Ryerson University
Department of Electrical, Computer & Biomedical Engineering
Faculty of Engineering & Architectural Science

Epidermis Lesion Classification System

By

Andrew Mullen, Farnaz Forooghi, Mirt Mehany and Nour Abu Hantash

BME 70A/B Biomedical Engineering Capstone Final Thesis Report
Ryerson University, 2020-2021

Acknowledgements

The completion of this undertaking project could not have been possible without the participation and assistance of so many people whose names may not all be enumerated. Their contributions are sincerely appreciated and gratefully acknowledged. However, the group would like to express their deep appreciation and indebtedness particularly to professor Omar Grant for his guidance and constant encouragement. Without his knowledge and constant support, we would not have reached our goals.

Table of Contents

Table of Figures

1. Abstract

Skin lesions usually appear in different parts of the body with different shapes. These lesions can be classified into 3 main categories: benign, malignant, and precancerous. Malignant skin cancer contains different stages. Based on the results from the Canadian Cancer Society, the survival rate in the early stage is higher than 90% [1]. Due to the unexpected pandemic, visiting physicians is mostly virtual making the detection of problems hard for physicians [2]. Artificial intelligence can facilitate this challenge. Pre-existing images from Harvard Dataverse (HAM10000) were pre-processed through downsampling, filtering and normalization. The processed images were passed through different deep convolutional neural networks (DCNN). The DCNN outputs were concatenated into a feature vector that was utilized by a linear regression model to facilitate classification. The machine learning model output was classified into 3 main categories: benign keratosis-like lesions (BKL), melanocytic nevi (NV) and melanoma (MEL). The designed and implemented pipeline was able to achieve an overall accuracy of 80.5%. The classification process is accessible by users through a mobile application. Users are able to upload their own image to be classified by the model. Upon classification the results are relayed back to the mobile application to be presented to the user.

Keywords: skin lesion, frontend, backend, classifier

2. Introduction & Background

Skin lesions are areas of the skin that look different from its surrounding area. These lesions can occur in different parts of the body, for various reasons and can present as patches or bumps. They are classified into three main types: benign, precancerous and malignant [3].

Benign skin lesions are non-cancerous skin blemishes that may appear in the patient's skin and discover during routine skin examinations. These types of lesions are not specified in gender and age. There are common features for benign skin lesions such as symmetry in color, structure and shape, stable or slow growth and no spontaneous bleeding. These lesions can be classified by their origin cellular to melanocytic, keratinocyte, vascular, fibrous and etc. [4]. For this project, the benign keratosis lesion was chosen as a benign lesion. A seborrheic keratosis is one of the common skin lesions that tend to appear in the skin as a person gets older. These lesions are usually darker than a person's skin color and close to brown or black. The growths of them look waxy, scaly and slightly swollen. They mostly appear on the head and neck or upper limb. Seborrheic keratoses are harmless and non-contagious lesions. Usually there is no treatment for them, however if they become irritated by clothing or bother the patient, then they may decide to remove them [5].

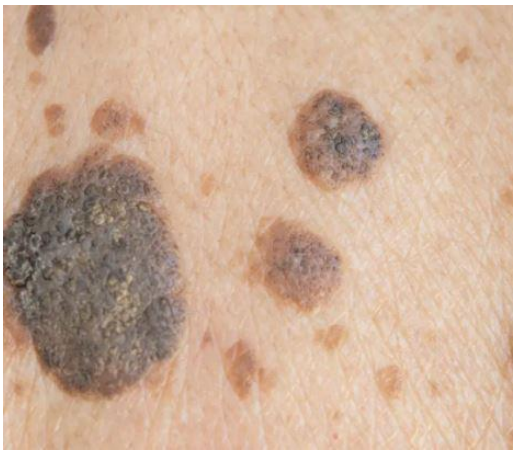


Figure 1. The benign keratosis-like lesion [6]

Precancerous skin lesions are regarded as different skin growths that have a high risk of developing into skin cancer. Typical precancerous skin lesions may develop into malignant melanoma and or squamous cell carcinoma. One of the main ways to prevent these lesions transferring to the malignant is skin removal surgery which should be done before developing to an advanced stage [7]. Melanocytic nevi are benign reproductions of melanocytic cells, known as nevus cells, and they are gathered in nests in some tissues such as epidermis or dermis. They mainly appear at birth in skin and develop until 6 months after. Melanocytic nevus is generally brown lesion without any swollen surface, and they are primarily diagnosed by physicians or dermatologists. These congenital lesions are identified as a high-risk factor of developing melanoma; however, the accurate quantity of the risk is still unknown [8].

Figure 2. The melanocytic nevi lesion [9]

Malignant skin lesions are skin tumors that can develop from squamous cell carcinoma and malignant melanoma. Malignant lesions are growing, spreading and pigmented in specific areas of the body for particular reasons. These lesions are easily distinguished from healthy skin; however, the physician mostly does the biopsy to confirm the level of malignancy. In addition, recently, there are some references of Artificial intelligence that allows users to detect the cancerous mark with a noninvasive method to prevent touching and develop the sensitivity of the skin [10]. A melanoma is the most common malignant skin cancer that is produced from the transformation of melanocytes. They usually occur on the epidermis, however it can occur in other locations where neural crest cells might migrate such as the brain [11].

Figure 3. The Melanoma lesion [12]

3. Objective and Scope of Epidermis Lesion Classification System

Based on the Government of Canada public health statistics, one third of newly diagnosed cancer cases in Canada are skin cancers. In more detail, one out of 73 Canadian women and one out of 59 Canadian men will develop melanoma specifically. Based on these statistics, action must be taken in effort to reduce these numbers and save lives [13]. There are multiple precautions one can take to prevent the development of most skin cancers. This includes, limiting the time in the sun, using sunscreen and staying hydrated. Moreover, early detection of cancers is proven to increase survival chances [13]. Therefore, it is always encouraged to examine your skin often and to seek a professional opinion upon detection of abnormal skin patches [13].

As our world changes, new innovative solutions are necessary to keep up with the new challenges of the world. Due to COVID-19 pandemic, new safety protocols were introduced in order to maintain social distance and limit the spread of the virus. Online doctor appointments were introduced to make up for the limited in person doctor visits introduced as a safety precaution [2]. This precaution presents a huge challenge for doctors and patients in need of diagnosis. Luckily, due to the advancement in today's technology, we are well equipped to overcome the challenges associated with the virtual medical world Based on a study conducted by the Canadian Cancer Society, 92 - 97% of cancer patients survive due to early diagnosis [1]. One of the potential sectors to overcome this challenge is Artificial intelligence (AI).

The work described in this design report is aimed at developing a mobile application available for the public to obtain a preliminary diagnosis on their skin lesions. The trained model is able to identify three classes of skin lesions including benign keratosis-like lesions (BKL), melanocytic nevi (NV) and melanoma (MEL). This application allows the user to upload an image of their lesions in which this image is communicated to a pre-trained model for preprocessing, feature extraction and classification. Based on the classification results, the model outputted the prediction of the input lesion. Providing such an application to the public with an accuracy higher than 80% can help with early diagnosis and save lives. Moreover, it can help increase the speed and efficiency of diagnosis if used in clinics.

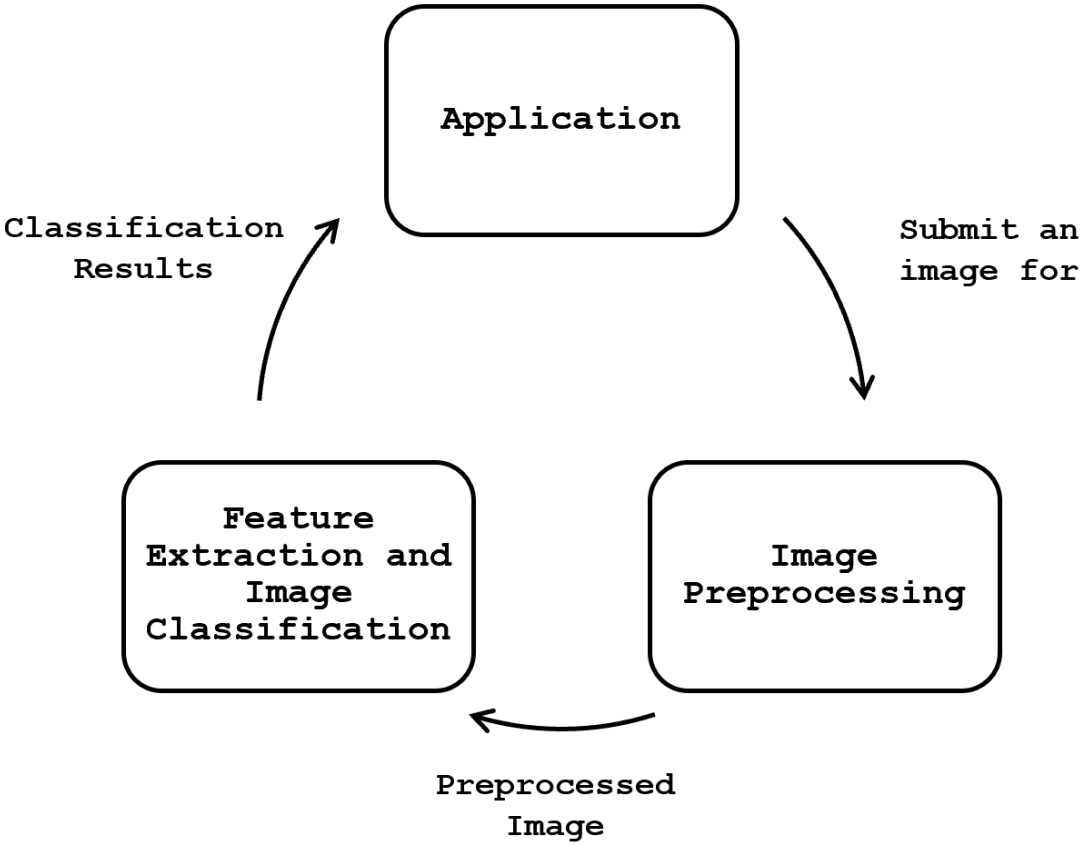


Figure 4. High-level overview of the application architecture

4. Theory and Design

4.1 Epidermis Lesion Database

The database used for image sources was the Human Against Machine with 10000 training images (HAM10000). This dataset contains 10015 images across various common pigmented skin lesions. The various skin lesions are actinic keratoses (n = 127), basal cell carcinoma (n = 514), benign keratosis like lesion (n = 1099), dermatofibroma (n = 115), melanoma (n = 1113), melanocytic nevi (n = 6705) and vascular lesions (n = 142). The labels are melanoma, benign keratosis-like lesion and melanocytic nevi. Due to the abundance of melanocytic nevi images only 1116 were selected to keep it in the same order as the other two classes. With greater than 500 images per class available for use in training the deep learning model, extensive data augmentation was not necessary to increase image quantity. With a total of 1664 images eligible for use, they were divided into 60% training (n = 1061), 20% validation (n = 266), and 20% testing (n = 337). For the logistic regression data, there are 1664 images available for use and these were divided into 80% training (n = 1330) and 20% testing (n = 334). Each class is represented equally across all training validation and testing groups.

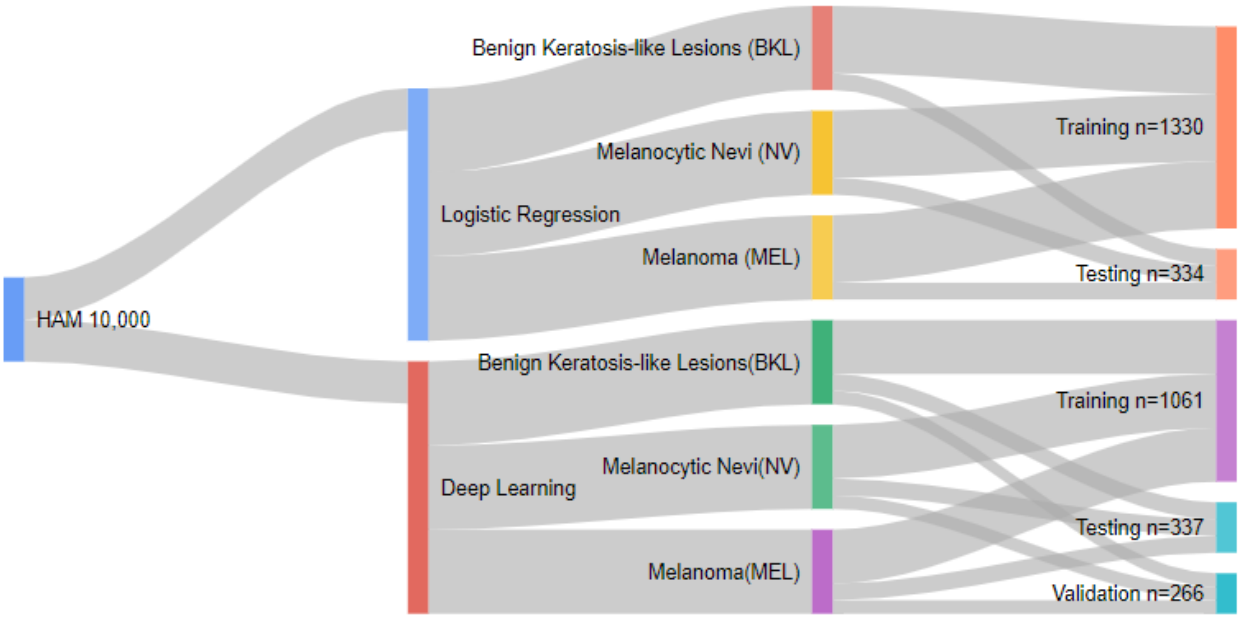


Figure 5. Database breakdown

4.2 Epidermis Lesion Classification System

The ensemble model for epidermis lesion classification system was developed using Tensorflow2.0. TensorFlow2.0 is the most popular machine learning open-source platform with flexible and comprehensive tools, libraries and community resources. TensorFlow provides different APIs such as Python and Keras. Keras is a high-level API that runs on top of TensorFlow. It is fast, easy to extend, easy to run and approachable [14].

4.2.1 Image Preprocessing Design

Once all images were downloaded from the HAM database, they were saved into 3 separate folders for each subclassification of skin lesions based on its ground truth. All images were *.jpg* format. The required steps of preprocessing the images was chosen based on the input requirements of machine learning libraries. All images should be square with 3 channels for RGB (**R**ed, **G**reen and **B**lue). Since all images in the HAM10000 database are already square, therefore this aspect was not changed. The images were down sampled to a 224x224 pixels resolution. The ImageDataGenerator class in Keras was used for shifting images, zooming in and out and rotations.

4.2.2 Model Training

Once all the images were preprocessed, they were saved in their respective class folders once again. Next these preprocessed images were divided into training (60%), validation (20%) and testing (20%) image folders while maintaining even distribution amongst the three classes in each folder. This process needs to be randomized to assure no bias is introduced in the process. In the training process for neural networks, training images got put through the model and the model's performance got calculated. This is an iterative process as the model adjusts its weights every time in an effort to improve performance by reducing loss. This is called gradient descent. The image data input stream was managed by the ImageDataGenerator class in Keras [15]. This class allows for data augmentations to be accomplished at the time of training. Shifting the image, zooming in and out, and rotations are some of the methods that are applied which contributed to the robustness of the model.

4.3 Epidermis Lesion Classification System Design

The classification system utilized an ensemble framework with three pipelines that fed into a logistic regression classifier. This method aims to improve neural networks performance by combining predictions from multiple models to reduce neural networks inherent non-linearity, high variance and low bias [16]. Three different architectures were explored, ResNet, VGG and InceptionResNetv2, and various combinations were attempted aiming to get the most optimal combination. The thinking behind this is that two pipelines were specific to our task and the third was general features that affected decisions when the first two were in conflict. Of the first two pipelines, one of them was trained using transfer learning by preloading weights from an ImageNet competition model. Transfer learning utilizes previous networks' learned features and applies them to a new task. These features are often generalizable to a wide domain of tasks and can reduce training time and improve performance of networks [17]. The other pipeline was a freshly trained network allowing it to learn specific features applicable to the task. Obtaining both general and specific information from images has proven to be valuable in medical image segmentation as shown by Ataloglou et al. [18], and the proposed method aimed to prove its efficacy for medical image classification. Combining these two neural networks in an ensemble format aimed to combine the benefits of both learning techniques to provide the highest accuracy [19]. The third pipeline was an architecture that has also been previously trained in the ImageNet classification competition. Instead of adjusting the classification layers of the model to a three-class prediction, it was left to output a 1000 class probability vector. This vector was treated as features of the image and fed to a logistic regression classifier.

This idea is based on previous work in which a group successfully fed a similar architecture's output to a Support Vector Machine and Random Forest classifiers with good success [20]. Our approach is a novel idea in which the outputs from the three pipelines were concatenated into a feature vector and fed to a classifier. We chose a binary logistic regression classifier upon seeing the success of this type of classifier in a paper from Arthur Marka et al. [21]. In theory the first two pipelines mentioned, would output three class probabilities and be highly correlated with the end result after the logistic regression classifier. The idea is that when the two three class prediction pipelines are in conflict, that the third pipeline (1000 class feature vector) would influence the logistic regression classifier in the right direction.

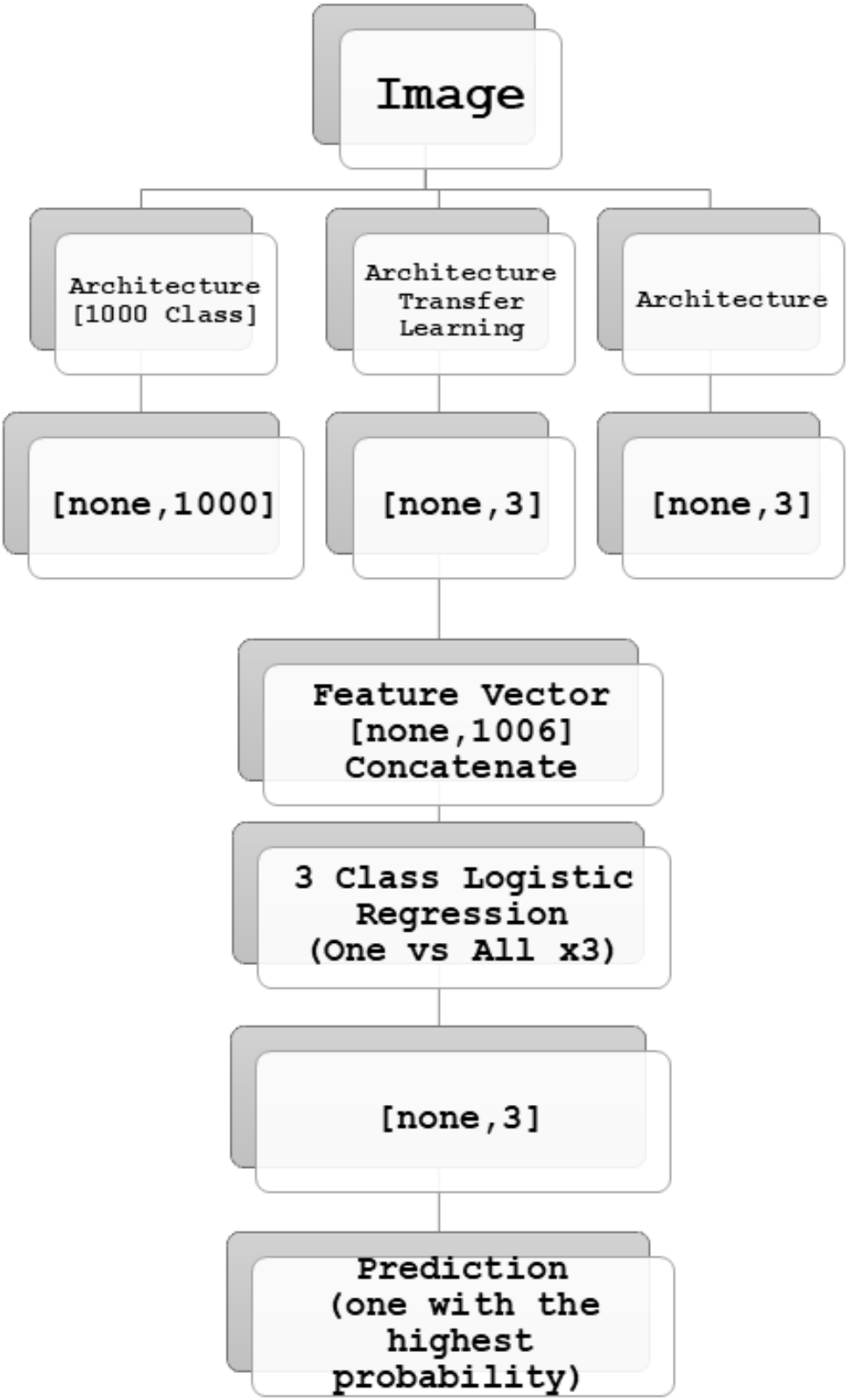


Figure 6. Ensemble framework for skin lesion classification

4.3.1 Neural Networks

The three architectures that were explored are the VGG-19, ResNet 152 and the Inception ResNet v2. These architectures are combinations of convolution, max pooling, concatenation, dropout and max pooling layers in various orders. The VGG-19 is the simplest model as it is a feed forward neural network with no recursive layers [22]. That is to say there is one path down the whole model. Although simple the VGG-19 is a deep layered model meaning it extracts deep, hidden information that proves valuable in classification [22]. ResNet 152 is based off of the VGG-19 except it introduces the shortcuts [23]. These shortcuts connect skipped layers and transform it into a residual network (ResNet) [23]. The Inception ResNet v2 is the most visually intimidating of the three, however it simply follows an inception architecture with added residual connections [24]. Inceptions are naturally very deep networks and should therefore extract quality features [24].

Figure 7. VGG Architecture [25]

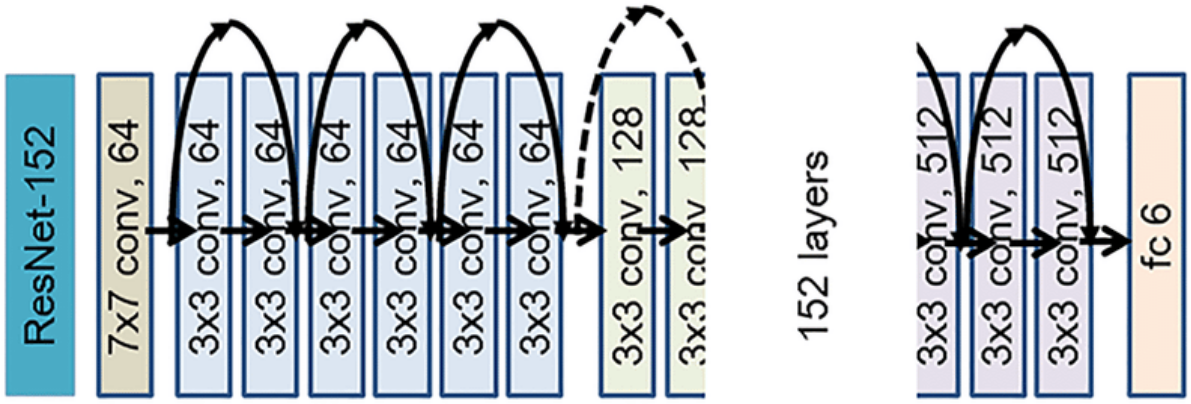


Figure 8. ResNet architecture [25]

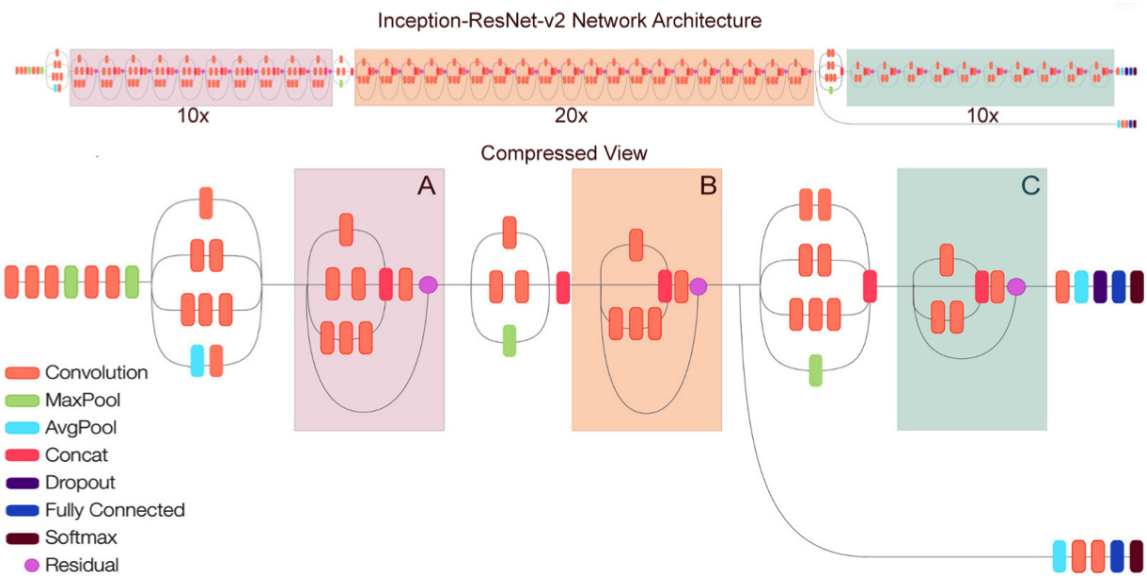


Figure 9. Inception ResNet v2 architecture [26]

4.3.2 Logistic Regression Classifier

For the last classification layer, it had been chosen to use a binary logistic regression. This classifier is inherently a binary, which means three different models were developed for the classifier [27]. One for each class versus the other two classes. These models took inputs from three pipelines in the form of a feature vector and output a probability for their respective classes. Each of the three models produced a probability that the input image belongs to its respective class. The class that has the highest probability was then taken as the model's predicted class. This layer was developed using the scikit learn API. This provides a simple to use multi class logistic regression as well as it gives you the ability to change many hyperparameters to optimize performance. A benefit of this API is that it applies feature regularization to prevent overfitting through the passing of an argument. In order to deploy this model to the database it had to be converted from its '.pkl' format to a '.hdf5' format. In order to do this it was converted to a tensorflow model by transferring its scikit learn model's coefficients and bias to a TensorFlow's dense layer.

