixels.length; i++) {
            int pixel = pixels[i];
            int red = (pixel >> 16) & 0xFF;
            int green = (pixel >> 8) & 0xFF;
            int blue = pixel & 0xFF;

            // Convert to grayscale using luminance formula
            int gray = (int) (0.299 * red + 0.587 * green + 0.114 * blue);
            pixels[i] = (255 << 24) | (gray << 16) | (gray << 8) | gray;
        }

        grayscaleBitmap.setPixels(pixels, 0, width, 0, 0, width, height);
        return grayscaleBitmap;
    } catch (Exception e) {
        Log.e(TAG, "Error converting to grayscale: " + e.getMessage());
        // Return original bitmap if conversion fails
        return bitmap;
    }
}
```

Figure 5.4.2.1 Code Snippet of the `toGrayscale` Function for Face Scanning

This part of the code is responsible for detecting faces in a given image, as shown in Figure 5.4.2.2. It first converts the input image into grayscale using the `toGrayscale` function, which is required for Android's Face Detector. The face detector is then

initialized with the image's width, height, and the maximum number of faces to detect. The faces array is cleared to ensure that no extra or outdated data interferes with the detection process. The face detector then scans the image, and if at least one face is detected, it retrieves the most confident face by extracting the confidence score, the midpoint of the face, and the distance between the eyes. These metrics are used to estimate the bounding box, where the width is set to 1.5 times the eye distance and the height to 1.8 times the eye distance. The bounding box is then clamped to ensure it stays within the image boundaries. Finally, a `DetectionResult` is returned, containing whether a face was detected, its bounding box, and the confidence score. If no face is detected, the result indicates failure with zero confidence.

```
try {
    // Convert to grayscale for Android FaceDetector
    Bitmap grayscaleBitmap = toGrayscale(bitmap);
    // Create Android FaceDetector with current bitmap dimensions
    androidFaceDetector = new android.media.FaceDetector(
        grayscaleBitmap.getWidth(),
        grayscaleBitmap.getHeight(),
        MAX_FACES);
    // Clear faces array
    for (int i = 0; i < faces.length; i++) {
        faces[i] = null;
    }
    // Detect faces
    int numFaces = androidFaceDetector.findFaces(grayscaleBitmap, faces);
    if (numFaces > 0) {
        // Get the first (most confident) face
        android.media.FaceDetector.Face face = faces[0];
        float confidence = face.confidence();
        // Get face center point
        android.graphics.PointF faceCenter = new android.graphics.PointF();
        face.getMidPoint(faceCenter);
        // Estimate face bounds using eye distance
        float eyeDistance = face.eyesDistance();
        // Face width is typically 1.5 times the eye distance
        // Face height is typically 1.8 times the eye distance
        float faceWidth = eyeDistance * 1.5f;
        float faceHeight = eyeDistance * 1.8f;
        // Calculate bounding box
        float left = faceCenter.x - faceWidth / 2;
        float top = faceCenter.y - faceHeight / 2;
        float right = faceCenter.x + faceWidth / 2;
        float bottom = faceCenter.y + faceHeight / 2;
        // Clamp to image bounds
        left = Math.max(0, left);
        top = Math.max(0, top);
        right = Math.min(bitmap.getWidth(), right);
        bottom = Math.min(bitmap.getHeight(), bottom);
        RectF boundingBox = new RectF(left, top, right, bottom);
        return new DetectionResult( faceDetected: true, boundingBox, confidence);
    } else {
        return new DetectionResult( faceDetected: false, boundingBox: null, confidence: 0.0f);
    }
} catch (Exception e) {
    return new DetectionResult( faceDetected: false, boundingBox: null, confidence: 0.0f);
}
```

Figure 5.4.2.2 Code Snippet of the Face Detection Process Using Android
FaceDetector

Once a face is detected and the user chooses to capture the image, it is sent to the YOLO model to classify the acne severity level and skin type. Before convert the YOLO model into the TensorFlow lite model, the trained weights must be selected. After completing the YOLOv8 training, the best.pt file can be found in the folder path C:\Users\yongs\runs\classify\train\weights. This best.pt file is used as the model for deployment, as shown in Figure 5.4.2.3.



 best.pt	26/7/2025 6:09 PM	PT File	2,897 KB
 last.pt	26/7/2025 6:09 PM	PT File	2,897 KB

Figure 5.4.2.3 Folder Path Showing the Trained YOLOv8 Model (best.pt)

Then, we will go to google colab to export the model into tensorflow lite because local environment libraries version is not compatible with the version required to convert the model into tensorflow lite model. Therefore, google colab will be used as the medium to convert it. First, we need to install all the required documents inside the google colab as shown in Figure 5.4.2.4.

```
!pip install ultralytics onnx onnxsim onnxruntime tensorflow
```

Figure 5.4.2.4 Installing Required Dependencies in Google Colab for YOLOv8 to TensorFlow Lite Conversion

Next, the best.pt file needs to be uploaded to Google Colab to proceed with the conversion, as shown in Figure 5.4.2.5. After uploading, the best.pt model is loaded, and then exported into the TensorFlow Lite (.tflite) format, as shown in Figure 5.4.2.6.

```
from google.colab import files
uploaded = files.upload()
```

Figure 5.4.2.5 Uploading the best.pt File to Google Colab

```
from ultralytics import YOLO

model = YOLO('best.pt') # Path to your uploaded best.pt file
model.export(format='tflite') # Export to TFLite
```

Figure 5.4.2.6 Exporting the YOLOv8 Model to TensorFlow Lite Format

The result of the successfully exported TensorFlow Lite model is shown in Figure 5.4.2.7, where the model path can be seen in the last line of the screenshot. This path is then copied and pasted to download the .tflite model, as illustrated in Figure 5.4.2.8.

```
TensorFlow SavedModel: export success 20.7s, saved as ('best_saved_model', <tf.keras.src.engine.functional.Functional object at 0x7ac611ed8ad0>) (0.0 MB)
TensorFlow Lite: starting export with tensorflow 2.19.0...
TensorFlow Lite: export success 0.0s, saved as 'best_saved_model/best_float32.tflite' (5.5 MB)

Export complete (21.1s)
Results saved to /content
Predict: yolo predict task=classify model=best_saved_model/best_float32.tflite imgsz=224
Validate: yolo val task=classify model=best_saved_model/best_float32.tflite imgsz=224 data=dataset
Visualize: https://netron.app
'best_saved_model/best_float32.tflite'
```

Figure 5.4.2.7 Successful Export of the YOLOv8 Model to TensorFlow Lite

```
from google.colab import files
files.download('best_saved_model/best_float32.tflite')
```

Figure 5.4.2.8 Copying the Model Path to Download the .tflite File

The exported models for both the skin acne severity and skin type classifiers are renamed as `skin_acne.tflite` and `skin_type.tflite`, respectively, and then placed into the `assets` folder of the mobile application, as shown in Figure 5.4.2.9. This figure shows that both the `skin_acne.tflite` model and the `skin_type.tflite` model are stored in the `assets` folder.

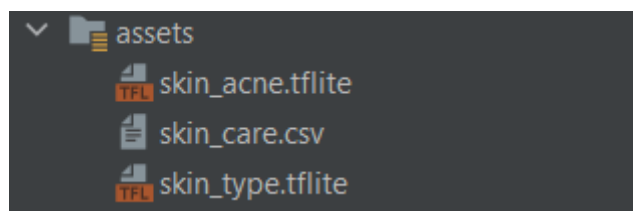


Figure 5.4.2.9 YOLOv8 Models Stored in the Assets Folder of the Mobile Application

The `preprocessBitmap` method in Figure 5.4.2.10 is used by both the YOLOv8 skin acne severity model and the skin type model. It first checks the image size to ensure it matches the model's required input size of (224, 224). Next, it performs normalization by scaling the RGB values into a range between 0 and 1 by dividing them by 255, making the image compatible with the model's expected input format. These normalized values are then stored sequentially in the input buffer. Finally, if a temporary resized bitmap was created, it is recycled to free memory resources.

```
private void preprocessBitmap(Bitmap bitmap) {
    // Resize bitmap to model input size if needed
    Bitmap resizedBitmap = bitmap;
    if (bitmap.getWidth() != INPUT_SIZE || bitmap.getHeight() != INPUT_SIZE) {
        resizedBitmap = Bitmap.createScaledBitmap(bitmap, INPUT_SIZE, INPUT_SIZE, filter: true);
    }

    inputBuffer.rewind();

    // Convert bitmap to ByteBuffer
    int[] pixels = new int[INPUT_SIZE * INPUT_SIZE];
    resizedBitmap.getPixels(pixels, offset: 0, INPUT_SIZE, x: 0, y: 0, INPUT_SIZE, INPUT_SIZE);

    for (int pixel : pixels) {
        // Extract RGB values and normalize to [0, 1]
        float r = ((pixel >> 16) & 0xFF) / 255.0f;
        float g = ((pixel >> 8) & 0xFF) / 255.0f;
        float b = (pixel & 0xFF) / 255.0f;

        inputBuffer.putFloat(r);
        inputBuffer.putFloat(g);
        inputBuffer.putFloat(b);
    }

    // Clean up if we created a resized bitmap
    if (resizedBitmap != bitmap) {
        resizedBitmap.recycle();
    }
}
```

Figure 5.4.2.10 Code Snippet for Preprocessing Bitmap Before YOLOv8 Model Inference

The `detectSkinType` method in Figure 5.4.2.11, is used to classify the skin type from a given image, while the acne severity level model follows the same code structure. The function first checks if the input bitmap is null; if so, it returns a default result of “normal” with zero confidence. Next, it calls the `preprocessBitmap` method to preprocess the image, and the processed image is then passed to the TensorFlow Lite interpreter to run the YOLOv8 model and generate predictions stored in the output

array. Finally, the function calls `processOutput()` to interpret the model's predictions and return the result containing the detected class and its confidence score. If an error occurs at any stage, the function logs the error and returns a default result of "normal" with zero confidence.

```
public SkinTypeResult detectSkinType(Bitmap bitmap) {  
    if (bitmap == null) {  
        Log.e(TAG, msg: "Input bitmap is null");  
        return new SkinTypeResult( skinType: "normal", confidence: 0.0f);  
    }  
  
    try {  
        // Preprocess the image  
        preprocessBitmap(bitmap);  
  
        // Run inference  
        interpreter.run(inputBuffer, outputArray);  
  
        // Process results  
        return processOutput();  
    } catch (Exception e) {  
        Log.e(TAG, msg: "Error during skin type detection: " + e.getMessage());  
        return new SkinTypeResult( skinType: "normal", confidence: 0.0f);  
    }  
}
```

Figure 5.4.2.11 Code Snippet for Detecting Skin Type Using YOLOv8 Models

The `uploadImageToStorage` method in Figure 5.4.2.12 is used to upload images to Firebase Storage. It first checks if the image file exists, and if not, it returns an error. If the file is valid, the function creates a unique filename using the current time to prevent overwriting existing files. The image is then uploaded to Firebase under a folder linked to the user's ID, keeping each user's images organized. Once uploaded, the function retrieves the image's download link, which can be used later in the app. If anything goes wrong during the upload or link retrieval, the function returns an error message.

```
private void uploadImageToStorage(String userId, String imagePath, ImageUploadCallback callback) {
    try {
        File imageFile = new File(imagePath);
        if (!imageFile.exists()) {
            callback.onFailure( error: "Image file does not exist");
            return;
        }

        // Create a unique filename using timestamp
        String timestamp = String.valueOf(System.currentTimeMillis());
        String fileName = "analysis-" + timestamp + ".jpg";
        String storagePath = STORAGE_PATH + "/" + userId + "/" + fileName;

        StorageReference imageRef = storage.getReference().child(storagePath);
        Uri fileUri = Uri.fromFile(imageFile);

        Log.d(TAG, "Uploading image to: " + storagePath);

        imageRef.putFile(fileUri).addOnSuccessListener(taskSnapshot -> {
            // Get download URL
            imageRef.getDownloadUrl().addOnSuccessListener(uri -> {
                String downloadUrl = uri.toString();
                Log.d(TAG, "Image uploaded successfully: " + downloadUrl);
                callback.onSuccess(downloadUrl);
            })
            .addOnFailureListener(e -> {
                Log.e(TAG, "Failed to get download URL", e);
                callback.onFailure( error: "Failed to get image URL: " + e.getMessage());
            });
        })
        .addOnFailureListener(e -> {
            Log.e(TAG, "Failed to upload image", e);
            callback.onFailure( error: "Failed to upload image: " + e.getMessage());
        });
    } catch (Exception e) {
        Log.e(TAG, "Error preparing image upload", e);
        callback.onFailure( error: "Error preparing image upload: " + e.getMessage());
    }
}
```

Figure 5.4.2.12 Code Snippet for Uploading Images to Firebase Storage

The `storeAnalysisDataInFirestore` method in Figure 5.4.2.13 is responsible for saving skin analysis results into Firestore. It starts by generating a timestamp, which is used as a unique identifier for each record. The function then creates a data map containing the analysis details, including the image URL, skin type, acne severity, their confidence scores, and the timestamp. This data is stored in Firestore under the structure “`skinAnalysisResult/{userId}/results/{timestamp}`”, ensuring that each user’s results are organized and can be retrieved later. If the data is stored successfully, a success message is logged and returned through the callback. If the storage process fails, an error message is logged and returned instead.

```
private void storeAnalysisDataInFirestore(String userId, String imageUrl, String skinType,
    String skinAcne, float skinTypeConfidence,
    float acneConfidence, SkinAnalysisCallback callback) {

    long timestamp = System.currentTimeMillis();
    String timestampStr = String.valueOf(timestamp);

    // Create the analysis result data
    Map<String, Object> analysisData = new HashMap<>();
    analysisData.put("imageUrl", imageUrl);
    analysisData.put("skinType", skinType);
    analysisData.put("skinAcne", skinAcne);
    analysisData.put("skinTypeConfidence", skinTypeConfidence);
    analysisData.put("acneConfidence", acneConfidence);
    analysisData.put("timestamp", timestamp);

    // Store in Firestore: skinAnalysisResult/{userId}/results/{timestamp}
    firestore.collection(COLLECTION_NAME).collection("results")
        .document(userId).document(timestampStr)
        .set(analysisData)
        .addOnSuccessListener(aVoid -> {
            Log.d(TAG, msg: "Successfully stored skin analysis result for user: " + userId);
            callback.onSuccess("Analysis result stored successfully");
        })
        .addOnFailureListener(e -> {
            Log.e(TAG, msg: "Error storing skin analysis result", e);
            callback.onFailure(error: "Failed to store analysis result: " + e.getMessage());
        });
}
```

Figure 5.4.2.13 Code Snippet for Storing Skin Analysis Results in Firestore

5.4.3 Skincare Recommendation Module

We apply the feature rules in the content-based filtering recommendations for skincare products by storing them in a map called `acne_features`. All the related features are then added to this map, as shown in Figure 5.4.3.1.

```
private static final Map<String, Map<String, Integer>> ACNE_FEATURES = new HashMap<>();

// Skin type feature rules with weights (1 = positive, 0 = negative)
5 usages
private static final Map<String, Map<String, Integer>> SKIN_TYPE_FEATURES = new HashMap<>();

static {
    // Initialize acne level features
    Map<String, Integer> acne0 = new HashMap<>();
    acne0.put("Brightening", 1);
    acne0.put("Anti-Aging", 1);
    ACNE_FEATURES.put("0", acne0);

    Map<String, Integer> acne1 = new HashMap<>();
    acne1.put("Acne Fighting", 1);
    acne1.put("Reduces Irritation", 1);
    ACNE_FEATURES.put("1", acne1);

    Map<String, Integer> acne2 = new HashMap<>();
    acne2.put("Acne Fighting", 1);
    acne2.put("Scar Healing", 1);
    acne2.put("Reduces Irritation", 1);
    ACNE_FEATURES.put("2", acne2);

    // Initialize skin type features
    Map<String, Integer> dry = new HashMap<>();
    dry.put("Hydrating", 1);
    dry.put("Redness Reducing", 1);
    SKIN_TYPE_FEATURES.put("dry", dry);

    Map<String, Integer> normal = new HashMap<>();
    normal.put("Hydrating", 1);
    SKIN_TYPE_FEATURES.put("normal", normal);

    Map<String, Integer> oily = new HashMap<>();
    oily.put("Good For Oily Skin", 1);
    oily.put("Reduces Large Pores", 1);
    oily.put("May Worsen Oily Skin", 0);
    SKIN_TYPE_FEATURES.put("oily", oily);
}
```

Figure 5.4.3.1 Mapping Acne Feature Rules for Content-Based Filtering

The `getFeatureFlags` method in Figure 5.4.18, is used to obtain the combined features of acne level and skin type by calling the `getCombinedFeatures` method. This method merges the feature rules from both parameters which are `acneLevel` and `skinType`. For example, if the inputs are Level 0 and Normal skin type, it may return a map containing features such as Brightening, Anti-Aging, and Hydrating, each with an integer value of 1. The function then iterates through the map, checks the key–value pairs, and sets the appropriate flags before returning them to the function call.

```
public static FeatureFlags getFeatureFlags(String acneLevel, String skinType) {
    FeatureFlags flags = new FeatureFlags();
    Map<String, Integer> features = getCombinedFeatures(acneLevel, skinType);
    for (Map.Entry<String, Integer> entry : features.entrySet()) {
        String feature = entry.getKey();
        Integer weight = entry.getValue();
        // Set positive features (weight = 1)
        if (weight == 1) {
            switch (feature) {
                case "Acne Fighting":
                    flags.needsAcneFighting = true;
                    break;
            }
        }
    }
}
```

Figure 5.4.3.2 Code Snippet for Combining Acne Level and Skin Type Features

The `getCategoryRecommendationsWithFilters` method, illustrated in Figure 5.4.3.3, is responsible for filtering products that satisfy the feature rules associated with each acne severity level and skin type. Initially, the method invokes `getFeatureFlags` to obtain the feature rules corresponding to the specified combination of `acneLevel` and `skinType`. Subsequently, it calls the `getProductsByCategoryFeaturesAndFilters` method in the `skinCareProductDao` to retrieve products from the Room database that match the defined category, brand, country, and the feature flags provided as input. Finally, the method returns the filtered products to the invoking component through the `callback.onSuccess` mechanism.

```
public void getCategoryRecommendationsWithFilters(String category, String acneLevel, String skinType,
String brand, String country, RepositoryCallback<List<SkinCareProduct>> callback) {
    executor.execute() -> {
        try {
            SkinFeatureManager.FeatureFlags flags = SkinFeatureManager.getFeatureFlags(acneLevel, skinType);

            List<SkinCareProduct> products = skinCareProductDao.getProductsByCategoryFeaturesAndFilters(
                category,
                brand,
                country,
                flags.needsAcneFighting,
                flags.needsReducesIrritation,
                flags.needsBrightening,
                flags.needsAntiAging,
                flags.needsHydrating,
                flags.needsGoodForOilySkin,
                flags.needsReducesLargePores,
                flags.needsScarHealing,
                flags.needsRednessReducing,
                flags.avoidMayWorsenOilySkin);

            products = applyBusinessLogic(products);
            callback.onSuccess(products);
        } catch (Exception e) {
            callback.onError(e);
        }
    });
}
```

Figure 5.4.3.3 code snippet for retrieving filtered skincare products

The `getProductsByCategoryFeaturesAndFilters` method, illustrated in Figure 5.4.3.4, executes a query to retrieve skincare products from the Room database based on both mandatory and optional filtering criteria. This query ensures that all retrieved products belong to a specific category, while brand and country filters are applied only if corresponding values are provided. It also integrates a series of feature-based boolean flags, such as `needsAcneFighting`, `needsHydrating`, or `needsBrightening`, which function as optional filters. For example, if `needsAcneFighting` is set to true, the query enforces that only products with the acne-fighting attribute are retrieved. Conversely, if it is set to false, the filter is bypassed, and products are not restricted by this attribute. An exception is the `avoidMayWorsenOilySkin` parameter, which is inversely applied to exclude products that may negatively affect oily skin. This approach provides a more flexible and dynamic mechanism for product filtering.

```
@Query("SELECT * FROM skincare_products WHERE type = :productCategory AND " +
        "(:brand IS NULL OR :brand = '' OR brand = :brand) AND " +
        "(:country IS NULL OR :country = '' OR country = :country) AND " +
        "(:needsAcneFighting = 0 OR acne_fighting = 1) AND " +
        "(:needsReducesIrritation = 0 OR reduces_irritation = 1) AND " +
        "(:needsBrightening = 0 OR brightening = 1) AND " +
        "(:needsAntiAging = 0 OR anti_aging = 1) AND " +
        "(:needsHydrating = 0 OR hydrating = 1) AND " +
        "(:needsGoodForOilySkin = 0 OR good_for_oily_skin = 1) AND " +
        "(:needsReducesLargePores = 0 OR reduces_large_pores = 1) AND " +
        "(:needsScarHealing = 0 OR scar_healing = 1) AND " +
        "(:needsRednessReducing = 0 OR redness_reducing = 1) AND " +
        "(:avoidMayWorsenOilySkin = 0 OR may_worsen_oily_skin = 0)")
List<SkinCareProduct> getProductsByCategoryFeaturesAndFilters(
    String productCategory,
    String brand,
    String country,
    boolean needsAcneFighting,
    boolean needsReducesIrritation,
    boolean needsBrightening,
    boolean needsAntiAging,
    boolean needsHydrating,
    boolean needsGoodForOilySkin,
    boolean needsReducesLargePores,
    boolean needsScarHealing,
    boolean needsRednessReducing,
    boolean avoidMayWorsenOilySkin);
```

Figure 5.4.3.4 code snippet for dynamic skincare product retrieval

5.4.4 Skincare Routine Module

The `createRoutine` method in Figure 5.4.4.1 is responsible for saving a new skincare routine into Firestore. It collects details such as the user ID, routine name, description, products, and timestamps, and organizes them into a map format suitable for storage. The data is stored under the user's record in a specific subcollection. Upon successful execution, the method logs the new routine ID and notifies the caller through a success callback. In the event of an error, it logs the issue and returns a failure message through the callback. A try-catch block is also implemented to safely handle any unexpected problems.

```
public void createRoutine(String userId, String routineName, String description,
    Map<String, String> productsMap, RoutineCallback callback) {
    try {
        long timestamp = System.currentTimeMillis();
        SkincareRoutine routine = new SkincareRoutine(userId, routineName, description, productsMap, timestamp);

        Map<String, Object> routineData = new HashMap<>();
        routineData.put("userId", routine.getUserId());
        routineData.put("routineName", routine.getRoutineName());
        routineData.put("description", routine.getDescription());
        routineData.put("productsMap", routine.getProductsMap());
        routineData.put("createdTimestamp", routine.getCreatedTimestamp());
        routineData.put("updatedTimestamp", routine.getUpdatedTimestamp());
        routineData.put("isActive", routine.isActive());

        firestore.collection(SKINCARE_ROUTINES_COLLECTION)
            .document(userId)
            .collection(ROUTINES_SUBCOLLECTION)
            .add(routineData)
            .addOnSuccessListener(documentReference -> {
                Log.d(TAG, "Routine created with ID: " + documentReference.getId() + " for user: " + userId);
                callback.onSuccess("Routine created successfully");
            })
            .addOnFailureListener(e -> {
                Log.e(TAG, "Error creating routine", e);
                callback.onFailure("Failed to create routine: " + e.getMessage());
            });
    } catch (Exception e) {
        Log.e(TAG, "Exception in createRoutine", e);
        callback.onFailure("Error creating routine: " + e.getMessage());
    }
}
```

Figure 5.4.4.1 code snippet for storing a new skincare routine in Firestore

The `checkForProductStepConflictWithVerification` method is illustrated in Figure 5.4.4.2, which shows the try-catch block. This code attempts to validate each existing product in the routine by parsing its product ID into an integer. If the conversion is successful, the system retrieves the product from the repository using the ID and verifies whether the stored name matches the expected product name. If the product is successfully verified, the method proceeds to check for step conflicts with the new product. In cases where the verification fails or an error occurs during retrieval, the process falls back to a name-based lookup to ensure consistency. If the product ID cannot be parsed due to an invalid format, the exception is caught, logged, and the product is skipped without interrupting the process. The use of counters ensures that once all products have been checked and no conflicts are detected, the new product is safely added to the routine. This approach enhances robustness by handling errors gracefully and ensuring that invalid or inconsistent product data does not disrupt the verification process.

```
// Step 1: Retrieve product by ID and verify name
try {
    int productId = Integer.parseInt(existingProductId);
    repository.getProductByIdWithNameVerification(productId, existingProductName,
        new SkinCareRepository.RepositoryCallback<SkinCareProduct>() {
            new {
                @Override
                public void onSuccess(SkinCareProduct verifiedProduct) {
                    if (verifiedProduct != null) {
                        // Name verification successful, check for step conflict
                        checkProductTypeConflict(newProduct, verifiedProduct, routine, userId,
                            newProductStep, existingProductName, existingProductId,
                            checkedProducts, totalProducts, conflictFound);
                    } else {
                        // Step 2: Name verification failed, retrieve by name
                        retrieveProductByNameFallback(newProduct, existingProductName, routine, userId,
                            newProductStep, existingProductId, checkedProducts, totalProducts, conflictFound);
                    }
                }
            }
        },
        new {
            @Override
            public void onError(Exception error) {
                Log.e(TAG, "Error retrieving product by ID: " + existingProductId, error);
                // Step 2: Fallback to name-based retrieval
                retrieveProductByNameFallback(newProduct, existingProductName, routine, userId,
                    newProductStep, existingProductId, checkedProducts, totalProducts, conflictFound);
            }
        }
    );
} catch (NumberFormatException e) {
    Log.e(TAG, "Invalid product ID format: " + existingProductId, e);
    // Skip this product and continue
    if (checkedProducts.incrementAndGet() == totalProducts && !conflictFound.get()) {
        addProductToExistingRoutine(newProduct, routine, userId);
    }
}
```

Figure 5.4.4.2 code snippet of Try-catch block in `checkForProductStepConflictWithVerification` method

After the system successfully retrieves the existing product ID and name, or retrieves the product name through the fallback process, it proceeds to call the `checkProductTypeConflict` method, as illustrated in Figure 5.4.4.3. This method compares the new product with the products already present in the skincare routine to ensure that the new product the user wishes to add does not duplicate an existing step. If both products belong to the same step, the existing product can be replaced by the new one. Conversely, if no conflict is detected, the new product is safely added to the existing routine.

