MELANOMA DETECTION THROUGH A THREE-STAGE MOBILE APPLICATION

A PROJECT REPORT

Submitted by

ARYAN SINHA [Reg No:RA2011003010066]

KAVYA NAIR [Reg No: RA2011003010067]

ARADHYA GOYAL [Reg No: RA2011003010065]

ADITI [Reg No: RA2011003010004]

Under the Guidance of

Dr. S. SIVAKUMAR

Assistant Professor, Department of Computing Technologies

in partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 18CSP109L project report titled **"MELANOMA DETECTION THROUGH A THREE-STAGE MOBILE APPLICATION"** is the bonafide work of **ARYAN SINHA [Reg No:RA2011003010066], KAVYA R. NAIR [Reg No:RA2011003010067], ARADHYA GOYAL [Reg No: RA2011003010065]** and **ADITI [Reg No:RA2011003010004]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

S. Sivakumar

Dr. S. SIVAKUMAR 29/04/24
SUPERVISOR
Assistant Professor
Department of Computing Technologies

Dr. B. MURUGANANTHAM
PANEL HEAD
Associate Professor
Department of Computing Technologies

Department of Computing Technology Kattankulathur 603 203 SRM IST

M. Pushpalatha

Dr. M. PUSHPALATHA
HEAD OF THE DEPARTMENT
Department of Computing Technologies

Internal Examiner

External Examiner

Department of Computing Technologies
SRM Institute of Science and Technology

Own Work Declaration Form

Degree/Course : B.Tech in Computer Science and Engineering

Student Names : ARYAN SINHA, KAVYA NAIR,
ARADHYA GOYAL, ADITI

Registration Number : RA2011003010066, RA2011003010067
RA2011003010065, RA2011003010004

Title of Work : Melanoma Detection Through A Three-Stage Mobile Application

I/We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web,etc.)
- Given the sources of all pictures, data etc that are not my own.
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course hand book / University website

I understand that any false claim for this work will be penalised in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
Student 1 Signature: Aryan S
Student 2 Signature:
Student 3 Signature: ARADHYA
Student 4 Signature: Ranya
Date: 29-4-24
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, **Dr. T.V. Gopal,** SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Pushpalatha,** Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, **Dr . S. Godfrey Winster**, **Dr. C. Pretty Diana Cyril, Dr. P. Murali, Dr. Selvin Paul Peter and Dr. Padmapriya**, Panel Head, **Dr. B. Murugananthan**, Associate Professor and Panel Members, **Dr. A. M. J. Muthu Kumaran** Assistant Professor, **Dr. S. Sivakumar** Assistant Professor, and **Dr. J. NithyaSri** Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. Sivakumar S.**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. S. Sivakumar**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of the Computing Technologies Department, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

ARYAN SINHA [Reg. No: RA2011003010066]

KAVYA NAIR [Reg. No: RA2011003010067]

ARADHYA GOYAL [Reg. No: RA2011003010065]

ADITI [Reg. No: RA2011003010004]

ABSTRACT

The project represents an integration of deep learning and computer vision techniques aimed at enhancing the detection of skin cancer. This initiative adopts a structured three-stage process, leveraging advanced computational models to improve the accuracy and precision in the classification of skin cancer. At the core of the project's methodology is the application of the Region-based Convolutional Neural Network (RCNN) model, which is trained extensively on a diverse collection of skin images. This training enables the RCNN to identify and analyze Regions of Interest (ROIs) potentially indicative of melanoma, refining its capability to recognize and categorize the nuanced visual characteristics of cancerous lesions. Further refining the detection process, the project incorporates the U-Net for advanced segmentation. This method surpasses traditional bounding box techniques by executing detailed extraction of melanoma contours from the ROIs, enhancing the accuracy of lesion detection. The final stage of the project employs MobileNetV2, an optimized convolutional neural network designed for efficient and precise image classification. Trained on a broad spectrum of skin lesion images, MobileNetV2 categorizes the segmented regions, facilitating the assessment of melanoma presence and aiding in prompt and informed medical decision-making. Overall, the project aims to advance dermatological care by employing innovative computational approaches to improve the detection and classification of skin cancer, contributing to better patient outcomes and the evolution of proactive healthcare methodologies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RCNN	Region-based Convolutional Neural Networks
ROI	Regions of Interest
ML	Machine Learning
RGB	Red Green Blue (colorspace)
IoU	Intersection over Union
MSE	Mean Squared Error