4.4 Application User Interface Design

The mobile application consists of two main components: the frontend and the backend. The frontend component is represented by the graphical user interface (GUI) design running natively on Android Operating Systems (OS). Meanwhile the backend is represented by the cloud servers on Firebase. The user interface facilitates the user interaction with the classification model and gets the appropriate results via a mobile device. Based on the user's choices through the frontend, the application ensures a successful back and forth communication with the backend components of the application.

The User Interface (UI) is the frontend component of this application. It is developed using Android Studio's IDE. Android Studio provides a friendly development environment. It also provides free tutorials for beginners [28]. In addition, the Flutter framework was selected to carry out the UI design. Flutter is a toolkit used to create cross-platform UIs created by Google in order to allow developers to create applications suitable for multiple OS[29]. By using this toolkit's plugin in Android Studio, the built-in advanced feature blocks, called widgets were used to carry out this project [29]. The language used to develop the front end is Dart coding language, which is an object-oriented language with C-style syntax [30].

The other component is Firebase. Firebase acts as the backend service/framework [31] of the application that holds the classifier model, user information, user history, and images.

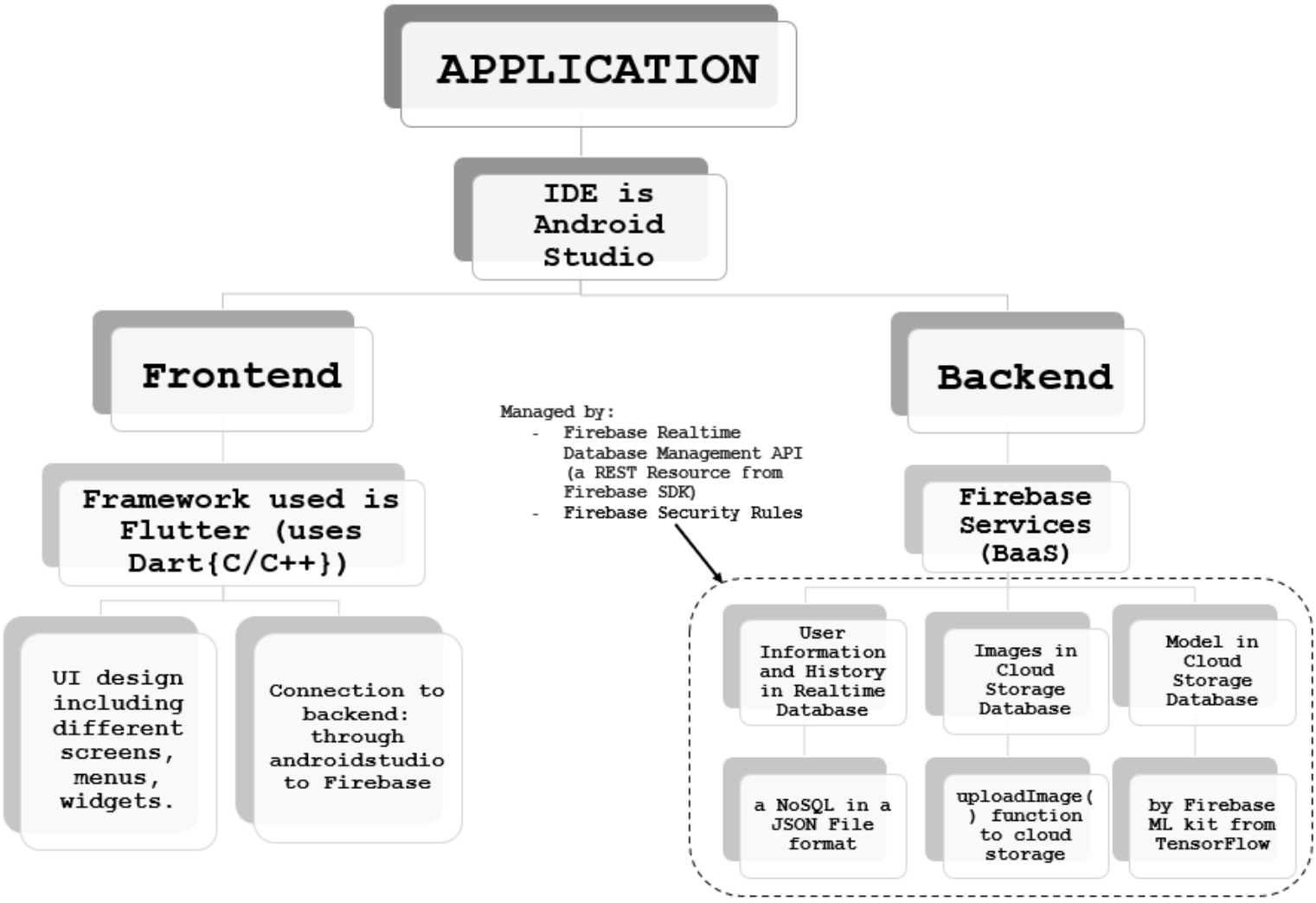


Figure 10. Flowchart of the application explaining the main tools used at each point of the application workflow.

4.4.1 Frontend User Interface Design

The user interface design is one of the most important aspects of any application due to its huge impact on traffic influx and user attraction. Therefore, ensuring an alluring design with an easy and intuitive interface are key components to a successful user interface design. There are a few basics that should never be neglected. This includes easy menu navigation, consistency, and a notification system. Keeping that in consideration, the flowchart shown in figure 11 represents the workflow of the application. The main features our application offers are photo submission, photo analysis and result reporting. Other components include user registration, user login, about us, insight, and history.

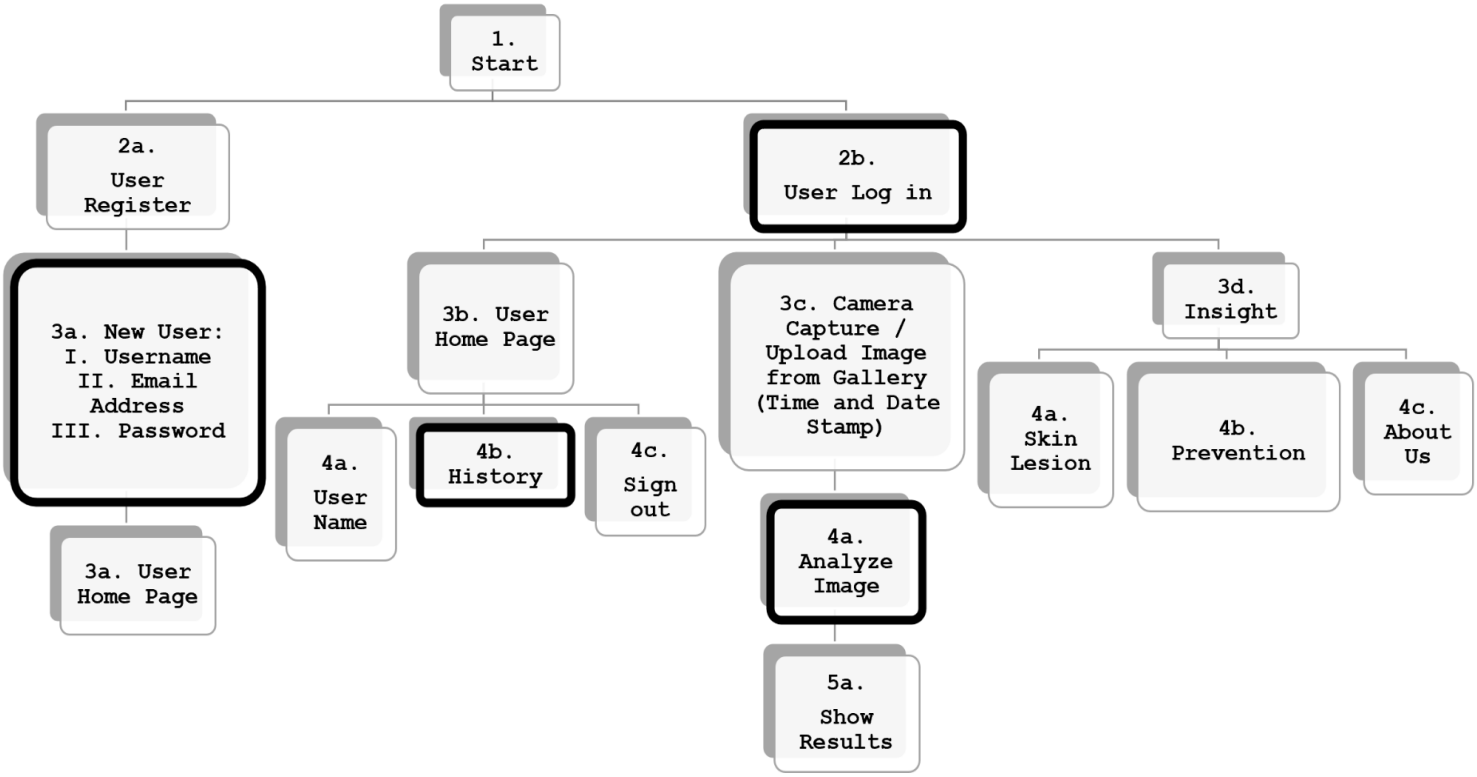


Figure 11. General flowchart of the application workflow where the bold-bordered boxes indicate points of interaction with the backend components of the application.

4.4.1a Image uploading, submission and results reporting activity

The main feature our application offers is image capture and submission to the prediction model followed by the subsequent actions of analysis and results reporting. The application allows the user to upload their images to be classified by the prediction model. The user is granted the use of both their camera to capture images and their gallery to upload an already captured image. The customized camera API is designed to provide the user with guide markers. These markers will direct the user to the correct way of capturing their skin lesion in order to avoid unclear and uncentered images of their lesions. Moreover, the user is granted the use of both the back and the front camera. In order to carry out these functionalities, some of the available Flutter plugins were used. In detail, Camera Plugin was used to gain access to the mobile camera while Image_Picker plugin was used to access the gallery.

As the user selects the desired image, they get to select the Analyze option. As the image is submitted for analysis, Firebase_storage plugin is employed to pass the image to the prediction model on the Firebase platform. The Firebase server is capable of communicating the results to be displayed on the user interface. The image used is logged into our database along with the output of the prediction model. The importance of image logging is that the user is granted the chance of accessing them later in time through the History screen.

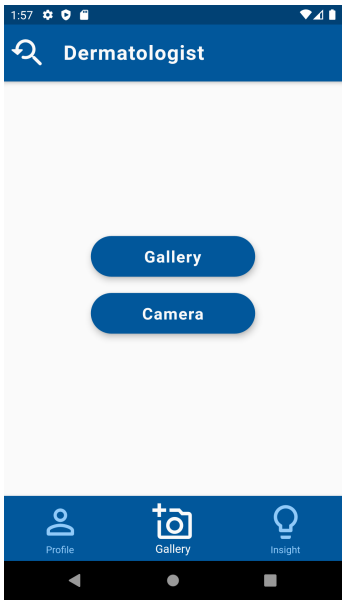


Figure 12a. Gallery screen to upload image from gallery or capture a new image using the camera

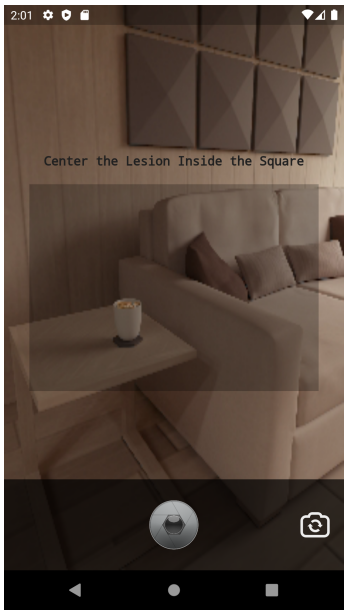


Figure 12b. Camera screen to allow the user capturing a new image using either the back or front camera in addition to a guiding overlay marker.

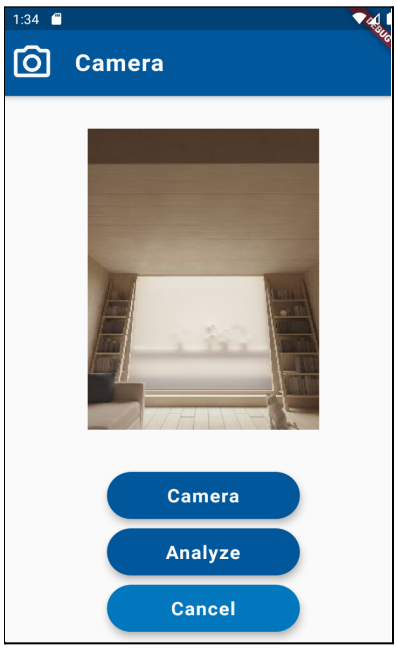


Figure 12c. Image view screen to view the captured image

Figure 12d. Results screen to view the results reported by the classification model

4.4.1b User Registration Feature

For privacy protection, our application offers user registration and user login features. User registration is an important part of the application. The user creates a new account to be able to use the application in which they will be able to retrieve their analysis history. Users are asked to enter username, email address and password. New user's credentials are stored on the user database. Each registered user is assigned a generated user ID. Before the user is registered on the database, the email address uniqueness check is done first in order to prevent multiple registrations with the same email.

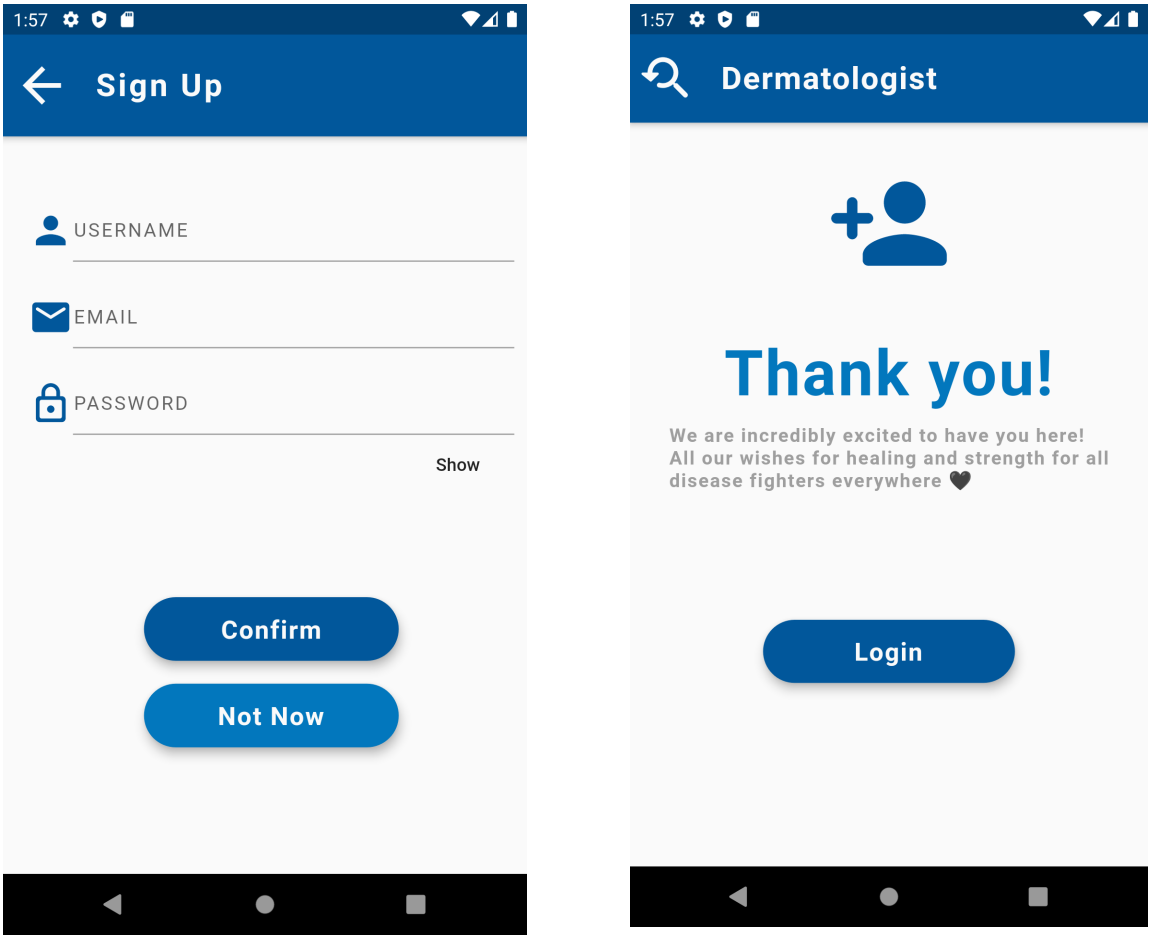


Figure 13. New user registration screen requests the user to enter their name, email and password to be stored on a user information database

4.4.1c User Login Feature

For users with an already existing account, the login feature enables them to access their profile using their email address and password. These credentials aim to protect our clients' privacy and secure the access to their information on our database. As the user tries to log into their account, based on the entered email address, the input password shall be verified with the database. After the credentials check is done, the user is granted access to their profile. In case the user enters the wrong credentials or leaves one or both of the log in fields empty, a notification message shall pop-up to effectively address the issue to the user.

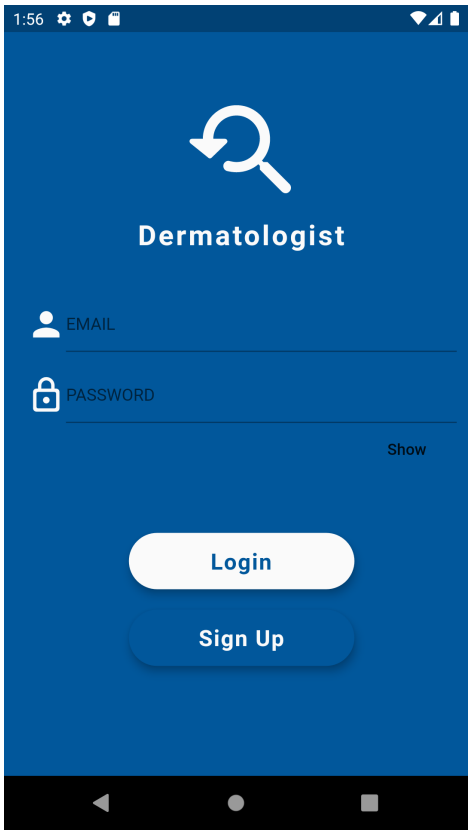


Figure 14. General Login screen interface

4.4.1d Secondary Features

The secondary features aim to grant the user with optimal user experience while using the application. To begin, as the user logs in, they get to access their history to be able to follow the progression of their lesion over time. In order to maintain our users' privacy while providing a customized experience, the profile picture is designed to show the first letter of the username capitalized as can be seen in figure 15 below. Furthermore, by selecting 'Insight', users can find useful information about skin lesions, prevention tips to stay healthy and finally 'About Us' that allows users to know more about the developers and the purpose behind the application.

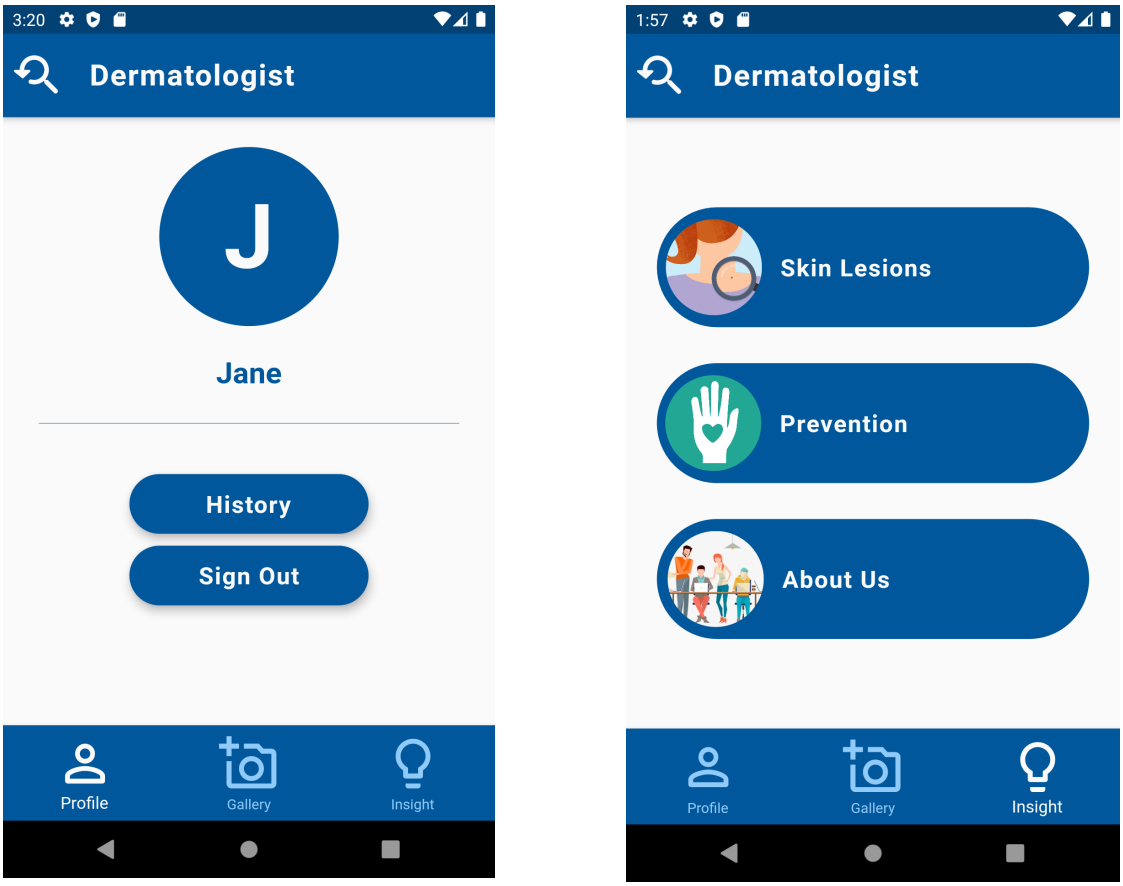


Figure 15. General home screen and insight screen interfaces

4.4.2 Back End User Interface Design

The designed application is served by cloud services. These cloud services include the Authentication, Realtime Database, and the Cloud Storage Database. These services are offered and managed through Firebase. Firebase is a Backend-as-a-Service (BaaS) provided by the Google Cloud Platform. This BaaS is a cloud service model that provides pre-written software for activities that take place on servers, including user authentication, database management, remote updating, cloud storage and hosting.

The Firebase platform is utilized as a backend framework along with Flutter which was used to build the frontend framework. The backend components were implemented using a combination of Kotlin, Java, and Python. To ensure the flow of the application and communication between the backend and frontend, MethodChannels were implemented to call platform-specific APIs available on Android in Java. For instance, a MethodChannel was implemented to obtain the values the user inputs in the Login screen (figure 14) and send them to the Store, Login and Authentication functions implemented in Java which connect to the Firebase Realtime Database and Authentication services.