```
private void checkProductTypeConflict(SkinCareProduct newProduct, SkinCareProduct existingProduct,
    SkincareRoutine routine, String userId, String newProductStep, String existingProductName,
    String existingProductId, AtomicInteger checkedProducts, int totalProducts, AtomicBoolean conflictFound) {

    // Step 3: Compare actual product types using the same logic as getSkincareStepForProduct
    String existingProductStep = getSkincareStepForProduct(existingProduct);

    if (newProductStep.equals(existingProductStep)) {
        // Conflict found! Set the flag to prevent multiple dialogs
        if (!conflictFound.getAndSet(true)) {
            runOnUiThread(() -> {
                showReplaceProductInStepDialog(newProduct, routine, userId, existingProductName, existingProductId);
            });
        }
        return;
    }

    // No conflict with this product, continue checking
    if (checkedProducts.incrementAndGet() == totalProducts && !conflictFound.get()) {
        runOnUiThread(() -> {
            addProductToExistingRoutine(newProduct, routine, userId);
        });
    }
}
```

Figure 5.4.4.3 code snippet for validating product step conflicts in a skincare routine

5.4.5 Skin Progress Module

The `updateChart()` method as shown in Figure 5.4.5.1 is designed for updating and displaying the skin type progress chart. First, it checks if there is any skin analysis history data; if not, it shows an empty state message. If data exists, it processes the history based on the currently selected period which is overall, daily or monthly. The method then converts each skin type record into a numeric value, creates chart entries, and stores corresponding dates for the X-axis. These entries are used to build a dataset, which is styled with colors, line width, circle radius, and smoothing for better visualization. The dataset is then placed into the chart, and the X-axis is formatted to display the correct dates. Once everything is set, the chart is refreshed to show the updated data. In addition, the method also updates the accuracy chart using the same period.

```
private void updateChart() {
    if (historyList.isEmpty()) {
        showEmptyState();
        return;
    }
    List<SkinAnalysisHistory> processedData = processDataByTimePeriod(historyList, currentTimePeriod);

    // Create chart entries
    List<Entry> entries = new ArrayList<>();
    List<String> dates = new ArrayList<>();

    for (int i = 0; i < processedData.size(); i++) {
        SkinAnalysisHistory item = processedData.get(i);
        float skinTypeValue = convertSkinTypeToNumeric(item.getSkinType());
        entries.add(new Entry(i, skinTypeValue));
        dates.add(String.valueOf(item.getTimestamp()));
    }
    if (entries.isEmpty()) {
        showEmptyState();
        return;
    }
    // Create dataset
    LineDataSet dataSet = new LineDataSet(entries, label: "Skin Type");
    dataSet.setColor(getResources().getColor(R.color.accent, theme: null));
    dataSet.setCircleColor(getResources().getColor(R.color.accent, theme: null));
    dataSet.setLineWidth(3f);
    dataSet.setCircleRadius(6f);
    dataSet.setDrawValues(false);
    dataSet.setMode(LineDataSet.Mode.CUBIC_BEZIER);

    LineData lineData = new LineData(dataSet);
    skinTypeProgressChart.setData(lineData);

    // Configure X-axis formatter
    XAxis xAxis = skinTypeProgressChart.getXAxis();
    xAxis.setValueFormatter(new DateTimeAxisValueFormatter(dates, processedData, currentTimePeriod));

    skinTypeProgressChart.invalidate();

    // Also update accuracy chart with same time period
    updateAccuracyChart(historyList);
}
```

Figure 5.4.5.1 snippet code for visualizing skin type progress over time

This code snippet in Figure 5.4.5.2 belongs to the method `updateAccuracyChart`. It is used to update the accuracy chart, which displays the model's accuracy over a selected period. First, the accuracy data is processed based on the chosen time frame, and if no data is available, the chart is cleared and a message stating "No accuracy data available" is shown. The processed data is then sorted by timestamp so that the chart displays results in chronological order. To keep the chart readable, the number of X-axis labels is adjusted dynamically, ensuring that only three to six labels are shown depending on the data size. Each accuracy record is converted into a percentage and added as a chart entry. If no valid entries are created, the chart is cleared again to avoid displaying incorrect information. Finally, a log message is generated to indicate the number of entries created and the selected period, which supports debugging and verification.

```
List<SkinAnalysisHistory> processedData = processAccuracyDataByTimePeriod(data, currentTimePeriod);

if (processedData.isEmpty()) {
    Log.d(TAG, "msg: \"No processed accuracy data to display\"");
    accuracyChart.clear();
    accuracyChart.setNoDataText("No accuracy data available");
    accuracyChart.invalidate();
    return;
}

// Sort processed data by timestamp (oldest first for chart display)
Collections.sort(processedData, (h1, h2) -> Long.compare(h1.getTimestamp(), h2.getTimestamp()));

// Calculate optimal label count based on data size
int dataSize = Math.max(1, processedData.size());
int maxLabels = Math.min(6, Math.max(3, dataSize / 2)); // Show 3-6 labels max
accuracyChart.getXAxis().setLabelCount(maxLabels, force: false);

// Create data entries for accuracy chart
List<Entry> entries = new ArrayList<>();

for (int i = 0; i < processedData.size(); i++) {
    SkinAnalysisHistory item = processedData.get(i);
    float accuracy = (float) (item.getAccuracyScore() * 100); // Convert to percentage
    entries.add(new Entry(i, accuracy));
}

Log.d(TAG, "msg: \"Created \" + entries.size() + \" accuracy chart entries for \" + currentTimePeriod + \" view\"");

if (entries.isEmpty()) {
    accuracyChart.clear();
    return;
}
```

Figure 5.4.5.3 code snippet to update the accuracy chart with processed accuracy data

5.4.6 Manage Profile

The `updateProfile` method in Figure 5.4.6.1 is responsible for updating a user's profile information in the system. If a new profile image is provided, the method first uploads the new image using the `uploadProfileImage` function. Once the upload is successful, it deletes the old image if available and updates the profile data with the new image URL along with the user's name, age, and gender. In case the image upload fails, an error message is logged, and the callback function is triggered to notify the failure. The method also logs upload progress, which can be displayed in the user interface if required. If no new image is provided, the method simply updates the profile data using the existing image URL.