CHAPTER 1
INTRODUCTION

The project is an exploratory initiative in the field of dermatological diagnostics, focusing on the potential enhancements in skin cancer detection through a novel three-step process. This project investigates whether this multi-stage approach offers significant improvements over the direct application of existing technologies like MobileNetV2. The primary aim is not to provide definitive diagnoses but to assess the likelihood of skin cancer presence more accurately. At the outset, it employs the Region-based Convolutional Neural Network (RCNN) model. This stage involves extensive training on diverse skin images, enabling the RCNN to identify Regions of Interest (ROIs) that could indicate melanoma. The effectiveness of RCNN in this context lies in its ability to detect subtle visual markers associated with cancerous lesions, thereby enhancing the precision of identifying areas that warrant further examination.

The second phase of the project incorporates the U-Net for advanced segmentation. This technique moves beyond basic bounding box approaches, focusing on accurately delineating the contours and shapes of suspected melanomas within the ROIs. The goal here is to determine whether this added level of segmentation detail contributes to a more nuanced understanding of the detected areas, potentially leading to more reliable assessments.

In the final stage, the project utilizes MobileNetV2, a convolutional neural network known for its efficiency in image classification. Here, the project examines if the preceding steps of RCNN and U-Net segmentation provide meaningful enhancement to the classification capabilities of MobileNetV2. The focus is on whether this staged approach aids in a more informed evaluation of the likelihood of melanoma presence, as opposed to relying solely on MobileNetV2's classification.

This project, therefore, is positioned as an investigative tool in the realm of skin cancer detection. Its purpose is to explore the effectiveness of a layered analytical process in improving the accuracy of preliminary skin cancer assessments. This project aims to contribute to the ongoing research in dermatological diagnostics by providing insights into the potential benefits and limitations of integrating multiple deep learning models in a sequential manner.

Building on the objective of enhancing diagnostic processes, the project delves into the realm of computational dermatology with a focus on developing a system that can be integrated into existing medical workflows. The overarching goal is to provide a supplementary tool for dermatologists, one that combines human expertise with artificial intelligence to improve the early detection rates of skin cancer. By doing so, the project aspires to bridge the gap between rapidly advancing technological capabilities and the practical needs of clinical practice. It seeks to construct a more refined diagnostic aid that can process and analyze dermatological imagery with increased accuracy, potentially reducing the rate of overlooked melanoma cases and thus, the associated morbidity.

Moreover, this initiative takes into account the urgent need for accessible and efficient medical diagnostic tools, particularly in under-resourced settings where specialist care may be sparse. By introducing a system that requires minimal hardware yet offers high diagnostic accuracy, the project aims to democratize medical technology, making it available to a wider audience. The intent is to provide a means by which individuals, regardless of their location, can benefit from advances in medical imaging and analysis. In doing so, the project underlines the role of technological inclusivity in the broader context of global health, positioning itself as a stepping stone towards more equitable healthcare solutions.

1.2 Proposed Three-Stage Approach

Our project aims to develop an application focused on early detection of skin cancer, enhancing the accuracy and efficiency of diagnosis through a structured three-step approach. The initial phase involves the use of a Region-based Convolutional Neural Network (RCNN) to identify Regions of Interest (ROIs) within skin images, facilitating focused analysis. Subsequently, U-Net is employed for segmentation, providing precise delineation of cancerous lesions beyond the capabilities of traditional bounding boxes. This detailed segmentation is vital for accurate identification and effective diagnosis. The final step involves applying MobileNetV2 for the classification of segmented regions, enabling accurate differentiation between benign and malignant cases.