The Firebase Authentication service provides an SDK. The SDK includes methods that were implemented to create and manage users. The users were able to create an account and log in to the account by providing the used email and password. With the creation of every user, a UID, a unique identifier for that user, is created and linked to the user's account. This UID is inaccessible by the user and cannot be modified by the user or the admin. Due to the uniqueness of the ID and inability to modify it, it was used to manage the users within the Realtime Database. Along with the UID, the user's email and password are stored by theAuthentication services. On a login activity, the user's information was verified with the Authentication services.

The Realtime Database is a NoSQL (non-tabular and non-relational) database which is managed by the Firebase Realtime Database Security Rules [31]. These rules define how the data is structured and when data can be written to or read from. When these rules are integrated with Firebase Authentication, what data is accessed by the user is defined. The data is stored within the database as JSON objects (or a JSON tree). Instead of implementing HTTP requests and responses to sync and retrieve data [32], WebSockets and HTTP Long Polling are utilized by the Firebase SDK to carry out data synchronization and transactions. This allows for the data to be synchronized in real time to any connected users and for all users to share one database regardless of the platform (iOS or Android) [31].

In the Realtime Database the UID is used as the highest level object in the JSON tree under which the rest of information is stored. The Realtime Database holds the user's email information, username, history (previous results), and the reference to the user's images which are held in the Cloud Storage Database.

The process of communication between the Realtime Database and application is detailed in figure 16. Using Firebase facilitates most of the communication between the application components but poses some challenges including: the possibility of having to pay to use it at some point in the future, managing the user information that's saved as a JSON file instead of tables, and using a BaaS platform as a backend limits data hosting and migration.

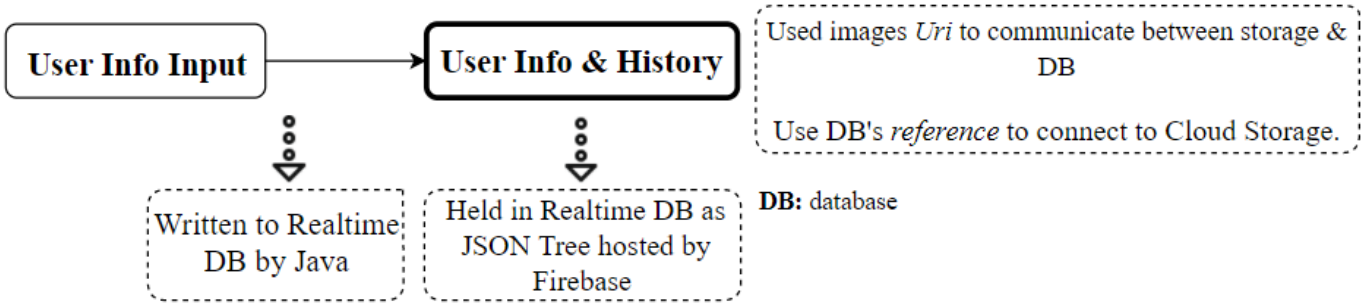


Figure 16. Communication between the application and real time database

The second cloud service being utilized is the Cloud Storage Database. As a backend component, the classifier model is held by Cloud Storage which is serviced by Firebase. Images are also saved in the Cloud storage and given the same id as the user's iterated one as a part of their label to be referenced to the user information and history in the Realtime Database. Similar to the Realtime Database, when the Firebase SDKs for Cloud Storage integrate with Firebase Authentication, security for the data is established.

The Keras classification models (HDF5 file format) are also stored in Cloud Storage. Using a FireBase Python SDK, the model was packaged and deployed to the Firebase ML that is connected to the same project. The active model is deployed from Firebase ML to the application when called by the Java code that feeds the image to the classifier. After the analysis is carried out, the classification output is displayed to the user and the model will no longer be accessible by the application or the user until the next analysis process is requested. This process is detailed in figure 17 below.

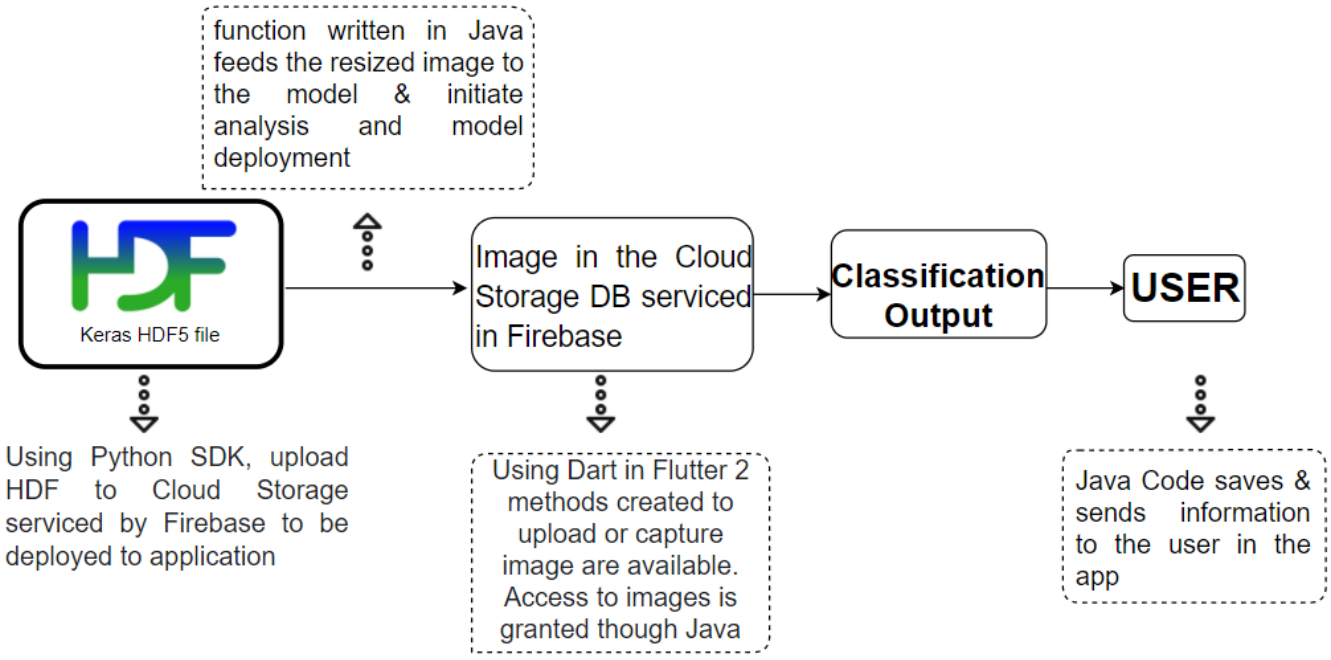


Figure 17. Communication between images and the model

5. Material/Component list

Throughout the process of carrying out this project no tangible materials were used and the cost of using the chosen tools is zero. The database used for images is an open source provided by Harvard Dataverse. Online services such as Google Collaboratory have been utilized to train the classifier and the Firebase platform along with Android Studio IDE have been used to implement the application. It is important to note that the online services used, especially Firebase, are free within certain limits and charges may occur in the future although that is very unlikely.

6. Observation and Results

6.1 Results of Classification system Design

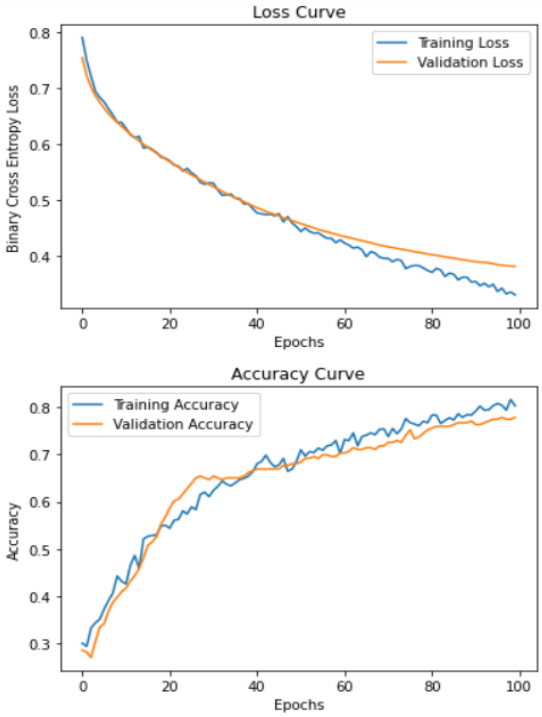
All three architectures were trained with transfer learning and normal learning to extract desired features. In this part, only one architecture was chosen based on its performance and accuracy percentage. The last pipeline extracted 1000 features for input to the logistic regression classifier. For this reason that model was left untouched and therefore no results are available from there.

6.1.1 Result from deep learning

Based on achieved values, the ResNet50 architecture had the higher accuracy in both transfer and normal learning training. The loss curve and confusion matrix was used to find the best architecture to be trained.

Table 1.1 Accuracy % for all datas with different training models

	Training Accuracy %	Validation Accuracy %	Testing Accuracy %
Transfer Learning	81.6	77.8	61.4
Normal Learning	93.2	74.1	61.9




```
Predicted  bkl  mel  nv  All
Actual
bkl         40   58  13  111
mel          3  109   1  113
nv           7   48  58  113
All         50  215  72  337

Predicted       bkl       mel       nv
Actual
bkl        0.360360  0.513274  0.115044
mel        0.027027  0.964602  0.008850
nv         0.063063  0.424779  0.513274
```

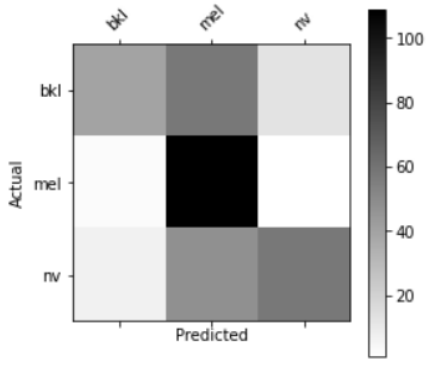
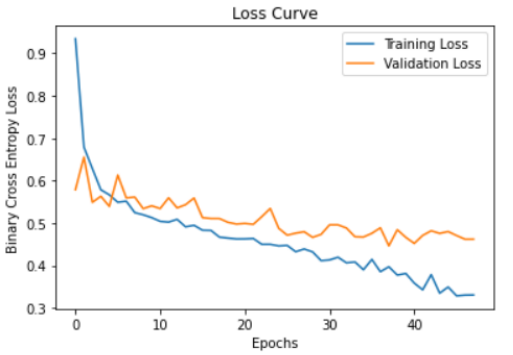
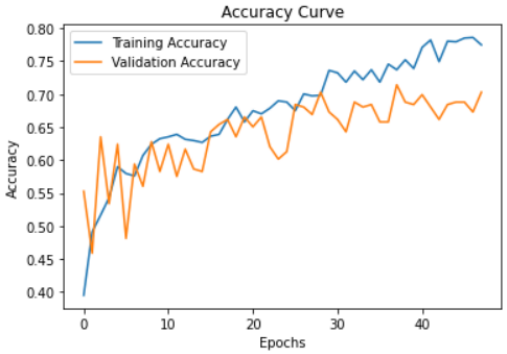


Figure 18. Loss and accuracy curve for model trained with transfer learning & confusion matrix



Predicted	bkl	mel	nv	All
Actual				
bkl	74	31	6	111
mel	14	92	7	113
nv	27	19	67	113
All	115	142	80	337

Predicted	bkl	mel	nv
Actual			
bkl	0.666667	0.274336	0.053097
mel	0.126126	0.814159	0.061947
nv	0.243243	0.168142	0.592920



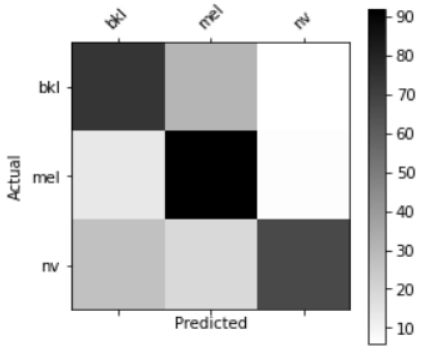


Figure 19. Loss and accuracy curve for model trained with normal learning & confusion matrix

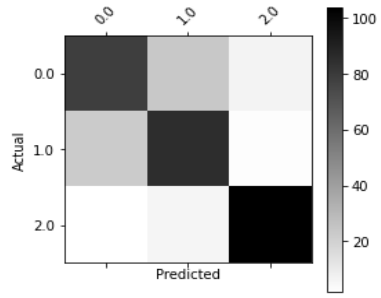
6.1.2 Result from logistics regression classifier

Once all features are concatenated, the final result is fed to the logistic regression for prediction. The logistic regression takes the probability of each class and predicts the class based on the highest probability. The ensemble model has the accuracy of 80.5% which is much higher than the single training accuracy of each pipeline. The other variables such as f1 score , precision and recall were used to support the accuracy of performance.

The precision is the vicinity of each measurement, while accuracy shows the closeness of the class value to a specific value [36]. The f1 score is the highest value of accuracy of each measurement. As figure 20 shows, the f1 score of this ensemble model is 80.5% which shows the model is improving in the accuracy. This shows the model can predict the class of the data with the accuracy of 80.5%.

Predicted	0.0	1.0	2.0	All
Actual				
0.0	79	24	7	110
1.0	23	86	3	112
2.0	2	6	104	112
All	104	116	114	334

Predicted	0.0	1.0	2.0	All
Actual				
0.0	0.359091	0.107143	0.031250	0.164671
1.0	0.104545	0.383929	0.013393	0.167665
2.0	0.009091	0.026786	0.464286	0.167665
All	0.472727	0.517857	0.508929	0.500000



Score: 0.8053892215568862

```
[[ 79  24   7]
 [ 23  86   3]
 [  2   6 104]]
```

	precision	recall	f1-score	support
0.0	0.760	0.718	0.738	110
1.0	0.741	0.768	0.754	112
2.0	0.912	0.929	0.920	112
accuracy			0.805	334
macro avg	0.804	0.805	0.804	334
weighted avg	0.805	0.805	0.805	334

Figure 20. Confusion matrix and other parameters of logistic regression classifier

7. Analysis of Performance

The classification results can be interpreted at different stages. The first deep learning stage has two different result outputs, one from the transfer learning pipeline and the other from the normal pipeline. At this stage the classification accuracy for both was approximately 61% and from both of their confusion matrices (figures 18 & 19) it can be seen that it has a general bias towards the melanoma class. The transfer learning model exhibited a heavier bias with 215 out of 337 images being predicted as melanoma and the normal learning model predicting 142 out of 337 images as melanoma. When looking at the output results from the last logistic regression classification layer, you can see that this bias is practically eliminated as well as the accuracy improved. This last layer accuracy improved to 80.5% which is about a 20% increase. Another benefit is that the bias is eliminated with an equal number of predictions across classes. This shows the benefit of adding this last layer. The transfer and normal learning pipelines output class probabilities, however when these are not in agreement or the probabilities are not strong the model has the ability to make a better prediction with the extra knowledge discerned through the 1000 class feature extraction pipeline.

8. Alternative Designs

8.1 Epidermis Lesion Classification System Design

Based on the reviews from previous articles [20,21,22] and available scholarly references, the alternative design for classification system was to take AlexNet and ResNet as the neural network architectures and train the model to get required vectors. AlexNet contains 11 layers: 5 convolutional layers, 3 max pooling layers and 3 fully connected layers. Based on this information, the combination vectors from AlexNet and ResNet would reduce the bias and increase the accuracy [33]. Both architectures will be trained with a 1000 classes and the trained model outputs 1000 probabilities for each architecture. The array of 2000 probability will be fed into the classier algorithm , which SVM was the selected one for alternative design. The support machine algorithm is a N-dimensional space that classifies the datasets. A SVM algorithm takes the features vectors as an input and predicts the probability of the class as an output [34].

Another alternative design for classification was to use matlab to predict the class of models. In matlab all the datas needs to be separated for training, testing and validation. The logistic regression application into matlab, takes all datas. By using convolution neural networks, the features will be extracted and the logistic regression , with k-fold cross validation output the probability of the class and predict it based on the highest probability [35].

8.2 Application User Interface Design

The application was initially connected to a local server. The local server was established by using XAMPP. XAMPP stands for cross-platform, Apache, MariaDB, PHP and Perl. It is an open-source web server platform that utilizes a host computer to establish a local web server [32]. XAMPP provides useful tools to be utilized including Apache, MariaDB and PHP. Apache HTTP Web Service is an open-source web server software which provides communication of information and the remote server over the internet [32]. MariaDB is a relational database (stores information in table) where stored procedures, functions, queries and triggers must be written in procedural SQL [32]. MariaDB holds the user login information, images, and previous analysis results. The communication between the frontend UI and backend stored information occurs through PHP, a server-side scripting language [32].

Another possible design for the application's backend involves continuing to use the XAMPP and its services. They will be used as an online server with databases that hold the user's information, history, and saved (logged and analyzed) photos. These entries are held by MariaDB, a relational database (stores and provides access to data points related to one another presented in tables). This option will require another database to hold the users' images. Another option for implementing the frontend will be using Android Studio's native environment and widgets without the addition of Flutter.

9. Conclusions and Future Work

In this thesis, we worked on developing a mobile phone application that is capable of detecting three different skin lesions by implementing an ensemble model. As was discussed, this work is split into two distinct sections, one dealing with the classification model and the other dealing with developing a mobile application to allow user interaction with the model. The aim has been to develop an application that allows users to upload an image of their skin lesions to be sent to a pre-trained classification model to predict the type of skin lesion present in the image. The model trained was able to achieve 80.5% prediction accuracy. Thus, users of this application can get a general diagnosis in case of absence of medical help.

Many different adaptations and tests have been left for the future work due to lack of time and resources regarding available open source datasets. Future work concerns training the model with larger datasets of the concerned should be considered for achieving better accuracy. Additionally, further investigation should be conducted to determine the effect of skin color on the model performance. Moreover, there are some ideas that we would have liked to try during the development of the project. These ideas include:

1. Increase the number of skin lesions predicted by our model
2. Expand the platform to accept feedback from certified clinicians. This will allow the model to be continuously trained with new data. Thus, improving accuracy over time
3. Integrate all clinics map that can show the location of available healthcare centers around the user to facilitate reaching out for medical assistance in case of emergency
4. Improve the user interface according to Human-Computer Interaction principles to provide the user with an ultimate user experience

10. References

[1] "Survival statistics for melanoma skin cancer", *www.cancer.ca*. [Online]. Available: https://www.cancer.ca/en/cancer-information/cancer-type/skin-melanoma.

[2] B. Baumann et al., "Management of primary skin cancer during a pandemic: Multidisciplinary recommendations", *Cancer*, vol. 126, no. 17, pp. 3900-3906, 2020. Available: 10.1002/cncr.32969.

[3] S. Perkins and A. Barrell, "Skin lesions: Pictures, treatments, and causes", *medicalnewstoday.com*, 2020. [Online]. Available: https://www.medicalnewstoday.com/articles/skin-lesions.

[4] "Benign skin lesions", Amboss.com. [Online]. Available: https://www.amboss.com/us/knowledge/Benign_skin_lesions.

[5] U. Wollina, "Seborrheic Keratoses – The Most Common Benign Skin Tumor of Humans. Clinical presentation and an update on pathogenesis and treatment options", *Open Access Macedonian Journal of Medical Sciences*, vol. 6, no. 11, pp. 2270-2275, 2018. Available: 10.3889/oamjms.2018.460.

[6] *I0.wp.com*. [Online]. Available: https://i0.wp.com/cdn-prod.medicalnewstoday.com/content/images/articles/320/320742/seborrheic-keratosis.jpg?w=1155&h=1541.

[7] "Precancerous skin lesions", Amboss.com. [Online]. Available:https://www.amboss.com/us/knowledge/Precancerous_skin_lesions

[8] A. Viana, B. Gontijo and F. Bittencourt, "Giant congenital melanocytic nevus", *Anais Brasileiros de Dermatologia*, vol. 88, no. 6, pp. 863-878, 2013. Available: 10.1590/abd1806-4841.20132233.

[9] "Pigmented lesions," *Neonatal Dermatology - Pigmented Lesions*, 29-Nov-2011. [Online]. Available: http://www.adhb.govt.nz/newborn/TeachingResources/Dermatology/PigmentedLesions.htm.

[10] L. Rose, "Recognizing neoplastic skin lesions: a photo guide", *Am Fam Physician*, vol. 58, no. 4, pp. 873-878, 1998. Available: PMID: 9767724.

[11] J. B. Heistein, "Malignant Melanoma,"20-Nov-2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK470409/

[12] "Melanoma", *wikimedia.org*. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/6/6c/Melanoma.jpg/1200px-Melanoma.jpg.

[13] Government of Canada, "Skin Cancer," 4 September 2018. [Online]. Available: https://www.canada.ca/en/public-health/services/sun-safety/skin-cancer.html.

[14] "Keras documentation: About Keras", Keras.io. [Online]. Available: https://keras.io/about/.

[15] "Keras documentation: Image data preprocessing", Keras.io. [Online]. Available: https://keras.io/api/preprocessing/image/#imagedatagenerator-class.

[16] J. Brownlee, "Ensemble Learning Methods for Deep Learning Neural Networks", *Machine Learning Mastery*, 2018. [Online]. Available: https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/.

[17] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning", *Journal of Machine Learning Research*, vol. 27, no. 11, pp. 17-37, 2011.

[18] D. Ataloglou, A. Dimou, D. Zarpalas and P. Daras, "Fast and Precise Hippocampus Segmentation Through Deep Convolutional Neural Network Ensembles and Transfer Learning", *Neuroinformatics*, vol. 17, no. 4, pp. 563-582, 2019. Available: 10.1007/s12021-019-09417-y.

[19] B. Harangi, "Skin lesion classification with ensembles of deep convolutional neural networks", *Journal of Biomedical Informatics*, vol. 86, pp. 25-32, 2018.

[20] D. Nussey, J. Kulpa, J. Cardinell and T. Oats, “Dermatological Tool for Automatically Identifying skin cancer”, *BME 700 Biomedical Engineering Capstone Design Report*, 2019.

[21] A. Marka, J. Carter, E. Toto and S. Hassanpour, "Automated detection of nonmelanoma skin cancer using digital images: a systematic review", *BMC Medical Imaging*, vol. 19, no. 1, 2019. Available: 10.1186/s12880-019-0307-7.

[22] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *International Conference on Learning Representations*, no. 6, 2015.

[23] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", 2015.

[24] C. Szegedy, S. Ioffe and V. Vanhoucke, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning", 2016.

[25] Han, Seung & Park, Gyeong & Lim, Woohyung & Kim, Myoung & Na, Jung-Im & Park, Ilwoo & Chang, Sung. (2018). Deep neural networks show an equivalent and often superior performance to dermatologists in onychomycosis diagnosis: Automatic construction of onychomycosis datasets by region-based convolutional deep neural networks. PLOS ONE. 13. e0191493. 10.1371/journal.pone.0191493.

[26] U. Nazir, N. Khurshid, M. A. Bhimra, and M. Taj, “Tiny-Inception-ResNet-v2: Using Deep Learning for Eliminating Bonded Labors of Brick Kilns in South Asia,” 2019.

[27] J. Brownlee, "Logistic Regression for Machine Learning ", *Machine Learning Mastery*, 2016. [Online]. Available: https://machinelearningmastery.com/logistic-regression-for-machine-learning/

[28] "Documentation for app developers", *Android Developers*. [Online]. Available: https://developer.android.com/docs.

[29] "Add Firebase to your Android project", *Firebase*, 2020. [Online]. Available: https://firebase.google.com/docs/android/setup.

[30] "Dart documentation", *Dart.dev*, 2020. [Online]. Available: https://dart.dev/guides.

[31] "Add Firebase to your Android project", *Firebase*, 2020. [Online]. Available: https://firebase.google.com/docs/android/setup.

[32] "XAMPP Documentation", *Apache Friends*, 2020. [Online]. Available: https://www.apachefriends.org/docs/.

[33] U.-O. Dorj, K.-K. Lee, J.-Y. Choi, and M. Lee, "The skin cancer classification using deep convolutional neural networks," Multimed. Tools Appl., vol. 77, no. 8, pp. 9909–9924, Apr. 2021.