```
public void updateProfile(String uid, String name, Integer age, String gender,
    Uri newImageUri, String currentPhotoUrl, ProfileUpdateCallback callback) {
    if (newImageUri != null) {
        // Upload new image first, then update profile
        ⚡ yong shen *
        uploadProfileImage(uid, newImageUri, new ImageUploadCallback() {
            ⚡ yong shen *
            @Override
            public void onSuccess(String imageUrl) {
                Log.d(TAG, msg: "Image upload successful, updating profile data");
                // Delete old image if exists
                deleteProfileImage(currentPhotoUrl);
                // Update profile with new image URL
                updateProfileData(uid, name, age, gender, imageUrl, callback);
            }
            ⚡ yong shen
            @Override
            public void onFailure(String error) {
                Log.e(TAG, msg: "Image upload failed: " + error);
                callback.onFailure( error: "Failed to upload image: " + error);
            }
        });
        1 usage ⚡ yong shen
        @Override
        public void onProgress(int progress) {
            Log.d(TAG, msg: "Image upload progress: " + progress + "%");
            // Progress can be handled by the calling activity if needed
        }
    }
    } else {
        Log.d(TAG, msg: "No new image, updating profile data only");
        // Update profile without changing image
        updateProfileData(uid, name, age, gender, currentPhotoUrl, callback);
    }
}
```

Figure 5.4.6.1 code snippet to update user profile

5.5 System Operation (with Screenshot)

5.5.1 Authentication Module

When users first access the Skin Glow mobile application, they will be redirected to the login page, as shown in Figure 5.1.1.1. For existing users, they can choose to fill in their email address and password, then click the “sign in” button to access the mobile application. They can also choose another alternative, which is using Google sign-in to access the mobile application, which provides a flexible way for them. For new users, they will be required to click “sign up” to create an account. Once they click “sign up,” the system will redirect the user to the register page, as shown in Figure 5.5.1.2.

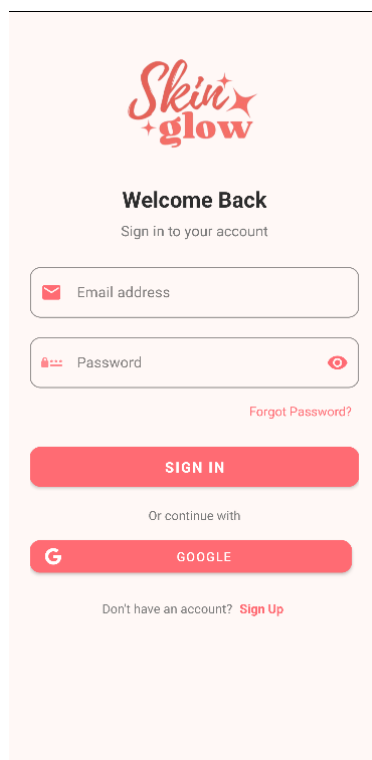


Figure 5.1.1.1 login page

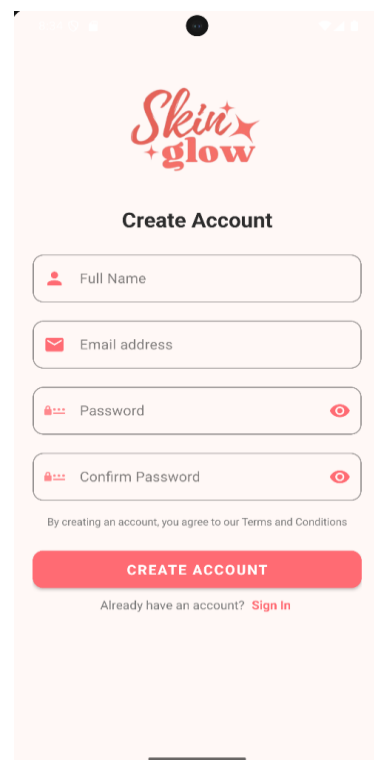


Figure 5.1.1.2 register page

CHAPTER 5

In the register page, the user will be required to fill in their full name, email address, password, and confirm password, as shown in Figure 5.1.1.3. After they enter the required information, they will need to double-check it to ensure that the registration process can proceed smoothly. They can now click the “create account” button to register an account. For the user who wants to go back to the login page, they can click “sign in.” Then the system will redirect them back to the login page. After the system completes the account creation process, it will redirect the user back to the login page and display a prompt to inform the user that the account has been created successfully.

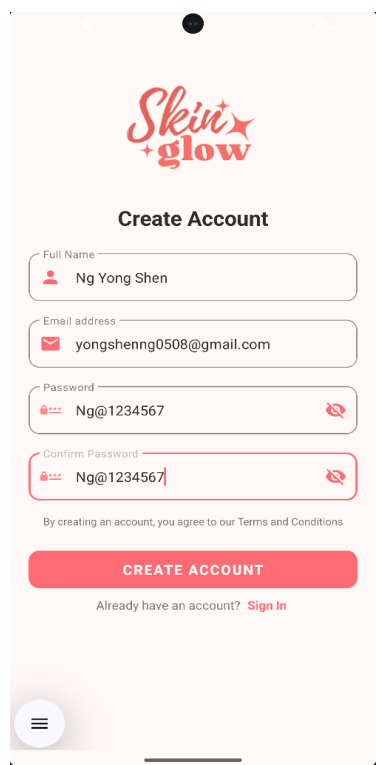


Figure 5.5.1.3 register details

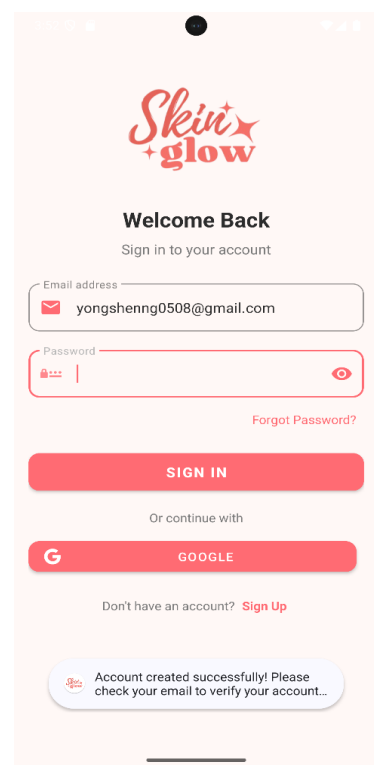


Figure 5.5.1.4 login page after registered

CHAPTER 5

After the user creates their account, they will be required to verify their email address. This is to ensure that the user's email address exists and to avoid fake email registrations. The user will need to click on the link sent from Firebase to their email, as shown in Figure 5.5.1.5. After the user clicks on the provided link, a message saying “Your email has been verified” will be displayed, as shown in Figure 5.5.1.6.



Figure 5.5.1.5 email verification

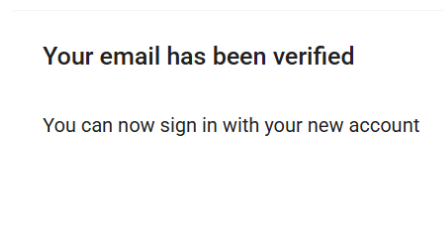


Figure 5.5.1.6 email verification success

After the user verifies their email address, they can proceed to fill in the required information, as shown in Figure 5.5.1.7. After they fill in the required information, they can click the “sign in” button. Then the system will check the user credentials to ensure that only authorized users can access the main menu. If the user is authenticated, the system will redirect them to the main menu and display a prompt informing the user that “Login successful,” as shown in Figure 5.5.1.8.

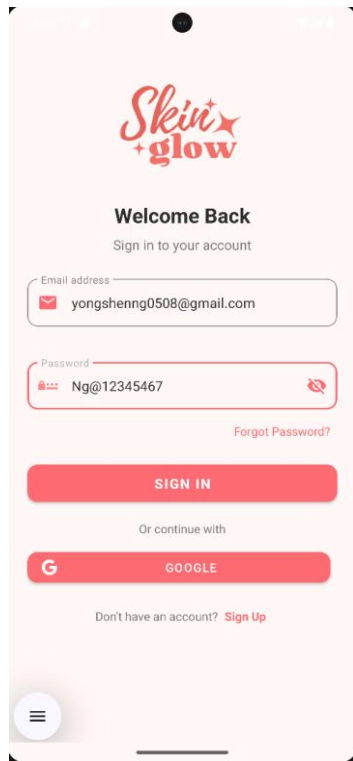


Figure 5.5.1.7 login details

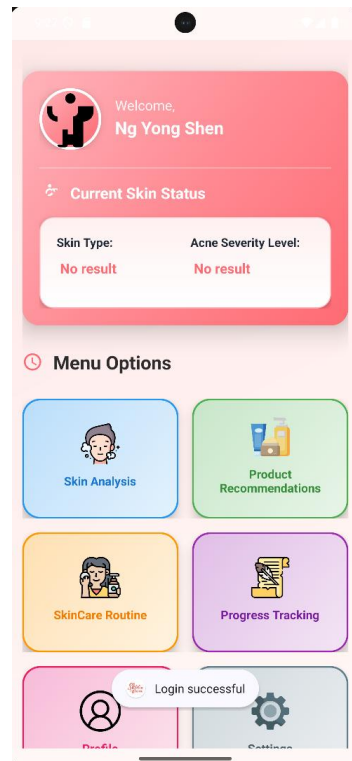


Figure 5.5.1.8 main menu after
successfully login

5.5.2 Skin Analysis Module

The user can perform skin analysis by clicking on the skin analysis icon, after which the system will redirect them to the skin analysis page. The user is required to scan their face by positioning it within the pink frame. Detection is handled by the face detector module, which uses Android's legacy `android.media.FaceDetector`. This component processes a `Bitmap`, detects up to 10 faces, and returns the most confident bounding box along with a confidence score. If the user's face is detected, they can capture the image by clicking on the Capture button, as shown in Figure 5.5.2.1. The system will then begin processing the image. While the image is being processed, the user has the option to click on the Retake button to recapture their image; otherwise, they can wait until the system completes the processing, as shown in Figure 5.5.2.2.

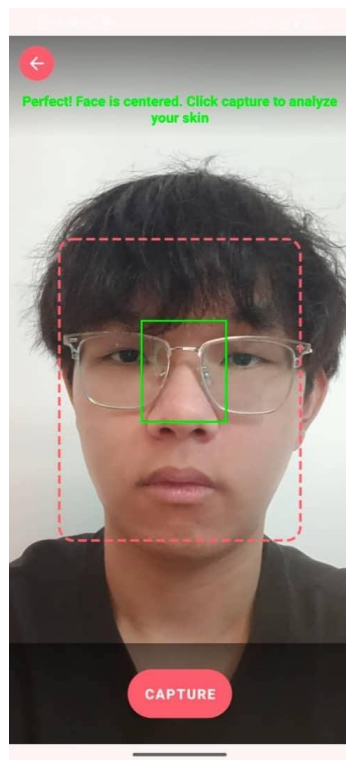


Figure 5.5.2.1 Face Detection and Capture Option

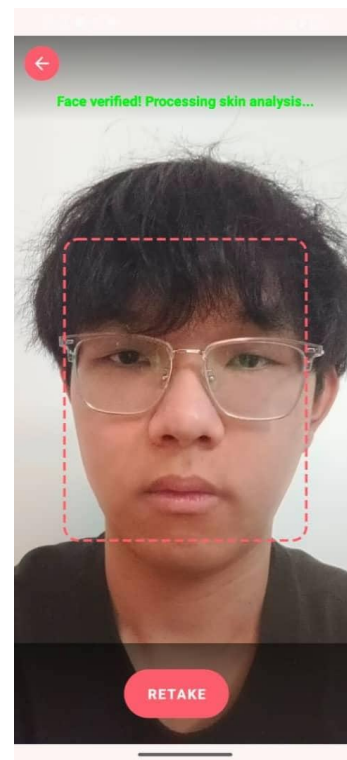


Figure 5.5.2.2 Image Processing After Capture

CHAPTER 5

Once the system completes processing the image, the skin analysis result will be displayed, as shown in Figure 5.5.2.3. The result includes the user's captured face along with the detected acne severity level and skin type, accompanied by accuracy values to inform the user of their skin condition. The user can scroll down, as shown in Figure 5.5.2.4, to view the acne progress chart based on their acne severity level. Finally, the user can click on Continue to return to the main menu.

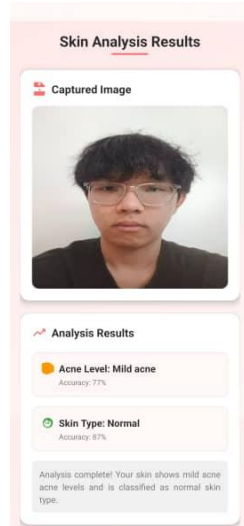


Figure 5.5.2.3 Skin Analysis Result
Display

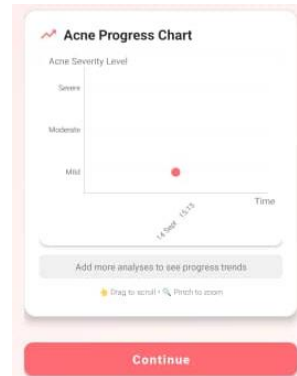


Figure 5.5.2.4 Acne Progress Chart

CHAPTER 5

After the user clicks on Continue, the system will redirect them back to the main menu. The user's status, including their skin type and acne severity level, will be updated with the latest result. For example, the updated status may show Normal skin type and Mild acne severity level, as illustrated in Figure 5.5.2.5.

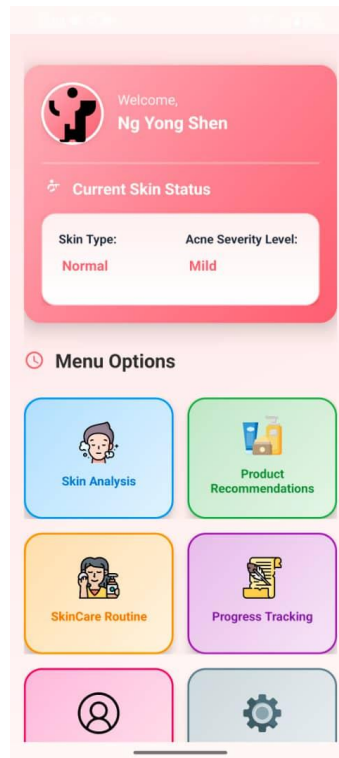


Figure 5.5.2.6 Updated User Status in Main Menu

5.5.3 Skincare Recommendation Module

The user can choose to get skincare recommendations by clicking on the skincare recommendation icon in the main menu. The system will redirect the user to the skincare characteristics page, where they are required to enter their acne severity level and skin type before proceeding to the next step. If the user has not performed any skin analysis, the skincare characteristics page will display a message “No previous analysis available”, as shown in Figure 5.5.3.1. In this case, the user must manually select their acne severity level and skin type. If the user has already performed a skin analysis, the page will appear as shown in Figure 5.5.3.2, where the user can click on the Use Current Result button. The system will then automatically fill in the selection using the latest skin analysis result. Finally, the user must click on Confirm to proceed to the next step.

Figure 5.5.3.1 Skincare Characteristics Page with No Previous Analysis

Figure 5.5.3.2 Skincare Characteristics Page with Current Analysis Result

After the user clicks on the Confirm button, the system will redirect them to the product category selection page before they can view the available products. The user is required to choose one of the product categories, which include face cleanser, exfoliator, toner, essence, serum, emulsion, moisturizer, and sunscreen as shown in Figure 5.5.3.3. Assume that the user clicks on face cleanser, then the system will filter out the face cleanser that suitable to the user. The filtered products as shown in Figure 5.5.3.4.

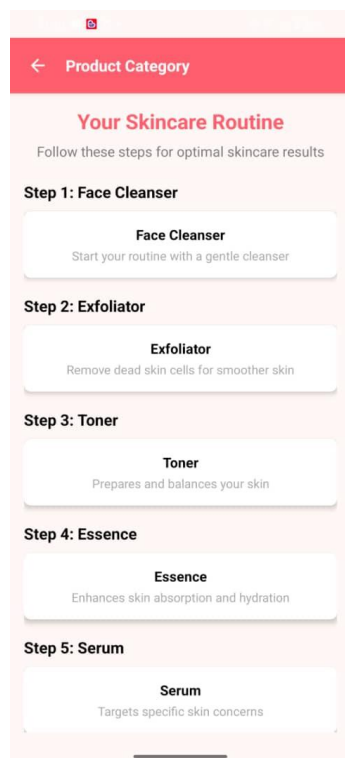


Figure 5.5.3.3 Product Category Selection Page

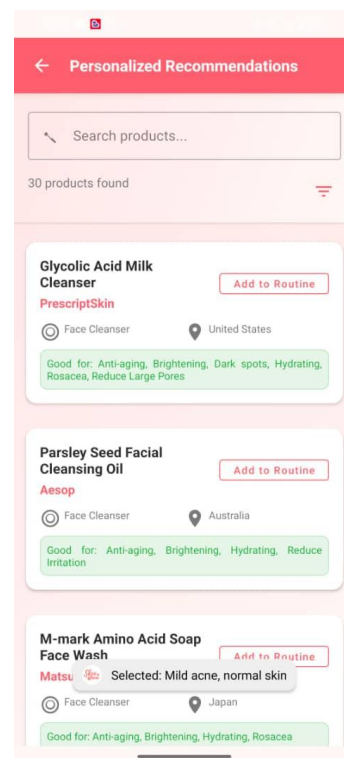


Figure 5.5.3.4 Filtered Face Cleanser Products

CHAPTER 5

On the product list page, the user can search for products by entering the product name or brand. For example, if the user enters the keyword “facial” in the product name field, the system will filter and display all products containing “facial” in their name, as shown in Figure 5.5.3.5. In this case, four products match the keyword. Similarly, if the user enters the brand name “face gym”, the system will filter and display the product associated with that brand, as shown in Figure 5.5.3.6.

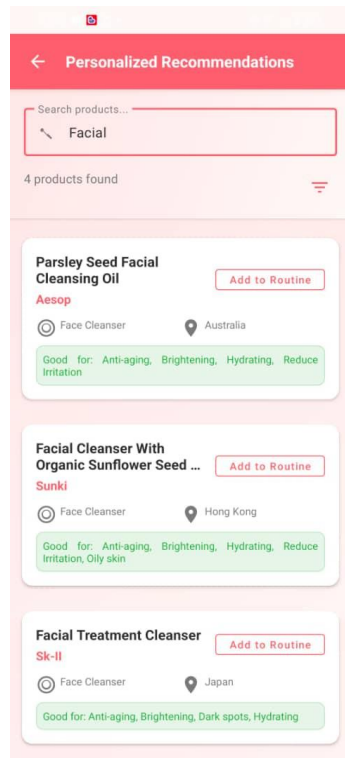


Figure 5.5.3.5 Filtered Products by Name (“facial”)

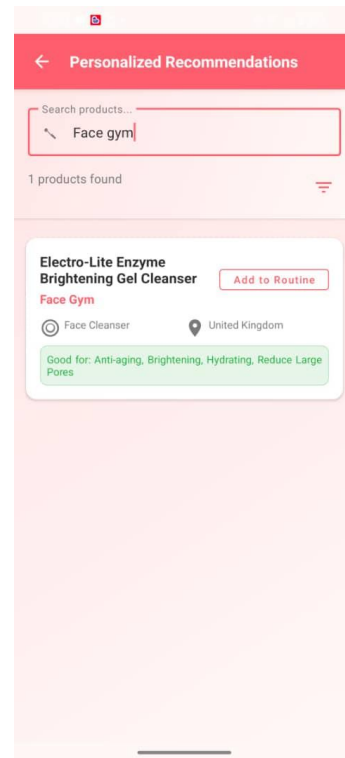


Figure 5.5.3.6 Filtered Products by Brand (“face gym”)

CHAPTER 5

The user can also filter products based on brand name and country by clicking on the Filter icon. After clicking the icon, the user can select the desired brand and country, as shown in Figure 5.5.3.7. Once the user has made their selections and clicks the OK button, the system will filter and display the corresponding products. For example, if the user selects the brand Dermalogica and the country Australia, the filtered products will be shown as in Figure 5.5.3.8. In this case, no products are found because there are no Dermalogica products from Australia.

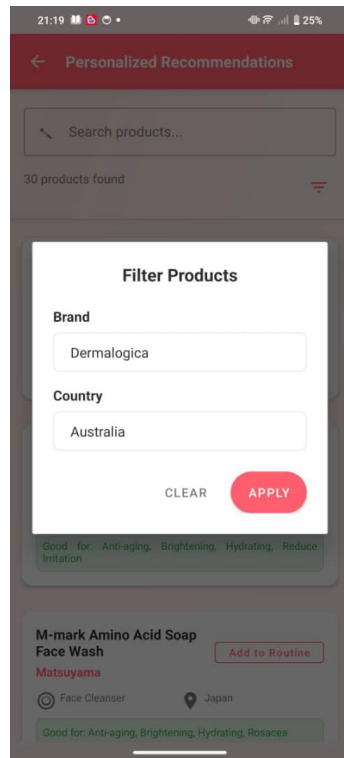


Figure 5.5.3.7 Filter Options for Brand and Country

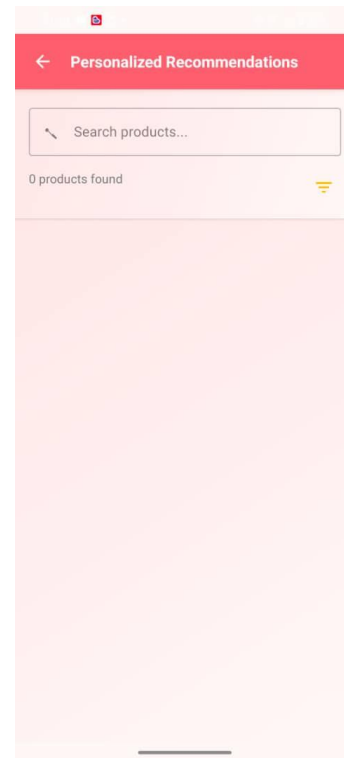


Figure 5.5.3.8 Filtered Products with No Results (Dermalogica, Australia)

CHAPTER 5

The user can select a product from the product list. For example, if the user selects the product named “Glycolic Acid Milk Cleanser”, the system will display the product details and trigger Gemini AI to provide recommendations, including where to buy, how to use, key benefits, and usage tips, as shown in Figure 5.5.3.9. The user can then choose a product link from different shopping platforms such as Lazada and Shopee. They can also select the Search Product Online option, which will redirect them to Google Search to find the product, as shown in Figure 5.5.3.10.

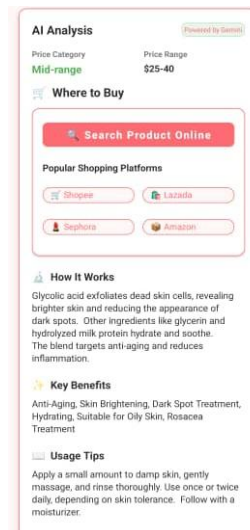
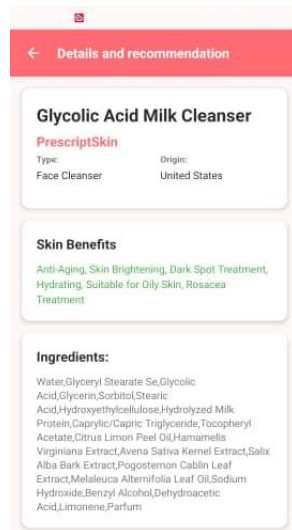


Figure 5.5.3.9 Product Details (upper half) Figure 5.5.3.10 product details(lower half)

The search result page redirected by the system to Google is shown in Figure 5.5.3.11.

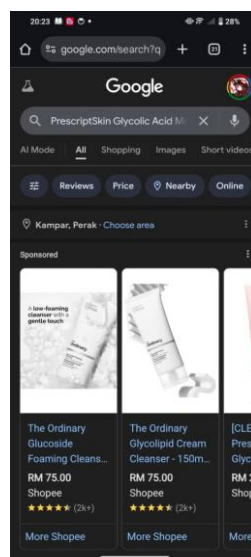


Figure 5.5.3.11 Redirected Google Search Results

5.5.4 Skincare Routine Module

The user can choose to view the skincare routine by clicking on the skincare routine icon inside the main menu. If the user has not created any skincare routine, a message saying “No routines yet” will be displayed. The user can then click on Create New Routine to add a new routine, as shown in Figure 5.5.4.1. After clicking the Create Routine button, the system will redirect the user to the create routine page, where they are required to fill in the routine name and optionally provide a routine description, as shown in Figure 5.5.4.2.

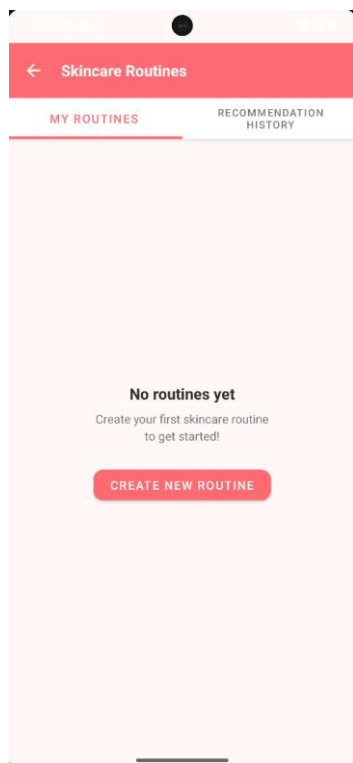


Figure 5.5.4.1 Skincare Routine Page with No Routine Created

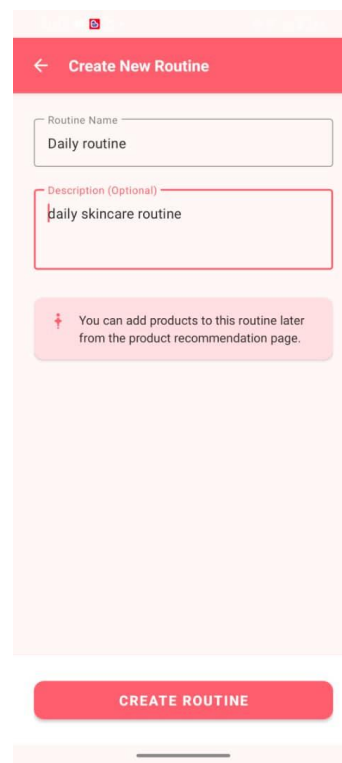


Figure 5.5.4.2 Create Routine Page

CHAPTER 5

Once the user clicks on **Save**, the system will create the routine and redirect them back to the skincare routine page. The newly created routine will then be displayed on the routine page, as shown in Figure 5.5.4.3. The user can then click on the created routine, and the system will redirect them to the selected routine, as shown in Figure 5.5.4.4.

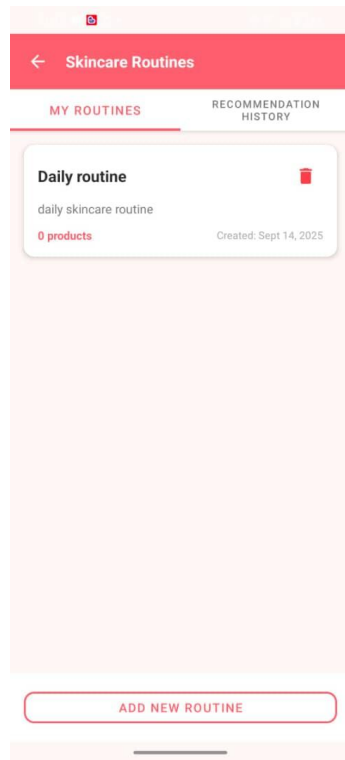


Figure 5.5.4.3 Skincare Routine Page
After Routine Creation

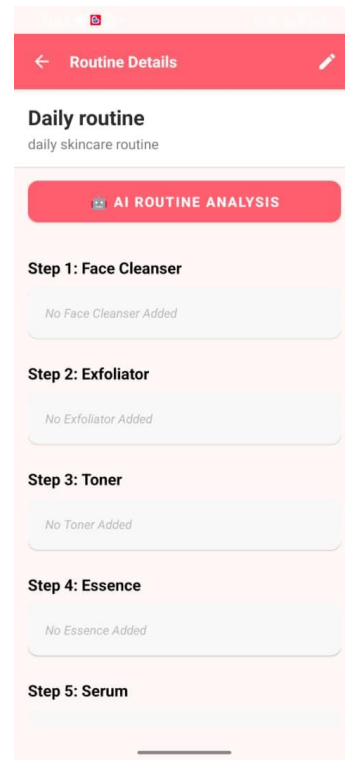


Figure 5.5.4.4 Viewing a Selected
Routine

CHAPTER 5

The user can choose to edit the routine name and its description by clicking on the icon at the top-right corner. For example, the user can change the routine name from “daily routine” to “daily routine 1”, as shown in Figure 5.5.4.5. Once the user saves the changes, the updated routine name will be displayed, as shown in Figure 5.5.4.6.

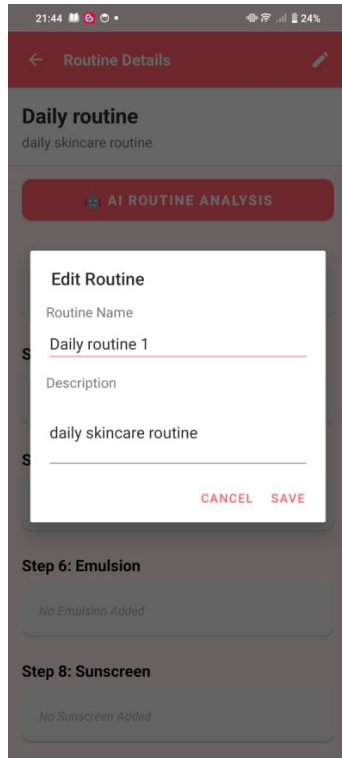


Figure 5.5.4.5 Editing Routine Name and Description

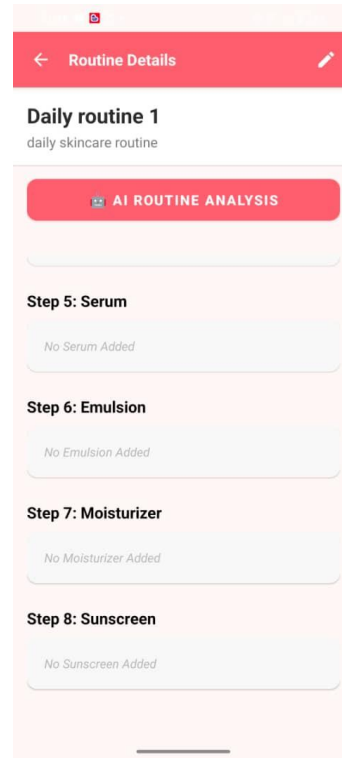


Figure 5.5.4.6 Updated Routine After Saving Changes

CHAPTER 5

Since there are no products inside the routine, the user can choose to add products from the product recommendation list by clicking on the **Add to Routine** button at the top-right corner. The user can then add the skincare product to an existing routine or create a new one, as shown in Figure 5.5.4.7. After the product has been added to the routine, the user can view the product name inside the routine details, as shown in Figure 5.5.4.8.

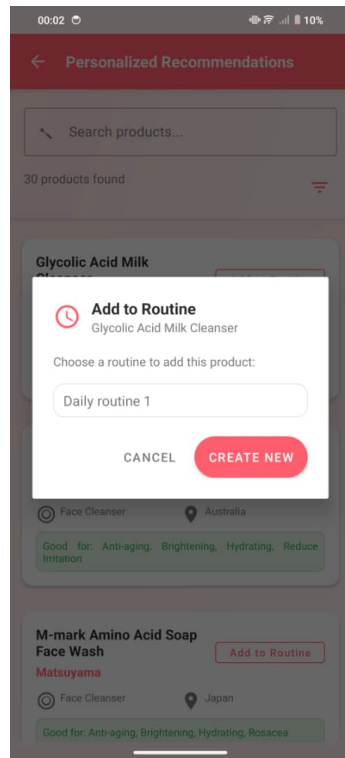


Figure 5.5.4.7 Adding a Product to a Routine

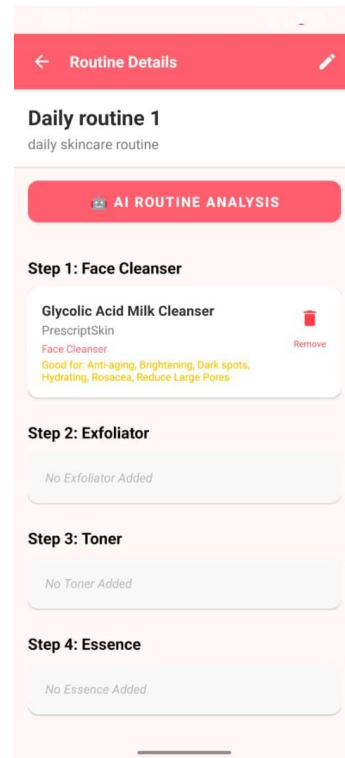


Figure 5.5.4.8 Routine Details Displaying Added Product

The user can also click on the product inside the routine details to view the product details similar as the product recommendation part. After at least one skincare product has been added to the selected routine, the user can perform the AI analysis by clicking on the AI Routine Analysis button. The result of the AI analysis will then be displayed, as shown in Figure 5.5.4.9 and 5.5.4.10.

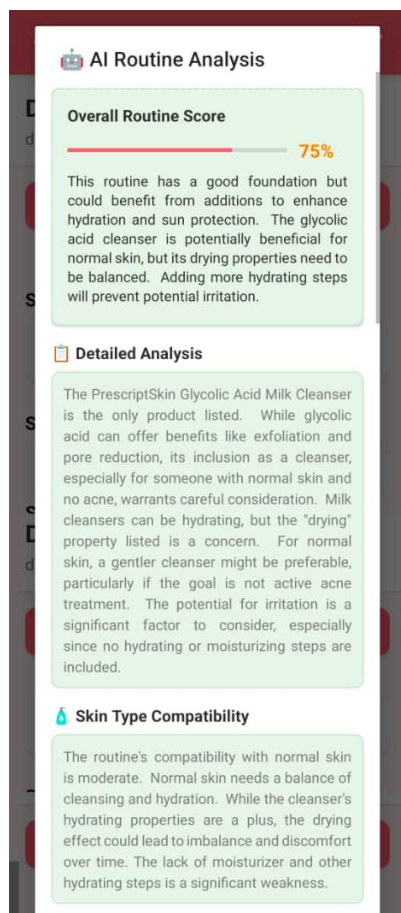


Figure 5.5.4.9 AI Routine Analysis
Result (upper half)

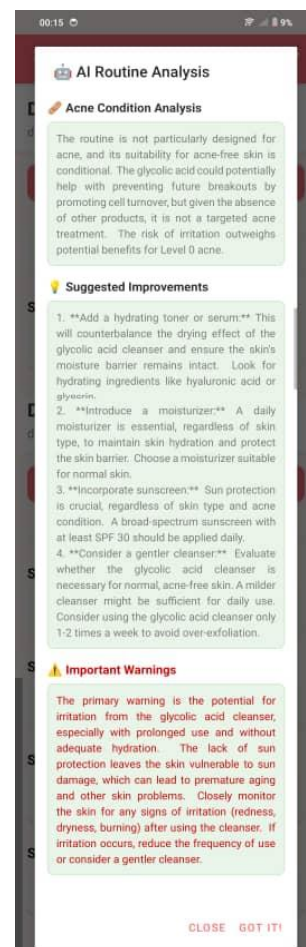


Figure 5.5.4.10 AI Routine Analysis
Result(lower half)

5.5.5 Skin Progress Module

When the user clicks on the Skin progress icon in the main menu, the system will redirect the user to the skin progress page. Since the user hasn't performed any skin analysis, it means there will be no data to show their skin progress. Therefore, the acne severity page and skin type page will show no skin acne severity data and no skin type data respectively, as shown in Figure 5.5.5.1 and Figure 5.5.5.2.

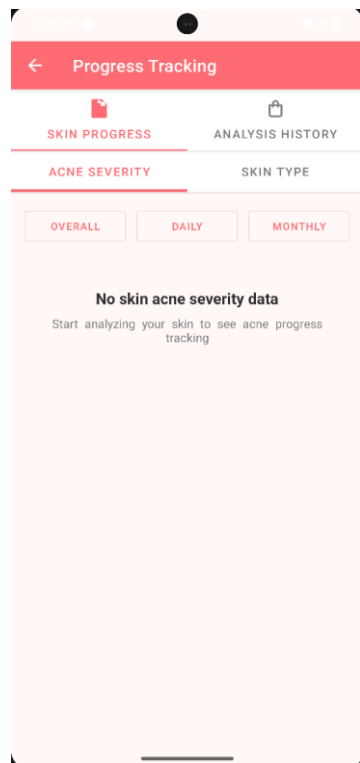


Figure 5.5.5.1 Acne progress page

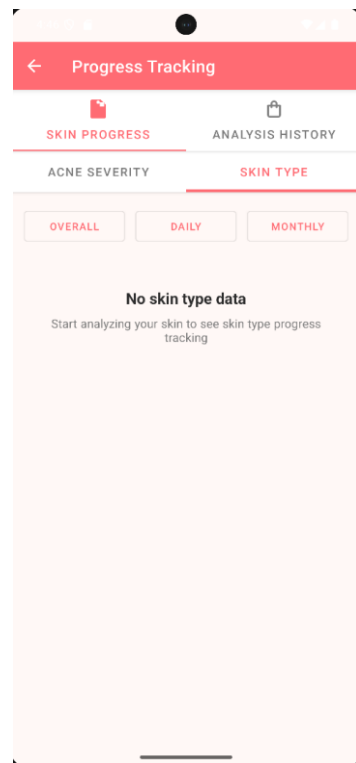


Figure 5.5.5.2 Skin type page

After the user performs a skin analysis, the data for skin progress will be displayed. The system will first show the overall skin acne severity graph and the skin acne severity accuracy graph. The user can choose to change the period of time to either daily or monthly. Figure 5.5.5.3 shows the overall period view. This is followed by an accuracy graph of the acne severity level. In addition, the AI will use the overall data to perform a descriptive analysis, providing key observations, recommendations, and suggested next steps based on the results, as shown in Figure 5.5.5.4.



Figure 5.5.5.3 Skin Acne Severity Graph (Overall Period)

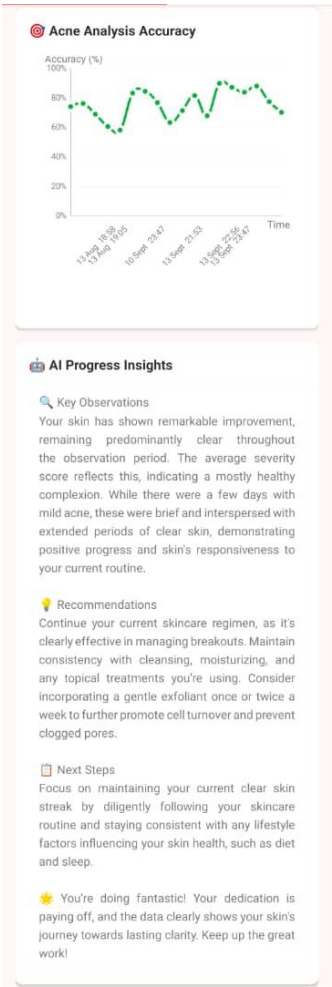


Figure 5.5.5.4 AI Descriptive Analysis with Recommendations

CHAPTER 5

The user can choose to view the daily period, as shown in Figure 5.5.5.5. The user can then switch to the Skin Type tab, where the system will display the skin type and its accuracy graph along with AI recommendations, as shown in Figure 5.5.5.6.



Figure 5.5.5.5 Daily Skin Acne Severity Graph

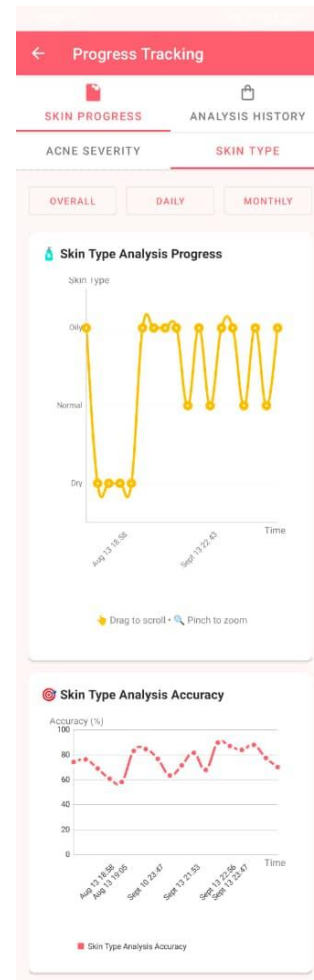
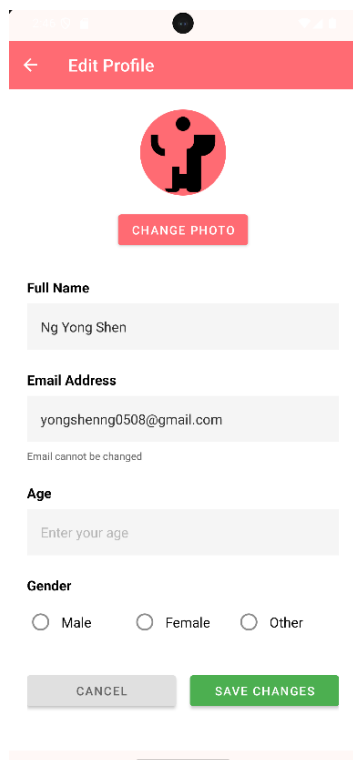


Figure 5.5.5.6 Skin Type Graph

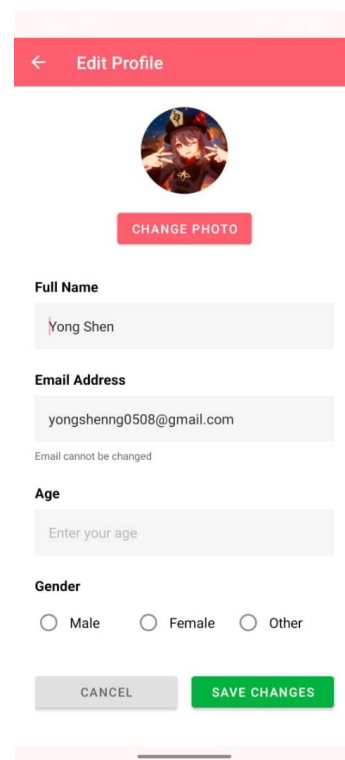
5.5.6 Manage Profile

The user can choose to manage their profile by clicking on the profile icon inside the main menu. After clicking the profile icon, the system will redirect the user to the edit profile page, where they can change their photo, full name, age, and gender. Then, the user only needs to click Save Changes to apply the updates or cancel to discard them. Figure 5.5.6.1 shows the edit profile page before any changes are made, while Figure 5.5.6.2 shows the edit profile page after changes have been applied.



The screenshot shows the 'Edit Profile' page with a red header bar containing a back arrow and the text 'Edit Profile'. Below the header is a circular profile icon with a red background and a black silhouette of a person. Underneath the icon is a red button labeled 'CHANGE PHOTO'. The form contains four sections: 'Full Name' with a text input field containing 'Ng Yong Shen'; 'Email Address' with a text input field containing 'yongshenng0508@gmail.com' and a small note below it stating 'Email cannot be changed'; 'Age' with a text input field containing the placeholder 'Enter your age'; and 'Gender' with three radio button options: 'Male', 'Female', and 'Other'. At the bottom are two buttons: a grey 'CANCEL' button and a green 'SAVE CHANGES' button.

Figure 5.5.6.1 Edit Profile Page (Before Changes)



The screenshot shows the 'Edit Profile' page after changes have been applied. The layout is identical to Figure 5.5.6.1, but the profile icon now shows a character from an anime. The 'Full Name' input field now contains 'Yong Shen' with a red cursor at the start. The 'Email Address' and 'Age' fields remain the same. The 'Gender' options are still 'Male', 'Female', and 'Other'. The 'CANCEL' and 'SAVE CHANGES' buttons are at the bottom.

Figure 5.5.6.2 Edit Profile Page (After Changes)

After the user saves the changes, the system will redirect them to the main menu. The updated information, including the user's full name and profile image, will be displayed in the main menu based on the changes made, as shown in Figure 5.5.6.3.

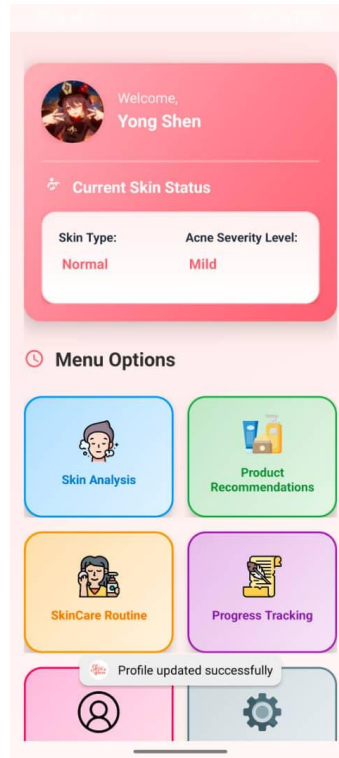


Figure 5.5.6.3 Updated User Profile Displayed in Main Menu

5.6 Implementation Issues and Challenges

There are several issues and challenges faced during the development of the project.

Firstly, one of the issues faced is that the model was unable to reach the desired classification accuracy. Despite applying the classic CNN model, other models were also tested to investigate the best performing model among them. The failure to achieve the expected accuracy may be caused by various factors, such as insufficient data in the dataset and the inability of the model to extract meaningful features from the images.

Besides, the OpenAI API platform posed a significant challenge due to the token limits imposed by the OpenAI API. Each conversation with the API consumes a certain number of tokens, and there is a maximum token limit. This limitation has impacted on the performance and completeness of the model. Therefore, further exploration of the OpenAI API documentation is needed to identify alternative ways to compute the classification model using this platform. Furthermore, learning how to estimate the tokens spent for each execution is important to ensure that the resources used stay within the budget set at the beginning. This will help avoid unexpected costs and ensure the smooth execution of the system.

Moreover, self-learning through the documentation of TensorFlow, Keras, PyTorch, and YOLO is essential for gaining a solid understanding of the functions and classes used in this project. A basic knowledge of these frameworks will make it easier to utilize them effectively. Additionally, tutorial videos from different platforms, such as YouTube, are helpful for enhancing knowledge, as they offer practical examples and demonstrations from experienced individuals.

In addition, the deployment of the TensorFlow Lite model into the mobile application is time-consuming because we need to import the required libraries and configure them correctly to avoid errors. This task requires knowledge from Android documentation and self-learning in order to build a solid understanding of both the TensorFlow Lite model deployment steps and the Android framework.

Furthermore, the integration of Firebase as the backend server for the mobile application to store data, and the Gemini API to provide users with valuable information, presents certain challenges. These challenges arise from the integration process and the methods required to connect both Firebase and the Gemini API. The first challenge we faced was managing the Firebase and Gemini credential key settings and storage methods to ensure that they are not stored in GitHub, preventing misuse by unauthorized individuals when the project becomes open source.

5.7 Concluding Remark