The objective of this approach is to improve the diagnostic process by making it go through more rigorous ML models.

Additionally, the project aims to assess the comparative effectiveness of this integrated three-step process against the traditional method of direct image classification. By leveraging advanced deep learning and image processing techniques, the application is designed to support healthcare professionals and individuals in early skin cancer detection, potentially facilitating timely intervention and treatment.

Furthermore, the project is committed to addressing the challenges posed by the variability in skin lesions by incorporating diverse datasets that span a wide range of skin types and cancer stages. This inclusivity in data ensures that the developed models are robust and can generalize well across different populations, making the application valuable in a variety of clinical and personal settings. The design and testing of the application also aims to ensure user-friendliness and accessibility, so that it can be easily adopted by users with minimal technical expertise. Ultimately, this project endeavors to contribute a novel tool to the field of digital health, one that harnesses the power of AI to support the early detection and prevention of skin cancer.

1.3 Metrics Measured

To rigorously evaluate the performance of the proposed melanoma detection models, several metrics were utilized, each tailored to the specific tasks and stages of the model pipeline. The metrics are essential for assessing various aspects of model accuracy and robustness in medical image analysis.

RCNN

- Accuracy: This metric measures the proportion of true results (both true positives and true negatives) among the total number of cases examined. It is calculated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

- Intersection over Union (IoU): This metric is used for evaluating the overlap between the predicted and actual bounding boxes around the lesions. It is defined as the size of the intersection divided by the size of the union of the predicted set and the truth set:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

UNET

- Dice Coefficient: This metric assesses the similarity between the predicted segmentation and the ground truth, making it particularly useful for evaluating segmentation tasks. It is defined as twice the area of overlap between the two sets divided by the total number of pixels in both sets:

$$\text{Dice Coefficient} = \frac{2 \times |\text{Prediction} \cap \text{Ground Truth}|}{|\text{Prediction}| + |\text{Ground Truth}|}$$

MobileNetV2, Integrated Model, and TFLite

- Precision: This metric evaluates the accuracy of the positive predictions and is defined as the ratio of true positives to the sum of true and false positives:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- Recall: This measures the ability of the model to detect all relevant instances, calculated as the ratio of true positives to the sum of true positives and false negatives:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- F1 Score: The F1 Score is the harmonic mean of precision and recall, providing a balance between the two when comparing model performance:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

1.4 Software Used

The development of the machine learning models for melanoma detection was carried out using Microsoft Visual Studio Code, the 2020 edition, which provided a robust and versatile integrated development environment. We leveraged the Python 3.10.0 interpreter, opting for a downgrade specifically to ensure compatibility with CUDA-accelerated TensorFlow, thereby harnessing the power of GPU processing to expedite model training and evaluation.

CHAPTER 2
LITERATURE SURVEY

2.1 Motivation

Traditional methods for the diagnosis of melanoma rely mainly on visual inspection by a dermatologist, often supplemented by dermoscopy and histopathological analysis. Although these methods are effective, they can be subjective and require specialized knowledge, leading to diagnostic variability [1]. Automated systems for melanoma diagnosis have emerged as a promising alternative that uses advances in computer vision and machine learning to improve accuracy and efficiency[2].

A] Computer vision and machine learning in melanoma diagnosis:
In recent years, significant progress has been made in the application of computer vision and machine learning techniques for melanoma diagnosis. These techniques enable automatic analysis of images of skin lesions and facilitate early identification of suspicious lesions [3]. In particular, deep learning algorithms have shown remarkable performance in image classification tasks, exceeding human-level performance in certain fields [4]. Convolutional Neural Networks (CNN) have emerged as a leading architecture for melanoma diagnosis because they can learn hierarchical features directly from image data [5].