[34] R. Gandhi, "Support Vector Machine - Introduction to Machine Learning Algorithms," *Medium*, 05-Jul-2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

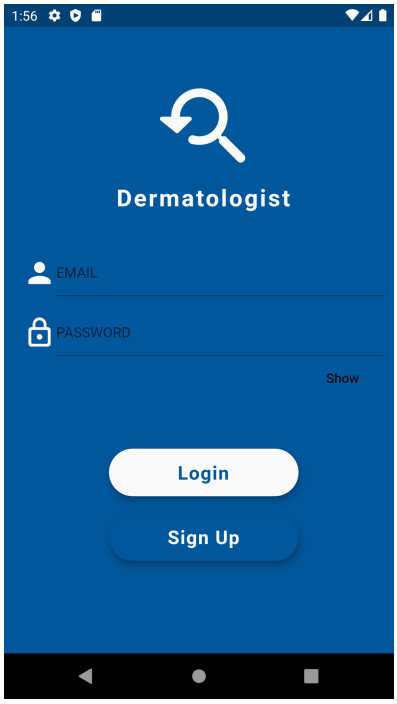
[35] "What Is a Neural Network?," *What Is a Neural Network? - MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/discovery/neural-network.html.

[36] "sklearn.metrics.precision_recall_fscore_support," *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html.

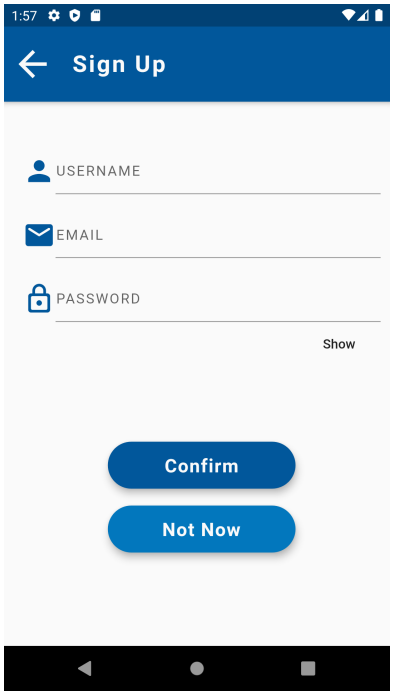
12. Appendices

12.1 Application Screens

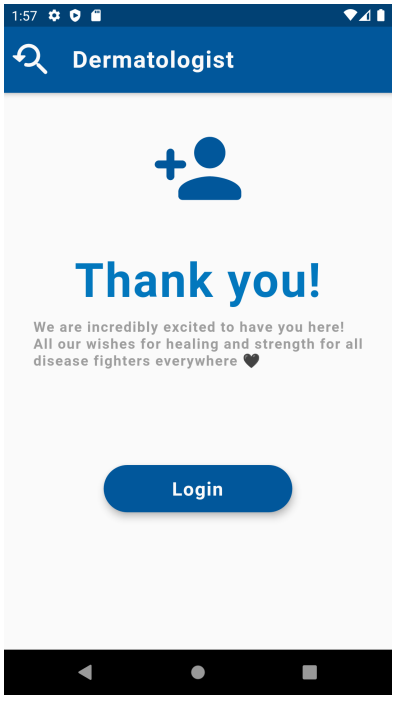
Start/Splash Screen



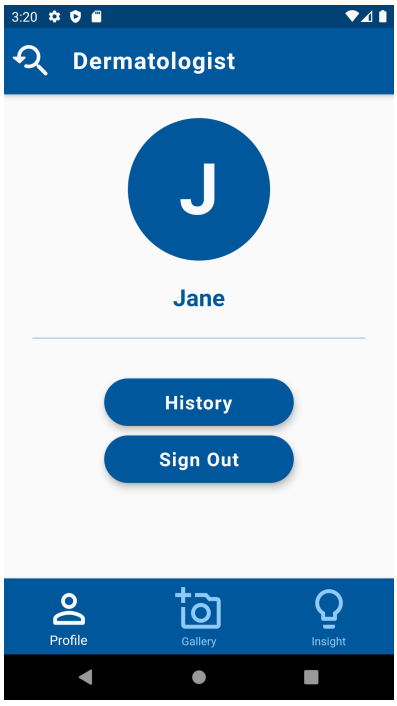
Login and Sign Up Screen



Create Account Screen

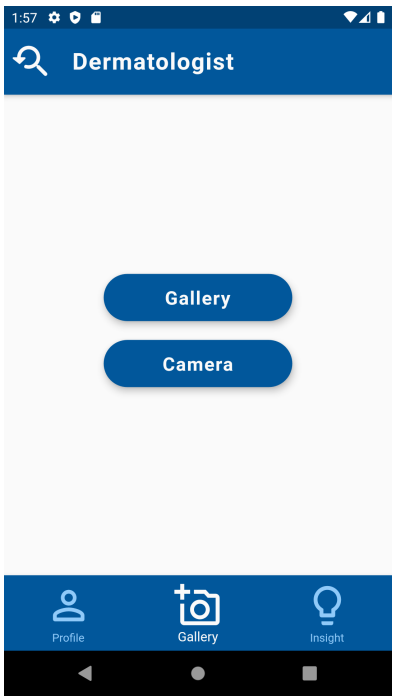


Confirmation Screen

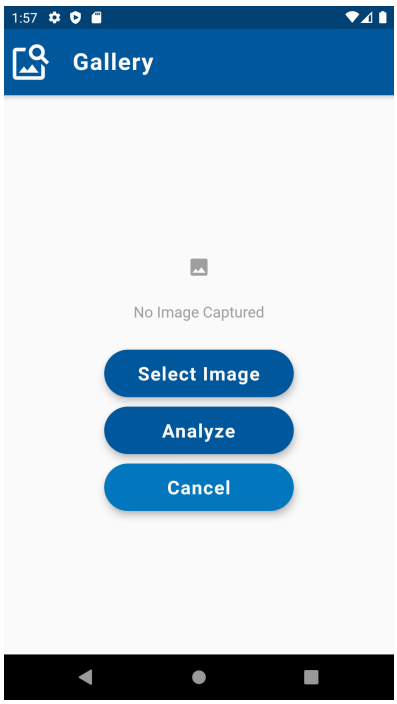


Profile Tab Screen

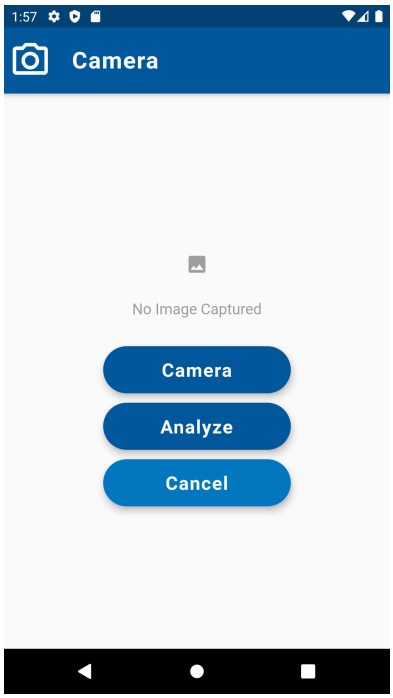
History Screen



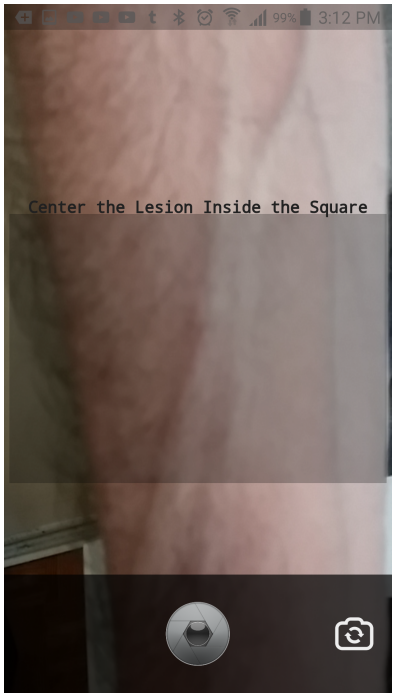
Gallery Tab Screen



Select Image from Gallery



Capture image using Camera



Camera Page

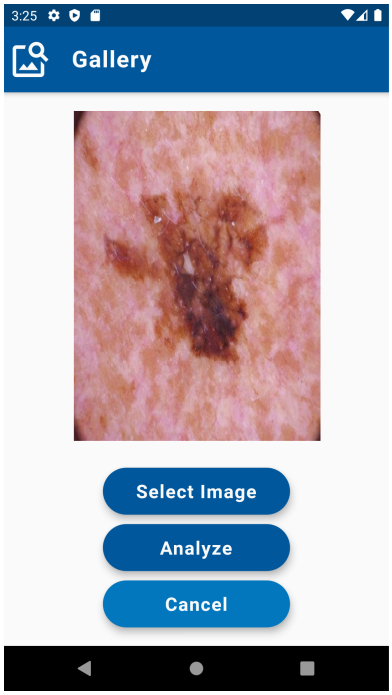
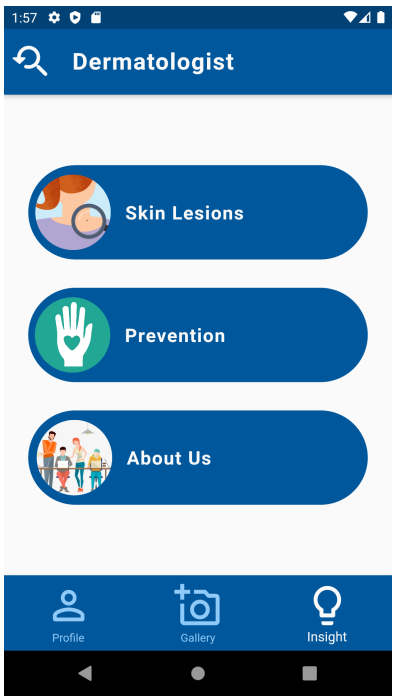
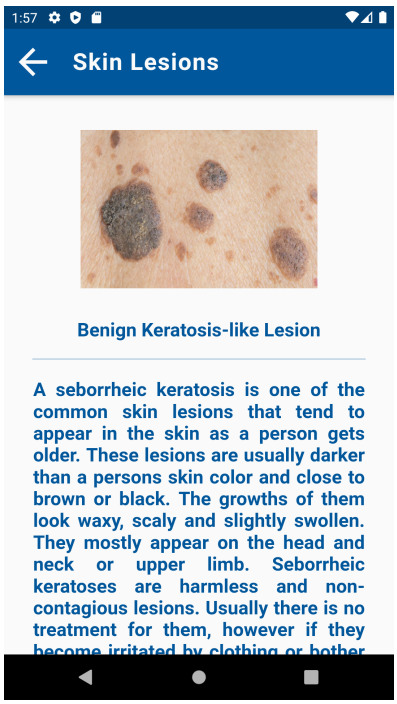


Image Selected from Gallery

Results Screen

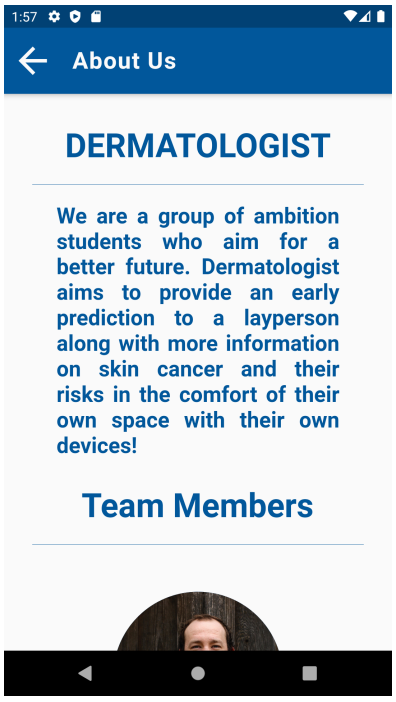


Insight Tab Screen



Skin Lesions Screen

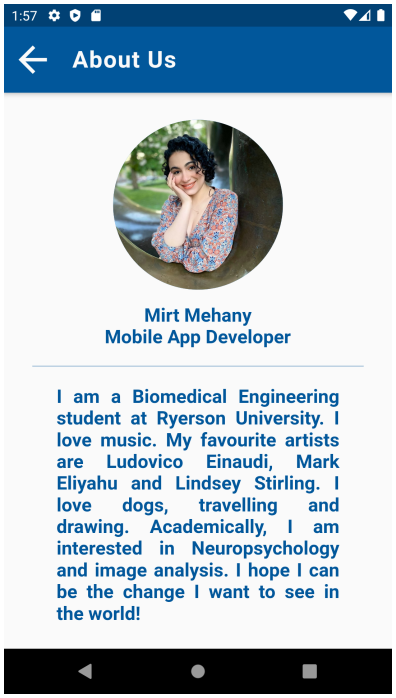
Prevention Screen



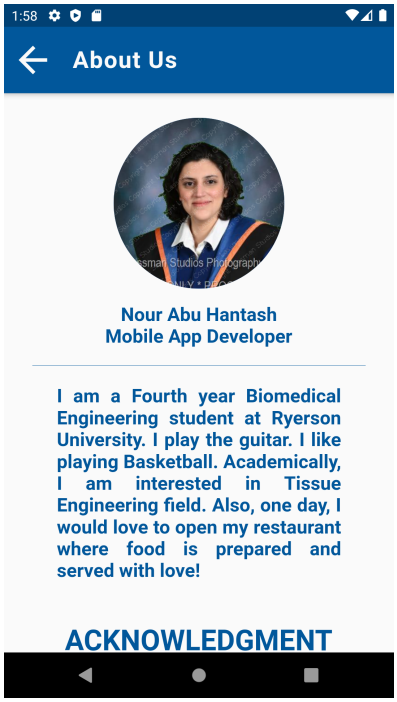
About Us Screen

About Us Screen Cont'd

About Us Screen Cont'd



About Us Screen Cont'd



About Us Screen Cont'd

About Us Screen Cont'd

12.2 Classification Source Code

12.2.1 Deep Learning Training

```
# memory footprint support libraries/code
!ln -sf /opt/bin/nvidia-smi /usr/bin/nvidia-smi
!pip install gputil
!pip install psutil
!pip install humanize
import psutil
import humanize
import os
import GPUtil as GPU
# retrieve GPU type
GPUs = GPU.getGPUs()
gpu = GPUs[0]
# check allocation of GPU resources
def printm():
 process = psutil.Process(os.getpid())
 print("Gen RAM Free: " + humanize.naturalsize( psutil.virtual_memory().available ), " | Proc size: " + humanize.naturalsize(
process.memory_info().rss))
 print("GPU RAM Free: {0:.0f}MB | Used: {1:.0f}MB | Util {2:3.0f}% | Total {3:.0f}MB".format(gpu.memoryFree, gpu.memoryUsed,
gpu.memoryUtil*100, gpu.memoryTotal))
printm()

# Import Modules and Mount Drive

# Mounting Google Drive
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Mount google drive
from google.colab import drive
drive.mount('/content/gdrive', force_remount = True)

# Authenticate and create the PyDrive client
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Import Necessary Libraries and personal scripts (Utils.py & Models.py)
# Tensorflow, matplotlib, numpy etc
%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/Scripts/
from Utils import *
from models import *
import os
import cv2
import numpy as np
from matplotlib.pyplot import imsave, imread
import matplotlib.pyplot as plt
import math as mth
import pandas as pd
import tensorflow as tf
from tensorflow.keras.applications import ResNet50, VGG16, VGG19, InceptionV3, InceptionResNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, CSVLogger
import random


%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/
print(len(os.listdir('Data/deep/bkl/')))
print(len(os.listdir('Data/deep/mel/')))
print(len(os.listdir('Data/deep/nv/')))
print(len(os.listdir('Data/deep/test/')))
print(len(os.listdir('Data/deep/train/')))
print(len(os.listdir('Data/log/Features/bkl/')))
print(len(os.listdir('Data/log/Features/mel/')))
print(len(os.listdir('Data/log/Features/nv')))

train = np.load('Data/log/Features/train.npy')
print(len(train))
test = np.load('Data/log/Features/test.npy')
print(len(test))


# Divide Data
```

```
# Change to Directory where data is stored
%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/

# Read in the metadata information about the images
meta = pd.read_csv('HAM10000_metadata.csv')
# Create a dictionary to store images
data = dict()

# Loop through and fill in data variable with key=filename and value=image array
for filename in os.listdir('HAM10000_images_part_1/') :
  data[filename[:-4]] = np.array(imread('HAM10000_images_part_1/' + filename))

for filename in os.listdir('HAM10000_images_part_2/') :
  data[filename[:-4]] = np.array(imread('HAM10000_images_part_2/' + filename))

# Specify path to save labelled images in
melPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/mel/'
bklPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/bkl/'
akiecPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/akiec/'
bccPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/bcc/'
dfPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/df/'
nvPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/nv/'
vascPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/vasc/'

for i in range(len(meta)) :
  if meta.image_id[i] in data :
    if meta.dx[i] == 'akiec' :
      # Save in akiec Folder
      name = akiecPath + meta.image_id[i] + '.jpg'
      imsave(name, data[meta.image_id[i]])
    if meta.dx[i] == 'mel' :
      # Save in Mel Folder
      name = melPath + meta.image_id[i] + '.jpg'
      imsave(name, data[meta.image_id[i]])
    if meta.dx[i] == 'bkl' :
      # Save in BKL folder
      name = bklPath + meta.image_id[i] + '.jpg'
      imsave(name, data[meta.image_id[i]])
    if meta.dx[i] == 'bcc' :
      # Save in bcc Folder
      name = bccPath + meta.image_id[i] + '.jpg'
      imsave(name, data[meta.image_id[i]])
    if meta.dx[i] == 'df' :
      # Save in df Folder
      name = dfPath + meta.image_id[i] + '.jpg'
      imsave(name, data[meta.image_id[i]])
    if meta.dx[i] == 'nv' :
      # Save in nv Folder
      name = nvPath + meta.image_id[i] + '.jpg'
      imsave(name, data[meta.image_id[i]])
    if meta.dx[i] == 'vasc' :
      # Save in vasc Folder
      name = vascPath + meta.image_id[i] + '.jpg'
      imsave(name, data[meta.image_id[i]])

# Load Preprocessed images
melPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/mel/'
bklPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/bkl/'
nvPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/nv/'

mel = dict()
bkl = dict()
nv = dict()

i = 0
for filename in os.listdir(melPath) :
  mel[filename] = imread(melPath + filename)
  print(i)
  i = i+1

for filename in os.listdir(bklPath) :
  bkl[filename] = imread(bklPath + filename)
  print(i)
  i = i+1

for filename in os.listdir(nvPath) :
  nv[filename] = imread(nvPath + filename)
  print(i)
  i = i+1
```

```
# Divide data (Images from 3 classes) into training(60%), Validation(20%), Testing(20%)
train_percent = 0.8
val_percent = 0.25 * train_percent
test_percent = 0.2

# Create index variables
indexMel = createIndex(mel, train_percent, val_percent, test_percent)
indexBkl = createIndex(bkl, train_percent, val_percent, test_percent)
indexNv = createIndex(nv, train_percent, val_percent, test_percent)

# Make sure its randomized to avoid any bias
# Save divided data into directories
dataPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/'
saveImagesToFolders(mel, indexMel, 'mel/', dataPath)
saveImagesToFolders(bkl, indexBkl, 'bkl/', dataPath)
saveImagesToFolders(nv, indexMel, 'nv/', dataPath)


print('akiec', len(os.listdir('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/DummyData/akiec')))
print('mel', len(os.listdir('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/DummyData/mel')))
print('bcc', len(os.listdir('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/DummyData/bcc')))
print('bkl', len(os.listdir('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/DummyData/bkl')))
print('df', len(os.listdir('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/DummyData/df')))
print('nv', len(os.listdir('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/DummyData/nv')))
print('vasc', len(os.listdir('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/DummyData/vasc')))

# New Data Division

melPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/mel/'
bklPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/bkl/'
nvPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/DummyData/nv/'

mel = dict()
bkl = dict()
nv = dict()

for filename in os.listdir(melPath) :
  mel[filename] = imread(melPath + filename)

for filename in os.listdir(bklPath) :
  bkl[filename] = imread(bklPath + filename)

for filename in os.listdir(nvPath) :
  nv[filename] = imread(nvPath + filename)


len(bkl)//2

# DeepLearning folders
melDeepPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/deep/mel/'
bklDeepPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/deep/bkl/'
nvDeepPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/deep/nv/'
# Logistic Regression Folders
melLogPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/log/mel/'
bklLogPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/log/bkl/'
nvLogPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/log/nv/'
# State Dictionarys
bklDeep = dict()
melDeep = dict()
nvDeep = dict()

i = 0
for key, value in bkl.items() :
  if i <= len(bkl)//2 :
    imsave(bklDeepPath + key, value)
    bklDeep[key] = value
  else :
    imsave(bklLogPath + key, value)
  i += 1
print('Done BKL')

i = 0
for key, value in mel.items() :
  if i <= len(mel)//2 :
    imsave(melDeepPath + key, value)
    melDeep[key] = value
  else :
    imsave(melLogPath + key, value)
  i += 1
print('Done MEL')
```

```
i = 0
for key, value in  nv.items() :
  if i <= len(nv)//2 :
    imsave(nvDeepPath + key, value)
    nvDeep[key] = value
  else :
    imsave(nvLogPath + key, value)
  i += 1
print('Done NV')

# Save into train val and test
# Divide data (Images from 3 classes) into training(60%), Validation(20%), Testing(20%)
train_percent = 0.8
val_percent = 0.25 * train_percent
test_percent = 0.2

# Create index variables
indexMel = createIndex(melDeep, train_percent, val_percent, test_percent)
indexBkl = createIndex(bklDeep, train_percent, val_percent, test_percent)
indexNv = createIndex(nvDeep, train_percent, val_percent, test_percent)
print('Indexes Created')

# Make sure its randomized to avoid any bias
# Save divided data into directories
dataPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/deep/'
saveImagesToFolders(melDeep, indexMel, 'mel/', dataPath)
print('Saved MEL')
saveImagesToFolders(bklDeep, indexBkl, 'bkl/', dataPath)
print('Saved BKL')
saveImagesToFolders(nvDeep, indexMel, 'nv/', dataPath)
print('Saved NV')

# Train Models

# Select Architecture
# Choices = 'ResNet50', 'VGG16', 'InceptionResNetV2'
# For transfer learning have load_weights = 'imagenet', for non transfer learning load_weights = None
model_choice = 'ResNet50'
load_weights = 'imagenet'
model = get_model(model_choice, load_weights)
model.summary()

# Set argumemts for the training and validation generator
# Probably add vertical and horizontal flip
train_args = dict(
              rotation_range=0.2,
              width_shift_range=0.05,
              height_shift_range=0.05,
              shear_range=0.05,
              zoom_range=0.05,
              fill_mode='nearest'
)
val_args = dict()

# Set paths for training and validation data
trainPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/deep/train'
valPath = '/content/gdrive/Shared drives/CAPSTONE/DeepLearning/Data/deep/val'

# Create Data Generators w/ data augmentations (flip, shift, translations, shear, rescale etc)
train_gen = ImageDataGenerator(train_args)
val_gen = ImageDataGenerator(val_args)

# Set Hyperparameters (epochs, learning rate, batchsize, steps, validation steps, cost function)
epochs = 100
batchsize = 16
steps = mth.ceil(1061/batchsize)
val_steps = mth.ceil(266/batchsize)
learningrate = 0.000001
model_name = 'e' + str(epochs) + '_BS' + str(batchsize) + '_lr' + str(learningrate) + '/'
ModelPath = '/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/Models/TransferLearning/' + model_choice + '/' + model_name

# Specify where the input images are coming from
train_gen = train_gen.flow_from_directory(trainPath, target_size=(224,224), batch_size=batchsize)
val_gen = val_gen.flow_from_directory(valPath, target_size=(224,224), batch_size=batchsize)


if os.path.isdir(ModelPath) == 0 :
  os.mkdir(ModelPath)
  print('Creating Directory')
```

```
# Set callbacks using tensorflow functions (csvlogger, early stopping, model checkpoint)
model_checkpoint = ModelCheckpoint(ModelPath + 'model.hdf5', monitor='val_loss', save_best_only=True)
early_stop = EarlyStopping(monitor = 'val_loss', patience = 10)
csv_logger = CSVLogger(ModelPath + 'model.csv')

# Create Directory Label for specific models
# Compile Model Using Hyperparamters [model.compile()]
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),      optimizer=tf.keras.optimizers.Adam(lr=learningrate),
metrics=['accuracy'])

# Train The model [model.fit()] - This trains the model, logs the data by epoch and saves the best version

model_history = model.fit_generator(train_gen,
                                    steps_per_epoch = steps,
                                    validation_data = val_gen,
                                    validation_steps = val_steps,
                                    epochs=epochs,
                                      callbacks = [model_checkpoint, early_stop, csv_logger])

# Once finished training...
# Display loss curves to double check it looks good

# Testing



# TestDataPaths
bklPath = '/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/Data/deep/test/bkl/'
melPath = '/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/Data/deep/test/mel/'
nvPath = '/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/Data/deep/test/nv/'

# Load Model
modelPath                                                                                                   =
'/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/newModels/TransferLearning/ResNet50/e100_BS16_lr1e-05/newmodel.hdf5'
model_choice = 'ResNet50'
shape = (224, 224, 3)
model = get_model(model_choice=model_choice)
model.load_weights(modelPath)

# Make Predictions
bklPredict = getPredictions(model, shape, bklPath)
melPredict = getPredictions(model, shape, melPath)
nvPredict = getPredictions(model, shape, nvPath)

# Create Actuals
bklActual = ['bkl'] * len(bklPredict)
melActual = ['mel'] * len(melPredict)
nvActual = ['nv'] * len(nvPredict)

# Concatenate Predictions and Actuals
Actual = bklActual + melActual + nvActual
Predicted = bklPredict + melPredict + nvPredict


# Evaluate Models


# Choose Model
dataPath                    =                      ModelPath                    +                    '/model.csv'
#'/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/Models/TransferLearning/ResNet50/e150_BS16_lr5e-07/model.csv'
data = pd.read_csv(dataPath)
# Plot Loss
plotLoss(data)
plotAccuracy(data)

# Display max/min
print(max(data['accuracy']))
print(max(data['val_accuracy']))
print(min(data['loss']))
print(min(data['val_loss']))



**Confusion matrix**

# Confusion Matrices
# using panda for confusion matrix
act_y = pd.Series(Actual, name = 'Actual')
pred_y = pd.Series(Predicted, name = 'Predicted')
df_conf = pd.crosstab(act_y, pred_y)
print(df_conf)
```

```
print()

#to find some of rows ( not sure if it is require for us)
#adding true = margin
df_conf1 = pd.crosstab(act_y, pred_y, rownames=['Actual'], colnames=['Predicted'], margins=True)
print(df_conf1)
print()

# if you need to normalize it
df_conf_norm = df_conf / df_conf.sum(axis=1)
print(df_conf_norm)
print()

# now for plotting
def plot_confusion_matrix(df_conf, title='Confusion matrix', cmap=plt.cm.gray_r):
    plt.matshow(df_conf, cmap=cmap) # imshow
    #plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(df_conf.columns))
    plt.xticks(tick_marks, df_conf.columns, rotation=45)
    plt.yticks(tick_marks, df_conf.index)
    #plt.tight_layout()
    plt.ylabel(df_conf.index.name)
    plt.xlabel(df_conf.columns.name)

plot_confusion_matrix(df_conf)

printConfMatrix()
```