The core features within the skin analysis and recommendation system have been successfully developed and integrated to ensure smooth interaction across different modules. Each module communicates effectively with the others, enabling data and results to flow seamlessly throughout the system. As a result, users can move between various interfaces such as skin analysis, product recommendations, skincare routine management, and progress tracking without disruption.

Chapter 6

System Evaluation and Discussion

6.1 Result Discussion for AI models

6.1.1 Classic CNN models

Based on Figure 6.1.1.1, the classic CNN model achieves the highest training accuracy of 0.5175 at Epoch 18 and the highest validation accuracy of 0.5000 at Epoch 19. However, the validation accuracy at epoch 18 is 0.4800, and the training accuracy at epoch 19 is 0.4534. Although epoch 18 achieves better training accuracy compared to epoch 19, its validation accuracy is lower than that of epoch 19. This means that epoch 19 can achieve better performance on unseen data compared to epoch 18.

Since the accuracy of the model remains below 0.5000, this may indicate that the architecture of the classic CNN model is not complex enough and requires hyperparameter tuning, such as adjusting the dropout rate, learning rate, or number of epochs, to improve the model's performance.

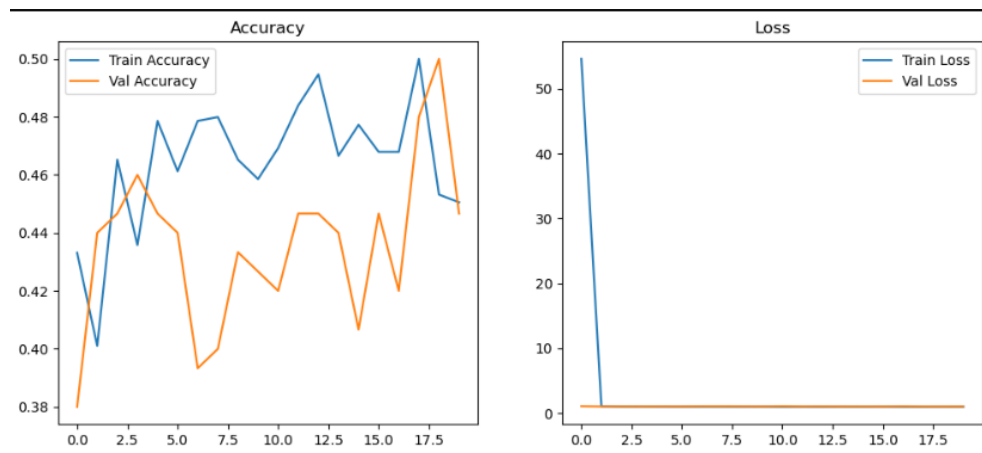


Figure 6.1.1.1 Training and validation accuracy and loss across epochs for classic CNN skin acne model

In Figure 6.1.1.2 shows that the confusion matrix of classic CNN on the test set for the skin acne dataset. The figure indicates that 44 out of 109 test images were correctly classified.

CHAPTER 6

For Level 0, 9 out of 39 images were accurately predicted, while 30 were misclassified as Level 1, resulting in a recall of 0.230. Of the 23 images predicted as Level 0, 13 images belong to Level 1, and 1 image belongs to Level 2, resulting in a precision of 0.390.

For Level 1, 35 out of 48 images were correctly classified, with 13 misclassified as Level 0, resulting in a recall of 0.790. Among the 78 images predicted as Level 1, 30 images belong to Level 0, and 13 images belong to Level 2, leading to a precision of 0.490.

For Level 2, no images were correctly predicted, with 1 misclassified as Level 0 and 13 images misclassified as Level 1, giving a recall of 0.000. No images were predicted as Level 2, resulting in a precision of 0.000.

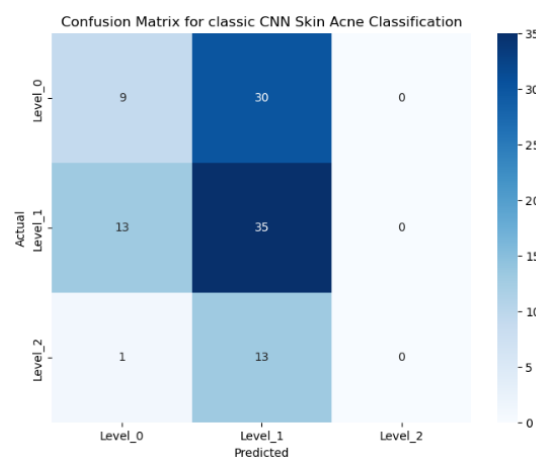


Figure 6.1.1.2 confusion matrix for classic CNN skin acne classification

Table 6.1.1.1 shows the results of precision, recall, and F1-score of the ResNet50 model on the test set for the skin acne dataset.

The F1-score for the Level 0 class is 0.290, indicating that the model has a low recall of 0.230 and low precision of 0.390. This suggests that many images are misclassified as other levels, and many images from other levels are predicted as Level 0.

CHAPTER 6

The F1 score for the Level 1 class is 0.600, which is the highest among the three classes. With a precision of 0.790 and a recall of 0.490, it shows that the model has a strong ability to predict Level 1 images, as it has high precision.

The F1-score for the Level 2 class is 0.000, as it also achieves the same score for both precision and recall, meaning the model failed to identify the images from this class.

Table 6.1.1.1 classification report for Classic CNN skin acne model

	precision	recall	f1-score	support
Level_0	0.39	0.23	0.29	39
Level_1	0.49	0.79	0.60	48
Level_2	0.00	0.00	0.00	14
accuracy			0.47	101
macro avg	0.29	0.34	0.30	101
weighted avg	0.38	0.47	0.40	101

In Figure 6.1.1.3 shows the training accuracy of classic CNN model around 0.40 to 0.50, while the validation accuracy is between 0.30 until 0.43. It achieves the highest validation accuracy of 0.4300 at epoch 10 while the highest training accuracy of 0.4977 at epoch 7. Compared with the training accuracy of 0.4533 at epoch 10 with the validation accuracy of 0.4200 at epoch 7, epoch 10 is more favourable than epoch 7. This is because epoch 10 achieves higher validation accuracy compared with epoch 7, which considers that it has better generalization on handling unseen data. The training accuracy and validation accuracy of epoch 10 also more consistent as compared with epoch 7, indicates that epoch 10 represents the better performance overall.

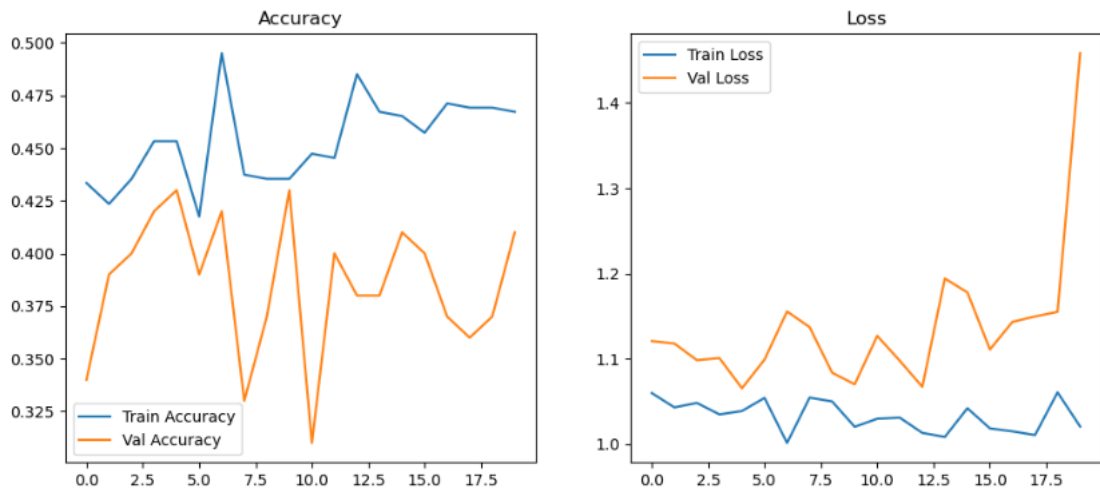


Figure 6.1.1.3 Training and validation accuracy and loss across epochs for classic CNN skin type model

In Figure 6.1.1.4 shows that the confusion matrix of the classic CNN on the test set for the skin type dataset. The figure indicates that 25 out of 60 test images were correctly classified.

For **Dry**, 9 out of 21 images were correctly classified, meaning that 10 images were misclassified as Normal and 2 images as Oily, resulting in a **recall of 0.430**. Among the 22 images predicted as Normal, 3 belonged to Dry, and 3 to Oily, giving a **precision of 0.320**.

For **Normal**, 12 out of 27 images were correctly classified, meaning that 10 images were misclassified as Dry and 5 as Oily, resulting in a **recall of 0.440**. Of the 30 images predicted as Normal, 10 were Dry and 8 were Oily, leading to a **precision of 0.400**.

For **Oily**, 4 out of 21 images were correctly classified, meaning that 9 images were misclassified as Dry and 8 as Normal, resulting in a **recall of 0.190**. Out of the 11 images predicted as Oily, 2 belonged to Dry and 5 to Normal, giving a **precision of 0.360**.

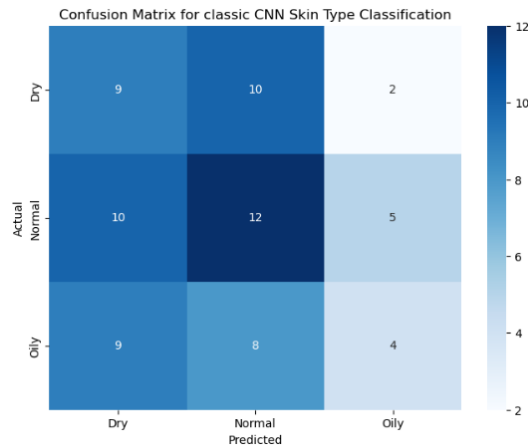


Figure 6.1.1.4 confusion matrix for classic CNN skin type classification

Table 6.1.1.2 presents the precision, recall, and F1-score results of the ResNet50e model on the test set for the skin acne dataset.

For the **Dry** class, the F1-score is **0.370**, with a precision of **0.32** and a recall of **0.43**. This relatively low F1-score indicates that the model frequently misclassified images belonging to the Dry class and often predicted images from other classes as Dry.

For the **Normal** class, the F1-score is **0.420**, representing the most balanced and slightly better performance among the three classes. This is supported by its precision of **0.40** and recall of **0.44**.

For the **Oily** class, the F1-score is **0.150**, indicating the poorest performance among the three classes. Although it achieved a precision of **0.36**, the low recall of **0.19** suggests that the model misclassified most images that actually belong to this class.

Table 6.1.1.2 classification report for Classic CNN skin type model

	precision	recall	f1-score	support
Dry	0.32	0.43	0.37	21
Normal	0.40	0.44	0.42	27
oily	0.36	0.19	0.25	21
accuracy			0.36	69
macro avg	0.36	0.35	0.35	69
weighted avg	0.37	0.36	0.35	69

6.1.2 ResNet50 models

Based on Figure 6.1.2.1, the ResNet50 model achieves its highest validation accuracy of **69.52%** at epoch 17, indicating optimal performance on the validation data. In comparison, the highest training accuracy of **69.65%** is reached at epoch 16. Although these peak accuracies occur at different epochs, epoch 17 is considered more favourable because its validation accuracy is higher than that at epoch 16, which is only 66.71%, while its training accuracy remains reasonably high at 67.65%. This suggests that the model performs more consistently on unseen data at epoch 17, despite a slightly lower training accuracy compared to epoch 16.

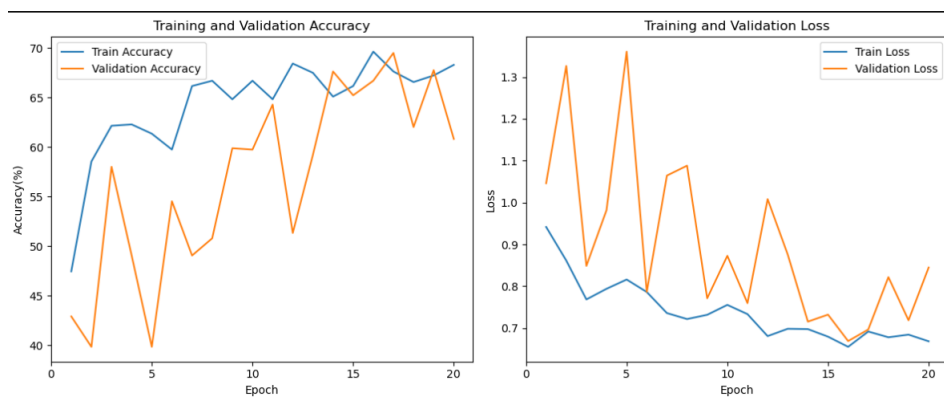


Figure 6.1.2.1 Training and validation accuracy and loss across epochs for ResNet50 skin acne model

In Figure 6.1.2.2 shows that the confusion matrix of the EfficientNetB0 on the test set of the skin acne dataset. The figure indicates that 61 out of 109 test images were correctly classified.

For Level 0, 38 out of 39 images were accurately predicted, while 1 image was misclassified as Level 1, resulting in a recall of 0.974. Among the 69 images predicted as Level 0, 30 actually belonged to Level 1 and 1 belonged to Level 2, resulting in a precision of 0.551.

For Level 1, 18 out of 48 images were correctly classified, with 30 images misclassified as Level 0, resulting in a recall of 0.375. Among the 27 images predicted as Level 1, 1 belonged to Level 0 and 8 belonged to Level 2, leading to a precision of 0.667.

For Level 2, 5 out of 14 images were correctly predicted, with 1 image misclassified as Level 1 and 8 images misclassified as Level 0, resulting in a recall of 0.357. Notably, all 5 images predicted as Level 2 truly belonged to that class, meaning no incorrect images were predicted as Level 2, resulting in a precision of 1.000.

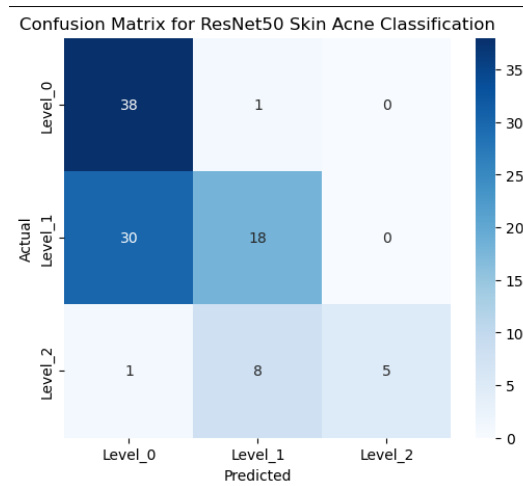


Figure 6.1.2.2 confusion matrix for ResNet50 skin acne classification

Table 6.1.2.1 presents the precision, recall, and F1-score results of the ResNet50 model on the test set of the skin acne dataset.

For the F1-score in **Level 0** class is **0.704**, indicating that the model is effective at identifying actual Level 0 images, as reflected by the high **recall of 0.974**. However, the **precision of 0.551** suggests that many images predicted as Level 0 actually belong to other classes, which affects the overall performance.

For the F1-score in **Level 1** class is **0.480**, reflecting the lowest performance among the three classes. With a **precision of 0.667** and a **recall of 0.375**, this result indicates that while the model is precise when predicting Level 1 images, it often fails to identify all actual Level 1 images, leading to a lower recall.

For the F1-score in **Level 2** class is **0.526**, representing moderate performance among the three classes. Although the **precision is perfect at 1.000**, meaning every image predicted as Level 2 truly belongs to that class, the low **recall of 0.357** indicates that the model misses many actual Level 2 images, impacting overall prediction quality.

Table 6.1.2.1 classification report for ResNet50 skin acne classification

Classification Report:				
	precision	recall	f1-score	support
Level_0	0.551	0.974	0.704	39
Level_1	0.667	0.375	0.480	48
Level_2	1.000	0.357	0.526	14
accuracy			0.604	101
macro avg	0.739	0.569	0.570	101
weighted avg	0.668	0.604	0.573	101

Based on Figure 6.1.2.3 the ResNet50 model achieves its highest training accuracy of **60.83%** and validation accuracy of **56.06%** at epoch 19. From the previous epochs, both the training accuracy and validation accuracy steadily increase, despite certain epochs where noticeable drops in accuracy occur. However, the overall trend shows an increase in accuracy. This suggests that, despite occasional fluctuations, the model continues to learn and improve. After reaching its highest accuracy at epoch 19, there is a slight drop in performance at the final epoch, with a training accuracy of **59.24%** and a validation accuracy of **55.27%**. While this drop is small, it indicates that further training could help the model achieve even better results if additional epochs were available.



Figure 4.5.2.3 Training and validation accuracy and loss across epochs for ResNet50 skin type model

CHAPTER 6

In Figure 6.1.2.4 shows that the confusion matrix of of the YOLOv8 model on the test set for the skin type dataset. The confusion matrix indicates that 25 out of 60 test images were correctly classified.

For Dry, 2 out of 21 images were correctly classified, meaning that 10 images were misclassified as Normal and 9 images as Oily, resulting in a recall of 0.095. Among the 3 images predicted as Dry, 1 image actually belonged to Dry, giving a precision of 0.667.

For Normal, 21 out of 27 images were correctly classified, with 1 image misclassified as Dry and 5 images misclassified as Oily, resulting in a recall of 0.778. Among the 22 images predicted as Normal, 3 images belonged to Dry and 3 images belonged to Oily, resulting in a precision of 0.512.

For Oily, 11 out of 21 images were correctly classified, with 10 images misclassified as Normal, resulting in a recall of 0.524. Among the 25 images predicted as Oily, 9 images belonged to Dry, and 5 images belonged to Normal, giving a precision of 0.440.

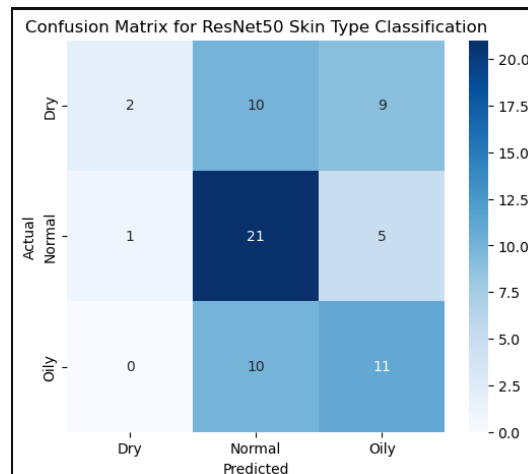


Figure 6.1.2.4 confusion matrix for ResNet50 skin type classification

Table 6.1.2.2 shows the results of precision, recall, and F1-score for the test set of the skin type dataset.

The F1-score for the **Dry skin** class is **0.167**, which reflects the lowest F1-score among the three classes. This is caused by a very low recall of **0.095**, despite a relatively high

precision of **0.667**. This indicates that a significant number of Dry skin images are misclassified.

The F1-score for the **Normal skin** class is **0.618**, which is relatively better than the other two classes, Dry and Oily skin types. It has a higher precision of **0.778** and a recall of **0.593**, which is slightly lower. This highlights that the model often predicts images from other classes as Normal skin type.

The F1-score for the **Oily skin** class is **0.478**, indicating moderate performance among the three classes. It has a recall of **0.524** and precision of **0.440**, which suggests that the Oily skin class is often misclassified as other classes, and other skin types are frequently predicted as Oily skin type.

Table 6.1.2.2 classification report for ResNet50 skin type classification

Classification Report:				
	precision	recall	f1-score	support
Dry	0.667	0.095	0.167	21
Normal	0.512	0.778	0.618	27
Oily	0.440	0.524	0.478	21
accuracy			0.493	69
macro avg	0.540	0.466	0.421	69
weighted avg	0.537	0.493	0.438	69

6.1.3 EfficientNetB0 model

Based on Figure 6.1.3.1, the EfficientNetB0 model achieves its highest validation accuracy of **0.6667** at epoch **15**, meaning that it is performing optimally on the validation data. The training accuracy keeps improving throughout the epoch. However, the validation accuracy does not follow the same trend as the training accuracy.

Upon reaching epoch 20, the validation accuracy decreases to **0.6333**, suggesting that the model begins to overfit starting from epoch 15. This emphasizes the importance of stopping training at epoch 15, as it represents the peak performance. Figure 4.5.3(a)

also shows that the validation accuracy reaches its optimum performance at epoch 15, and the validation loss is at its lowest at epoch 15.

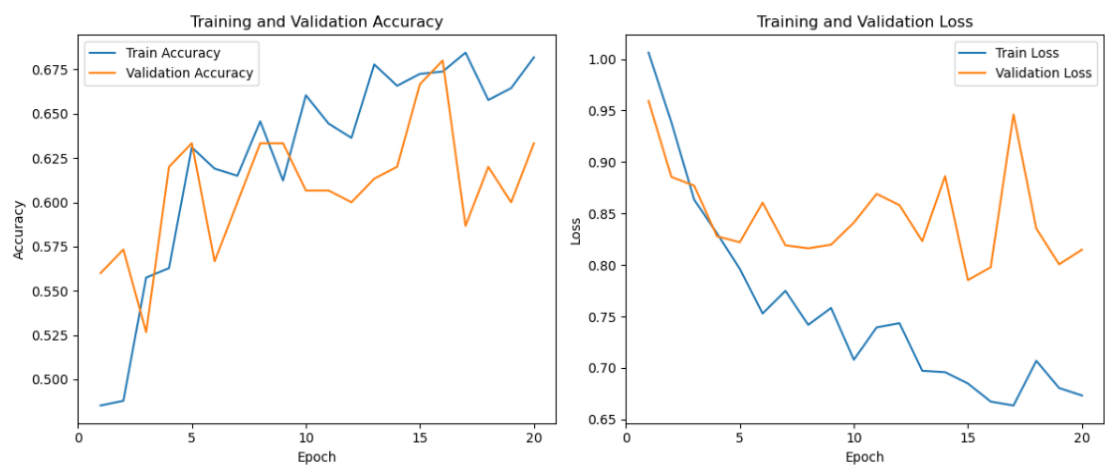


Figure 6.1.3.1 Training and validation accuracy and loss across epochs for EfficientNetB0 skin acne model

In Figure 6.1.3.2 presents the confusion matrix of EfficientNetB0 on the test set for the skin acne dataset. The figure indicates that 72 out of 109 test images were correctly classified.

For Level 0, 32 out of 39 images were correctly predicted, with 7 misclassified as Level 1, resulting in a **recall** of 0.821. From 46 images predicted as Level 0, 13 images belonged to Level 1 and 1 image belonged to Level 2, resulting in a **precision** of 0.696. For Level 1, 34 out of 48 images were correctly classified as Level 1 images, meaning that 7 images were misclassified as Level 0 and 7 images were misclassified as Level 2, resulting in a **recall** of 0.708. From 46 images predicted as Level 1, 13 images belonged to Level 0 and 1 image belonged to Level 2, resulting in a **precision** of 0.708.

For Level 2, 6 out of 14 images were correctly predicted as Level 2, meaning that 7 images were misclassified as Level 1 and 1 image was misclassified as Level 0, resulting in a **recall** of 0.429. From 7 images predicted as Level 2, 1 image belonged to Level 1, resulting in a **precision** of 0.857.

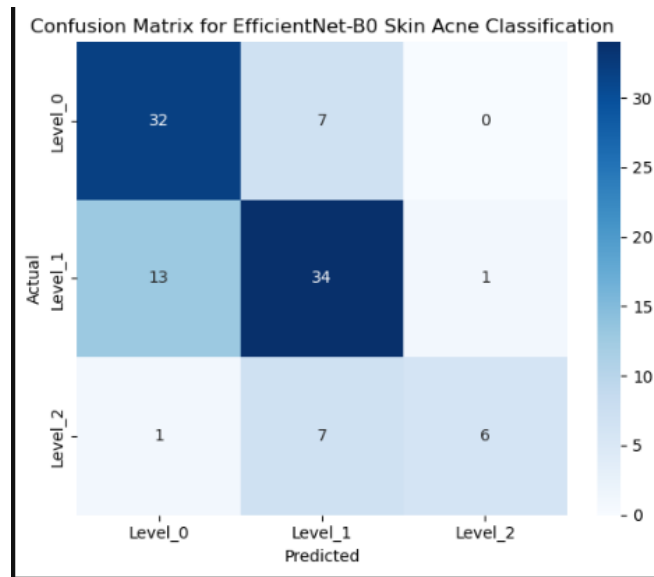


Figure 6.1.3.2 confusion matrix for EfficientNetB0 skin acne classification

From Table 6.1.3.1, the results of precision, recall, and F1-score on the test set for the skin acne dataset are shown.

For the F1-score in the Level 0 class, it is 0.753, indicating that the model is able to identify the actual Level 0 images well using EfficientNetB0, as it has a high recall of 0.821. However, its precision of 0.696 suggests that images from other classes are incorrectly predicted as Level 0 images.

For the F1-score in the Level 1 class, the model achieved balanced values across the three metrics: precision, recall, and F1-score, all resulting in an accuracy of 0.708. This balanced accuracy indicates that the model is not favoring false positives or false negatives in this class, leading to stable and reliable classification.

For the F1-score in the Level 2 class, it is 0.571, reflecting the lowest performance among the three classes. Although the precision is the highest at 0.857, the recall is significantly lower at 0.429. This imbalance of low recall and high precision contributed to the lowest F1-score, indicating that the model struggles to classify Level 2 images.

Table 6.1.3.1 classification report for EfficientNetB0 skin acne model

Classification Report:				
	precision	recall	f1-score	support
Level_0	0.696	0.821	0.753	39
Level_1	0.708	0.708	0.708	48
Level_2	0.857	0.429	0.571	14
accuracy			0.713	101
macro avg	0.754	0.652	0.678	101
weighted avg	0.724	0.713	0.707	101

Based on Figure 6.1.3.3, the EfficientNetB0 model on the skin type dataset achieves the highest validation accuracy of 0.5600 at epoch 17, while the training accuracy is 0.7995, indicating that the model performs optimally at this epoch. After this epoch, the validation accuracy does not follow the same trend as the training accuracy.

By epoch 20, the validation accuracy decreases to 0.5300, suggesting that the model begins to overfit starting at epoch 17. This emphasizes the importance of stopping training when reaching epoch 17, as it marks the peak performance. Figure 4.5.3(d) also shows that the validation accuracy reaches its optimum performance at epoch 17, and the validation loss is the lowest at this epoch.

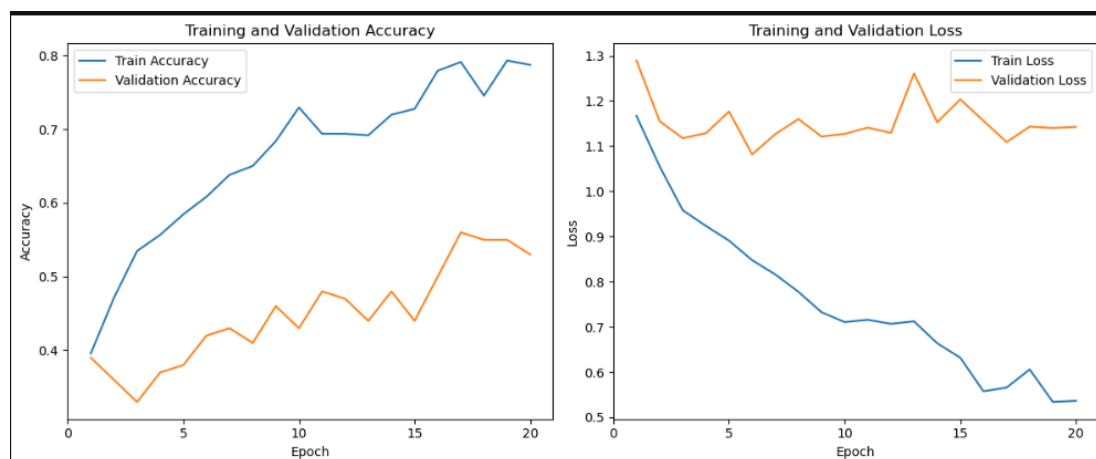


Figure 4.5.3.3 Training and validation accuracy and loss across epochs for EfficientNetB0 skin type model

CHAPTER 6

In Figure 6.1.3.4 presents the confusion matrix of the EfficientNetB0 on the test set for the skin type dataset. The figure indicates that 39 out of 63 test images were correctly classified.

For Dry, 12 out of 21 images were correctly predicted, with 2 misclassified as Normal and 7 as Oily, resulting in a **recall** of 0.571. Out of 26 images predicted as Dry, 7 belonged to Normal and 7 to Dry, giving a **precision** of 0.462.

For Normal, 13 out of 27 images were correctly classified as Normal, with 7 misclassified as Dry and 7 as Oily, resulting in a **recall** of 0.481. Out of 15 images predicted as Normal, 2 belonged to Dry, yielding a **precision** of 0.867.

For Oily, 14 out of 21 images were correctly predicted, with 1 image misclassified as Dry, resulting in a **recall** of 0.667. Out of 28 images predicted as Oily, 7 belonged to Dry and 7 to Normal, resulting in a **precision** of 0.500.

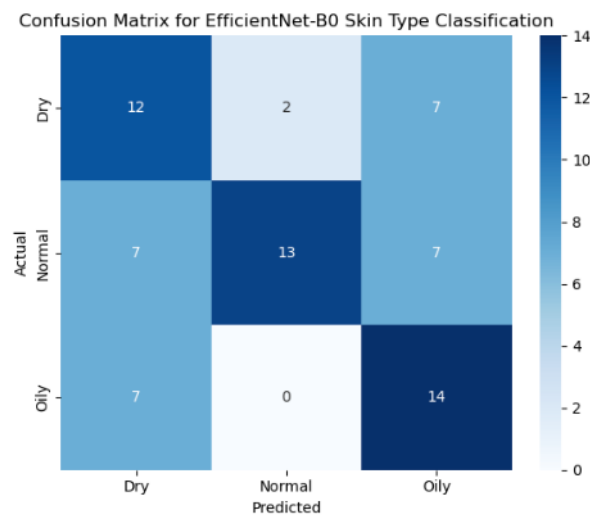


Figure 6.1.3.4 confusion matrix for EfficientNetB0 skin type classification

Table 6.1.3.2 shows the precision, recall, and F1-score results on the test set for the skin type dataset.

For the F1-score in Dry class is 0.511, which is the lowest among the classes. Although the recall is 0.571, suggesting that the model correctly identified more than half of the

actual Dry images, the lower precision of 0.462 shows that several images from other classes were misclassified as Dry.

For the F1-score in Normal class is 0.619, the highest among the classes. This indicates that it achieved the best performance overall. With a high precision of 0.867 but a lower recall of 0.481, the model was very accurate when predicting Normal images but misclassified many actual Normal images as other classes.

For the F1-score in Oily class is 0.571, with a recall of 0.667 and a precision of 0.500. This indicates that the model was able to detect a fair number of actual Oily images, but it also made more false positives, which affected the overall classification quality.

Table 6.1.3.2 classification report for EfficientNetB0 skin type classification

Classification Report:				
	precision	recall	f1-score	support
Dry	0.462	0.571	0.511	21
Normal	0.867	0.481	0.619	27
Oily	0.500	0.667	0.571	21
accuracy			0.565	69
macro avg	0.609	0.573	0.567	69
weighted avg	0.632	0.565	0.572	69

6.1.4 YOLOv8 model

The accuracy of YOLOv8 for skin acne classification reached 80% for both the train set and validation set, as shown in Figure 6.1.4.1. The accuracy for the test set achieved 76.2%, as shown in Figure 6.1.4.2.