B] Three-step approach in melanoma diagnosis:
A three-step approach for melanoma detection includes lesion segmentation, feature extraction, and classification, each step contributing to the overall accuracy and reliability of the detection system. The aim of lesion segmentation techniques is to delineate the boundaries of skin lesions from the surrounding healthy tissue and facilitate subsequent analysis [7]. Feature extraction algorithms extract relevant features from segmented lesions and capture key features indicative of malignant tumors [8]. A classification algorithm then uses the extracted features to distinguish between benign and malignant lesions, enabling accurate diagnosis [9].

C] Bharata et al. (2014) proposed two different systems for the diagnosis of melanoma from dermoscopic images. One uses global features and the other uses local features. These systems classify lesions as benign or melanoma.

Mainly, color and texture features are used for feature extraction and classification.[7] The dataset used was PH2 (Hospital Pedro Hispano) and in the end it was found that color features have better results than texture features. Only color and texture features are used here. Global methods may not know whether an object is complex or not.

D] Abuzagaleh et al. (2015) proposed 2 significant parts of a non-invasive continuous computerized skin lesion examination framework for early detection and staging of melanoma. The main segment is a continuous alarm that enables consumers to avoid skin consumption by sunlight; A new equation that calculates the ideal opportunity for skin consumption is presented along these lines.

The next part is the automatic image investigation element, which includes image procurement, hair identification and rejection, lesion division, highlight extraction and classification. The proposed framework uses the PH2 image database[6].

E] Rubegni (2010) proposed a system to classify tumors into two categories based on melanoma thickness: thin melanoma and thick melanoma. Digital dermoscopy analysis uses computerized scanning of digital images and is integrated with software to examine the morphological characteristics of lesions.

Melanoma images were evaluated for 49 different features including their color, structure, texture and integrity. 141 melanoma images were used, the accuracy of which was 86.5%. 97 of 108 melanomas were thin and 25 of 33 melanomas were thick[11].

F] Reshma and Shan (2017) proposed a system based on dermoscopy sum score to identify melanoma stage. Images are pre-processed using rgb2gray conversion and median filter to reduce noise. Sobel edge detection algorithm is used. Asymmetry, border irregularities, discoloration and structural differences are the features used to calculate the total dermoscopy score[10].

G] Prasanna et al. proposed a deep learning-based method to achieve high-level skin segmentation and malignant tumor detection by developing a neural network. It accurately identifies the boundaries of the parent disease and builds mobile models using deep neural network transfer learning and optimization to improve the accuracy of predictions.

H] Andre Esteva et al. used 129,450 clinical skin disease images to train a deep convolutional neural network to classify skin lesions. The result is an algorithm that can classify lesions from photographic images similar to those taken with a mobile phone. The accuracy of the system in detecting malignant melanomas and carcinomas was equal to that of trained dermatologists. The authors suggest that the technique could be used outside the clinic as a visual screen for cancer.

I] Sandler, Mark et al. describe a new mobile architecture, MobileNetV2, which improves the performance of mobile models on various tasks and benchmarks, as well as across a spectrum of different model sizes. They also describe efficient ways to apply these mobile models to object detection in a new framework called SSDLite. In addition, they demonstrate how to build mobile semantic segmentation models through a reduced form of DeepLabv3, called Mobile DeepLabv3. The MobileNetV2 architecture is based on an inverse residual structure, where the input and output of the residual block are thin narrow layers, unlike traditional residual models that use extended representations at the input, and MobileNetV2 uses lightweight depth convolutions to filter features in the intermediate extension layer. In addition, they found that it is necessary to remove nonlinearities in narrow layers in order to maintain a representative force. They show that this improves performance and provide the intuition that led to this proposal.

Through this literature review, we provide an overview of the current state of melanoma detection methods and highlight the potential of three-step mobile applications with deep learning algorithms to increase early detection rates and improve patient outcomes.