12.2.2 Logistic Regression Training

```
# Import Modules and Mount Drive

#CAPSTONE
# Mounting Google Drive
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Mount google drive
from google.colab import drive
drive.mount('/content/gdrive', force_remount = True)

# Authenticate and create the PyDrive client
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Import Necessary Libraries and Personal scripts
%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/Scripts/
from Utils import *
from models import *
import tensorflow as tf
import numpy as np
import os
from sklearn import metrics
from sklearn.linear_model import LogisticRegressionCV
from sklearn import svm
import pickle

# Feature Extraction


%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/
# Load Feature Extraction models
# Model A --> 1000 Class feature extractor
modelA = tf.keras.applications.ResNet50()

# Model B --> Transfer Learning Model
transferPath = 'newModels/TransferLearning/ResNet50/e100_BS16_lr1e-07/newmodel.hdf5'
modelB = get_model('ResNet50')
modelB.load_weights(transferPath)

# Model C --> Normal Learning Model
normPath = 'newModels/nonTransferLearning/ResNet50/e100_BS16_lr1e-05/newmodel.hdf5'
modelC = get_model('ResNet50')
```

```
modelC.load_weights(normPath)

# Load In Data
bklImagePath = 'Data/log/Images/bkl/'
melImagePath = 'Data/log/Images/mel/'
nvImagePath = 'Data/log/Images/nv/'

# Declare dictionaries for each type of image
bklImage = dict()
melImage = dict()
nvImage = dict()

# Load in images
for filename in os.listdir(bklImagePath) :
  bklImage[filename] = imread(bklImagePath + filename)

for filename in os.listdir(melImagePath) :
  melImage[filename] = imread(melImagePath + filename)

for filename in os.listdir(nvImagePath) :
  nvImage[filename] = imread(nvImagePath + filename)



# Declare Feature Paths
bklFeaturePath = 'Data/log/Features/bkl/'
melFeaturePath = 'Data/log/Features/mel/'
nvFeaturePath = 'Data/log/Features/nv/'

# Make BKL class Predictions
for key, value in bklImage.items() :

  # Preprocessing of the images
  img = np.array(value)
  img = cv2.resize(img, (224,224), interpolation=cv2.INTER_LINEAR)
  img = img.reshape(224,224,3)
  img = np.expand_dims(img, axis=0)

  # Feature extraction
  predA = np.array(modelA.predict(img)) # 1000 class feature extraction
  predB = np.array(modelB.predict(img)) # Transfer Learning 3 Class Prediction Feature extraction
  predC = np.array(modelC.predict(img)) # Normal Learning 3 class prediction Feature extraction
  label = np.array([0]) # Declare a numerical value that corresponds to the class

  # Concatenate 1000+3+3+1=1007 features into one array
  featureArray = np.concatenate([predA[0], predB[0], predC[0], label])
  # Create the bkl feature name
  featureName = bklFeaturePath + key[:-4] + '.npy'

  # Save the feature vectors to google drive
  np.save(featureName, featureArray)

print('Done BKL')

for key, value in melImage.items() :
  # Preprocessing of the images
  img = np.array(value)
  img = cv2.resize(img, (224,224), interpolation=cv2.INTER_LINEAR)
  img = img.reshape(224,224,3)
  img = np.expand_dims(img, axis=0)

  # Feature Extraction
  predA = np.array(modelA.predict(img)) # 1000 class feature extraction
  predB = np.array(modelB.predict(img)) # Transfer learning 3 class prediction feature extraction
  predC = np.array(modelC.predict(img)) # Normal learning 3 class prediction feature extraction
  label = np.array([1]) # Declare a numerical value that corresponds to the class

  # Concatenate 1000+3+3+1=1007 features into one array
  featureArray = np.concatenate([predA[0], predB[0], predC[0], label])
  # Create the mel feature name
  featureName = melFeaturePath + key[:-4] + '.npy'

  # Save the feature vectors to google drive
  np.save(featureName, featureArray)

print('Done Mel')

for key, value in nvImage.items() :
  # Pre processing of images
  img = np.array(value)
  img = cv2.resize(img, (224,224), interpolation=cv2.INTER_LINEAR)
```

```
  img = img.reshape(224,224,3)
  img = np.expand_dims(img, axis=0)

  # Feature Extraction
  predA = np.array(modelA.predict(img)) # 1000 Class feature extraction
  predB = np.array(modelB.predict(img)) # Transfer learning 3 class prediction feature extraction
  predC = np.array(modelC.predict(img)) # Normal learning 3 class prediction feature extraction
  label = np.array([2]) # Declare a numerical value that corresponds to the class

  # Concatenate 1000+3+3+1=1007 features into one array
  featureArray = np.concatenate([predA[0], predB[0], predC[0], label])
  # Create the nv feature name
  featureName = nvFeaturePath + key[:-4] + '.npy'

  # Save feature vectors to google drive
  np.save(featureName, featureArray)

print('Done NV')

%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/
bklImagePath = 'Data/log/Images/bkl'
melImagePath = 'Data/log/Images/mel'
nvImagePath = 'Data/log/Images/nv'
bklFeaturePath = 'Data/log/Features/bkl'
melFeaturePath = 'Data/log/Features/mel'
nvFeaturePath = 'Data/log/Features/nv'

print(len(os.listdir(melImagePath)))
print(len(os.listdir(melFeaturePath)))
print(len(os.listdir(bklImagePath)))
print(len(os.listdir(bklFeaturePath)))
print(len(os.listdir(nvImagePath)))
print(len(os.listdir(nvFeaturePath)))

# Data Division

%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/
# Load In all features from the feature folder
bklFeaturePath = 'Data/log/Features/bkl/'
melFeaturePath = 'Data/log/Features/mel/'
nvFeaturePath = 'Data/log/Features/nv/'

# Declare empty vectors with size rows = # of instances and columns = # of features
bklSize = (549, 1007)
melSize = (556, 1007)
nvSize = (559, 1007)
bklFeature = np.zeros(bklSize)
melFeature = np.zeros(melSize)
nvFeature = np.zeros(nvSize)

# Load in bkl features
i = 0
for filename in os.listdir(bklFeaturePath) :
  bklFeature[i,:] = np.load(bklFeaturePath + filename)
  i += 1

# Load in nv features (only one out of every 6 to reduce the number of instances
# to a similar size as other classes)
i = 0
j = 0
for filename in os.listdir(nvFeaturePath) :
  if i <= 3352//6 :
    nvFeature[j,:] = np.load(nvFeaturePath + filename)
    j += 1

  i += 1

# Load in mel features
i = 0
for filename in os.listdir(melFeaturePath) :
  melFeature[i,:] = np.load(melFeaturePath + filename)
  i += 1

# Divide the data into 80% for training and 20% testing
# Randomly get indices for the 80% training
bklIndex = indexCreation(bklFeature, 0.8)
melIndex = indexCreation(melFeature, 0.8)
nvIndex = indexCreation(nvFeature, 0.8)

# Declare an empty array for bkl test and training data
bklTestFeature = np.zeros((110, 1007))
```

```
bklTrainFeature = np.zeros((439, 1007))

# Declare counters
trainCount = 0
testCount = 0

# Divide the data
for i in range(len(bklFeature)) :
  if i in bklIndex :
    # Place in the bkl train variable
    bklTrainFeature[trainCount, :] = bklFeature[i,:]
    trainCount += 1

  else :
    # Place in bkl test variable
    bklTestFeature[testCount, :] = bklFeature[i,:]
    testCount += 1

# Declare an empty array for mel test and training
melTrainFeature = np.zeros((int(0.8*556), 1007))
melTestFeature = np.zeros((556-int(0.8*556), 1007))

# Declare counters
trainCount = 0
testCount = 0

# Divide the data
for i in range(len(melFeature)) :
  if i in melIndex :
    # Place in the mel train variable
    melTrainFeature[trainCount, :] = melFeature[i,:]
    trainCount += 1

  else :
    # Place in mel test variable
    melTestFeature[testCount, :] = melFeature[i,:]
    testCount += 1

# Declare empty arrays for mel train and test
nvTrainFeature = np.zeros((int(0.8*559), 1007))
nvTestFeature = np.zeros((559-int(0.8*559), 1007))

# Declare counters
trainCount = 0
testCount = 0

# Divide the data
for i in range(len(nvFeature)) :
  if i in nvIndex :
    # Place in nv train variable
    nvTrainFeature[trainCount, :] = nvFeature[i,:]
    trainCount += 1

  else :
    # Place in nv test variable
    nvTestFeature[testCount, :] = nvFeature[i,:]
    testCount += 1

# Stack the 3 classes on top of eachother
train = np.vstack([melTrainFeature, bklTrainFeature, nvTrainFeature])
test = np.vstack([melTestFeature, bklTestFeature, nvTestFeature])

# Save to google drive
np.save('Data/log/Features/test.npy', test)
np.save('Data/log/Features/train.npy', train)

# Logistic Regression Training

%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/
# Load in training and testing data
dataTrain = np.load('Data/log/Features/train.npy')
dataTest = np.load('Data/log/Features/test.npy')

# Divide into X and y for training and testing
# X is the features and y is the labels at the end
X_train = dataTrain[:, 0:1006]
y_train = dataTrain[:, 1006]
X_test = dataTest[:, 0:1006]
y_test = dataTest[:, 1006]

# Declare parameters
```

```
num_Folds = 7
num_iterations = 10000

# Declare model
model = LogisticRegressionCV(cv = num_Folds, verbose = 2, max_iter = num_iterations)

#model = svm.SVC(decision_function_shape='ovo')
# Fit/train model
model.fit(X_train,y_train)

# Declare model path
modelPath = 'Models/LogRegModel/' + str(num_Folds) + '_Fold_' + str(num_iterations) + '_iter.pkl'
# Save model
pickle.dump(model, open(modelPath, 'wb'))

print(model.score(X_test, y_test))

# Testing/Evaluation

%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/
# Load in testing data
dataTest = np.load('Data/log/Features/test.npy')
X_test = dataTest[:, 0:1006]
y_test = dataTest[:, 1006]

# Load in model
model = pickle.load(open('Models/LogRegModel/7_Fold_10000_iter.pkl', 'rb'))

# Test Model
y_pred = model.predict(X_test)

print('Score:', model.score(X_test, y_test))

# Print the confusion matrix
print(metrics.confusion_matrix(y_test, y_pred))

# Print the precision and recall, among other metrics
print(metrics.classification_report(y_test, y_pred, digits=3))

## Create TF Layer

def indexMax(input) :
  return np.argmax(np.array(input))

#############
# create a TF model with the same architecture
tf_model = tf.keras.models.Sequential()
tf_model.add(tf.keras.Input(shape=(1006)))
tf_model.add(tf.keras.layers.Dense(3))


# assign the parameters from sklearn to the TF model
tf_model.layers[0].weights[0].assign(model.coef_.transpose())
tf_model.layers[0].bias.assign(model.intercept_.transpose())

count = 0

for i in range(len(X_test)) :
  temp = np.expand_dims(X_test[i], axis=0)
  tflow = indexMax(tf_model(temp))
  sk = model.predict(temp)
  print('Input # %d' %i)
  print(tf_model(temp))
  print(tflow)
  print(sk)
  if tflow != sk :
    print('FLAG')
    count+=1

print ('Logistic TensorFlow Model is type:', type(tf_model))

print(count)

# verify the models do the same prediction
#assert np.all((tf_model(x) > 0)[:, 0].numpy() == model.predict(x))
#################

tf_model.save('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/TF/logRegModel.hdf5')

act_y = pd.Series(y_test, name = 'Actual')
pred_y = pd.Series(y_pred, name = 'Predicted')
```

```
df_conf = pd.crosstab(act_y, pred_y)
print(df_conf)

df_conf1 = pd.crosstab(act_y, pred_y, rownames=['Actual'], colnames=['Predicted'], margins=True)
print(df_conf1)

# if you need to normalize it
df_conf_norm = df_conf / df_conf.sum(axis=1)
print(df_conf_norm1)
print()

# now for plotting
def plot_confusion_matrix(df_conf, title='Confusion matrix', cmap=plt.cm.gray_r):
    plt.matshow(df_conf, cmap=cmap) # imshow
    #plt.title(title)
    plt.colorbar()

    tick_marks = np.arange(len(df_conf.columns))
    plt.xticks(tick_marks, df_conf.columns, rotation=45)
    plt.yticks(tick_marks, df_conf.index)

    #plt.tight_layout()
    plt.ylabel(df_conf.index.name)
    plt.xlabel(df_conf.columns.name)

plot_confusion_matrix(df_conf)

def changeLabels(data) :
  out = []
  for i in range(len(data)) :
    if data[i] == 0.0 :
      out.append('bkl')
    elif data[i] == 1.0 :
      out.append('mel')
    else :
      out.append('nv')
x = pd.Series(changeLabels(y_test))
y = pd.Series(changeLabels(y_pred))

df_conf = pd.crosstab(x, y)
print(df_conf)

# Package Models

%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/
# Load Deep Learning Models
# Model A --> 1000 Class feature extractor
modelA = tf.keras.applications.ResNet50()

# Model B --> Transfer Learning Model
transferPath = 'newModels/TransferLearning/ResNet50/e100_BS16_lr1e-07/newmodel.hdf5'
modelB = get_model('ResNet50')
modelB.load_weights(transferPath)

# Model C --> Normal Learning Model
normPath = 'newModels/nonTransferLearning/ResNet50/e100_BS16_lr1e-05/newmodel.hdf5'
modelC = get_model('ResNet50')
modelC.load_weights(normPath)

# Model D Load Logistic Regression Model
logPath = 'Models/LogRegModel/7_Fold_10000_iter.pkl'
model = pickle.load(open(logPath, 'rb'))

from tensorflow.keras import Model
from keras.layers.merge import concatenate
from keras import layers
import itertools


def define_stacked_model2(modelA, modelB, modelC):

  # Update all laters in all models to not be trainable

  for layer in modelA.layers:
    # Make not trainable
    layers.trainable = False
    # Rename to avoid unique layer name issue
    layer._name ='ensemble_A_' +layer.name

  for layer in modelB.layers:
    # Make not trainable
```

```
    layers.trainable = False
    # Rename to avoid unique layer name issue
    layer._name ='ensemble_B_' +layer.name

  for layer in modelC.layers:
    # Make not trainable
    layers.trainable = False
    # Rename to avoid unique layer name issue
    layer._name ='ensemble_C_' +layer.name

  # Define multi-headed input
  ensemble_visible = [modelA.input, modelB.input, modelC.input]
  # Define output
  ensemble_output = [modelA.output, modelB.output, modelC.output]

  # Create model
  model = Model(inputs=ensemble_visible, outputs=ensemble_output)

  # Compile
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

  return model

ensembleModel = define_stacked_model2(modelA, modelB, modelC)
#ensembleModel.summary()
ensembleModel.save('ensembleModels/deepModel.hdf5')

ensembleModel.summary()
# Import Modules and Mount Drive
```

12.2.3 Utils.py

```
# This Script will contain all useful functions so the main code is a lot more readable
# Import Modules
import random
from matplotlib.pyplot import imsave, imread
import matplotlib.pyplot as plt
import os
import cv2
import numpy as np
import pandas as pd


# Self Made Functions
def createIndex(data, train_percent, val_percent, test_percent) :
    index = dict()
    num = int(len(data)*train_percent)
    random.seed(1)
    samp = random.sample(range(len(data)), num)

    index['train'] = samp[int(val_percent*len(samp))+1:]
    index['val'] = samp[0:int(val_percent*len(samp))]
    index['test'] = []

    for i in range(len(data)) :
        if i not in index['train'] and i not in index['val'] :
            index['test'].append(i)

    return index


def saveImagesToFolders(data, index, folder, path) :
    i = 0;
    for key, value in data.items():
        # Save in train
        if i in index['train'] :
          imsave(path + 'train/' + folder + key, value)

        # Save in val
        if i in index['val'] :
          imsave(path + 'val/' + folder + key, value)

        # Save in test
        if i in index['test'] :
          imsave(path + 'test/' + folder + key, value)
        i = i+1

    print('Done Saving')


def plotLoss(data) :
```

```
    plt.plot(data['loss'], label='Training Loss')
    plt.plot(data['val_loss'], label='Validation Loss')
    plt.ylabel('Binary Cross Entropy Loss')
    plt.xlabel('Epochs')
    plt.title('Loss Curve')
    plt.legend()
    plt.show()

def plotAccuracy(data) :
    plt.plot(data['accuracy'], label='Training Accuracy')
    plt.plot(data['val_accuracy'], label='Validation Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epochs')
    plt.title('Accuracy Curve')
    plt.legend()
    plt.show()


def getTestData(path) :
    data = dict()
    bkl = []
    nv = []
    mel = []
    bklPath = path + '/bkl/'
    melPath = path + '/mel/'
    nvPath = path + '/nv/'

    for filename in os.listdir(bklPath) :
        bkl.append(np.array(imread(bklPath + filename)))

    for filename in os.listdir(melPath) :
        mel.append(np.array(imread(melPath + filename)))

    for filename in os.listdir(nvPath) :
        nv.append(np.array(imread(nvPath + filename)))

    data['bkl'] = bkl
    data['nv'] = nv
    data['mel'] = mel

    return data


def getPredictions(model, shape, path) :
    classPred = list()
    for filename in os.listdir(path) :

        img = np.array(imread(path + filename))
        img = cv2.resize(img, shape[0:2], interpolation=cv2.INTER_LINEAR)
        img = img.reshape(shape)
        img = np.expand_dims(img, axis=0)

        pred = model.predict(img)
        prob = np.max(pred)
        loc = np.where(pred==prob)
        loc = loc[1]
        classPred.append(labelPred(loc))

    print('Done Class Predictions')
    return classPred


def labelPred(loc) :
    label = ''
    if loc == 0 :
        label = 'bkl'
    elif loc == 1 :
        label = 'mel'
    elif loc == 2 :
        label = 'nv'

    return label


def indexCreation(data, train_percentage) :
    samp = list(range(len(data)))
    samp = random.sample(samp, len(samp))

    index = samp[0:int(train_percentage*len(data))]
    return index
```

12.2.4 Models.py

```
# This script contains the outline of all models

# import modules
import tensorflow as tf
from tensorflow.keras.applications import ResNet50, VGG16, VGG19, InceptionV3, InceptionResNetV2



def get_model(model_choice='ResNet50', load_weights='imagenet', include_top=False) :

    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    prediction_layer = tf.keras.layers.Dense(3)

    if model_choice == 'ResNet50' :
        base_model = ResNet50(input_shape=(224,224,3), include_top=include_top, weights=load_weights)

        inputs = tf.keras.Input(shape=(224, 224, 3))
        x = tf.keras.applications.resnet50.preprocess_input(inputs)
        x = base_model(x, training=False)

    if model_choice == 'VGG16' :
        base_model = VGG16(input_shape=(224,224,3), include_top=include_top, weights=load_weights)

        inputs = tf.keras.Input(shape=(224, 224, 3))
        x = tf.keras.applications.vgg16.preprocess_input(inputs)
        x = base_model(x, training=False)

    if model_choice == 'InceptionResNetV2' :
        base_model = InceptionResNetV2(input_shape=(299,299,3), include_top=include_top, weights = load_weights)

        inputs = tf.keras.Input(shape=(299, 299, 3))
        x = tf.keras.applications.inception_resnet_v2.preprocess_input(inputs)
        x = base_model(x, training=False)


    x = global_average_layer(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    outputs = prediction_layer(x)
    model = tf.keras.Model(inputs, outputs)

    return model
```