```

ultralytics.utils.metrics.ClassifyMetrics object with attributes:
confusion_matrix: <ultralytics.utils.metrics.ConfusionMatrix object at 0x0000022C168D2C30>
curves: []
curves_results: []
fitness: 0.9000000059604645
keys: ['metrics/accuracy_top1', 'metrics/accuracy_top5']
results_dict: {'metrics/accuracy_top1': 0.800000011920929, 'metrics/accuracy_top5': 1.0, 'fitness': 0.9000000059604645}
save_dir: WindowsPath('C:/Users/yongs/runs/classify/train35')
speed: {'preprocess': 0.30614599999997455, 'inference': 0.4861426666661828, 'loss': 0.006658666666605105, 'postprocess': 0.022021999999980318}
task: 'classify'
top1: 0.800000011920929
top5: 1.0

```

Figure 6.1.4.1 YOLOv8 skin acne classification accuracy on training and validation sets

```

ultralitics.utils.metrics.ClassifyMetrics object with attributes:
confusion_matrix: <ultralitics.utils.metrics.ConfusionMatrix object at 0x0000022D17E38F80>
curves: []
curves_results: []
fitness: 0.8811881244182587
keys: ['metrics/accuracy_top1', 'metrics/accuracy_top5']
results_dict: {'metrics/accuracy_top1': 0.7623762488365173, 'metrics/accuracy_top5': 1.0, 'fitness': 0.8811881244182587}
save_dir: WindowsPath('C:/Users/yongs/runs/classify/train352')
speed: {'preprocess': 0.3010237623757888, 'inference': 3.145549504951629, 'loss': 0.000928712871868946, 'postprocess': 0.001831683167069113}
task: 'classify'
top1: 0.7623762488365173
top5: 1.0

```

Figure 6.1.4.2 YOLOv8 skin acne classification accuracy on test set

In Figure 6.1.4.3 presents the confusion matrix of YOLOv8 on the test set for the skin acne dataset. The figure indicates that 77 out of 109 test images were correctly classified.

For Level 0, 34 out of 39 images were correctly predicted, with 5 misclassified as Level 1, resulting in a recall of 0.872. Of the 44 images predicted as Level 0, 10 images belonged to Level 1, resulting in a precision of 0.773.

For Level 1, 35 out of 48 images were correctly classified as Level 1, meaning that 10 images were misclassified as Level 0 and 3 images were misclassified as Level 2, resulting in a recall of 0.729. Of the 46 images predicted as Level 1, 5 images belonged to Level 0 and 6 images belonged to Level 2, resulting in a precision of 0.761.

For Level 2, 8 out of 14 images were correctly predicted as Level 2, meaning that 6 images were misclassified as Level 1, resulting in a recall of 0.571. Of the 11 images predicted as Level 2, 3 images belonged to Level 1, resulting in a precision of 0.727.

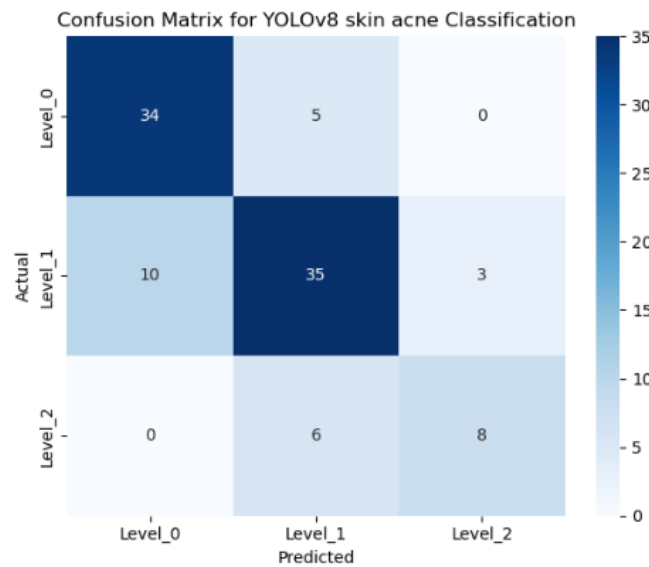


Figure 6.1.4.3 confusion matrix for YOLOv8 skin acne classification

Table 6.1.4.1 shows the precision, recall, and F1-score results on the test set for the skin acne dataset.

The F1-score for the Level 0 class is 0.819, which indicates that the model is good at identifying Level 0 images, as there is a good balance between precision and recall, contributing to the strong performance of the model in classifying Level 0 images.

The F1-score for the Level 1 class is 0.745, showing decent performance compared to Level 0. However, its relatively low precision and recall cause some Level 1 images to be misclassified as images from other classes.

The F1-score for the Level 2 class is 0.640, indicating the lowest performance among the three classes. Its recall and precision are the lowest compared to the others, suggesting that Level 2 images are frequently misclassified as images from other classes. This is also due to the insufficient number of images in this class.

Table 6.1.4.1 Precision, recall and F1-score for YOLOv8 skin acne classification

Class	Precision	Recall	F1-Score
Level_0	0.773	0.872	0.819
Level_1	0.761	0.729	0.745

Level_2	0.727	0.571	0.640
---------	-------	-------	-------

The accuracy of YOLOv8 for skin acne classification achieved 69.9% on the train and validation sets, as shown in Figure 6.1.4.4. The accuracy on the test set was 63.7%, as shown in Figure 6.1.4.5.

```
confusion_matrix: <ultralytics.utils.metrics.ConfusionMatrix object at 0x0000013840E1FDA0>
curves: []
curves_results: []
fitness: 0.8499999940395355
keys: ['metrics/accuracy_top1', 'metrics/accuracy_top5']
results_dict: {'metrics/accuracy_top1': 0.699999988079071, 'metrics/accuracy_top5': 1.0, 'fitness': 0.8499999940395355}
save_dir: WindowsPath('C:/Users/yongs/runs/classify/train36')
speed: {'preprocess': 0.21914999999353313, 'inference': 0.7465540000021065, 'loss': 0.009320999997726176, 'postprocess': 0.023448000001735636}
task: 'classify'
top1: 0.699999988079071
top5: 1.0
```

Figure 6.1.4.4 YOLOv8 skin type classification accuracy on training and validation sets