2.2 Objective

This literature review not only highlights the evolution of melanoma detection techniques but also emphasizes the transformative potential of integrating deep learning approaches within mobile health applications. These advancements suggest a paradigm shift from traditional diagnostic practices towards more automated, accessible, and accurate methods. By leveraging sophisticated computational models such as MobileNetV2, developers can create highly efficient, scalable applications that facilitate skin cancer screening on ubiquitous mobile devices. This shift is particularly significant as it democratizes health care by making cutting-edge diagnostic tools available to a broader audience.

TABLE I. LITERATURE SURVEY SUMMARY

S. No.	Author	Methodology	Limitation
1	Bharata et al. (2014)	Used dermoscopic images for melanoma diagnosis, employing global and local features.	Relied solely on color and texture features, may struggle with complex lesions.
2	Abuzagaleh et al. (2015)	Developed a continuous skin lesion examination framework using image analysis and a sun warning system.	Dependent on image quality, hair detection issues, limited feature scope, may not generalize beyond dataset.
3	Rubegni (2010)	Classified melanomas into thin or thick categories based on morphological characteristics.	Limited dataset size, potential bias, reliance on surface features, may not apply to broader melanoma characteristics.
4	Reshma and Shan (2017)	Introduced a dermoscopy sum score system for melanoma stage identification.	Relied on specific image processing, limited feature set, may not accurately represent various melanoma stages.
5	Prasanna et al (2019)	Used a deep learning-based method to achieve high-level skin segmentation and malignant tumor detection by developing a neural network.	Relied heavily on the transfer learning idea. Lack of standardization in evaluation metrics used to assess the performance of melanoma detection models.
6.	Codella et al (2018)	Provided with a dataset of dermoscopic images and tasked with developing automated detection algorithm	Dataset may not fully represent the diversity of lesions encountered in clinical practice, limiting the generalizability of the results..
7.	Esteva, Andre, et al (2017)	The model was evaluated on a large dataset and compared with dermatologists' performance.	The dataset might not represent the full spectrum of skin types and conditions, potentially limiting the model's generalizability.
8.	Sandler, Mark, et al (2019).	This study incorporates inverted residuals and linear bottlenecks to improve accuracy of deep neural networks on mobile devices	While MobileNetV2 is optimized for mobile applications, its performance may vary depending on the specific task and dataset.

CHAPTER 3
ARCHITECTURE AND ANALYSIS OF MELANOMA DETECTION

3.1 Architecture Diagram

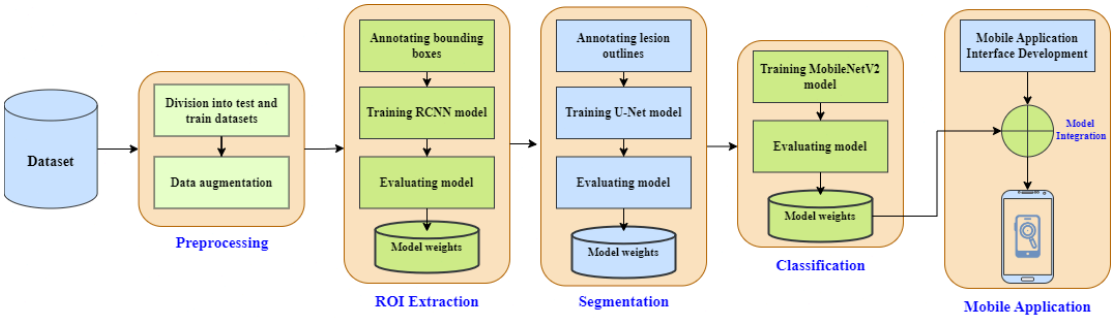


Fig 3.1: Architecture Diagram

3.2 Backend Design

Pre-processing

This phase includes data enhancement and noise removal.