12.3 Application Source Code

12.3.1 Frontend Code Source

12.3.1.1 AboutUs.dart

```
import 'package:flutter/material.dart';
import 'dart:math' as math;
import 'dart:io';

class AboutUs extends StatelessWidget {

 @override

 Widget _Andrew = Container(
     child: Column (
       children: <Widget> [
         CircleAvatar(
           radius: 90.0,
           backgroundImage: AssetImage('assets/blue.jpg'),
         ),
         SizedBox(height: 15.0,),
         Center(
           child: Text(
             'Andrew Mullen',
             style: TextStyle(
               fontSize: 20.0,
               fontWeight: FontWeight.bold,
               color: Colors.lightBlue[900],
             ),), ),
         Center(
           child: Text(
             'Machine Learning Researcher',
             style: TextStyle(
               fontSize: 20.0,
               fontWeight: FontWeight.bold,
               color: Colors.lightBlue[900],
             ), ),),
         Divider(
           color: Colors.lightBlue[900],
           height: 40.0,
           indent: 30.0,
           endIndent: 30.0,
         ),
         Center(
           child: Container(
             width: 350.0,
             child: Text(
               'Machine Learning Researcher',
               textAlign: TextAlign.justify,
               style: TextStyle(
                 fontSize: 20.0,
                 fontWeight: FontWeight.bold,
                 color: Colors.lightBlue[900],
               ),  ),  ),), ],   ) );

 Widget _Farnaz = Container(
     child: Column (
       children: <Widget> [
         CircleAvatar(
           radius: 90.0,
           backgroundImage: AssetImage('assets/Farnaz.jpg'),
         ),
         SizedBox(height: 15.0,),
         Center(
           child: Text(
             'Farnaz Forooghi',
             style: TextStyle(
               fontSize: 20.0,
               fontWeight: FontWeight.bold,
               color: Colors.lightBlue[900],
             ), ), ),
         Center(
           child: Text(
             'Machine Learning Researcher',
             style: TextStyle(
               fontSize: 20.0,
               fontWeight: FontWeight.bold,
               color: Colors.lightBlue[900],
```

```
            ), ),  ),
          Divider(
            color: Colors.lightBlue[900],
            height: 40.0,
            indent: 30.0,
            endIndent: 30.0,
          ),
          Center(
            child: Container(
              width: 350.0,
              child: Text(
                'Machine Learning Researcher',
                textAlign: TextAlign.justify,
                style: TextStyle(
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,
                  color: Colors.lightBlue[900],
                ),   ),  ), ), ],  ));

Widget _Mirt = Container(
    child: Column (
      children: <Widget> [
        CircleAvatar(
          radius: 90.0,
          backgroundImage: AssetImage('assets/Mirt.jpg'),
        ),
        SizedBox(height: 15.0,),
        Center(
          child: Text(
            'Mirt Mehany',
            style: TextStyle(
              fontSize: 20.0,
              fontWeight: FontWeight.bold,
              color: Colors.lightBlue[900],
            ),), ),
        Center(
          child: Text(
            'Mobile App Developer',
            style: TextStyle(
              fontSize: 20.0,
              fontWeight: FontWeight.bold,
              color: Colors.lightBlue[900],
            ), ), ),
        Divider(
          color: Colors.lightBlue[900],
          height: 40.0,
          indent: 30.0,
          endIndent: 30.0,
        ),
        Center(
          child: Container(
            width: 350.0,
            child: Text(
              'Mobile App Developer',
              textAlign: TextAlign.justify,
              style: TextStyle(
                fontSize: 20.0,
                fontWeight: FontWeight.bold,
                color: Colors.lightBlue[900],
              ), ), ),  ), ],) );
Widget _Nour = Container(
    child: Column (
      children: <Widget> [
        CircleAvatar(
          radius: 90.0,
          backgroundImage: AssetImage('assets/blue.jpg'),
        ),
        SizedBox(height: 15.0,),
        Center(
          child: Text(
            'Nour Abu Hantash',
            style: TextStyle(
              fontSize: 20.0,
              fontWeight: FontWeight.bold,
              color: Colors.lightBlue[900],
            ), ),),
        Center(
          child: Text(
            'Mobile App Developer',
            style: TextStyle(
              fontSize: 20.0,
```

```
              fontWeight: FontWeight.bold,
              color: Colors.lightBlue[900],
            ),  ), ),
        Divider(
          color: Colors.lightBlue[900],
          height: 40.0,
          indent: 30.0,
          endIndent: 30.0,
        ),
        Center(
          child: Container(
            width: 350.0,
            child: Text(
              'Mobile App Developer',
              textAlign: TextAlign.justify,
              style: TextStyle(
                fontSize: 20.0,
                fontWeight: FontWeight.bold,
                color: Colors.lightBlue[900],
              ),  ),  ),  ),    ], ) );
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       backgroundColor: Colors.lightBlue[900],
       toolbarHeight: 70.0,
       leading: GestureDetector (
         child: Icon (
           Icons.supervisor_account,
           size: 45.0,
           color: Colors.grey[50],
         ),),
       title: Text(
         'About Us',
         style: TextStyle(
           fontSize: 25.0,
           fontWeight: FontWeight.bold,
           letterSpacing: 1.0,
           color: Colors.grey[50],
         ), ),  ),
     body: Center(
       widthFactor: 200.0,
       child: SingleChildScrollView (
         child: Column (
           mainAxisAlignment: MainAxisAlignment.center,
           children: <Widget> [
             SizedBox(height: 30.0,),
             _Andrew,
             SizedBox(height: 30.0,),
             _Farnaz,
             SizedBox(height: 30.0,),
             _Mirt,
             SizedBox(height: 30.0,),
             _Nour,
             SizedBox(height: 20.0,),
           ],),,)),);}
```

12.3.2.2 camera.dart

```
import 'dart:async';
import 'dart:io';
import 'package:camera/camera.dart';
import 'package:capstone/screens/cameraPage.dart';
import 'package:flutter/material.dart';
import 'package:capstone/screens/ResultsView.dart';

class Camera extends StatefulWidget {
 @override
 _CameraState createState() => _CameraState();
}

class _CameraState extends State<Camera> {
 String _imagePath;
 File imageFile;
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       backgroundColor: Colors.lightBlue[900],
       toolbarHeight: 70.0,
       leading: GestureDetector(
         child: Icon(
           Icons.camera_alt_outlined,
```

```
          size: 45.0,
          color: Colors.grey[50],
        ),),
      title: Text(
        'Camera',
        style: TextStyle(
          fontSize: 25.0,
          fontWeight: FontWeight.bold,
          letterSpacing: 1.0,
          color: Colors.grey[50],),),),
    body: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
        _imagePath != null
            ? capturedImageWidget(_imagePath)
            : noImageWidget(),
        SizedBox(height: 30.0),
        Center(
          child: SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: openCamera,
              heroTag: 'Camera',
              backgroundColor: Colors.lightBlue[900],
              label: Text(
                'Camera',
                style: TextStyle(
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,),),),),),),
        SizedBox(height: 10.0),
        Center(
          child: SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: () {
                if (_imagePath != null)
                {Navigator.push(
                  context,
                  MaterialPageRoute(
                      builder: (context) => ResultsView(
                          imageFile: imageFile)),);}
                else
                {noImageWidget();}
              },
              heroTag: 'Analyze',
              backgroundColor: Colors.lightBlue[900],
              label: Text(
                'Analyze',
                style: TextStyle(
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,),),),),),
        SizedBox(height: 10.0),
        Center(
          child: SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: () {Navigator.of(context).pop();},
              heroTag: 'Cancel',
              backgroundColor: Colors.lightBlue[800],
              label: Text(
                'Cancel',
                style: TextStyle(
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,),),),),),],),);}

Widget noImageWidget() {
  return SizedBox(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Container(
```

```
            child: Icon(
              Icons.image,
              color: Colors.grey,
            ),
            width: 60.0,
            height: 60.0,
          ),
          Container(
            margin: EdgeInsets.only(top: 8.0),
            child: Text(
              'No Image Captured',
              style: TextStyle(
                color: Colors.grey,
                fontSize: 16.0,),),),],));}

Widget capturedImageWidget(String imagePath) {
  return SizedBox(
    child: Image.file(
      File(imagePath,),
      width: 450,
      height: 350,),);}

Widget fabWidget() {
  return Positioned(
    bottom: 30.0,
    right: 16.0,
    child: FloatingActionButton(
      onPressed: openCamera,
      child: Icon(
        Icons.photo_camera,
        color: Colors.white,
      ),
      backgroundColor: Colors.green,),);}

Future openCamera() async {
  availableCameras().then((cameras) async {
    final imagePath = await Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => CameraPage(cameras),
      ),);
    setState(() {
      _imagePath = imagePath;
      imageFile = File(imagePath);});});}}
```

12.3.2.3 camerPage.dart

```
import 'dart:async';
import 'dart:io';
import 'package:camera/camera.dart';
import 'package:path_provider/path_provider.dart';
import 'package:flutter/material.dart';
// https://www.c-sharpcorner.com/article/flutter-camera-overlay-or-overlap-using-stack-bar/
class CameraPage extends StatefulWidget {
 final List<CameraDescription> cameras;
 CameraPage(this.cameras);
 @override
 _CameraPageState createState() => _CameraPageState();
}

class _CameraPageState extends State<CameraPage> {
 String imagePath;
 bool _toggleCamera = false;
 CameraController controller;

 @override
 void initState() {
   onCameraSelected(widget.cameras[0]);
   super.initState();
 }

 @override
 void dispose() {
   controller?.dispose();
   super.dispose();
 }

 @override
 Widget build(BuildContext context) {
   if (widget.cameras.isEmpty) {
     return Container(
       alignment: Alignment.center,
```

```
    padding: EdgeInsets.all(16.0),
    child: Text(
      'No Camera Found',
      style: TextStyle(
        fontSize: 16.0,
        color: Colors.white,),),);}

if (!controller.value.isInitialized) {
  return Container();}

return AspectRatio(
  aspectRatio: controller.value.aspectRatio,
  child: Container(
    child: Stack(
      children: <Widget>[
        CameraPreview(controller),
        Align (
            alignment: Alignment.center,
            child: Container(
              width: 350.0,
              height: 250.0,
              padding: EdgeInsets.all(20.0),
              color: Color.fromRGBO(00, 00, 00, 0.25),)),
        Align (
          alignment: Alignment.topCenter,
          child: Padding(
            padding: const EdgeInsets.only(left: 20.0, top: 180.0, right: 20.0, bottom: 20.0),
            child: Text(
              "Center the Lesion Inside the Square",
              textAlign: TextAlign.center,
              style: TextStyle(
                decoration: TextDecoration.none,
                fontWeight: FontWeight.bold,
                fontSize: 15.0,
                letterSpacing: 0.0,
                color: Colors.grey[900],),),),),),
        Align(
          alignment: Alignment.bottomCenter,
          child: Container(
            width: double.infinity,
            height: 110.0,
            padding: EdgeInsets.all(10.0),
            color: Color.fromRGBO(00, 00, 00, 0.7),
            child: Stack(
              children: <Widget>[
                Align(
                  alignment: Alignment.center,
                  child: Material(
                    color: Colors.transparent,
                    child: InkWell(
                      borderRadius: BorderRadius.all(Radius.circular(100.0)),
                      onTap: () {
                        _captureImage();
                      },
                      child: Flexible(
                        child: Container(
                          padding: EdgeInsets.all(0.0),
                          child: Image.asset(
                            'assets/camera-button-icon-3.jpg',
                            width: 72.0,
                            height: 72.0,),),),),),),),
              Align(
                alignment: Alignment.centerRight,
                child: Material(
                  color: Colors.transparent,
                  child: InkWell(
                    borderRadius: BorderRadius.all(Radius.circular(50.0)),
                    onTap: () {
                      if (!_toggleCamera) {
                        onCameraSelected(widget.cameras[1]);
                        setState(() {
                          _toggleCamera = true;
                        });
                      } else {
                        onCameraSelected(widget.cameras[0]);
                        setState(() {
                          _toggleCamera = false;});});}},
                    child: Container(
                      padding: EdgeInsets.all(4.0),
                      child: Image.asset(
                        'assets/switch_camera.png',
```

```
                          color: Colors.grey[200],
                          width: 42.0,
                          height: 42.0,),),),),),],),),),),],),),);}

 void onCameraSelected(CameraDescription cameraDescription) async {
   if (controller != null) await controller.dispose();
   controller = CameraController(cameraDescription, ResolutionPreset.medium);

   controller.addListener(() {
     if (mounted) setState(() {});
     if (controller.value.hasError) {
       showMessage('Camera Error: ${controller.value.errorDescription}');
     }
   });
   try {
     await controller.initialize();
   } on CameraException catch (e) {
     showException(e);
   }

   if (mounted) setState(() {});}
 String timestamp() => new DateTime.now().millisecondsSinceEpoch.toString();
 void _captureImage() {
   takePicture().then((String filePath) {
     if (mounted) {
       setState(() {
         imagePath = filePath;
         print('(2) $imagePath/n');});
       if (filePath != null) {
         showMessage('Picture saved to $filePath');
         setCameraResult();}}});}
 void setCameraResult() {
   Navigator.pop(context, imagePath);
 }
 Future<String> takePicture() async {
   if (!controller.value.isInitialized) {
     showMessage('Error: select a camera first.');
     return null;
   }
   final Directory extDir = await getApplicationDocumentsDirectory();
   final String dirPath = '${extDir.path}/Images';
   await new Directory(dirPath).create(recursive: true);
   final String filePath = '$dirPath/${timestamp()}.jpg';

   if (controller.value.isTakingPicture) {
     // A capture is already pending, do nothing.
     return null;
   }
   try {
     await controller.takePicture(filePath);
   } on CameraException catch (e) {
     showException(e);
     return null;
   }
   return filePath;}
 void showException(CameraException e) {
   logError(e.code, e.description);
   showMessage('Error: ${e.code}\n${e.description}');}
 void showMessage(String message) {
   print(message);}
 void logError(String code, String message) =>
     print('Error: $code\nMessage: $message');}
```

12.3.2.4 Confirm.dart

```
import 'package:flutter/material.dart';
import 'dart:math' as math;
import 'dart:io';
import 'package:capstone/screens/Login.dart';
class Confirm extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       backgroundColor: Colors.lightBlue[900],
       toolbarHeight: 70.0,
       leading: GestureDetector (
         child: Icon (
           Icons.youtube_searched_for_rounded,
           size: 45.0,
           color: Colors.grey[50],
         ),),
```

```
      title: Text(
        'Dermatologist',
        style: TextStyle(
          fontSize: 25.0,
          fontWeight: FontWeight.bold,
          letterSpacing: 1.0,
          color: Colors.grey[50],),),),

    backgroundColor: Colors.grey[50],
    body: Center(
      child: Column (
        //mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget> [
          SizedBox(height: 30.0),
          Icon (
            Icons.person_add_rounded,
            size: 100.0,
            color: Colors.lightBlue[900],
          ),
          SizedBox(height: 40.0),
          Text (
            'Thank you!',
            style: TextStyle (
              fontWeight: FontWeight.bold,
              fontSize: 50.0,
              letterSpacing: 1.5,
              color: Colors.lightBlue[800],
            ),),
          SizedBox(height: 10.0),
          SizedBox(
            width: 350.0,
            height: 100.0,
            child: Text (
              'We are incredibly excited to have you here! '
                  'All our wishes for healing and strength for all disease fighters everywhere 🖤',
              style: TextStyle (
                fontWeight: FontWeight.bold,
                fontSize: 15.0,
                letterSpacing: 1.0,
                color: Colors.grey[500],),),),
          SizedBox(height: 55.0,),
          SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => Login()),);},
              backgroundColor: Colors.lightBlue[900],
              label: Text (
                'Login',
                style: TextStyle (
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,
                ),),),),],),),);}}
```

12.3.2.5 gallery.dart

```
import 'dart:async';
import 'package:capstone/components/utilities.dart';
import 'package:capstone/screens/image_analysis_result.dart';
import 'package:flutter/material.dart';
import 'package:capstone/screens/Main.dart';
import 'package:flutter/services.dart';
import 'dart:io';
//import 'package:universal_widget/universal_widget.dart';

import 'package:image_picker/image_picker.dart';

class Gallery extends StatefulWidget {
 @override
 _GalleryState createState() => _GalleryState();
}

class _GalleryState extends State<Gallery> {

 static const platform = const MethodChannel('flutter.native/helper');

 String imagePath;
```

```
String timestamp() => new DateTime.now().millisecondsSinceEpoch.toString();

Future<void> predictAsync(String path) async {
  print(path);
  try {

    Map<dynamic, dynamic> result = await platform.invokeMethod('predictImageAsync', {"path": path});
    var res = result['result'];
    print(result);
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => Result(result: res, imagePath: imagePath)),
    );

  } on Exception catch (e) {
    var response = e.toString();
    print(response);
    showToastMessage(response);
  }
}
final ImagePicker _picker = ImagePicker();
Widget showImage () {
  return FutureBuilder <PickedFile> (
      future: Future.value(imageFile),
      // ignore: missing_return
      builder: (BuildContext context, AsyncSnapshot <PickedFile> snapShot) {
        if (snapShot.connectionState == ConnectionState.done &&
            snapShot.data != null ) {
          imagePath = snapShot.data.path;
            return Image.file(
              File(snapShot.data.path),
              width: 450,
              height: 350,
            );
          } else if (snapShot.error != null){
          return const Text(
            'Error Picking Image',
            textAlign: TextAlign.center,
          );
        } else {
          return const Text(
            'No Image Selected',
            textAlign: TextAlign.center,);}},);}
PickedFile imageFile;
Future<void>pickImageFromGallery(ImageSource source) async {
  try {
    final pickedFile = await _picker.getImage(
      source: source,
    );
    setState(() {
      imageFile = pickedFile;
    });
  } catch (e) {
    setState(() {
     // _pickImageError = e;});}}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.lightBlue[900],
      toolbarHeight: 70.0,
      leading: GestureDetector (
        child: Icon (
          Icons.image_search,
          size: 45.0,
          color: Colors.grey[50],
        ),
      ),
      title: Text(
        'Gallery',
        style: TextStyle(
          fontSize: 25.0,
          fontWeight: FontWeight.bold,
          letterSpacing: 1.0,
          color: Colors.grey[50],),),),
    body: Column (
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget> [
        imageFile != null
```

```
            ? capturedImageWidget(imageFile)
            : noImageWidget(),
        //showImage(),
        SizedBox(height: 30.0),
        Center(
          child: SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: () {
                  pickImageFromGallery(ImageSource.gallery);
                  //this happens through Flutter not Android & Java
              },
              heroTag: 'Select Image',
              backgroundColor: Colors.lightBlue[900],
              label: Text (
                'Select Image',
                style: TextStyle (
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,),),),),),),
        SizedBox(height: 10.0),
        Center(
          child: SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: () {
                if(imageFile == null){
                  showToastMessage("Select an image first");
                  return;
                }
                //pickImageFromGallery(ImageSource.gallery);
                predictAsync(imageFile.path);
              },
              heroTag: 'Analyze',
              backgroundColor: Colors.lightBlue[900],
              label: Text (
                'Analyze',
                style: TextStyle (
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,),),),),),),
        SizedBox(height: 10.0),
        Center(
          child: SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => Main()),
                );
              },
              heroTag: 'Cancel',
              backgroundColor: Colors.lightBlue[800],
              label: Text (
                'Cancel',
                style: TextStyle (
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,),),),),),],),);}
Widget noImageWidget() {
  return SizedBox(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Container(
            child: Icon(
              Icons.image,
              color: Colors.grey,
            ),
            width: 60.0,
            height: 60.0,
          ),
          Container(
            margin: EdgeInsets.only(top: 8.0),
```

```
            child: Text(
              'No Image Captured',
              style: TextStyle(
                color: Colors.grey,
                fontSize: 16.0,),),),],));}

 Widget capturedImageWidget(File imageFile) {
   return SizedBox(
     child: Image.file(
       imageFile,
       width: 450,
       height: 350,),);}}
```

12.3.2.6 History.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/rendering.dart';
import 'dart:io';
import 'dart:math';

//https://flutter.dev/docs/cookbook/effects/parallax-scrolling

class HistoryEntity{
 String date;
 String name;
 String URI;

 HistoryEntity(String name, String date, String URI) {
   this.date = date;
   this.name = name;
   this.URI = URI;
 }
}

class History extends StatefulWidget {
 @override
 _HistoryState createState() => _HistoryState();
}

class _HistoryState extends State<History> {
 final historyFileCount = 1;
 var historyFiles = List<HistoryEntity>();
 initializeData() {
   historyFiles.add(new HistoryEntity(name, date, URI));}
 renderHistory() {
   var widgets = List<Widget>();
   var entity = Row(
     mainAxisAlignment: MainAxisAlignment.start,
     children: [
       Image.network(
           "https://upload.wikimedia.org/wikipedia/commons/thumb/6/6c/Melanoma.jpg/1200px-Melanoma.jpg",
           width: 200,
           height: 200),
       SizedBox(width: 20.0,),
       Column(
         children: [
           Text("Melanoma", style: TextStyle(fontSize: 20.0, fontWeight: FontWeight.bold),),
           SizedBox(height: 10.0,),
           Text("14-04-2021", style: TextStyle(fontSize: 20.0, fontWeight: FontWeight.bold),)],)],);
   widgets.add(
       Column(
           children: [
             entity,
             Divider(
               color: Colors.lightBlue[900],
               height: 5.0,
               indent: 15.0,
               endIndent: 15.0,),]));
   return widgets;
 }
 @override
 Widget build(BuildContext context) {
   initializeData();
   return Scaffold(
       appBar: AppBar(
         backgroundColor: Colors.lightBlue[900],
         toolbarHeight: 70.0,
         leading:IconButton(
           onPressed: () {Navigator.of(context).pop();},
           icon: Icon(
```

```
            Icons.arrow_back,
            size: 45.0,
            color: Colors.grey[50],),),
        title: Text(
          'History',
          style: TextStyle(
            fontSize: 25.0,
            fontWeight: FontWeight.bold,
            letterSpacing: 1.0,
            color: Colors.grey[50],),),),
      body: SingleChildScrollView(
          child: Padding(
            padding: const EdgeInsets.all(10.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: renderHistory(),),)));}}
```

12.3.2.7 Login.dart

```
import 'package:capstone/components/utilities.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'dart:math' as math;
import 'dart:io';
import 'package:capstone/screens/SignUp.dart';
import 'package:capstone/screens/Main.dart';
import 'package:flutter/services.dart';


class Login extends StatefulWidget {
 @override
 _LoginState createState() => _LoginState();
}
class _LoginState extends State<Login> {
 String loginEmail;
 String loginPassword;
 String response = "";
 bool _obscureText = true;

 static const platform = const MethodChannel('flutter.native/helper');
//Toggles the password show status
 void _toggle(){
   setState(() {
     _obscureText = !_obscureText;
   });
 Future<void> loginAsync(String email, String password) async {
   //Future tells the application to not expect an immediate response
   try { //try, catch, exception:
         // good to use try and catch when you expect something might go wrong

     Map<dynamic, dynamic> result = await platform.invokeMethod('loginAsync', {"email": email, "password":password});
     // here we call the Java (Android Side) platform method/function
     // we send off entered user information to the Java code by keys and dictionaries
     showToastMessage(result['message']);
     Navigator.push(
       context,
       MaterialPageRoute(builder: (context) => Main()),
       // nothing really happens here since its Future. aync and await tell the
       // application to wait the occurrence of an event/result
       // When successful it goes back to Main
     );

   } on Exception catch (e) {
     //need to catch so an error won't crash the application
     response = e.toString();
     print(response);
     showToastMessage(response);
   }
 }

 @override
 Widget build(BuildContext context) {
   TextStyle style = TextStyle(
       color: Colors.grey[50],
       letterSpacing: 0.0 ,
       fontSize: 15.0);

   final loginEmailField = TextField(
       obscureText: false,
       style: style,
       cursorColor: Colors.grey[50],
```

```
    keyboardType: TextInputType.text,
    onChanged: (value) {
      setState (() {
        loginEmail = value;
      });
    },
    decoration: InputDecoration(
        hintText: "EMAIL",
        border:
        UnderlineInputBorder  (borderSide: BorderSide(width: 10.0))));

final loginPasswordField = TextFormField(
    obscureText: _obscureText,
    style: style,
    cursorColor: Colors.grey[50],
    keyboardType: TextInputType.text,
    onChanged: (value){
      setState( () {
        loginPassword = value;
      });},
    decoration: InputDecoration(
        hintText: "PASSWORD",
        border:
        UnderlineInputBorder  (borderSide: BorderSide(width: 10.0))));

return Scaffold(
    backgroundColor: Colors.lightBlue[900],
    body: SingleChildScrollView(
    child: ConstrainedBox(
    constraints: BoxConstraints(),
      child: Padding(
       padding: const EdgeInsets.fromLTRB(20.0,0.0,10.0,00.0),
        child: Column (
          //mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget> [
            SizedBox(height: 75.0),
            Icon(
          Icons.youtube_searched_for_rounded,
          color: Colors.grey[50],
          size: 110.0,
        ),
        SizedBox(height: 5.0),
        Text(
            'Dermatologist',
            style: TextStyle(
              color: Colors.grey[50],
              letterSpacing: 2.0,
              fontSize: 25.0,
              fontWeight: FontWeight.bold,
            )),
        SizedBox(height: 40.0),
        SizedBox.fromSize(
          child: Row(
            children: <Widget> [
              Icon(
                Icons.person,
                color: Colors.grey[50],
                size: 35.0,),
              Expanded(child: loginEmailField),],),),
        SizedBox(height: 15.0,),
        Column(
          crossAxisAlignment: CrossAxisAlignment.end,
          children: <Widget> [
            SizedBox.fromSize(
              child: Row(
                children: <Widget> [
                  Icon(
                  Icons.lock_outline_rounded,
                   color: Colors.grey[50],
                  size: 35.0,),
                Expanded(child: loginPasswordField),],),),
          new FlatButton(onPressed: _toggle, child: new Text(_obscureText? "Show" : "Hide")),
          ],
        ),
          SizedBox(height: 50.0,), //37
          SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
            //
              onPressed: () {
```

```
                //showToastMessage(loginEmail);
                var email = loginEmail?? "";
                var password = loginPassword?? "";
                // get the email and password, chacking if they are null
                // or not to avoid crashing
                if(email.isEmpty || password.isEmpty) {
                  showToastMessage("Email and password should not be empty");
                  return;
                }
                try{//similar ti if else staments
                  loginAsync(email, password);
                  //call login function
                }
                on Exception catch(e){
                  showToastMessage("Error");
                  print(e.toString());}},
              heroTag: 'btn',
              backgroundColor: Colors.grey[50],
              label: Text (
                  'Login',
                  style: TextStyle(
                    color: Colors.lightBlue[900],
                    letterSpacing: 1.0,
                    fontSize: 20.0,
                    fontWeight: FontWeight.bold,)),),),
            SizedBox(height: 18.0,),
            SizedBox(
              width: 200.0,
              height: 50.0,
              child: FloatingActionButton.extended(
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(builder: (context) => SignUp()),);},
                heroTag: 'Sign Up',
                backgroundColor: Colors.lightBlue[900],
                label: Text (
                  'Sign Up',
                  style: TextStyle (
                    color: Colors.grey[50],
                    letterSpacing: 1.0,
                    fontSize: 20.0,
                    fontWeight: FontWeight.bold,),),),),],),));}}
```

12.3.2.8 Main.dart