```
confusion_matrix: <ultralytics.utils.metrics.ConfusionMatrix object at 0x00000138CFCCE00>
curves: []
curves_results: []
fitness: 0.8188405930995941
keys: ['metrics/accuracy_top1', 'metrics/accuracy_top5']
results_dict: {'metrics/accuracy_top1': 0.6376811861991882, 'metrics/accuracy_top5': 1.0, 'fitness': 0.8188405930995941}
save_dir: WindowsPath('C:/Users/yongs/runs/classify/train362')
speed: {'preprocess': 0.4240623188311672, 'inference': 4.425785507240674, 'loss': 0.0011521739076630656, 'postprocess': 0.0029188405753065244}
task: 'classify'
top1: 0.6376811861991882
top5: 1.0
```

Figure 6.1.4.5 YOLOv8 skin type classification accuracy on test set

Figure 6.1.4.6 presents the confusion matrix of the YOLOv8 model on the test set for the skin type dataset. It shows that 44 out of 60 test images were correctly classified.

For Dry, 15 out of 21 images were correctly classified, meaning that 3 images were misclassified as Normal and 3 images as Oily, resulting in a recall of 0.714. Out of 28 images predicted as Dry, 8 images belonged to Normal, and 5 images belonged to Oily, resulting in a precision of 0.536.

For Normal, 16 out of 27 images were correctly classified, meaning that 8 were misclassified as Dry and 3 images were misclassified as Oily, resulting in a recall of 0.593. Out of 22 images predicted as Normal, 3 images belonged to Dry, and 3 images belonged to Oily, resulting in a precision of 0.727.

CHAPTER 6

For Oily, 13 out of 21 images were correctly classified, meaning that 5 images were misclassified as Dry and 3 images were misclassified as Normal, resulting in a recall of 0.619. Out of 19 images predicted as Oily, 3 images belonged to Dry, and 3 images belonged to Normal, resulting in a precision of 0.684.

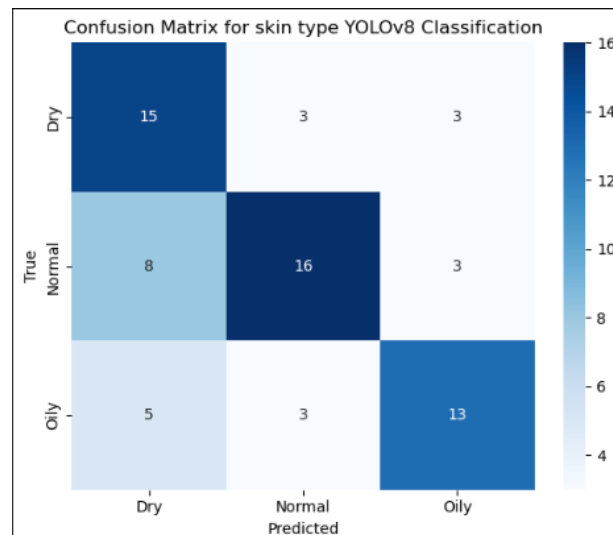


Figure 6.1.4.6 confusion matrix for YOLOv8 skin type classification

From Table 6.1.4.2, the precision, recall, and F1-score results for the test set of the skin type dataset are shown.

The F1-score for the Dry skin class is 0.612, indicating a moderate level of performance. While recall is strong at 0.714, precision is lower at 0.536, suggesting that the model frequently misclassifies other skin types as Dry.

The model classifies Normal skin type better than Dry and Oily skin types, with the highest F1-score of 0.653. However, its recall is slightly lower at 0.593, and precision is higher at 0.727. This highlights that the model often misclassifies Normal skin type images as other skin types.

With a comparable F1-score of 0.650, the Oily class exhibits balanced recall and precision. This indicates that the model is more consistent in classifying Oily skin types, as both recall, and precision are relatively balanced.

Table 6.1.8 Precision, recall and F1-score for YOLOv8 skin type classification

Class	Precision	Recall	F1-Score
Dry	0.536	0.714	0.612
Normal	0.727	0.593	0.653
Oily	0.684	0.619	0.650

6.1.5 GPT Assistant-Based

After the learning process of GPT-4o with the training dataset, all test images were submitted to GPT via the API key for prediction. A sample of the GPT assistant-based skin acne classification results across different severity levels is shown in Figure 4.5.5.1. It displays the predicted and actual values of the images stored in Google Cloud by showing the Google Cloud link sent to GPT-4o during the learning process. The detailed results are provided in the appendix.

Image: https://storage.googleapis.com/dataset_skinacne/Level_0/levle0_106.jpg | Predicted: Level_1 | Actual: Level_0
 Image: https://storage.googleapis.com/dataset_skinacne/Level_0/levle0_107.jpg | Predicted: Level_0 | Actual: Level_0
 Image: https://storage.googleapis.com/dataset_skinacne/Level_0/levle0_130.jpg | Predicted: Level_0 | Actual: Level_0
 Image: https://storage.googleapis.com/dataset_skinacne/Level_0/levle0_131.jpg | Predicted: Level_1 | Actual: Level_0
 Image: https://storage.googleapis.com/dataset_skinacne/Level_0/levle0_132.jpg | Predicted: Level_0 | Actual: Level_0
 Image: https://storage.googleapis.com/dataset_skinacne/Level_0/levle0_134.jpg | Predicted: Level_1 | Actual: Level_0
 Image: https://storage.googleapis.com/dataset_skinacne/Level_0/levle0_142.jpg | Predicted: Level_0 | Actual: Level_0
 Image: https://storage.googleapis.com/dataset_skinacne/Level_0/levle0_149.jpg | Predicted: Level_1 | Actual: Level_0

Figure 4.1.5.1 sample results GPT assistant-based skin acne classification across different severity levels

In Figure 6.1.5.2, the confusion matrix is shown using 25 images for each level from the training set. After training the GPT-4o with the training set, all test images are sent to the GPT via the API key to compute the results. From the results, 44 out of 109 images were correctly predicted.

For Level 0, 14 out of 39 images were correctly predicted, with 25 misclassified as Level 1, resulting in a recall of 0.360. No images from other classes were predicted as Level 0, resulting in a precision of 1.000.

For Level 1, 16 out of 48 images were correctly classified as Level 1 images, meaning that 32 images were misclassified as Level 2, resulting in a recall of 0.330. From 41

CHAPTER 6

images predicted as Level 1, 25 images belonged to Level 0, resulting in a precision of 0.390.

For Level 2, all 14 images were correctly classified as Level 2, resulting in a recall of 1.000. From 46 images predicted as Level 2, 32 images belonged to Level 1, resulting in a precision of 0.300.

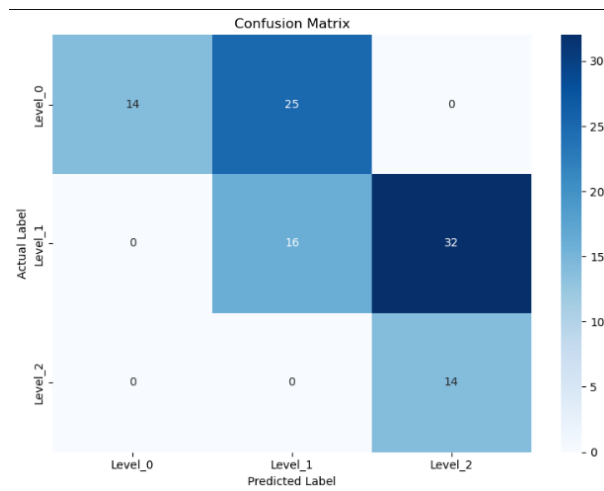


Figure 6.1.5.2 confusion for ChatGPT assistant-based skin acne classification

From Table 6.1.5.1, the precision, recall, and F1-score for the skin acne classification model are shown. The overall accuracy for this model is 0.440.

The F1-score for the Level 0 class is 0.530, reflecting the best performance among the three classes. This strong result is largely derived from the high precision of 1.000, indicating that all predictions made by the model for this class are correct. However, the recall is only 0.360, meaning that the model often misclassifies Level 0 images as other classes. This significantly impacts the F1-score, causing it to decrease despite the perfect precision.

The F1-score for the Level 1 class is 0.360, which is the lowest among the three classes, indicating the weakest model performance in this category. The precision of 0.39 suggests that many images from other classes are often predicted as Level 1, while the recall of 0.33 shows that many Level 1 images are misclassified as images from other

classes. The combination of relatively low precision and recall leads to the overall poor F1-score for this class.

The F1-score for the Level 2 class is 0.470, which places it in the middle range among the three classes. The model demonstrates a precision of 0.30, reflecting that many images from other classes are predicted as Level 2 images. However, the recall is exceptionally high at 1.000, indicating that the model successfully identified all actual Level 2 cases. This imbalance results in a moderate F1-score.

Table 6.1.5.1 classification report for GPT assistant-based skin acne

Classification Report:				
	precision	recall	f1-score	support
Level_0	1.00	0.36	0.53	39
Level_1	0.39	0.33	0.36	48
Level_2	0.30	1.00	0.47	14
accuracy			0.44	101
macro avg	0.56	0.56	0.45	101
weighted avg	0.61	0.44	0.44	101

Figure 4.5.5.3 shows a sample of the results, consisting of the image link stored in Google Cloud, the prediction results from GPT-4o, and the actual value of the image. The detailed results are provided in the appendix.

Image: https://storage.googleapis.com/dataset_skin/Dry/dry_104.jpg | Predicted: Normal | Actual: Dry
 Image: https://storage.googleapis.com/dataset_skin/Dry/dry_106.jpg | Predicted: Normal | Actual: Dry
 Image: https://storage.googleapis.com/dataset_skin/Dry/dry_109.jpg | Predicted: Normal | Actual: Dry
 Image: https://storage.googleapis.com/dataset_skin/Dry/dry_116.jpg | Predicted: Dry | Actual: Dry
 Image: https://storage.googleapis.com/dataset_skin/Dry/dry_118.jpg | Predicted: Oily | Actual: Dry
 Image: https://storage.googleapis.com/dataset_skin/Dry/dry_133.jpg | Predicted: Oily | Actual: Dry

Figure 4.5.5.3 Sample results for GPT assistant-based skin type classification

In Figure 6.1.5.4, the confusion matrix is shown using 25 images from each level of the training set and testing with all the images from the test set. The result shows that 30 images out of 109 were predicted correctly.

For Dry, 4 out of 21 images were correctly predicted, with 6 images misclassified as Normal and 11 as Oily, resulting in a recall of 0.190. From 9 images predicted as Level 0, 1 image belongs to Normal and 3 images belong to Dry, giving a precision of 0.500.

For Normal, 6 out of 27 images were correctly classified as Normal, with 1 misclassified as Dry and 14 as Dry, resulting in a recall of 0.290. Out of 16 images predicted as Normal, 2 images belonged to Dry, yielding a precision of 0.867.

For Oily, 14 out of 21 images were correctly predicted, with 1 image misclassified as Dry and 14 images misclassified as Normal, resulting in a recall of 0.290. Out of 16 images predicted as Level 2, 6 images belong to Dry, and 4 images belong to Normal, resulting in a precision of 0.380.

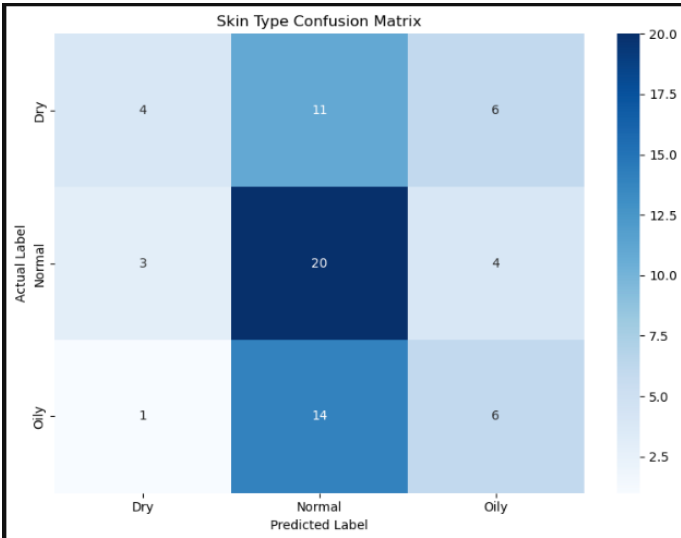


Figure 6.1.5.4 confusion for ChatGPT assistant-based skin type classification

From Table 6.1.5.2, the results of precision, recall, and F1-score for the test set of the skin acne dataset are shown. The overall accuracy for this model is 0.430.

For F1-score of the Dry class, it is 0.280, reflecting that the model is struggling to identify this class effectively. While it has a precision of 0.500, meaning half of the samples predicted as Dry were correct, the recall is only 0.190, meaning a large number of images from other classes were predicted as Dry. This significant gap between precision and recall causes the F1-score to be the lowest among the three classes.

For F1-score of the Normal class, it is 0.560, demonstrating the best overall performance among the three classes. With a precision of 0.44 and a recall of 0.74, the model correctly identified many actual Normal images. Although the precision is lower,

CHAPTER 6

the high recall contributes to the most successful classification ability among the three classes.

For F1-score of the Oily class, it is 0.320, indicating that the model's performance can still be improved. The precision of 0.38 and recall of 0.29 reveal that many actual Oily class images were classified as other classes, and images from other classes were predicted as Oily class images. The relatively close values of precision and recall indicate limited performance in classifying this class.

Table 6.1.5.2 classification report for GPT assistant-based skin type

Classification Report:				
	precision	recall	f1-score	support
Dry	0.50	0.19	0.28	21
Normal	0.44	0.74	0.56	27
Oily	0.38	0.29	0.32	21
accuracy			0.43	69
macro avg	0.44	0.41	0.39	69
weighted avg	0.44	0.43	0.40	69

6.1.6 Model Comparison

The performance of four models which are classic CNN, ResNet50, EfficientNetB0, and YOLOv8 for skin acne classification was evaluated based on their training and validation accuracy, as shown in Table 6.1.6.1. The first model, the classic CNN model, achieves a training accuracy of 51.75% and a validation accuracy of 50.00%, reflecting the worst performance among the four models. This suggests that the classic CNN model may not be complex enough, preventing it from achieving better performance.

In comparison, ResNet50 outperformed the classic CNN model with a training accuracy of 67.65% and a validation accuracy of 69.52%, indicating that this model has better feature extraction capabilities and generalization. Similarly, EfficientNetB0 achieved a comparable training accuracy of 67.44%, but its validation accuracy dropped to 60.00%, indicating that this model may be prone to overfitting. Among all the models, YOLOv8 achieved the best performance, with a training accuracy of 80.00%, showcasing its strength in learning discriminative features effectively for classification tasks.

Table 6.1.6.1 Training and validation accuracy comparison between models for skin acne

Model	Training accuracy(%)	Validation accuracy(%)
Classic CNN	51.75%	50.00%
ResNet50	67.65%	69.52%
EfficientNetB0	67.44%	60.00%
YOLOv8	80.00%	

Comparing the models based on their validation and training accuracy, YOLOv8 achieved the best performance among them. Table 6.1.6.2 shows the testing accuracy across all models.

The testing accuracy of the YOLOv8 model still outperforms all other models, with a testing accuracy of 76.2%, highlighting its robustness and superior performance across the dataset. It is followed by the EfficientNetB0 model, with a testing accuracy of

CHAPTER 6

71.3%, indicating that this model still performs well and is able to deliver reliable results.

Next, the ResNet50 model achieves a testing accuracy of 60.4%, demonstrating decent performance, but it is not as effective as the other two models. Finally, the classic CNN model achieves the lowest testing accuracy at 47.0%, suggesting that it is unable to perform well with only three convolutional blocks in its design. This indicates that the model may be too simple to handle the complex task compared to the other three pretrained models.

Table 6.1.6.2 Testing accuracy comparison between models for skin acne

Model	Testing accuracy(%)
Classic CNN	47.0%
ResNet50	60.4%
EfficientNetB0	71.3%
YOLOv8	76.2%

After comparing the training accuracy, validation accuracy, and testing accuracy among the three models, Table 6.1.6.3 shows the comparison of testing accuracy between the YOLOv8 model and the GPT Assistant-Based model for the skin acne dataset. The results show that YOLOv8 outperformed the GPT Assistant-Based model, achieving a testing accuracy of 76.2%, compared to the GPT Assistant-Based model, which only achieves a testing accuracy of 44%. The testing accuracy results suggest that the GPT Assistant-Based model is unable to learn effectively from the images, likely due to the limited number of 25 training images per level used for training with all testing images.



Table 6.1.6.3 Testing accuracy of YOLOv8 and GPT Assistant-Based for skin acne

Model	Testing accuracy(%)
YOLOv8	76.2%
GPT Assistant-Based	44.0%

CHAPTER 6

After identifying YOLOv8 as the best-performing model for the skin acne grading dataset, one image each from Level 1 and Level 2 class were selected for evaluation. As shown in Table 6.1.6.4, the model correctly predicted the Level 1 image with 74.5% confidence, followed by 23.57% for Level 2. For the Level 2 image, the model predicted it as Level 1 with 87.22% confidence, while the predictions for Level 0 and Level 2 were 5.92% and 6.86%, respectively.

Table 6.1.6.4 YOLOv8 model confidence level for skin acne

Actual label	Level 1	Level 2
Image		
Predicted label	Level 1	Level 1
Level 0 (%)	1.93	5.92
Level 1 (%)	74.50	87.22
Level 2 (%)	23.57	6.86

In Table 6.1.6.5, the training and validation accuracy among the four models for the skin type dataset are shown. The classic CNN model demonstrates the worst performance among the four, achieving only a training accuracy of 45.3% and a validation accuracy of 43.0%. This suggests that the model is not complex enough, which contributes to its inability to achieve better accuracy.

In comparison, the ResNet50 model performs better than the classic CNN model, with a training accuracy of 60.8% and a validation accuracy of 43.0%. This indicates that the ResNet50 model can learn image features more effectively and has better generalization.

Remarkably, EfficientNetB0 achieves the highest training accuracy at 80.0%, but its validation accuracy only reaches 56.0%. This suggests that the model may be overfitting, as the training and validation accuracies are mismatched.

Finally, YOLOv8 provides training and validation accuracy of 70%, demonstrating better performance compared to the other models. This suggests that YOLOv8 has the potential to outperform EfficientNetB0, as it offers more reliable performance.

Table 6.1.6.5 Training and validation accuracy comparison between models for skin type

Model	Training accuracy(%)	Validation accuracy(%)
Classic CNN	45.3%	43.0%
ResNet50	60.8%	56.1%
EfficientNetB0	80.0%	56.0%
YOLOv8	70%	

When comparing the models based on testing accuracy, YOLOv8 provides the best performance among the four models, with a testing accuracy of 64.0%. In comparison, EfficientNetB0 achieves a testing accuracy of only 56.5%, which is relatively lower than YOLOv8. This suggests that YOLOv8 is better at performing on unseen data compared to EfficientNetB0.

Additionally, ResNet50 shows a moderate result with a testing accuracy of 49.3%, indicating that this model may not be handling the skin type dataset effectively. Finally, the classic CNN model exhibits the worst performance with a testing accuracy of 45.3%. This suggests that the simple architecture of the CNN model is not sufficient to handle the complexity of the dataset, as it fails to learn enough features from the images.

Table 6.1.6.6 Testing accuracy comparison between models for skin type

Model	Testing accuracy(%)
Classic CNN	36.0%
ResNet50	49.3%
EfficientNetB0	56.5%
YOLOv8	64.0%

After comparing the training accuracy, validation accuracy, and testing accuracy among the three models, Table 6.1.6.7 shows the comparison of testing accuracy between the