Data augmentation will create new training models by applying various changes to existing data. This technique is especially useful when the dataset is limited because it helps train many without collecting new data. Noise in the data is data that is unwanted or irrelevant to the interpretation of the underlying model. Removing noise from your data is critical to improving the accuracy and reliability of machine learning models. Data augmentation and noise removal are important preliminary processes that help improve the overall data quality and usability of data in the learning process. These enhancements and noise removal techniques are particularly pertinent to medical imaging, where the distinction between relevant signals and background noise can be subtle yet critical for accurate diagnosis. By implementing these steps, the project ensures that the machine learning models are not only trained on more extensive datasets but also on data that closely mimics the variability and complexities encountered in real-world clinical scenarios.

ROI Extraction

ROI (Region of Interest) extraction is the process of identifying and isolating specific regions in images or data that are relevant to the current task. The project's first task is dataset labeling, a critical step that entails preparing a dataset complete with manual annotations. Each data point is marked with annotated bounding boxes around the items of interest, and a corresponding label is assigned. This process ensures that each image is ready for the subsequent phase of machine learning model training.

Following dataset preparation, the next phase involves training the Region-based Convolutional Neural Network (R-CNN) model. The manually annotated bounding boxes serve as a guide for the R-CNN to learn how to independently detect and delineate Regions of Interest (ROIs) within the images. This training equips the model with the capability to assign bounding boxes autonomously, a fundamental step in automating the identification process.

Once the R-CNN model is trained, its performance is rigorously evaluated through a separate testing or validation process. This evaluation is an in-depth analysis involving the calculation of the model’s accuracy and its performance using metrics like the Intersection over Union (IoU). Such metrics provide a quantifiable measure of the model’s ability to predict bounding boxes in alignment with the annotations.

The final step in this stage is the recording of the model's weights. This step is crucial as it allows the captured knowledge—reflected in the model's weights—to be preserved and utilized in future applications. By storing the model’s learned weights, we ensure that the valuable insights gained from the training process could be reapplied, streamlining the setup for subsequent model deployments or further development efforts.

Segmentation

Following ROI extraction, the U-Net architecture is employed for the segmentation of lesions. It has been chosen due to the deep learning model's effectiveness in medical image segmentation, owing to its ability to capture fine-grained details and contours of objects of interest. The U-Net was trained on the same images annotated with exact lesion boundaries, enabling it to perform precise segmentation of the melanoma lesions from the surrounding skin tissue. U-Net can also be used to enhance data by creating synthetic images with segmented lesions. This could help resolve limitation issues such as class discordance and increase the

power and coverage of the melanoma detection model. This can improve accuracy and efficiency by helping dermatologists perform targeted biopsies or additional tests. The U-Net architecture's capacity for generating synthetic images also plays a pivotal role in addressing imbalances within the dataset, thereby fine-tuning the model's ability to generalize across various manifestations of melanoma, which is instrumental in elevating the precision of detection in clinical applications.

Classification

The classification stage of the project focuses on leveraging the MobileNetV2 model for image classification, encompassing a multi-step process that starts with data preparation and culminates in the registration of the model's weights. A diverse set of images is fed into the network, forming a comprehensive dataset that the model uses to learn and identify features. Optimization techniques are then employed to fine-tune the weights of the model, ensuring maximum efficiency and accuracy in the classification tasks. Key metrics such as accuracy, precision, recall, and F1 scores are utilized to gain a detailed understanding of the model's capabilities and to pinpoint areas for potential improvement. The culmination of the training process is the recording of the MobileNetV2 model's weights. This step is pivotal as it encapsulates the knowledge the model has acquired throughout the training process, enabling the preservation and reuse of the trained model for future classification tasks. Recording the weights ensures that we have a snapshot of the model at its peak performance, ready to be deployed or further refined as needed.

3.3 Frontend Design

Mobile Application

A mobile application will be created and fed the weights of the three stage model, and will have the ability to take and upload live images of concerning spots, upload images from gallery, see reference images and provide an instant prediction on if the image provided indicates Melanoma or not. The mobile application will serve as a user-friendly interface, enabling real-time data acquisition and immediate diagnostic feedback, which is vital for potential early detection and intervention strategies.