```
import 'package:capstone/components/utilities.dart';
import 'package:capstone/screens/AboutUs.dart';
import 'package:capstone/screens/History.dart';
import 'package:capstone/screens/Prevention.dart';
import 'package:capstone/screens/SkinLesion.dart';
import 'package:capstone/screens/camera.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'dart:math' as math;
import 'dart:io';
import 'package:image_picker/image_picker.dart';
import 'package:capstone/screens/Start.dart';
import 'package:capstone/screens/Login.dart';
import 'package:capstone/screens/SignUp.dart';
import 'package:capstone/screens/Confirm.dart';
import 'package:capstone/screens/Main.dart';
import 'package:capstone/screens/History.dart';
import 'package:capstone/screens/Gallery.dart';
//import 'package:flutter/services.dart';
//                                                    important                                                    link:
https://medium.com/coding-with-flutter/flutter-case-study-multiple-navigators-with-bottomnavigationbar-90eb6caa6dbf

class Main extends StatefulWidget {
 //final int idx;
 //Main ({Key key, @required this.idx}): super(key: key);

 @override
 _MainState createState() => _MainState();
}

class _MainState extends State<Main> {
 String response = "";

 static const platform = MethodChannel('flutter.native/helper');

 int _selectedIndex = 0;
```

```
void _onItemTapped(int index) {
  setState(() {
    _selectedIndex = index;
  });}

final ImagePicker _picker = ImagePicker();
PickedFile imageFile;
pickImageFromGallery(ImageSource source) {
  Future<void>pickImageFromGallery(ImageSource source) async {
    try {
      final pickedFile = await _picker.getImage(
        source: source,
        maxWidth: 254,
        maxHeight: 254,
        imageQuality: 3,);
      setState(() {
        imageFile = pickedFile;
      });
    } catch (e) {
      setState(() {
        // _pickImageError = e;});}}}

@override
Widget build(BuildContext context) {

  var name = "saveUser":user;

  Widget showImage() {
    return FutureBuilder<PickedFile>(
      future: Future.value(imageFile),
      builder: (BuildContext context, AsyncSnapshot<PickedFile> snapshot) {
        if (snapshot.connectionState == ConnectionState.done &&
            snapshot.data != null) {
          return Image.file(
            File(snapshot.data.path),
            width: 300,
            height: 300,
          );
        } else if (snapshot.error != null) {
          return const Text(
            'Error Picking Image',
            textAlign: TextAlign.center,
          );
        } else {
          return const Text(
            'No Image Selected',
            textAlign: TextAlign.center,);}},);}

  Widget _profile = Scaffold(
    body: Padding(
      padding: const EdgeInsets.fromLTRB(30.0, 25.0, 30.0, 0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          Center(
            child: Row(
              mainAxisAlignment: MainAxisAlignment.center,
              crossAxisAlignment: CrossAxisAlignment.end,
              children: <Widget> [
                ClipOval(
                  child: Container(
                      alignment: Alignment.center,
                      color: Colors.lightBlue[900],
                      height: 150.0,
                      width: 150.0,
                      child: Text(name.substring(0, 1).toUpperCase(), style: TextStyle(fontSize: 75, fontWeight: FontWeight.bold,
color: Colors.grey[50]),),),),],),),
          SizedBox(height: 25.0,),
          Center(
            child: Text(
              name,
              style: TextStyle(
                fontSize: 25.0,
                fontWeight: FontWeight.bold,
                color: Colors.lightBlue[900],),),),
          Divider(
            color: Colors.lightBlue[900],
            height: 55.0,),
          SizedBox(height: 15.0,),
          Center(
```

```
            child: SizedBox(
              width: 200.0,
              height: 50.0,
              child: FloatingActionButton.extended(
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(builder: (context) => History()),
                  );
                },
                heroTag: 'History',
                backgroundColor: Colors.lightBlue[900],
                label: Text(
                  'History',
                  style: TextStyle(
                    color: Colors.grey[50],
                    letterSpacing: 1.0,
                    fontSize: 20.0,
                    fontWeight: FontWeight.bold,),),),),),),
          SizedBox(
            height: 10.0,
          ),
          Center(
            child: SizedBox(
              width: 200.0,
              height: 50.0,
              child: FloatingActionButton.extended(
                onPressed: () {
                  logout();
                },
                heroTag: 'Sign Out',
                backgroundColor: Colors.lightBlue[900],
                label: Text(
                  'Sign Out',
                  style: TextStyle(
                    color: Colors.grey[50],
                    letterSpacing: 1.0,
                    fontSize: 20.0,
                    fontWeight: FontWeight.bold,),),),),),],),),);

  final _gallery = Scaffold(
    body: Padding(
      padding: const EdgeInsets.fromLTRB(30.0, 40.0, 30.0, 0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        children: <Widget> [
          SizedBox(height: 150.0),
          Center(
            child: SizedBox(
              width: 200.0,
              height: 50.0,
              child: FloatingActionButton.extended(
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(builder: (context) => Gallery()),
                  );
                  //pickImageFromGallery(ImageSource.gallery);
                },
                heroTag: 'Gallery',
                backgroundColor: Colors.lightBlue[900],
                label: Text (
                  'Gallery',
                  style: TextStyle (
                    color: Colors.grey[50],
                    letterSpacing: 1.0,
                    fontSize: 20.0,
                    fontWeight: FontWeight.bold,),),),),),
          SizedBox(height: 20.0,),
          Center(
            child: SizedBox(
              width: 200.0,
              height: 50.0,
              child: FloatingActionButton.extended(
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(builder: (context) => Camera()),
                  );
                },
                heroTag: 'Camera',
```

```
              backgroundColor: Colors.lightBlue[900],
              label: Text(
                'Camera',
                style: TextStyle(
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,),),),),),],),),);

Widget _insightNew = Scaffold(
    body: new Stack(
      fit: StackFit.expand,
      children: <Widget>[
        new Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            SizedBox(
              height: 100.0,
              width: 360.0,
              child: new RaisedButton(
                  elevation: 0.0,
                  shape: new RoundedRectangleBorder(borderRadius: new BorderRadius.circular(60.0)),
                  padding: EdgeInsets.only(top: 7.0, bottom: 7.0, right: 130.0, left: 7.0),
                  onPressed: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(builder: (context) => SkinLesions()),);},
                  child: new Row(
                    mainAxisSize: MainAxisSize.min,
                    children: <Widget>[
                      CircleAvatar(
                        radius: 40.0,
                        backgroundImage: AssetImage('assets/skin_cancer.jpg'),
                      ),
                      Padding(
                          padding: EdgeInsets.only(left: 15.0),
                          child: new Text(
                            "Skin Lesions",
                            style: TextStyle(
                              fontSize: 20.0,
                              fontWeight: FontWeight.bold,
                              letterSpacing: 1.0,
                              color: Colors.grey[50],),))],),
                  color: Colors.lightBlue[900]),),
            Padding(
              padding: const EdgeInsets.only(top: 30.0,),
              child: SizedBox(
                height: 100.0,
                width: 360.0,
                child: new RaisedButton(
                  elevation: 0.0,
                  shape: new RoundedRectangleBorder(borderRadius: new BorderRadius.circular(60.0)),
                  padding: EdgeInsets.only(top: 7.0, bottom: 7.0, right: 150.0, left: 7.0),
                  onPressed: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(builder: (context) => Prevention()),);},
                  child: new Row(
                    mainAxisSize: MainAxisSize.min,
                    children: <Widget>[
                      CircleAvatar(
                        radius: 40.0,
                        backgroundImage: AssetImage('assets/prevention_green.png'),
                      ),
                      Padding(
                          padding: EdgeInsets.only(left: 15.0),
                          child: new Text(
                            "Prevention",
                            style: TextStyle(
                              fontWeight: FontWeight.bold,
                              fontSize: 20.0,
                              letterSpacing: 1.0,
                              color: Colors.grey[50],),))],),
                  color: Colors.lightBlue[900],),),),
            Padding(
              padding: const EdgeInsets.only(top: 30.0,),
              child: SizedBox(
                height: 100.0,
                width: 360.0,
                child: new RaisedButton(
                  elevation: 0.0,
```

```
                    shape: new RoundedRectangleBorder(
                        borderRadius: new BorderRadius.circular(60.0)),
                    padding: EdgeInsets.only(
                        top: 7.0, bottom: 7.0, right: 150.0, left: 0.0),
                    onPressed: () async {
                      Navigator.push(
                        context,
                        MaterialPageRoute(builder: (context) => AboutUs()),);},
                    child: new Row(
                      mainAxisSize: MainAxisSize.min,
                      children: <Widget>[
                        CircleAvatar(
                          radius: 40.0,
                          backgroundImage: AssetImage('assets/about_us.png'),
                        ),
                        Padding(
                            padding: EdgeInsets.only(left: 15.0),
                            child: new Text(
                              "About Us ",
                              style: TextStyle(
                                fontWeight: FontWeight.bold,
                                fontSize: 20.0,
                                letterSpacing: 1.0,
                                color: Colors.grey[50],),)),],),color: Colors.lightBlue[900],),),),)],)],));

  List<Widget> _widgetOptions = <Widget>[
    _profile, // index 0
    _gallery, // index 1
    _insightNew
    // _insight, // index 2
  ];

  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.lightBlue[900],
      toolbarHeight: 70.0,
      leading: GestureDetector(
        child: Icon(
          Icons.youtube_searched_for_rounded,
          size: 45.0,
          color: Colors.grey[50],),),
      title: Text(
        'Dermatologist',
        style: TextStyle(
          fontSize: 25.0,
          fontWeight: FontWeight.bold,
          letterSpacing: 1.0,
          color: Colors.grey[50],),),),
    body: Center(
      child: _widgetOptions.elementAt(_selectedIndex),
    ),
    bottomNavigationBar: new Theme(
      data: Theme.of(context).copyWith(
        canvasColor: Colors.lightBlue[900],
      ),
      child: BottomNavigationBar(
        items: <BottomNavigationBarItem> [
          BottomNavigationBarItem(icon: Icon(Icons.person_outline_rounded, size: 50.0, ), label:'Profile'),
          BottomNavigationBarItem(icon: Icon(Icons.add_a_photo_outlined, size: 50.0, ), label:'Gallery'),
          BottomNavigationBarItem(icon: Icon(Icons.lightbulb_outline_rounded, size: 50.0, ), label:'Insight'),
          //BottomNavigationBarItem(icon: Icon (Icons.settings, size: 40.0), label:'Settings'),
        ],
        currentIndex: _selectedIndex,
        unselectedItemColor: Colors.blue[200],
        selectedItemColor: Colors.grey[50],
        onTap: _onItemTapped,),),);}

Future<void> logout() async {
  try {
    Map<dynamic, dynamic> result = await platform.invokeMethod('logOutAsync');
    showToastMessage(result['message']);
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => Main()),
    );
  } on Exception catch (e) {
    response = e.toString();
    print(response);
    showToastMessage(response);}}}
```

12.3.2.9 Prevention.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'dart:math' as math;
import 'dart:io';

class Prevention extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       backgroundColor: Colors.lightBlue[900],
       toolbarHeight: 70.0,
       leading:IconButton(
         onPressed: () {Navigator.of(context).pop();},
         icon: Icon(
           Icons.arrow_back,
           size: 45.0,
           color: Colors.grey[50],),),
       title: Text(
         'Prevention',
         style: TextStyle(
           fontSize: 25.0,
           fontWeight: FontWeight.bold,
           letterSpacing: 1.0,
           color: Colors.grey[50],),),),
     body: Center(
       child: SingleChildScrollView(
             child: Column(
               mainAxisAlignment: MainAxisAlignment.center,
               children: <Widget>[
                 SizedBox(height: 20.0),
                 Center(
                   child: ClipRRect(
                     borderRadius: BorderRadius.circular(20.0),
                     child: Container(
                       height: 250.0,
                       //width: 500.0,
                      child: Image.asset('assets/sun.gif'),),),),
                 SizedBox (height: 15.0),
                 Center(
                   child: Container(
                     width: 350.0,
                     child: Text(
                       '1. Wear sunscreen and/or protective clothing\n\n'
                           '2. Seek shade when possible to reduce UV exposure\n\n'
                           '3. Use a daily moisturizer with SPF for your face, ears and back of the neck\n\n'
                           '4. Avoid tanning beds\n\n'
                           '5. Give yourself a self-examination, looking for a new or changing spots on your skin\n',
                        textAlign: TextAlign.justify,
                       style: TextStyle(
                         fontSize: 20.0,
                         fontWeight: FontWeight.bold,
                         color: Colors.lightBlue[900],),),),),
                 SizedBox(height: 20.0),
               ],),),) );}}
```

12.3.2.10 ResultsView.dart

```
import 'dart:async';
import 'package:capstone/components/utilities.dart';
import 'package:flutter/material.dart';
import 'package:capstone/screens/Main.dart';
import 'package:flutter/services.dart';
import 'dart:io';

import 'package:image_picker/image_picker.dart';

class Result extends StatefulWidget {
 final String result;
 final String imagePath;
 Result({Key key, @required this.result, @required this.imagePath}) : super(key: key);
 @override
 _ResultState createState() => _ResultState(result: result, imagePath: imagePath);
}

class _ResultState extends State<Result> {

 final String result;
 final String imagePath;
 _ResultState({@required this.result, @required this.imagePath}) : super();
```

```
final ImagePicker _picker = ImagePicker();
Widget showImage () {
  return FutureBuilder <PickedFile> (
      future: Future.value(imageFile),
      // ignore: missing_return
      builder: (BuildContext context, AsyncSnapshot <PickedFile> snapShot) {
        if (snapShot.connectionState == ConnectionState.done &&
            snapShot.data != null ) {
            return Image.file(
              File(this.imagePath),
              width: 450,
              height: 350,);}
          else if (snapShot.error != null){
          return const Text(
            'Error Picking Image',
            textAlign: TextAlign.center,
          );
        } else {
          return const Text(
            'No Image Selected',
            textAlign: TextAlign.center,);}},);}
PickedFile imageFile;
Future<void>pickImageFromGallery(ImageSource source) async {
  try {
    final pickedFile = await _picker.getImage(
      source: source,
    );
    setState(() {
      imageFile = pickedFile;
    });
  } catch (e) {
    setState(() {
     // _pickImageError = e;
    });}}
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.lightBlue[900],
      toolbarHeight: 70.0,
      leading: GestureDetector (
        child: Icon (
          Icons.image_search,
          size: 45.0,
          color: Colors.grey[50],),),
      title: Text(
        'Result',
        style: TextStyle(
          fontSize: 25.0,
          fontWeight: FontWeight.bold,
          letterSpacing: 1.0,
          color: Colors.grey[50],),),),
    body: Column (
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget> [
        Center(
          child: Image.file(
            File(imagePath),
            width: 450,
            height: 350,),),
        SizedBox(height: 30.0),
        Center(
          child: Text(
            result,
            textAlign: TextAlign.center,),),
        SizedBox(height: 30.0),
        Center(
          child: SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: () {
                //pickImageFromGallery(ImageSource.gallery);
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) => Main()));
              },
              heroTag: 'Done',
              backgroundColor: Colors.lightBlue[900],
```

```
              label: Text (
                'Cancel',
                style: TextStyle (
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,
                ),),), ),),], ),);}}
```

12.3.2.11 SignUp.dart

```
import 'package:capstone/components/utilities.dart';
import 'package:flutter/material.dart';
import 'dart:math' as math;
import 'dart:io';
import 'package:capstone/screens/Confirm.dart';
import 'package:flutter/services.dart';

class SignUp extends StatefulWidget {
 @override
 _SignUpState createState() => _SignUpState();
}
class _SignUpState extends State<SignUp> {
 TextStyle style = TextStyle(color: Colors.lightBlue[900], letterSpacing: 1.5 ,fontSize: 15.0);
 String signUpEmail;
 String signUpUsername;
 String signUpPassword;
 String response = "";
 bool _obscureText = true;
 static const platform = const MethodChannel('flutter.native/helper');


 Future<void> signUpAsync(String email, String password, String username) async {
   try {
      final Map<String, String> result = await platform.invokeMethod('createAccountAsync', {"email": email, "password":password,
"username":username});
     showToastMessage(result['status']);
     response = result['message'];
     Navigator.push(
       context,
       MaterialPageRoute(builder: (context) => Confirm()),
     );
   } on Exception catch (e) {
     var error = e;
     print(error);
     showToastMessage(error.toString());}}
 //Toggles the password show status
 void _toggle(){
   setState(() {
     _obscureText = !_obscureText;});}

 @override
 Widget build(BuildContext context) {

   final signUpEmailField = TextFormField(
       obscureText: false,
       style: style,
       cursorColor: Colors.grey[50],
       keyboardType: TextInputType.text,
       onChanged: (value) {
         setState (() {
           signUpEmail = value;});},
       decoration: InputDecoration(
           hintText: "EMAIL",
           border:
           UnderlineInputBorder  (borderSide: BorderSide(width: 10.0))));

   final signUpPasswordField = TextFormField(
       obscureText: false,
       style: style,
       cursorColor: Colors.grey[50],
       keyboardType: TextInputType.text,
       onChanged: (value){
         setState( () {
           signUpPassword = value;});},
       decoration: InputDecoration(
           hintText: "PASSWORD",
           border:
           UnderlineInputBorder  (borderSide: BorderSide(width: 10.0))));

   final signUpUsernameField = TextFormField(
       obscureText: false,
```

```
      style: style,
      cursorColor: Colors.grey[50],
      keyboardType: TextInputType.text,
      onChanged: (value){
        setState( () {
          signUpUsername = value;});},
      decoration: InputDecoration(
          hintText: "USERNAME",
          border:
          UnderlineInputBorder  (borderSide: BorderSide(width: 10.0))));

  return Scaffold(
    backgroundColor: Colors.grey[50],
    appBar: AppBar (
      leading:IconButton(
        onPressed: () {Navigator.of(context).pop();},
        icon: Icon(
          Icons.arrow_back,
          size: 45.0,
          color: Colors.grey[50],),),
      title: Text (
        'Sign Up',
        style: TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 25.0,
          letterSpacing: 2.0,
          color: Colors.grey[50],),),
      backgroundColor: Colors.lightBlue [900],
      toolbarHeight: 80.0,),
    body: SingleChildScrollView(
      child: ConstrainedBox(
      constraints: BoxConstraints(),
      child: Padding(
      padding: const EdgeInsets.fromLTRB(20.0,0.0,10.0,00.0),
      child: Column (
        mainAxisAlignment: MainAxisAlignment.start,
        children: <Widget> [
          SizedBox(height: 50.0),
          Row(
            children: <Widget> [
              Icon(
                Icons.person,
                color: Colors.lightBlue[900],
                size: 35.0,),
              Expanded(child: signUpUsernameField),],),
          SizedBox(height: 20.0,),
          Row(
              children: <Widget> [
                Icon(
                  Icons.email_rounded,
                  color: Colors.lightBlue[900],
                  size: 35.0,),
                Expanded(child: signUpEmailField),]),
          SizedBox(height: 20.0,),
          Column(
            crossAxisAlignment: CrossAxisAlignment.end,
            children:<Widget> [
               Row(
                children: <Widget> [
                  Icon(
                   Icons.lock_outline_rounded,
                   color: Colors.lightBlue[900],
                   size: 35.0,),
                Expanded(child: signUpPasswordField),]),
          new FlatButton(onPressed: _toggle, child: new Text(_obscureText? "Show" : "Hide")),],),
          SizedBox(height: 80.0,),
          SizedBox(
            width: 200.0,
            height: 50.0,
            child: FloatingActionButton.extended(
              onPressed: () {

                var email = signUpEmail ?? "";
                var password = signUpPassword?? "";
                var username = signUpUsername?? "";
                if(email.isEmpty || password.isEmpty || username.isEmpty) {
                  showToastMessage("Email, password or username should not be empty");
                  return;
                }
                try{
                  signUpAsync(email, password, username);
```

```
                }
                on Exception catch(e){
                  showToastMessage("Error");
                  print(e.toString());}},
              heroTag: 'Confirm',
              backgroundColor: Colors.lightBlue[900],
              label: Text (
                'Confirm',
                style: TextStyle (
                  color: Colors.grey[50],
                  letterSpacing: 1.0,
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,),),),),),
            SizedBox(height: 18.0),
            SizedBox(
              width: 200.0,
              height: 50.0,
              child: FloatingActionButton.extended(
                onPressed: () {
                  Navigator.pop(context,);},
                heroTag: 'back to login',
                backgroundColor: Colors.lightBlue[800],
                label: Text (
                  'Not Now',
                  style: TextStyle (
                    color: Colors.grey[50],
                    letterSpacing: 1.0,
                    fontSize: 20.0,
                    fontWeight: FontWeight.bold,),),),),],),),);}}
```

12.3.2.12 SkinLesion.dart

```
import 'package:flutter/material.dart';
import 'dart:math' as math;
import 'dart:io';

class SkinLesions extends StatelessWidget {
 @override

 Widget _BKL = Container(
     child: Column (
       children: <Widget> [
         Center(
           child: ClipRRect(
             borderRadius: BorderRadius.circular(15.0),
             child: Container(
               height: 200.0,
               width: 250.0,
               child: Image.asset('assets/BKL.jpg'),),),),
         SizedBox(height: 15.0),
         Center(
           child: Text(
             'Benign Keratosis-like Lesion',
             style: TextStyle(
               fontSize: 20.0,
               fontWeight: FontWeight.bold,
               color: Colors.lightBlue[900],),),),
         Divider(
           color: Colors.lightBlue[900],
           height: 40.0,
           indent: 30.0,
           endIndent: 30.0,),
         Center(
           child: Container(
             width: 350.0,
             child: Text(
                 'A seborrheic keratosis is one of the common skin lesions that tend to appear in the skin as a person gets older.
'
                   'These lesions are usually darker than a persons skin color and close to brown or black. '
                     'The growths of them look waxy, scaly and slightly swollen. They mostly appear on the head and neck or upper
limb. '
                   'Seborrheic keratoses are harmless and non-contagious lesions. '
                     'Usually there is no treatment for them, however if they become irritated by clothing or bother the patient,
then they may decide to remove them.',
               textAlign: TextAlign.justify,
               style: TextStyle(
                 fontSize: 20.0,
                 fontWeight: FontWeight.bold,
                 color: Colors.lightBlue[900],),),),),],));
 Widget _NV = Container(
     child: Column (
```

```
      children: <Widget> [
        Center(
          child: ClipRRect(
            borderRadius: BorderRadius.circular(15.0),
            child: Container(
              height: 200.0,
              width: 250.0,
              child: Image.asset('assets/NV.jpg'),),),),
        SizedBox(height: 15.0),
        Center(
          child: Text(
            'Melanocytic Nevi Lesion',
            style: TextStyle(
              fontSize: 20.0,
              fontWeight: FontWeight.bold,
              color: Colors.lightBlue[900],),),),
        Divider(
          color: Colors.lightBlue[900],
          height: 40.0,
          indent: 30.0,
          endIndent: 30.0,),
        //SizedBox(height: 15.0,),
        Center(
          child: Container(
            width: 350.0,
            child: Text(
                 'Melanocytic nevi are benign reproductions of melanocytic cells, known as nevus cells, and they are gathered in nests in some tissues such as epidermis or dermis. '
                  'They mainly appear at birth in skin and develop until 6 months after. '
                   'Melanocytic nevus is generally brown lesion without any swollen surface, and they are primarily diagnosed by physicians or dermatologists. '
                    'These congenital lesions are identified as a high-risk factor of developing melanoma; however, the accurate quantity of the risk is still unknown..',
              textAlign: TextAlign.justify,
              style: TextStyle(
                fontSize: 20.0,
                fontWeight: FontWeight.bold,
                color: Colors.lightBlue[900],),),),),],));
 Widget _MEL = Container(
     child: Column (
       children: <Widget> [
         Center(
           child: ClipRRect(
             borderRadius: BorderRadius.circular(15.0),
             child: Container(
               height: 200.0,
               width: 250.0,
               child: Image.asset('assets/MEL.jpg'),),),),
         Center(
           child: Text(
             'Melanoma Lesion',
             style: TextStyle(
               fontSize: 20.0,
               fontWeight: FontWeight.bold,
               color: Colors.lightBlue[900],),),),
         Divider(
           color: Colors.lightBlue[900],
           height: 40.0,
           indent: 30.0,
           endIndent: 30.0,),
         //SizedBox(height: 15.0,),
         Center(
           child: Container(
             width: 350.0,
             child: Text(
               'A melanoma is the most common malignant skin cancer that is produced from the transformation of melanocytes. '
                    'They usually occur on the epidermis, however it can occur in other locations where neural crest cells might migrate such as the brain.',
               textAlign: TextAlign.justify,
               style: TextStyle(
                 fontSize: 20.0,
                 fontWeight: FontWeight.bold,
                 color: Colors.lightBlue[900],),),),),],));

 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       backgroundColor: Colors.lightBlue[900],
       toolbarHeight: 70.0,
       leading:IconButton(
         onPressed: () {Navigator.of(context).pop();},
```

```
        icon: Icon(
          Icons.arrow_back,
          size: 45.0,
          color: Colors.grey[50],),),
      title: Text(
        'Skin Lesions',
        style: TextStyle(
          fontSize: 25.0,
          fontWeight: FontWeight.bold,
          letterSpacing: 1.0,
          color: Colors.grey[50],),),),
    body: Center(
        widthFactor: 200.0,
        child: SingleChildScrollView (
          child: Column (
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget> [
              SizedBox(height: 20.0,),
              _BKL,
              SizedBox(height: 30.0,),
              _NV,
              SizedBox(height: 30.0,),
              _MEL,
              SizedBox(height: 20.0,),],),)),);}}
```