YOLOv8 model and the GPT Assistant-Based model for the skin type dataset. The results indicate that YOLOv8 still outperforms the GPT Assistant-Based model, achieving a testing accuracy of 64.0%, whereas the GPT Assistant-Based model only achieves a testing accuracy of 44%. This further supports the conclusion that the GPT Assistant-Based model is unable to achieve better performance, likely due to the insufficient number of training images (only 25 per class), which prevents the model from effectively learning the features of the images.

Table 6.1.6.7 Testing accuracy of YOLOv8 and GPT Assistant-Based for skin type

Model	Testing accuracy(%)
YOLOv8	64.0%
GPT Assistant-Based	43.0%

After identifying YOLOv8 as the best-performing model for the skin type dataset, one image each from the Normal and Oily class were selected for evaluation. As shown in Table 6.1.6.8, the model correctly predicted the Normal class image with confidence of 87.21%, followed by 10.90% for the Dry class. However, the image from the Oily class was incorrectly predicted as Normal with a confidence of 83.47% and followed by 12.10% for Dry class.

Table 6.1.6.8 YOLOv8 model confidence level for skin type

Actual label	Normal	Oily
Image		
Predicted label	Normal	Normal
Dry (%)	10.90	12.10
Normal (%)	87.21	83.47
Oily (%)	1.85	4.43

6.2 Testing Setup and Result

6.2.1 Authentication Module

Table 6.2.1.1 Registration process test case 1

Test case		reg_001
Test case name		Register an account
Test case description		Verify registration process when correct email and password are entered
Expected result		The new user can register an account
Actual result		The user able to register an account
Test case result		Passed
No.	Test case step	Test case data
1.	Enter full name	Ng Yong Shen
2.	Enter email address	yongshennng0508@gmail.com
3.	Enter password	Ng@1234567
4.	Confirm password	Ng@1234567

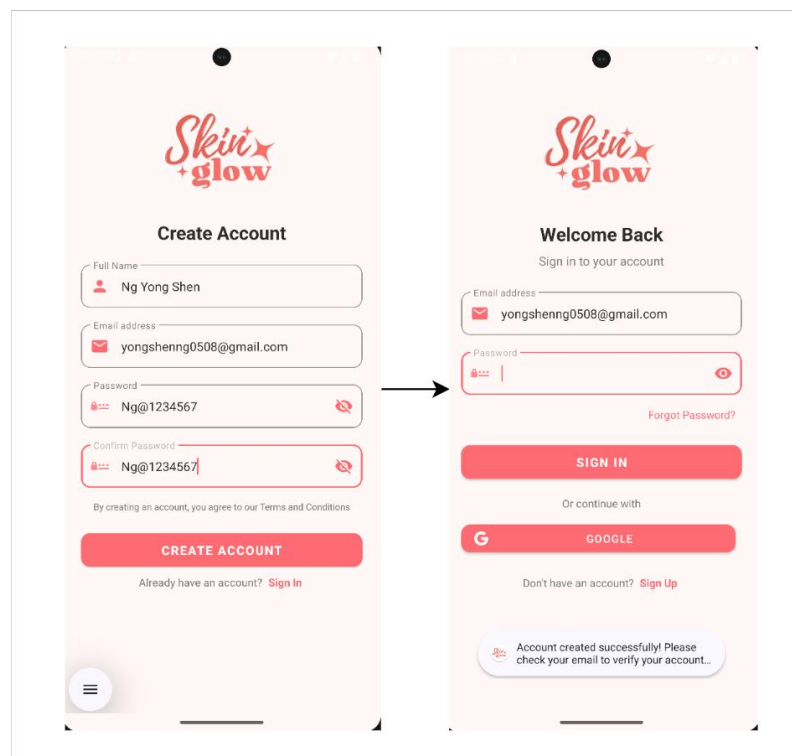


Figure 6.2.1.1 Registration process test case 1

Table 6.2.1.2 Registration process test case 2

Test case		reg_002
Test case name		Register an account
Test case description		Verify registration process when invalid email and password are entered
Expected result		The new user cannot register an account
Actual result		The user not able to register an account as the email and password are invalid
Test case result		Passed
No.	Test case step	Test case data
1.	Enter full name	Ng Yong Shen
2.	Enter email address	yongshenng0508
3.	Enter password	Ng1234567
4.	Confirm password	Ng1234567

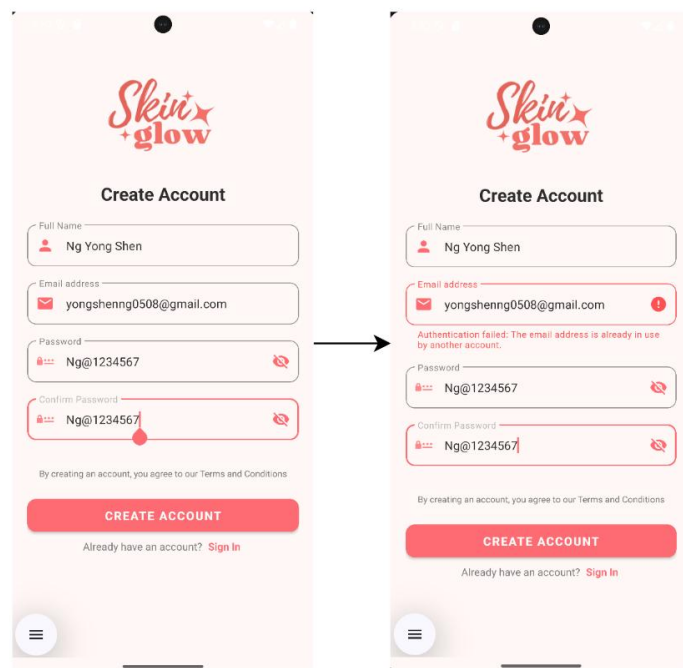


Figure 6.2.1.2 Registration process test case 2

Table 6.2.1.3 Registration process test case 3

Test case		reg_003
Module name		Authentication
Test case description		Verify registration process when duplicate email and password are entered
Expected result		The new user cannot register an account
Actual result		The user not able to register an account as duplicated email entered
Test case result		Passed
No.	Test case step	Test case data
1.	Enter full name	Ng Yong Shen
2.	Enter email address	yongshenng0508@gmail.com
3.	Enter password	Ng@1234567
4.	Confirm password	Ng@1234567

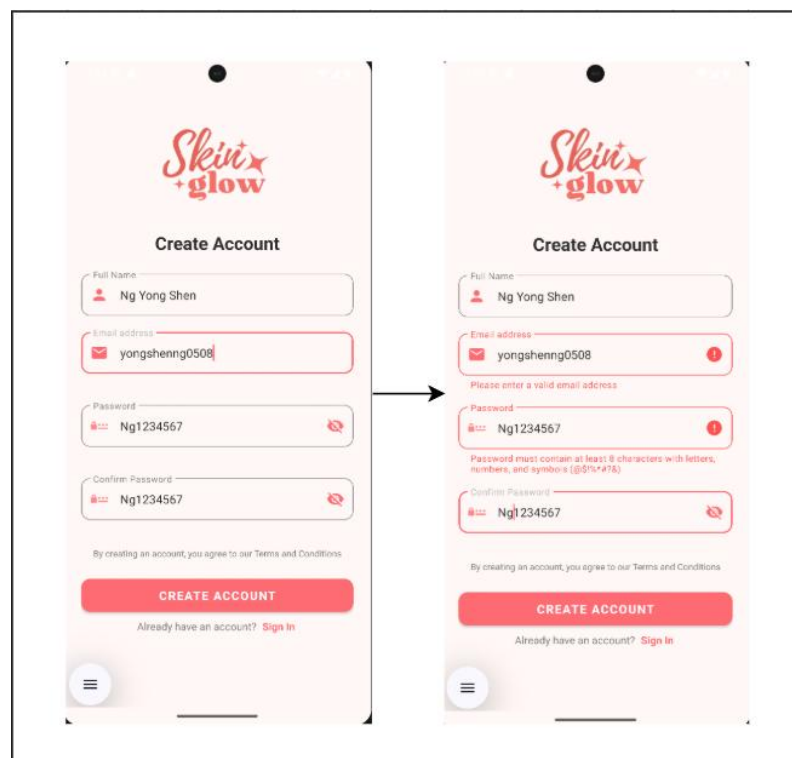


Figure 6.2.1.3 Registration process test case 3

Table 6.2.1.4 Registration process test case 4

Test case		reg_004
Module name		Authentication
Test case description		Verify registration process when a valid email is entered but the password and confirm password fields do not match
Expected result		The new user cannot register an account
Actual result		The user not able to register an account as confirm password not similar as password
Test case result		Passed
No.	Test case step	Test case data
1.	Enter full name	Ng Yong Shen
2.	Enter email address	yongshennng0508@gmail.com
3.	Enter password	Ng@1234567
4.	Confirm password	Ng@1233333

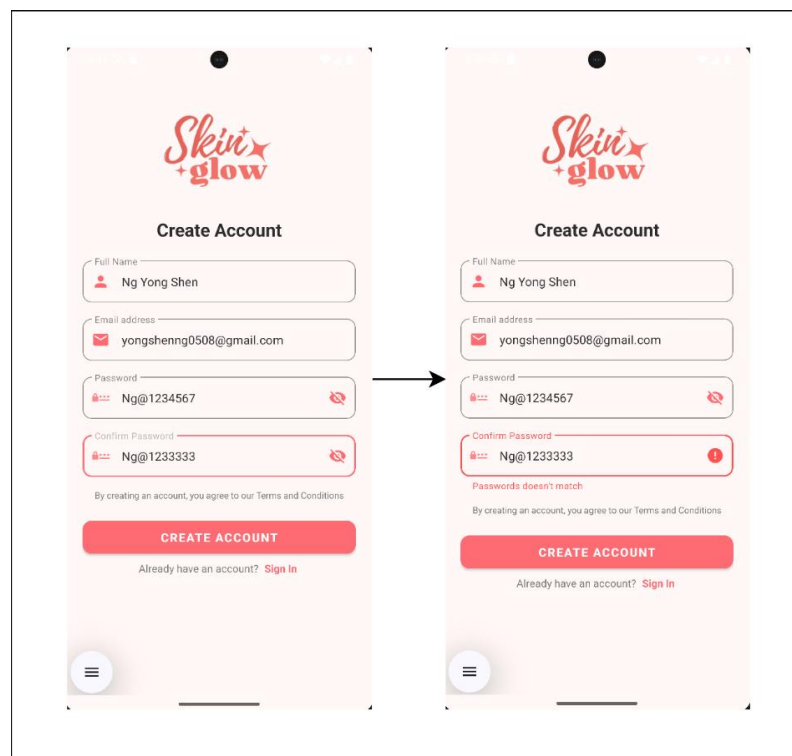


Figure 6.2.1.4 Registration process test case 4

Table 6.2.1.5 Login process test case 1

Test case		login_001
Module name		Authentication
Test case description		Verify login process when correct user email address and password are entered
Expected result		The user can login an account
Actual result		The user able to login an account
Test case result		Passed
No.	Test case step	Test case data
1.	Enter email address	yongshenng0508@gmail.com
2.	Enter password	Ng@1234567

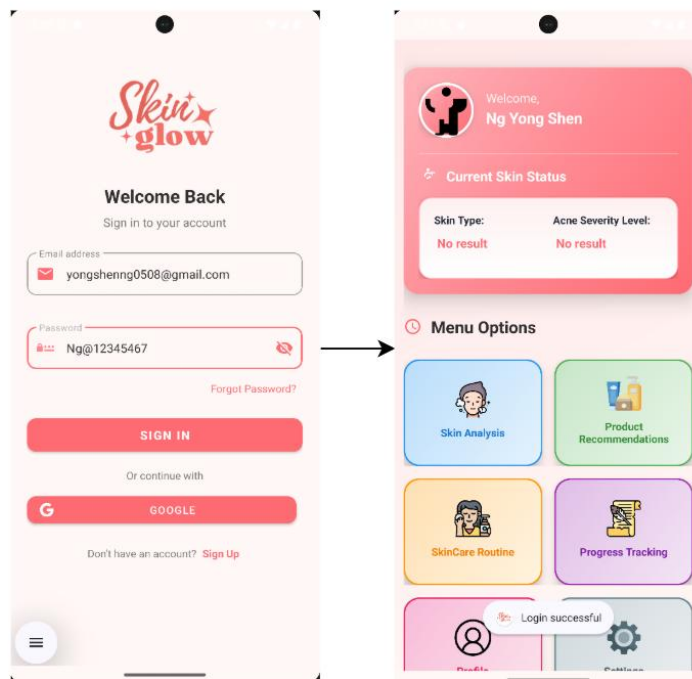


Figure 6.2.1.5 Login process test case 1

Table 6.2.1.6 Login process test case 2

Test case		login_002
Module name		Authentication
Test case description		Verify login process when invalid user email address and password are entered
Expected result		The user cannot login an account
Actual result		The user not able to login an account
Test case result		Passed
No.	Test case step	Test case data
1.	Enter email address	yongshennng0508
2.	Enter password	Ng1234567

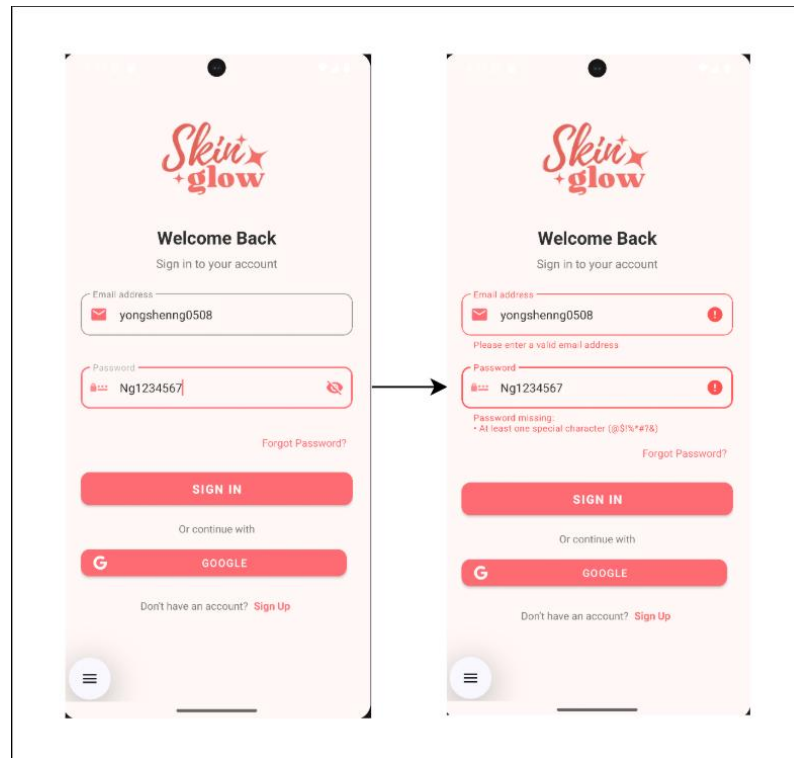


Figure 6.2.1.6 Login process test case 2

Table 4.6.1.7 Reset password process test case 1

Test case		reset_001
Test case name		Authentication
Test case description		Verify login process when valid user email address entered
Expected result		The user can reset the account password
Actual result		The user able to reset the account password
Test case result		Passed
No.	Test case step	Test case data
1.	Enter email address	yongshenng0508@gmail.com

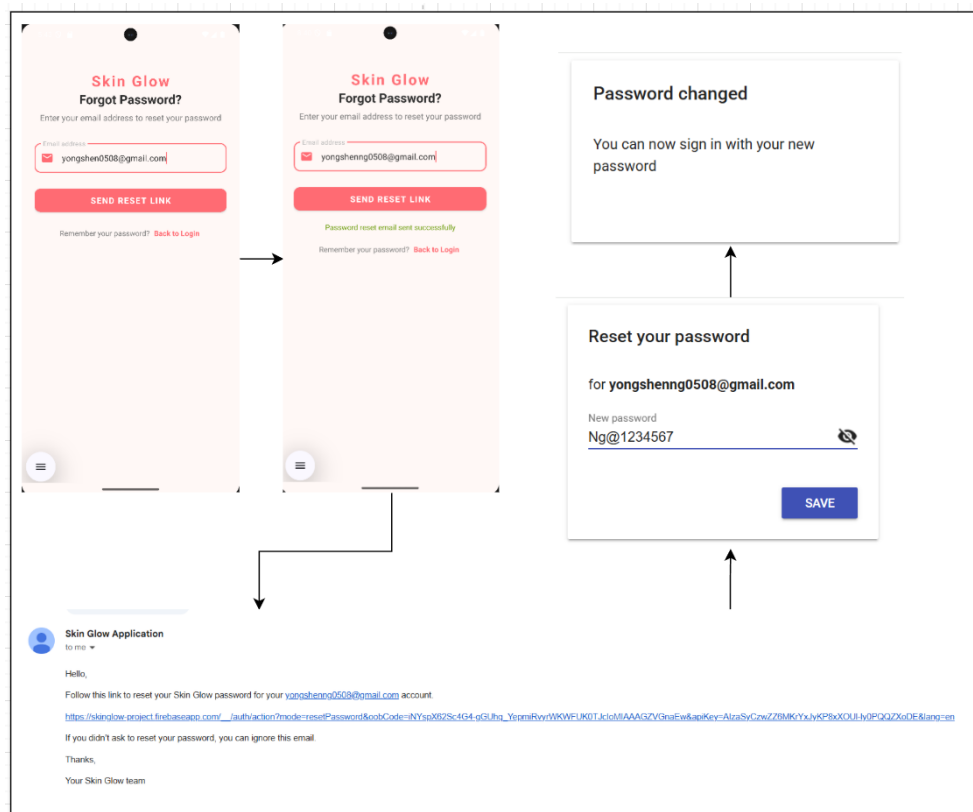


Figure 6.2.1.7 Reset password process test case 1

Table 4.6.1.8 Reset password process test case 2

Test case		reset_002
Test case name		Reset password of an account
Test case description		Verify reset password process when entered invalid email address
Expected result		The user cannot reset password of an account
Actual result		The user not able to reset password
Test case result		Passed
No.	Test case step	Test case data
1.	Enter email address	yongshenng0508

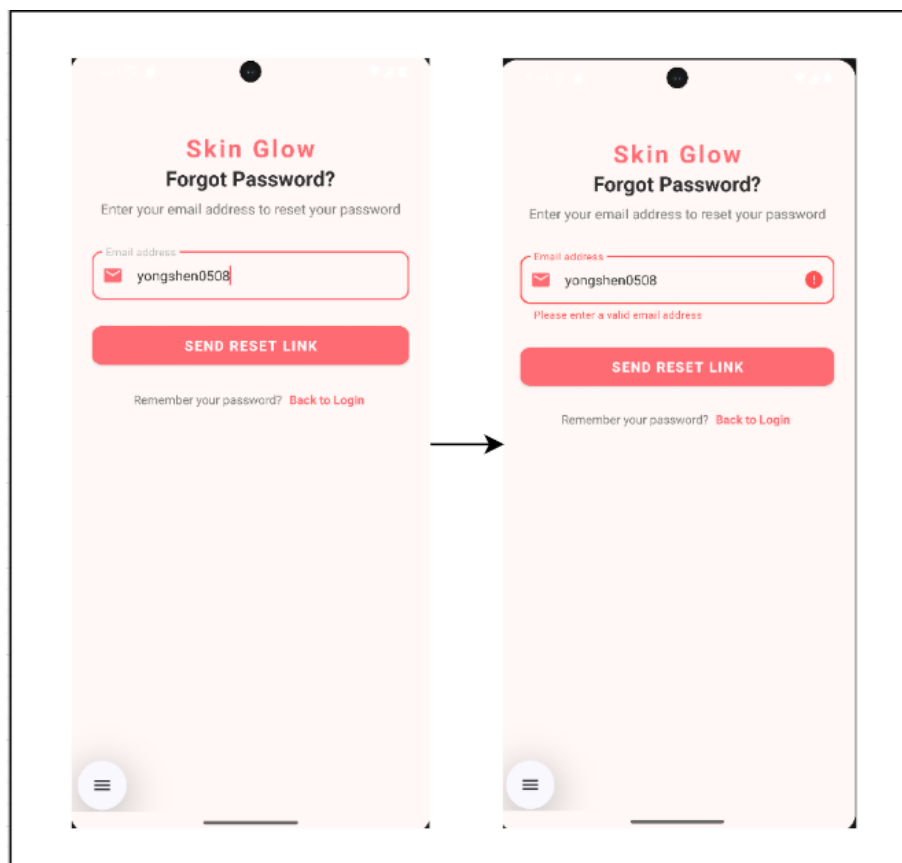


Figure 4.6.1.8 Reset password process test case 2

6.2.2 Skin Analysis Module

Table 6.2.2.1 Skin analysis process test case 1

Test case		skin_001
Module name		Skin analysis
Test case description		Verify skin analysis process when user positioning their face correctly
Expected result		The user can generate skin analysis result
Actual result		The user able to generate skin analysis result
Test case result		Passed
No.	Test case step	Test case data
1.	Click on “skin analysis” icon	Redirect to skin analysis page
2.	Positioning the face within the face	capture
3.	View the skin analysis result	-

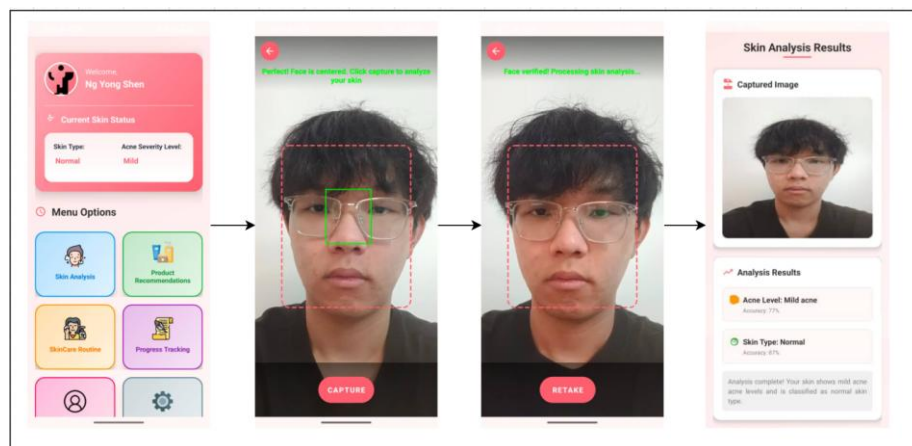


Figure 6.2.2.1 Skin analysis process test case 1

Table 6.2.2.2 Skin analysis process test case 2

Test case		skin_002
Module name		Skin analysis
Test case description		Verify skin analysis process when do not position correctly in skin analysis process
Expected result		The user can generate skin analysis result
Actual result		The user able to generate skin analysis result
Test case result		Passed
No.	Test case step	Test case data
1.	Click on “skin analysis” icon	Redirect to skin analysis page
2.	Positioning the face within the face	-

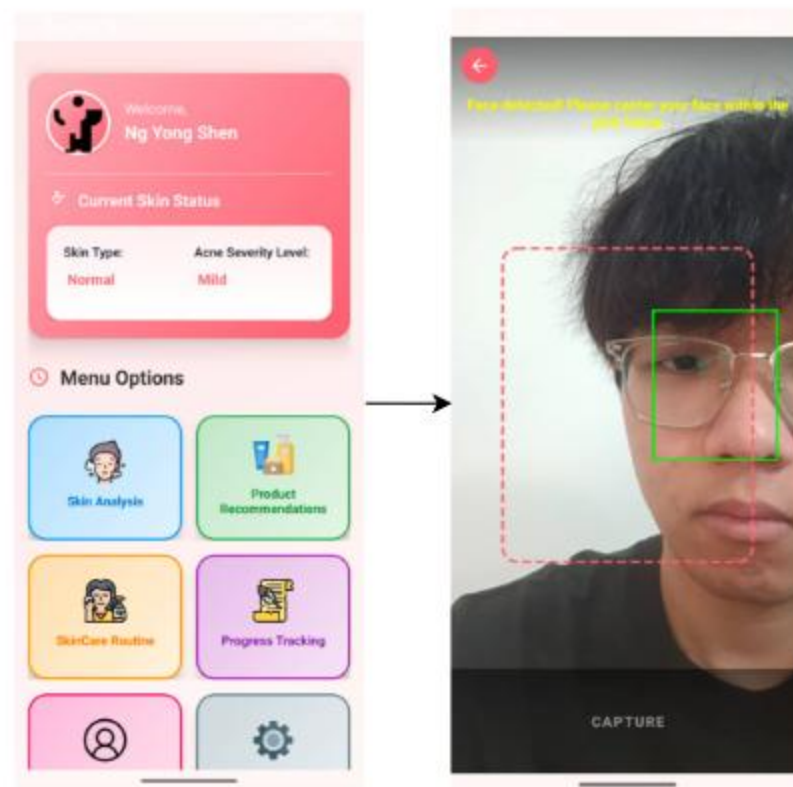


Figure 6.2.2.2 Skin analysis process test case 2

6.2.3 Skincare Recommendation Module

Table 6.2.3.1 Skincare recommendation process test case 1

Test case		skinrec _001
Module name		Skincare recommendation
Test case description		Verify skincare recommendation process when user has not performed skin analysis
Expected result		The user needs to select the skin acne severity and skin type to produce skincare recommendation
Actual result		The user able to proceed to skincare recommendation
Test case result		Passed
No.	Test case step	Test case data
1.	Click on “skin analysis” icon	Redirect to skin analysis page
2.	Positioning the face within the face	-

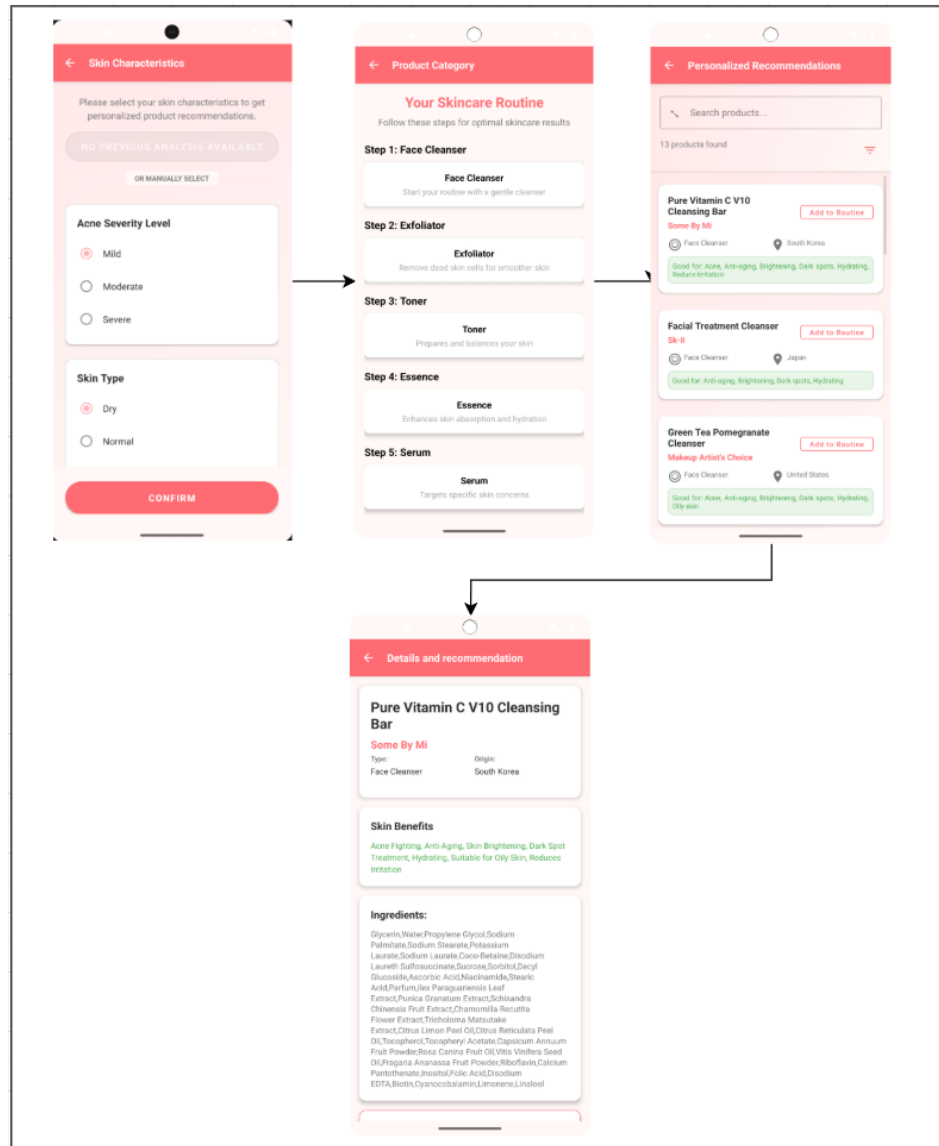


Figure 6.2.3.1 Skincare recommendation process test case 1

Table 6.2.3.2 Skincare recommendation process test case 2

Test case		skinrec _002
Module name		Skincare recommendation
Test case description		Verify skincare recommendation process when user performed skin analysis
Expected result		The user can select to use current skin analysis result to produce skincare recommendation
Actual result		The user able to proceed to skincare recommendation
Test case result		Passed
No.	Test case step	Test case data
1.	Click on “skincare recommendation” icon	Redirect to skin analysis page
2.	Select “use current result” button	Click on “Confirm” to proceed
3.	Select the product category	Click on “Face Cleanser”
5.	View the product list	-

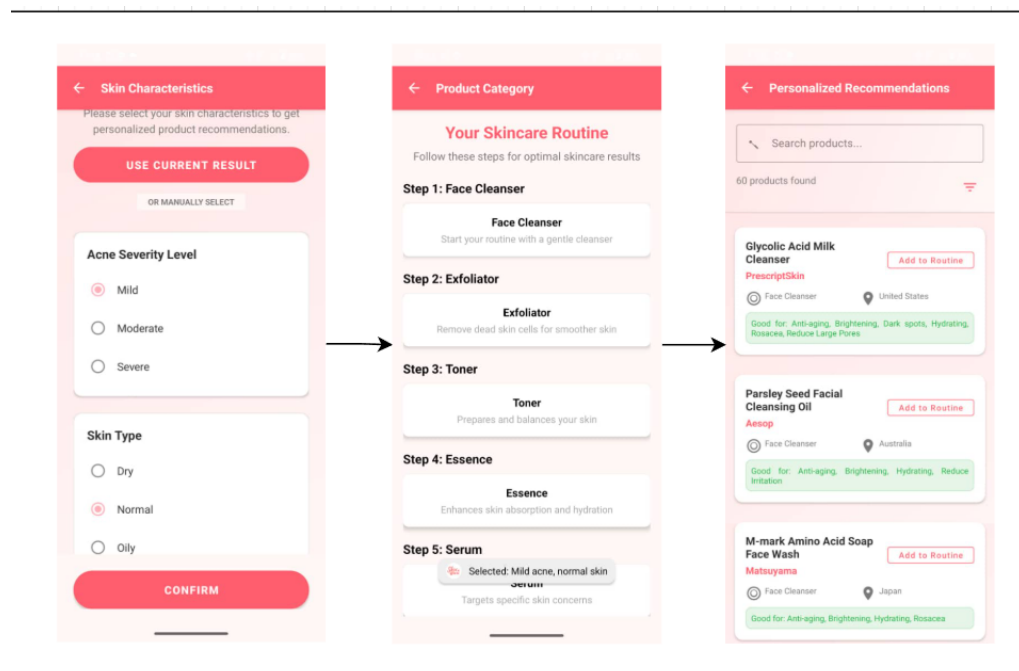


Figure 6.2.3.3 Skincare recommendation process test case 2

Table 6.2.3.4 Skincare recommendation process test case 3

Test case		skinrec _003
Module name		Skincare recommendation
Test case description		Verify that product recommendation details switch to OpenAI when the selected LLM model is changed.
Expected result		The user should see a small text indicating which model powers the recommendation.
Actual result		The user will be able to view the recommendations powered by GPT
Test case result		Passed
No.	Test case step	Test case data
1.	Check on OpenAI API platform on the token trigger	OpenAI platform
2.	Select settings	Settings
3.	Select AI Configuration	AI Configuration
4.	Select AI models	GPT-4o
5.	View the product details	-
6.	Verify that the usage of GPT-4o	OpenAI platform

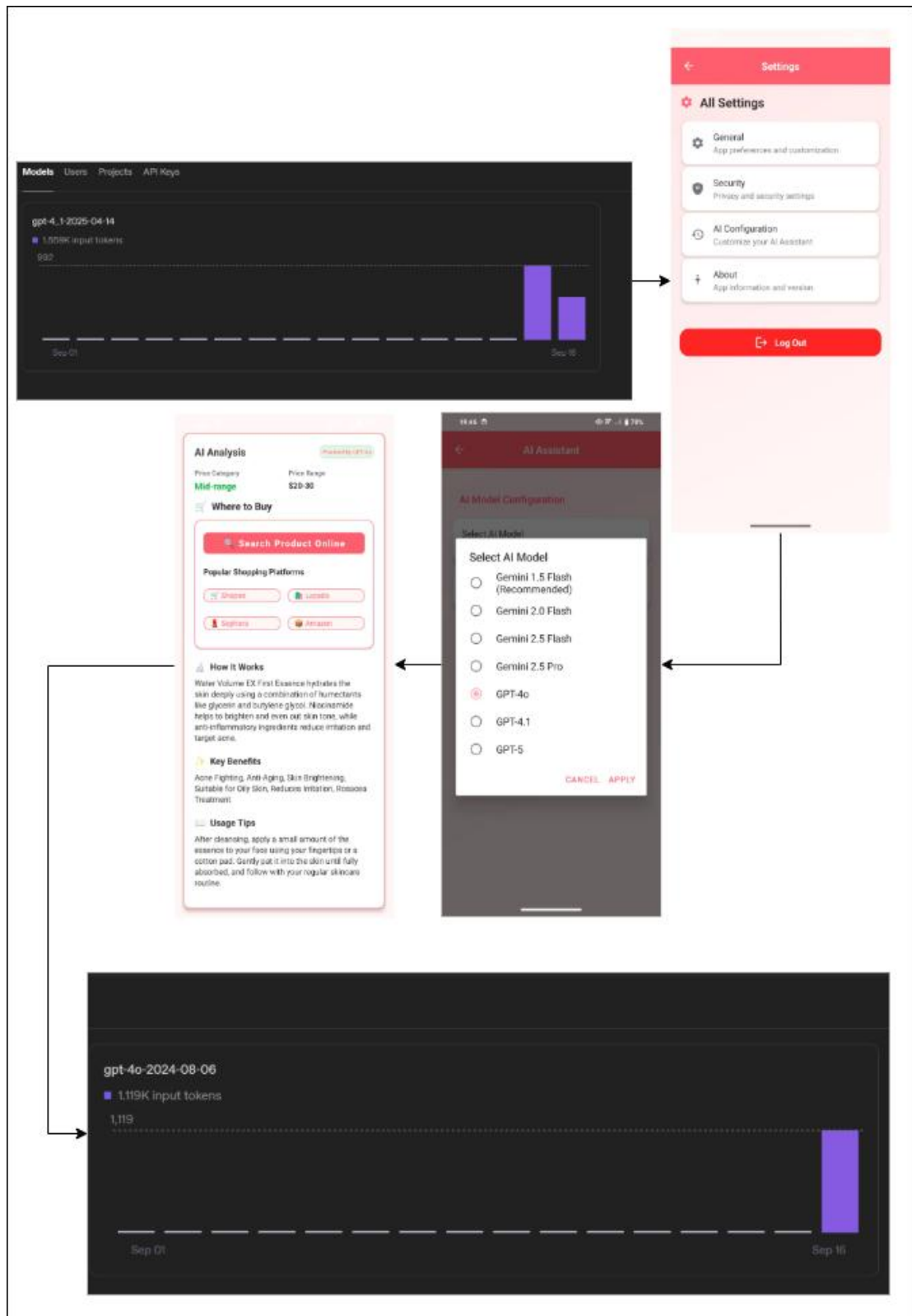


Figure 6.2.3.4 Skincare recommendation process test case 3

6.2.4 Skincare Routine Module

Table 6.2.4.1 Skincare routine test case 1

Test case		skinroutine _001
Module name		Skincare recommendation
Test case description		Verify adding a skincare product into a skincare routine, ensuring that no duplicate skincare product is allowed for each skincare step
Expected result		When a duplicate product is added to the same step, the system should prompt the user with an option to either replace the existing product or cancel the action
Actual result		The user is still able to proceed to skincare recommendation without being prompted to choose replacement or cancellation
Test case result		Passed
No.	Test case step	Test case data
1.	Select first item in the recommendation history	View product list
2.	Click on “add to routine” button inside product list	Add to routine
3.	Select the existing routine	daily routine 1
4.	Select one product from product list	Parsley seeds facial cleansing oil
5.	Message prompt to alert user to choose whether to replace or not	Replace
6.	Select skincare routine	Daily routine 1
7.	The product under Step 1: Face Cleanser changed	-

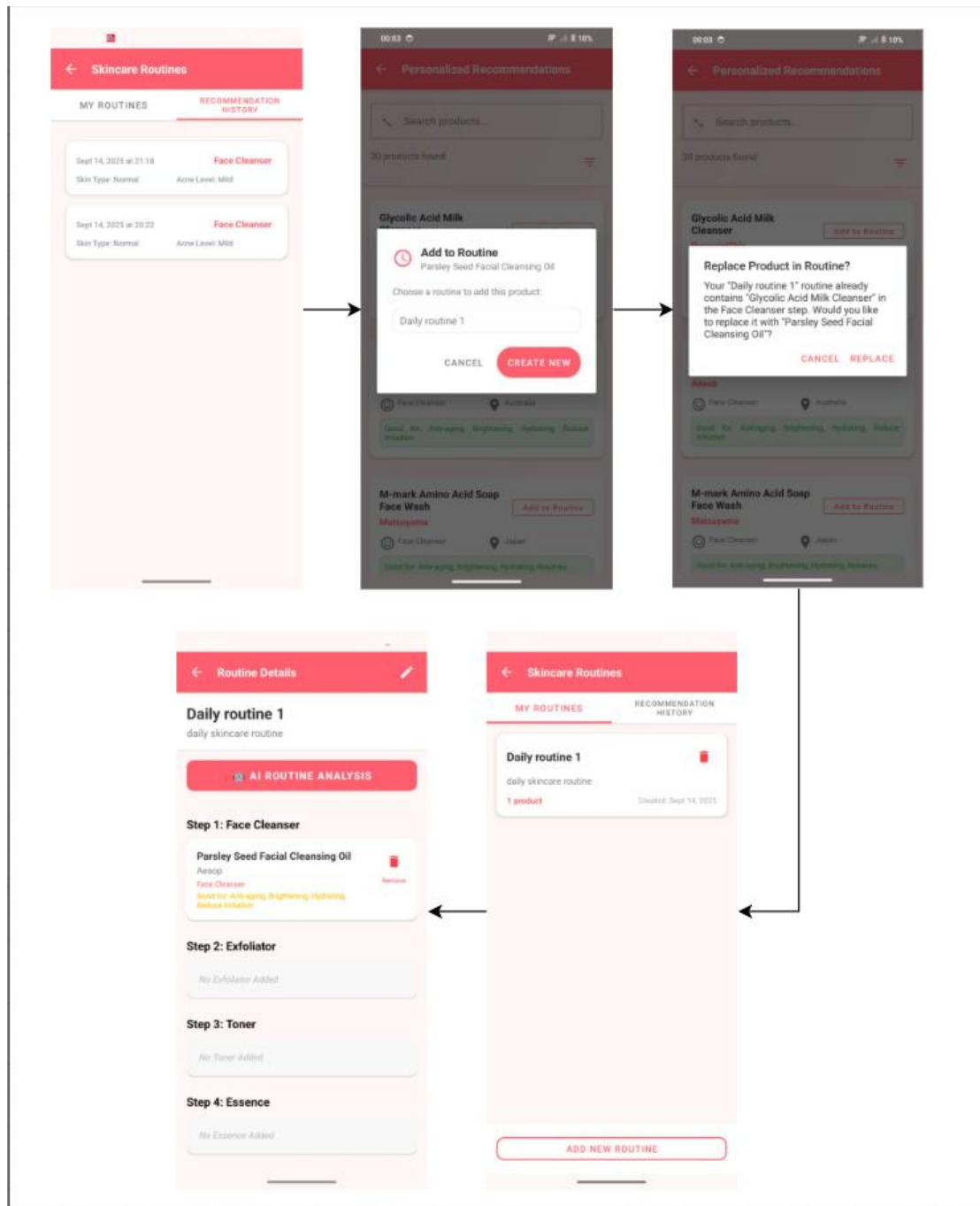


Figure 6.2.4.1 Skincare routine test case 1

Table 6.2.4.2 Skincare routine test case 2

Test Case ID		skinroutine_002
Module Name		Skincare Recommendation
Test Case Description		Verify that performing AI analysis is not allowed when no products exist inside the skincare routine
Expected Result		The user cannot perform Routine AI Analysis
Actual Result		The user cannot perform Routine AI Analysis for the empty skincare within routine
Test case result		Passed
No.	Test case step	Test case data
1.	Select the routine	Daily routine 1
2.	Select AI routine analysis	AI routine Analysis

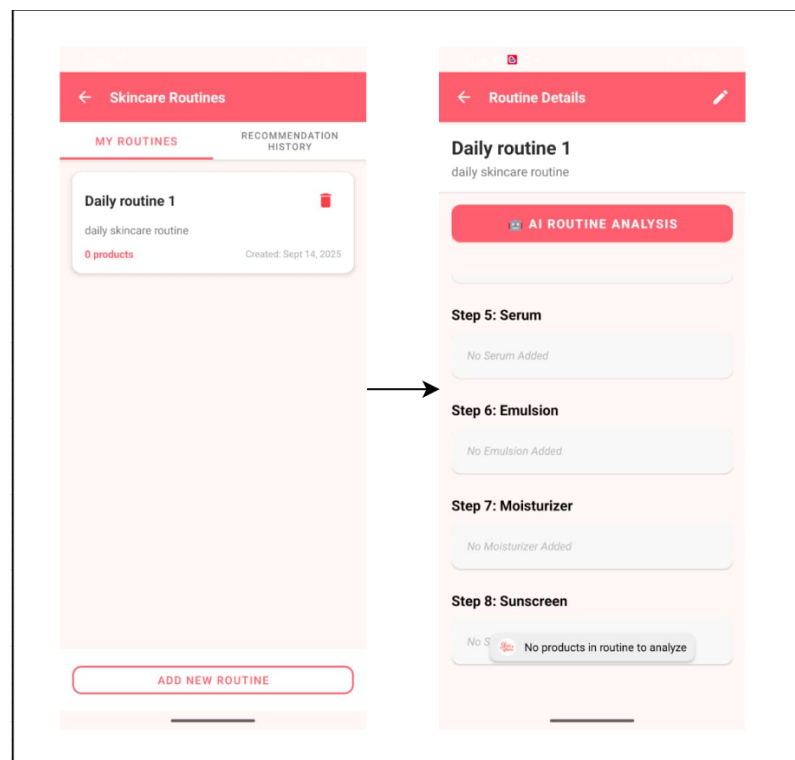


Figure 6.2.4.2 Skincare routine test case 2

6.2.5 Skin Progress Module

Table 6.2.5.1 skin progress test case 1

Test case		skinprogress _001
Module name		Skin progress
Test case description		Verify skin progress process when no skin analysis performed
Expected result		The user cannot view the graph of the skin progress, and no results show in the skin analysis history
Actual result		The user will not view the skin progress graph and no result show in skin analysis history
Test case result		Passed
No.	Test case step	Test case data
1.	Click on “progress tracking” icon	Redirect to progress tracking page and view the acne severity page
2.	Select “skin type” tab	Redirect to skin type page

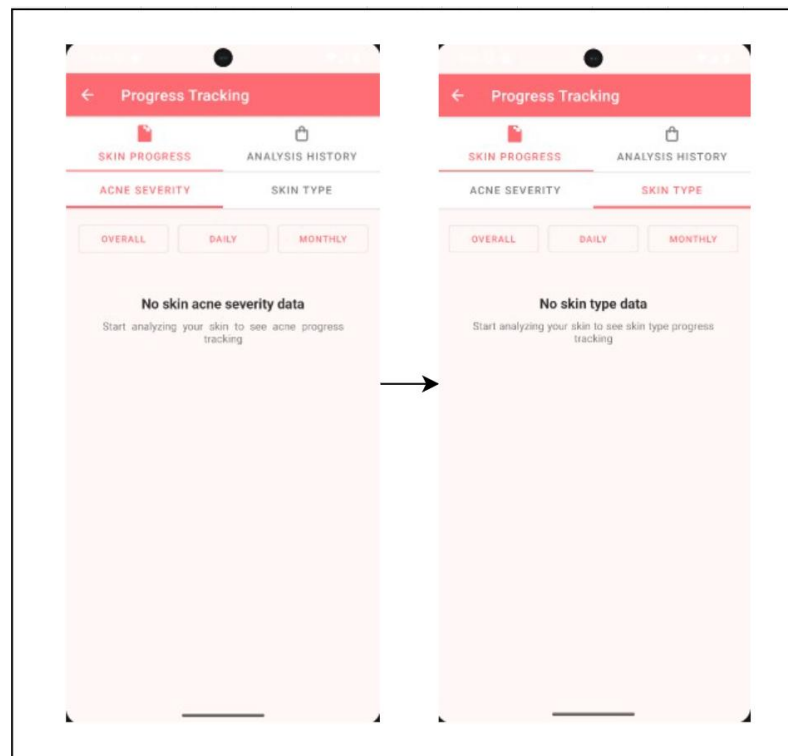


Figure 6.2.5.1 skin progress test case 2

Table 6.2.5.2 Skin Progress test case 2

Test case		skinprogress _002
Module name		Skin progress
Test case description		Verify skin progress Ai analysis where change to GPT-5
Expected result		The user can view the word “powered by GPT-5”
Actual result		The user will view the word “powered by GPT-5”
Test case result		Passed
No.	Test case step	Test case data
1.	Click on “progress tracking” icon	Redirect to progress tracking page and view the acne severity page
2.	Select “skin type” tab	Redirect to skin type page



Figure 6.2.5.2 skin progress test case 2

6.3 Project Challenges

There are several challenges faced in this project, some of which were successfully resolved while others remain unresolved.

Firstly, the learning curve with unfamiliar frameworks such as TensorFlow, Keras, and PyTorch required extensive self-learning through documentation and tutorials. This challenge was successfully resolved, as we were able to gain the necessary understanding of the architecture required to develop and evaluate the models to determine the best-performing one.

Besides, the integration with Firebase and LLM models posed difficulties in ensuring that credential keys were not exposed publicly when publishing code to GitHub. This issue was resolved by storing credential keys in the `local.properties` file and configuring the `google-services.json` file to be ignored during GitHub pushes.

In addition, there is a challenge related to waiting time when triggering the LLM model. When the analysis is executed, the return time is relatively long, sometimes more than four seconds—which could cause user frustration. This issue remains unresolved and requires further optimization to improve response time and enhance user experience.

6.4 Objectives Evaluation

Three objectives in this project were successfully achieved.

The first objective was to develop a skin analysis model for evaluating and determining users' skin conditions. In this project, five AI models were compared, including classic CNN, EfficientNetB0, ResNet50, YOLOv8, and an image classification approach integrated with the OpenAI API. Through the processes of data preparation, preprocessing, model training, and evaluation, the YOLOv8 model outperformed the other models in classifying both acne severity levels and skin types.

The second objective, which focused on recommending skincare products suitable for users with different skin concerns, was accomplished by applying content-based filtering based on feature rules for each acne severity level and skin type. Personalized products were suggested according to the results of the skin analysis, ensuring that recommendations were tailored to the users' specific skin conditions. By providing appropriate product suggestions, the system effectively supports skin improvement while minimizing the risk of potential side effects.

The third objective was to develop a mobile application that integrates the skin analysis and product recommendation system. The application includes four key modules: skin analysis, product recommendations, skincare routine management, and skin progress tracking. The skin analysis module enables users to perform real-time analysis and view results, while the product recommendation module applies content-based filtering to suggest suitable products. Users can add selected products into their skincare routines, and each skin analysis result can be recorded and tracked through the skin progress module. To enhance trust and flexibility, large language models (LLMs) such as GPT and Google Gemini were integrated into the application, allowing users to choose their preferred model. Furthermore, Firebase and Room Database were adopted for efficient and reliable data storage within the mobile application.

6.5 Concluding Remark

In conclusion, the objectives of this project have been successfully achieved through the development of a skin analysis and recommendation system. The solution enables users to analyze their skin conditions using AI models, where YOLOv8 demonstrated the best performance in classifying acne severity and skin type. Based on the analysis results, personalized skincare products are recommended using a content-based filtering approach, ensuring that users receive suggestions tailored to their unique skin concerns. In addition, the mobile application provides four integrated modules which are skin analysis, product recommendations, skincare routine management, and skin progress tracking allowing users to monitor their skin health and manage their skincare journey more effectively. To enhance trust and flexibility, LLM models such as OpenAI and Gemini are incorporated, giving users the option to choose their preferred model. Furthermore, Firebase and Room Database integration ensures reliable storage and synchronization of user data. Overall, the system not only fulfills its intended objectives but also provides a practical and user-friendly solution to support users in improving their skin health.

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