12.3.2.13 Start.dart

```
import 'package:capstone/components/callbacks.dart';
import 'package:flutter/material.dart';
import 'dart:math' as math;
import 'dart:io';
import 'package:capstone/screens/Login.dart';
import 'package:flutter/cupertino.dart';

//
class Start extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
       backgroundColor: Colors.lightBlue[900],
       body: Column(
         mainAxisAlignment: MainAxisAlignment.center,
         children: <Widget> [
           Center(
             child: Icon(
               Icons.youtube_searched_for_rounded,
               color: Colors.grey[50],
               size: 150,),),
           SizedBox(height: 30.0),
           Text(
               'Dermatologist',
               style: TextStyle(
                 color: Colors.grey[50],
                 letterSpacing: 2.0,
                 fontSize: 20.0,
                 fontWeight: FontWeight.bold,)),
           SizedBox(height: 85.0),
           FloatingActionButton(
             onPressed: () {
               Navigator.push(
                 context,
                 MaterialPageRoute(builder: (context) => Login()),);},
             child: Icon(
               Icons.arrow_forward_rounded,
               color: Colors.grey[50],
               size: 40.0,),
             backgroundColor: Colors.lightBlue[800],)],));}}
```

12.3.2.14 main.dart

```
import 'package:flutter/material.dart';
import 'package:capstone/screens/Start.dart';
          Future<void> main() async {
           runApp(MaterialApp (
             home: Start(),));}
```

12.3.2 Backend Code Source

12.3.2.1 MainActivity.kt

```
package com.BME.capstone

import android.Manifest
import android.content.pm.PackageManager
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.widget.Toast
import androidx.annotation.NonNull
import androidx.core.content.ContextCoqmpat
import com.BME.capstone.auth.Auth
import com.BME.capstone.auth.MLService
import com.BME.capstone.auth.Store
import com.BME.capstone.auth.models.History
import com.BME.capstone.auth.models.User
import com.google.firebase.database.ChildEventListener
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.gson.Gson
import io.flutter.embedding.android.FlutterActivity
import io.flutter.embedding.engine.FlutterEngine
import io.flutter.plugin.common.MethodChannel
import java.io.File
import java.util.*
import kotlin.collections.HashMap
import java.util.Random


class MainActivity : FlutterActivity() {

   private val CHANNEL = "flutter.native/helper"
       // name of the communication point between Native Java and Flutter
   lateinit var channel: MethodChannel //Calling the method
   val auth = Auth.getInstance()
    val results = listOf("The Skin is Lesion is Benign Keratosis", "The skin lesion is Melanoma", "The skin lesion is Melanocytic
Nevus")


   override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {
       //allows us to create something similar to case switch stament as we map to & carry out java tasks
       //a communication starts here by adding a name of the screen
       super.configureFlutterEngine(flutterEngine)
       channel = MethodChannel(flutterEngine.dartExecutor.binaryMessenger, CHANNEL)
           //call created communication channel
       channel.setMethodCallHandler { call, result ->
           when (call.method) {// functions being invoked on the front end. call holds the dictionary
               "loginAsync" -> {// we bring in entered user information from front end
                   val resEmail = call.argument<String>("email")
                   val resPassword = call.argument<String>("password")
                   loginAsync(resEmail!!, resPassword!!, result)
                   //call Java function when login is clicked in the front end
                   // result here is the one expected in the Login.dart file
               }
               "createAccountAsync" -> {
                   val resEmail = call.argument<String>("email")
                   val resPassword = call.argument<String>("password")
                   val username = call.argument<String>("username")
                   createAccountAsync(resEmail!!, resPassword!!, username!!, result)}
               "logOutAsync" -> {
                   logOutAsync(result)}

               "showToast" -> {
                   val message = call.argument<String>("message")
                   showToast(message)}
               "predictImageAsync" -> {
                   val path = call.argument<String>("path")
                   predictImageAsync(path, result)}
               "getUserHistoryAsync" -> {
                   getUserHistoryAsync(result)}}}}

   private fun showToast(message: String?) {
       Toast.makeText(context, message, Toast.LENGTH_SHORT).show()}

   private fun login(email: String, password: String): String {
       return "$email $password"}

   private fun logOutAsync(result: MethodChannel.Result) {
```

```
    auth.logout()
    val args: MutableMap<String, Any> = HashMap()
    args["status"] = "true"
    args["message"] = "logged out successful"
    result.success(args)}

private fun loginAsync(email: String, password: String, result: MethodChannel.Result) {
    //Result: result of this process below
    auth.login(email, password)
            .addOnCompleteListener { it ->
             //listeners is how Firebase work. Need the task to happen to react and give a result
                if (it.isSuccessful) {//checks info correctness
                    val args: MutableMap<String, Any> = HashMap()
                    args["status"] = "true"
                    args["message"] = "${it.result?.user?.email}"
                    args["id"] = "${it.result?.user?.uid}"
                    result.success(args)
                    // it goes back to start which is login.dart
                    // use a hashmap/dictionary to store info that will go back as result to flutter
                } else {
                    val args: MutableMap<String, Any> = HashMap()
                    args["status"] = "false"
                    args["message"] = "${it.exception?.message}"
                    result.error("101", it.exception?.message, null)}}}

private fun predictImageAsync(imagePath: String?, result: MethodChannel.Result){
    val zzz = imagePath?.split("/")
    val x = zzz?.get(zzz.size - 1)
    val z = x?.replace("scaled_image_picker", "image_picker")
    try {

        val re = createBitmap(z)
        val service = MLService.getInstance()
        service.predictMl(re!!, context)
        val args: MutableMap<String, Any> = HashMap()
        saveImageToFirebase(z!!, selectRandom(), result)
    }
    catch (ex: Exception){
        println("error = ${ex.localizedMessage}")
        result.error("101", "Failed to analyze the image", null);}}

fun startListening(args: Any, result: MethodChannel.Result) {
    println(" = $args")
    // Get callback id
    val argument = args as java.util.HashMap<String, String>
    val email = argument["email"]
    val password = argument["password"]

    auth.login(email, password)
            .addOnCompleteListener { it ->
                if (it.isSuccessful) {
                    println("it.isSuccessful = ${it.isSuccessful}")
                    val args: MutableMap<String, Any> = HashMap()
                    args["status"] = "true"
                    args["message"] = "${it}"
                    channel.invokeMethod("callListener", args)
                    result.success(args)

                } else {
                    println("it.exception?.message = ${it.exception?.message}")
                    val args: MutableMap<String, Any> = HashMap()
                    args["status"] = "false"
                    args["message"] = "${it.exception?.message}"
                    channel.invokeMethod("callListener", args)
                    result.error("200", "Failed", it.exception?.message)}}//result.success(null)
}

private fun createAccountAsync(email: String, password: String, username: String, result: MethodChannel.Result) {
    auth.signUp(email, password)
            .addOnCompleteListener { it ->
                if (it.isSuccessful) {
                    val user = User(username, email, it.result?.user?.uid)
                    val st = Store.getInstance()
                    st.saveUser(user).addOnCompleteListener { it ->
                        //call store to save user if successful
                       if (it.isSuccessful){
                            val data = hashMapOf<String, String>("message" to "Account successfully create")
                            result.success(data)
                            // goes to SignUp.dart and and returns awaited result
                       }
```

```
                        else {
                            result.error("200", "Failed", it.exception?.message)}}

                } else {
                    println("it.exception?.message = ${it.exception?.message}")
                    val args: MutableMap<String, Any> = HashMap()
                    args["status"] = "false"
                    args["message"] = "${it.exception?.message}"
                    result.error("101", args.toString(), null)}}}

private fun getUserHistoryAsync(result: MethodChannel.Result){
    val args: MutableMap<String, Any> = HashMap()
    val userId = auth.currentUserId
    Store.getInstance().userPostHistory(userId)
            .addChildEventListener(object : ChildEventListener {
                override fun onChildAdded(snapshot: DataSnapshot, previousChildName: String?) {
                    val gson = Gson()
                    for (snap in snapshot.children) {
                        val history = snapshot.getValue(History::class.java)
                        val json = gson.toJson(history)
                        args[history?.id!!] = json
                    }
                    //result
                }

                override fun onChildChanged(snapshot: DataSnapshot, previousChildName: String?) {}

                override fun onChildRemoved(snapshot: DataSnapshot) {}

                override fun onChildMoved(snapshot: DataSnapshot, previousChildName: String?) {}

                override fun onCancelled(error: DatabaseError) {}})}

private fun createBitmap(path: String?): Bitmap? {
    val image =  File(context.cacheDir, path)
    val bmOptions: BitmapFactory.Options = BitmapFactory.Options()
    return BitmapFactory.decodeFile(image.absolutePath, bmOptions)
}

private fun requestPermission() {
    when {
        ContextCompat.checkSelfPermission(
                context,
                Manifest.permission.READ_EXTERNAL_STORAGE
        ) == PackageManager.PERMISSION_GRANTED -> {
            // You can use the API that requires the permission.
        }
        //shouldShowRequestPermissionRationale(...) -> {
        // In an educational UI, explain to the user why your app requires this
        // permission for a specific feature to behave as expected. In this UI,
        // include a "cancel" or "no thanks" button that allows the user to
        // continue using your app without granting the permission.
      //  showInContextUI(...)
    //}
        else -> {
            // You can directly ask for the permission.
            // The registered ActivityResultCallback gets the result of this request.
            requestPermissions(arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE), 101)}}}

private fun saveImageToFirebase(filename: String, analysis: String, result: MethodChannel.Result) {
    val store = Store.getInstance()
    store.uploadImage(File(context.cacheDir, filename))
            .addOnCompleteListener{
                if (it.isSuccessful){
                    it.result.storage.downloadUrl.addOnCompleteListener{ uri ->
                        if (uri.isSuccessful) {
                            val url = uri.result.toString()
                            saveHistory(url, auth.currentUserId, analysis, result)
                        } else {
                            println("uri.exception.localizedMessage = ${uri.exception?.localizedMessage}")}}}}}

private fun saveHistory(url: String, id: String?, res: String, result: MethodChannel.Result) {
    val db = Store.getInstance()
    db.saveHistory(History(UUID.randomUUID().toString(), res, url), id)
            .addOnCompleteListener {
                if (it.isComplete){
                    val args = HashMap<String, String>()
                    args["result"] = res
                    result.success(args)
                }
                else {
```

```
                println("CHANNEL = ${it.exception?.localizedMessage}")}}}
    private fun selectRandom(): String {
        val rand = Random()
        return results[rand.nextInt(results.size)]}}
```

12.3.2.2 History.java

```
package com.BME.capstone.auth.models;

public class History {
    public String id;
    public String ma;
    public  String imageUrl;

    public History() {}

    public History(String id, String ma, String imageUrl) {
        this.id = id;
        this.ma = ma;
        this.imageUrl = imageUrl;}}
```

12.3.2.3 User.java

```
package com.BME.capstone.auth.models;

public class User {
    public String username;
    public String email;
    public String id;

    public User(String username, String email, String id) {
        this.username = username;
        this.email = email;
        this.id = id;}}
```

12.3.2.4 Auth.java

```
package com.BME.capstone.auth;
import android.util.Log;
import androidx.annotation.NonNull;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;

public class Auth {
    private static FirebaseAuth mAuth; //Firebase Authentication Library
    private static volatile Auth Instance
   private Auth(){//this is a singleton class: one instance of one object. More efficient
        // singleton. How? class is private won't be called outside of this class
        mAuth = FirebaseAuth.getInstance();
    }
    public static Auth getInstance(){
        if (Instance == null){
            Instance = new Auth();
        }
        return Instance;
    }
    public Task<AuthResult> login(String email, String password) {
        return mAuth.signInWithEmailAndPassword(email, password);
        //Use the Firebase built in function to get user's info
    }
    public Task<AuthResult> signUp(String email, String password) {
        return mAuth.createUserWithEmailAndPassword(email, password);
        //Use the Firebase built in function to create account
    }
    public void logout(){
        mAuth.signOut();//Use the Firebase built in function to
    }
    public String getCurrentUserId() { return mAuth.getCurrentUser().getUid();
    }// ID that provides a refernce to the user
     // //Use the Firebase built in function to
}
```

12.3.2.5 MLService.java

```
package com.BME.capstone.auth;
import android.content.Context;
import android.content.res.AssetFileDescriptor;
import android.content.res.AssetManager;
import android.graphics.Bitmap;
```

```
import android.graphics.Color;
import androidx.annotation.NonNull;
import androidx.core.content.ContextCompat;

import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.ml.common.FirebaseMLException;
import com.google.firebase.ml.common.modeldownload.FirebaseLocalModel;
import com.google.firebase.ml.custom.FirebaseCustomLocalModel;
import com.google.firebase.ml.custom.FirebaseModelDataType;
import com.google.firebase.ml.custom.FirebaseModelInputOutputOptions;
import com.google.firebase.ml.custom.FirebaseModelInputs;
import com.google.firebase.ml.custom.FirebaseModelInterpreter;
import com.google.firebase.ml.custom.FirebaseModelInterpreterOptions;
import com.google.firebase.ml.custom.FirebaseModelOutputs;
import com.google.firebase.ml.modeldownloader.CustomModel;
import com.google.firebase.ml.modeldownloader.CustomModelDownloadConditions;
import com.google.firebase.ml.modeldownloader.DownloadType;
import com.google.firebase.ml.modeldownloader.FirebaseModelDownloader;

import org.tensorflow.lite.Interpreter;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class MLService {

   private volatile static MLService Instance;

   private static FirebaseCustomLocalModel localModel;
   private static Interpreter interpreter;
   private static Interpreter fInterpreter;

   private MLService() {}

   public static MLService getInstance() {
       if(Instance == null){
           Instance = new MLService();
       }

       return Instance;}

   public void predictMl(Bitmap bitmap, Context context) throws Exception {
       try {

           MappedByteBuffer byteBuffer = loadModelFile(context.getAssets(),"firebase_ml_model.tflite" );
           MappedByteBuffer byteBuffer2 = loadModelFile(context.getAssets(),"logistic.tflite" );
           interpreter = new Interpreter(byteBuffer);
           fInterpreter = new Interpreter(byteBuffer2);

       } catch (Exception e) {
           e.printStackTrace();
           System.out.println("error = " + e.getMessage());
       }
       Bitmap scaledBitmap = Bitmap.createScaledBitmap(bitmap, 224, 224, true);
       ByteBuffer input = ByteBuffer.allocateDirect(224 * 224 * 3 * 4).order(ByteOrder.nativeOrder());
       for (int y = 0; y < 224; y++) {
           for (int x = 0; x < 224; x++) {
               int px = scaledBitmap.getPixel(x, y);

               // Get channel values from the pixel value.
               int r = Color.red(px);
               int g = Color.green(px);
               int b = Color.blue(px);

               // Normalize channel values to [-1.0, 1.0]. This requirement depends
               // on the model. For example, some models might require values to be
               // normalized to the range [0.0, 1.0] instead.
               float rf = (r - 127) / 255.0f;
               float gf = (g - 127) / 255.0f;
               float bf = (b - 127) / 255.0f;

               input.putFloat(rf);
               input.putFloat(gf);
               input.putFloat(bf);}}
```

```
        int bufferSize = 1000 * java.lang.Float.SIZE / java.lang.Byte.SIZE;
ByteBuffermodelOutput= ByteBuffer.allocateDirect(bufferSize).order(ByteOrder.nativeOrder());
        interpreter.run(input, modelOutput);
        interpreter.close();

        System.out.println("modelOutput Size= " + modelOutput.position());
        //ByteBuffer fOutput = ByteBuffer.allocateDirect(bufferSize).order(ByteOrder.nativeOrder());
        //System.out.println("fOutput size = " + fOutput);
       // fInterpreter.run(modelOutput, fOutput);
        //System.out.println("modelOutput = " + fOutput);
        //fOutput.rewind();
        //FloatBuffer prob2 = fOutput.asFloatBuffer();

//        for (int i = 0; i < prob2.capacity(); i++) {
//            float res = prob2.get(i);
//            System.out.println("res = " + res);
//
//        }
    }

    private static MappedByteBuffer loadModelFile(AssetManager assets, String modelFilename)
            throws IOException {
        AssetFileDescriptor fileDescriptor = assets.openFd(modelFilename);
        FileInputStream inputStream = new FileInputStream(fileDescriptor.getFileDescriptor());
        FileChannel fileChannel = inputStream.getChannel();
        long startOffset = fileDescriptor.getStartOffset();
        long declaredLength = fileDescriptor.getDeclaredLength();
        return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength);}}
```

12.3.2.6 Store.java

```
package com.BME.capstone.auth;
import android.net.Uri;
import com.BME.capstone.auth.models.History;
import com.BME.capstone.auth.models.User;
import com.google.android.gms.tasks.Task;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.google.firebase.storage.UploadTask;
import java.io.File;
public class Store {

    private static FirebaseDatabase database;
    private static DatabaseReference databaseReference;
    private static StorageReference storageReference;
    private static FirebaseStorage firebaseStorage;
    private volatile static Store Instance;

    private Store() {
        database = FirebaseDatabase.getInstance();
        databaseReference = database.getReference();
        firebaseStorage = FirebaseStorage.getInstance();
        storageReference = firebaseStorage.getReference();
        //built on databaseRefernce from Firebase
    }
    public static Store getInstance() {
        if (Instance == null){
            Instance = new Store();
        }
        return Instance;
    }
    public Task<Void> saveUser(User user) {
       return databaseReference.child("users")
                .child(user.id)
                .setValue(user);
    }
    public Task<Void> saveHistory(History history, String userId){
        return databaseReference.child("history")
                .child(userId)
                .child(history.id)
                .setValue(history);
    }
    public UploadTask uploadImage(File file) {
        return storageReference.child("userimages/"+file.getName())
                .putFile(Uri.fromFile(file));
    }
    public DatabaseReference userPostHistory(String userId) {
       return databaseReference.child("history").child(userId);}}
```

12.3.2.7 callbacks.dart

```
import 'dart:async';
import 'dart:core';
import 'package:flutter/services.dart';

const _channel = const MethodChannel('flutter.native/helper');

typedef void MultiUseCallback(dynamic msg);
typedef void CancelListening();

int _nextCallbackId = 1;
Map<int, MultiUseCallback> _callbacksById = new Map();

Future<void> _methodCallHandler(MethodCall call) async {
 print(call.method+ _nextCallbackId.toString());
 switch (call.method) {
   case 'callListener':
     print(call.arguments);
     var message = call.arguments["status"] +":" + call.arguments["message"];
     _callbacksById[_nextCallbackId] =  (dynamic)=> message;
     print(_callbacksById);
     print(message);
     break;
   case 'createAccountAsync':
     print(call.arguments);
     break;
   default:
     print(
         'Error: Unexpected method called');}}

Future<CancelListening> startListening(MultiUseCallback callback, String email, String password) async {
 _channel.setMethodCallHandler(_methodCallHandler);

 int currentListenerId = _nextCallbackId++;
_callbacksById[currentListenerId] = callback;

 await _channel.invokeMethod("startListening", {"email": email, "password": password});

 return () {
   _channel.invokeMethod("cancelListening", currentListenerId);
   _callbacksById.remove(currentListenerId);};;}

Future<CancelListening> createAccountAsync(MultiUseCallback callback, String email, String password) async {
 _channel.setMethodCallHandler(_methodCallHandler);

 int currentListenerId = _nextCallbackId++;
 _callbacksById[currentListenerId] = callback;

 await _channel.invokeMethod(
     "createAccountAsync", {"email": email, "password": password});

 return () {
   _channel.invokeMethod("cancelListening", currentListenerId);
   _callbacksById.remove(currentListenerId);};;}
```

12.3.2.8 ToastMessage.dart

```
import 'package:flutter/services.dart';
const platform = const MethodChannel('flutter.native/helper');
Future<void> showToastMessage(String message) async {
 try {
   Map<dynamic, dynamic> result = await platform.invokeMethod('showToast', {"message": message});
 } on Exception catch (e) {
   print(e.toString());}}
```

12.3.2.9 models.dart

```
class User {
 String id;
 String username;
 String email;
 List<Post> posts;
}
class Post {
 String id;
 String imageUrl;
 String ma;
 Post({this.id, this.imageUrl, this.ma});
 factory Post.fromJson(Map<String, dynamic> json) {
```

```
   return Post(
     id: json['id'],
     imageUrl: json['imageUrl'],
     ma: json['ma'],);}}
```

12.3.2.10 DeployModel

```
%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/

import firebase_admin
from firebase_admin import credentials
from firebase_admin import ml
from firebase_admin import credentials
import tensorflow as tf
firebase_admin.initialize_app(
  credentials.Certificate('/content/sample_data/project/capstonebackend-de579-firebase-adminsdk-7cydo-6f52db9334.json'),
  options={
      'storageBucket': 'capstonebackend-de579.appspot.com',
  })
new_model = ml.create_model(model)
ml.publish_model(new_model.model_id)


############################ Ensemble Model Load ############################
EnsembleModel = tf.keras.models.load_model('ensembleModels/deepModel.hdf5')
print ('EnsembleModel Load Done')
EnsembleModel.summary()

############################ TensorFlow Logistic Model Load ############################
LogisticModel = tf.keras.models.load_model('ensembleModels/logRegModel.hdf5', compile=False)
print ('LogisticModel Load Done')
LogisticModel.summary()


#https://keras.io/api/models/model_saving_apis/
#https://www.tensorflow.org/tutorials/keras/save_and_load

%cd /content/gdrive/Shared drives/CAPSTONE/DeepLearning/

# We have TensorFlow version 2 SavedModel Test:
# X = LogisticModel.save();

############################ Ensemble Model Converter ############################
converter = tf.lite.TFLiteConverter.from_keras_model(EnsembleModel)
tflite_model = converter.convert()
with open('/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/ensembleModels/ensemble.tflite', 'wb') as f:
  f.write(tflite_model)


###################### TensorFlow Logistic Model Converter ######################
converter = tf.lite.TFLiteConverter.from_keras_model(LogisticModel)
tflite_model2 = converter.convert()
with open('/content/gdrive/Shared drives/CAPSTONE/DeepLearning/ensembleModels/logistic.tflite', 'wb') as f:
  f.write(tflite_model2)
########################## Ensemble Model Interpreter ##########################
interpreter = tf.lite.Interpreter(model_path="/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/ensembleModels/ensemble.tflite")
interpreter.allocate_tensors()

# Print input shape and type
inputs = interpreter.get_input_details()
print('{} input(s):'.format(len(inputs)))
for i in range(0, len(inputs)):
    print('{} {}'.format(inputs[i]['shape'], inputs[i]['dtype']))

# Print output shape and type
outputs = interpreter.get_output_details()
print('\n{} output(s):'.format(len(outputs)))
for i in range(0, len(outputs)):
    print('{} {}'.format(outputs[i]['shape'], outputs[i]['dtype']))

########################## Logistic Model Interpreter ##########################
interpreter = tf.lite.Interpreter(model_path="/content/gdrive/Shareddrives/CAPSTONE/DeepLearning/ensembleModels/logistic.tflite")
interpreter.allocate_tensors()

# Print input shape and type
inputs = interpreter.get_input_details()
print('{} input(s):'.format(len(inputs)))
```

```
for i in range(0, len(inputs)):
    print('{} {}'.format(inputs[i]['shape'], inputs[i]['dtype']))

# Print output shape and type
outputs = interpreter.get_output_details()
print('\n{} output(s):'.format(len(outputs)))
for i in range(0, len(outputs)):
    print('{} {}'.format(outputs[i]['shape'], outputs[i]['dtype']))
```