The skin analysis and recommendation system successfully address the issue of uneven performance in AI models, where unreliable accuracy often causes users to lose trust and rely on trial-and-error with skincare products. With this system, users can perform skin analysis and view accuracy scores for each result, helping them evaluate the reliability of the analysis. Before performing the skin analysis, several AI models which are classic CNN, EfficientNetB0, ResNet50, YOLOv8, and GPT assistant-based models were compared through the training and testing processes to identify the best-performing model. Throughout the evaluation, the YOLOv8 was identified as the most robust model, achieving testing accuracies of 76.2% for acne severity and 64.0% for skin type. Therefore, this model was selected for deployment in the mobile application to perform the skin analysis process for users and generate accurate and reliable results.

Furthermore, the skin analysis results are utilized to generate skincare product recommendations through a content-based filtering approach. In this process, the system applies predefined feature rules that map acne severity levels and skin types to specific product attributes. By filtering products according to these rules, the system ensures that only those most suitable for the user's condition are recommended, thereby providing a more personalized and reliable skincare solution. To enhance transparency and provide deeper insights, the system integrates LLM models, offering users the flexibility to select either the Gemini API or the OpenAI API. Users can choose their preferred model to obtain ingredient-grounded explanations and insights about recommended products. Desired skincare products can then be added into a personalized routine, where LLM integration ensures compatibility and suitability across different products, providing detailed recommendations to optimize treatment outcomes.

In addition, the system enables users to monitor their skin progress by collecting analysis results and presenting daily, monthly, and overall trends. Progress tracking is essential for users to understand improvements in their skin condition over time. The system not only visualizes acne severity levels and skin type changes through graphs

but also includes accuracy tracking to reinforce trust in the AI model's reliability. To further support users, descriptive analyses of skin progress are generated using LLM models, providing meaningful insights and recommendations. This ensures the system is both reliable and trustworthy, while also encouraging user confidence in AI-assisted skincare.

The system makes significant contributions to the advancement of AI-assisted skin analysis and recommendation. First, it enhances user trust by incorporating accuracy feedback mechanisms and explanatory outputs generated by LLM models. It also ensures skincare routines are systematically validated to minimize compatibility issues when combining multiple products within a routine. With support from LLM models, users can make informed decisions based on AI-driven recommendations, thereby reducing reliance on professional consultations. Furthermore, the system addresses trial-and-error issues in product selection by improving efficiency and transparency.

Beyond system development, this project has strengthened my technical skills in Java, GitHub, LLM model integration, Firebase integration, and building different model architectures, including classic CNN, EfficientNetB0, ResNet50, YOLOv8, and the GPT assistant via the OpenAI API. Working with these AI models has enhanced my understanding of deep learning. In addition, this project has improved my model deployment skills, ensuring that models can be successfully integrated into a mobile application to provide reliable skin analysis.

7.2 Recommendation

For this skin analysis and recommendation system, several improvements and enhancements can be implemented in the future. Firstly, the accuracy of acne severity and skin type classification can be further improved, as the current dataset is still limited and does not contain enough images to train a more reliable model. Therefore, expanding the dataset with a larger number of images is essential to ensure that the AI models are more accurate, trustworthy, and reliable for users. In addition, expanding the classification capabilities to cover more diverse skin conditions, such as fine lines and large pores, would allow the system to handle a wider range of issues. The implementation of push notifications could also enhance the system by providing

CHAPTER 7

reminders, such as skin analysis alerts, even when the application is closed, ensuring that users continue to follow their skin analysis trends.

Furthermore, the system can be enhanced by extending the capabilities of LLM models to provide richer recommendations and insights to users. This would allow users to gain deeper knowledge about their skincare routines and encourage them to spend more time engaging with the application, ultimately improving both their skin condition and overall user experience.

REFERENCES

- [1] Z. X. Li, K. C. Koban, T. L. Schenck, R. E. Giunta, Q. F. Li, Y. B. Sun, Eds., "Artificial Intelligence in Dermatology Image Analysis: Current Developments and Future Trends." in *Journal of Clinical Medicine*, Nov 18, 2022. [Online]. Available: <https://pubmed.ncbi.nlm.nih.gov/36431301/>
- [2] F. S. Juwanda and H. M. Zin, Eds., "The Development of Skin Analyser for Skin Type and Skin Problem Detection," in *Journal of ICT in Education*, July. 2021. [Online]. Available: <https://ejournal.upsi.edu.my/index.php/JICTIE/article/download/6129/3244/26899>
- [3] T. H. Lim, N. S. F. B. Badaruddin, S. Y. Foo, M. A. Bujang, and P. Muniandy, Eds., "Prevalence and psychosocial impact of acne vulgaris among high school and university students in Sarawak, Malaysia," in *Medical Journal of Malaysia*, July. 2022. [Online]. Available: <https://www.e-mjm.org/2022/v77n4/acne-vulgaris.pdf>
- [4] S. K. Mbatha, M. J. Booysen, and R. P. Theart, Eds., "Skin Tone Estimation under Diverse Lighting Conditions," in *Journal of Imaging*, April. 2024. [Online]. Available: <https://www.mdpi.com/2313-433X/10/5/109>
- [5] "Acne: Learn More – Skin care for acne-prone skin." *National Centre for Biotechnology Information*. Accessed: Dec 5, 2022. [Online]. Available: <https://www.ncbi.nlm.nih.gov/books/NBK279208/>
- [6] B. Shunatona, "This app can track how well your skin-care products are working." *COSMOPOLITAN*. Accessed: Aug 8, 2017. [Online]. Available: <https://www.cosmopolitan.com/style-beauty/beauty/a11648158/youcam-makeup-app-skin-analysis/>
- [7] P. MARTEL. "Spotscan+ is the only available free public tool, that is both user friendly & accurate to evaluate the severity of acne-prone skin." *Loreal*. [Online]. Available: <https://www.loreal.com/en/articles/science-and-technology/la-roche-posay-spotscan/>
- [8] "Spotscan+, acne-prone skin analysis powered by A.I." *La Roche Posay*. [Online]. Available: <https://www.laroche-posay.com.my/en-my/event/spotscan-plus>
- [9] "Best Free Skincare & Skin Analysis App in 2024." *PERFECT*. Accessed: July 19, 2024. [Online]. Available: <https://www.perfectcorp.com/consumer/blog/ai-skincare/how-to-use-skincare-app>

REFERENCES

- [10] A. Kothari, D. Shah, T. Soni, and S. Dhage, "Cosmetic Skin Type Classification Using CNN With Product Recommendation" in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, July 08, 2021. [Online]. doi: 10.1109/ICCCNT51525.2021.9580174
- [11] H. M. W. T. C. B. Samarakoon, A. U. Ovitigala, G. A. T. T. Wijesinghe, K. M. L. P. Weerasinghe, G. K. B. V. Karunathilake and I. Weerathunga, "Facial Skincare Product Suggestion with Product Popularity and Post Recommendation Care," in *2023 4th International Informatics and Software Engineering Conference (IISEC)*, Ankara, Turkiye, pp. 1 - 6, Dec. 12, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10391033>
- [12] A. T. Chan, D. K. H. Chiu, and W. K. Yip, "A New Efficient Convolutional Neural Network Model for Image Classification," in *2018 9th International Conference on Awareness Science and Technology (iCAST)*, Sept 19, 2018. [Online]. doi: 10.1109/ICAwST.2018.8517246
- [13] C.L. Chin, Z.Y. Yang, R.C. Su, and C-S. Yang, "A Facial Pore Aided Detection System Using CNN Deep Learning Algorithm," in *2018 9th International Conference on Awareness Science and Technology (iCAST)*, Fukuoka, Japan, pp. 90-94, Sept 19, 2018. [Online]. Available: <https://ieeexplore.ieee.org/document/8517224>
- [14] A. Kaur, "Revolutionizing Dermatology with High-Accuracy Skin Condition Classification Through ResNet50," in *2024 5th IEEE Global Conference for Advancement in Technology (GCAT)*, Bangalore, India, March 20, 2025. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10924079>
- [15] S. Chauhan. "Deep Learning-Based Skin type Classification using Fine-Tuned ResNet50 Architecture," in *2025 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, Bangalore, India, Jan 16, 2025. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10915335>
- [16] R. Rismayani, A. A. Ilham, A. Achmad, and M. R. Y. Rachman, "Convolutional neural network architectural models for multiclass classification of aesthetic facial skin disorders," in *International Journal of Advanced Technology and Engineering Exploration (IJATEE)*, Jan 19, 2025. [Online]. Available: <https://accentsjournals.org/PaperDirectory/Journal/IJATEE/2025/1/7.pdf>

REFERENCES

- [17] M. Y. Shams, E. Hassan, S. Gamil, A. Ibrahim, E. Gabr, S. Gamal, E. Ibrahim, F. Abbas, A. Mohammed, A. Khamis, M. Hamed, M. Mokhtar, and R. Bhatnagar, "Skin disease classification: a comparison of Resnet50, Mobilenet, and Efficient-B0," in *Journal of Current Multidisciplinary Research*, Jan 27, 2025. [Online]. Available: https://journals.ekb.eg/article_407474_cea60474375672fe45ccaaf88efce162.pdf
- [18] Y. H. Gan, S. Y. Ooi, Y. H. Pang, Y. H. Tay, and Q. F. Yeo. "Facial Skin Analysis in Malaysians using YOLOv5: A Deep Learning Perspective," in *Journal of Informatics and Web Engineering*, June. 2024. [Online]. Available: <https://journals.mmupress.com/index.php/jiwe/article/view/796/538>
- [19] A. Sankar, K. Chaturvedi, A. Nayan, M. H. Hesamian, A. Braytee, and M. Prasad, "RETRACTED: Utilizing Generative Adversarial Networks for Acne Dataset Generation in Dermatology," in *BioMed Informatics 2024*, pp. 1059 – 1070, April 4, 2024. [Online]. Available: <https://www.mdpi.com/2673-7426/4/2/59>
- [20] Y.H. Liao, P.C. Chang, C.C. Wang, and H.H. Li, "An Optimization-Based Technology Applied for Face Skin Symptom Detection," in *healthcare 2022*, Oct 8, 2022. [Online]. Available: <https://www.mdpi.com/2227-9032/10/12/2396>
- [21] J. Akshya, V. Mehra, M. Sundarrajan, P. T. Sri, and M. D. Choudhry. "Efficient Net-based Expert System for Personalized Facial Skincare Recommendations," in *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, May 17, 2023. [Online]. doi: 10.1109/ICICCS56967.2023.10142790
- [22] S Bhuvana; Brindha G S; Shubhikshaa S M; Swathi J V . "Cosmetic Suggestion System Using Convolution Neural Network," in *2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)*, Aug 17, 2022. [Online]. doi: 10.1109/ICESC54411.2022.9885369
- [23] P. H. Aditya, I. Budi, Q. Munajat, "A Comparative Analysis of Memory-based and Model-based Collaborative Filtering on the Implementation of Recommender System for Ecommerce in Indonesia : A Case Study PT X." in *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, Oct 15, 2016. [Online]. Doi: 10.1109/ICACSIS.2016.7872755

REFERENCES

- [24] R. Lathiya, “Acne grading classificationdataset,” Apr. 2021. Available: <https://www.kaggle.com/datasets/rutviklathiyateksun/acne-grading-classificationdataset>
- [25] H. Alwi, “Dataset Skin Type & Skin Condition,” Jan. 2025. Available: <https://www.kaggle.com/datasets/humamalwi/dataset-skin-type-and-skin-condition>
- [26] B. Or, “On Common Split for Training, Validation, and Test Sets in Machine Learning,” Medium.com, Apr 19, 2023. [Online]. Available: <https://pub.towardsai.net/breaking-the-mold-challenging-the-common-split-for-training-validation-and-test-sets-in-machine-271fd405493d>
- [27] Ultralytics. “Explore Ultralytics YOLOv8.” Accessed: Jan 10, 2023. [Online]. Available: <https://docs.ultralytics.com/models/yolov8/#how-do-i-train-a-yolov8-model>
- [28] Y. X. Fu. “Image classification via fine-tuning with EfficientNet.” Accessed: Jun 30, 2024. [Online]. Available: https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/
- [29] Keras, “ResNet and ResNetV2,” [Online]. Available: <https://keras.io/api/applications/resnet/>
- [30] GeeksforGeeks, “Implementing Dropout in TensorFlow,” Jun 19, 2024. [Online]. Available: <https://www.geeksforgeeks.org/implementing-dropout-in-tensorflow/>
- [31] GeeksforGeeks, “Learning Rate in Neural Network,” Nov 2, 2024. [Online]. Available: <https://www.geeksforgeeks.org/impact-of-learning-rate-on-a-model/>
- [32] GeeksforGeeks, “How to choose Batch Size and Number of Epochs When Fitting a Model?” Apr. 21, 2025. [Online]. Available: <https://www.geeksforgeeks.org/how-to-choose-batch-size-and-number-of-epochs-when-fitting-a-model/>
- [33] OpenAI, “Assistants,” [Online]. Available: <https://platform.openai.com/docs/api-reference/assistants>
- [34] GeeksforGeeks, “Understanding the Confusion Matrix in Machine Learning,” Feb 27, 2025. [Online]. Available: <https://www.geeksforgeeks.org/confusion-matrix-machine-learning/>

REFERENCES

- [35] K. R. Hasan, “19000+ Skincare Products Database of Skinsort,” 2023. Available: <https://www.kaggle.com/datasets/kazireyazulhasan/19000-skincare-products-database-of-skinsort>

APPENDICES

APPENDIXES

[illegible]

APPENDIXES

[illegible]

Skin type results using GPT-Assistant Based (25 training images each level with all test images)

[illegible]

Bachelor of Computer Science (Honours)

Faculty of Information and Communication Technology (Kampar Campus), UTAR

APPENDIXES

Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_35.jpg	Predicted: Normal Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_43.jpg	Predicted: Normal Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_44.jpg	Predicted: Normal Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_45.jpg	Predicted: Normal Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_57.jpg	Predicted: Normal Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_58.jpg	Predicted: Normal Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_64.jpg	Predicted: Dry Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_87.jpg	Predicted: Oily Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Normal/normal_95.jpg	Predicted: Normal Actual: Normal
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_10.jpg	Predicted: Oily Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_107.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_113.jpg	Predicted: Oily Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_116.jpg	Predicted: Oily Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_123.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_131.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_135.jpg	Predicted: Oily Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_136.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_141.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_142.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_157.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_163.jpg	Predicted: Oily Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_166.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_173.jpg	Predicted: Oily Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_199.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_24.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_33.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_34.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_51.jpg	Predicted: Dry Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_78.jpg	Predicted: Normal Actual: Oily
Image: https://storage.googleapis.com/dataset_skintype/Oily/oily_90.jpg	Predicted: Normal Actual: Oily

UNIVERSITI TUNKU ABDUL RAHMAN

SKIN ANALYSIS AND RECOMMENDATION SYSTEM

Author : Ng Yong Shen

Supervisor : Dr Kh'ng Xin Yi

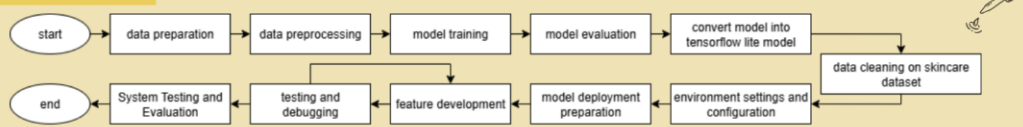
INTRODUCTION

Skin conditions such as acne have become a significant concern, often affecting individuals' confidence and social interactions. To address this issue, AI-driven skin analysis systems are emerging as essential tools for providing accurate assessments and personalized skincare recommendations.

OBJECTIVE

- develop skin analysis model for evaluating and determining users' skin conditions
- recommend skincare products suitable for users with various skin concerns
- develop a mobile application for skin analysis and recommendation system

METHODOLOGY



PROPOSED METHOD

Implemented database as a backend service for storing information

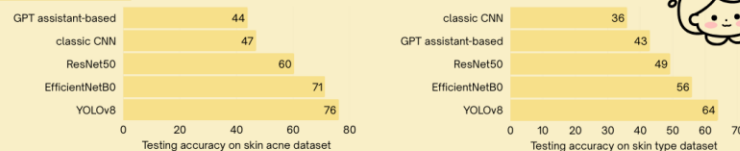
develop Java-based mobile application

Applied content-based filtering for skincare product recommendations.

Integrated Gemini and OpenAI LLM models to provide insights to users

YOLOv8 outperformed Classic CNN, ResNet50, EfficientNetB0, and GPT-4o in acne severity and skin type classification, and integrated into the system for skin analysis

ANALYSIS



WHY OUR APP BETTER

- Provide broad range of skincare product recommendations
- Reliable Acne Severity & Skin Type Results with clear accuracy metrics
- Utilize the LLM models to provide insights for product selection and progress tracking

FUTURE WORK

- Expand dataset with more images for reliable model training to improve accuracy
- Cover a wider range of skin conditions, such as fine lines and large pores
- Send push notifications even when the app is closed
- Enhance LLM capabilities to provide richer recommendations and deeper skincare insights