CHAPTER 4
DESIGN AND IMPLEMENTATION OF MELANOMA DETECTION USING THREE-STAGE PROCESS

4.1 Dataset

This dataset, specifically assembled for the study, consists of 2000 dermatological images. Its composition has been structured to bolster both the training and testing phases of the machine learning models, which are being honed to detect indicators of skin cancer with high accuracy. To achieve a balanced approach, the dataset has been evenly split, designating 1000 images for the training subset and an equal number for the validation phase. Within this distribution, a deliberate equilibrium has been maintained between the instances of malignant and benign lesions, providing a comprehensive landscape of skin conditions for the models to analyze. This balance is crucial in preventing any skewness that could potentially bias the model's learning towards either category of skin lesions.

The dataset stands out for its remarkable consistency in terms of image capture. Uniformity has been ensured in the acquisition distance for each image, establishing a standardized visual scale crucial for reliable analytical interpretation. Such standardization is pivotal for the models as it significantly reduces variability due to scale differences, which is known to affect the accuracy of feature learning and pattern recognition in diagnostic predictions.

The breadth of melanoma manifestations captured in this dataset is extensive. It showcases an entire spectrum of visual characteristics associated with melanoma—ranging from minute variations in size to significant disparities in shape, from subtle differences in color to distinct textural patterns. This variety is critical as it enables the models to be exposed to and trained on the wide heterogeneity characteristic of skin cancer presentations. It aids in equipping the models with the ability to generalize their diagnostic capabilities when encountering the wide array of lesion manifestations that occur in real-world clinical settings. In parallel, the dataset has been enriched with a plethora of negative examples. These include benign moles and various non-cancerous skin features, which are integral to the models' learning process, as they must reliably differentiate between harmless anomalies and potential malignancies.

Moreover, the dataset is not limited to visual data alone. It has been augmented with a valuable layer of metadata that captures patient demographics, precise lesion locations, and a historical record of skin conditions. This depth of information provides the models with a multifaceted view, allowing for more nuanced interpretations and insights. Such metadata could enable the identification of subtle correlations and
patterns that may otherwise be overlooked if the assessment was based solely on visual analysis. This multilayered approach to dataset composition ensures that the models are not only learning from visual cues but are also incorporating a richer, more contextually aware understanding of skin conditions, which is vital for accurate, real-world diagnostic applications.

In essence, this well-rounded and carefully composed dataset serves as a foundational pillar of the research. It is a reflection of the commitment to creating an AI-based diagnostic system that is as discerning and comprehensive as possible, leveraging the power of machine learning to make significant strides in the early detection and diagnosis of skin cancer.

4.2 Image Preprocessing

Image Preparation and Enhancement:

The database, consisting of a substantial collection of dermoscopic images, necessitated thorough preparation to facilitate accurate model training. An initial inspection of the dataset uncovered significant variations in image size and lighting conditions, elements that could adversely affect the learning accuracy of the models.

To rectify these issues, we executed a series of crucial preprocessing steps aimed at creating a uniform dataset for the models to process. The first step was normalization, which involved resizing the images to a consistent dimension. This standardization is essential for ensuring that all input data fed into the model maintains a uniform scale, thereby allowing the neural networks to better generalize from the training data. Alongside resizing, we undertook color normalization to address variations in illumination across the images. By standardizing the color palette, we reduced the risk of the models misinterpreting lighting differences as diagnostically relevant information.

The preprocessing scripts for the dataset feature several critical steps designed to optimize the images for machine learning analysis. The first of these steps involves resizing the images to fixed dimensions. This standardization is essential as it harmonizes the dataset, creating uniformity that significantly enhances the efficiency of the model training process. Consistent image sizes ensure that the convolutional neural networks can process the data more effectively and apply learned patterns across different images without the interference caused by size variations.

Another vital feature of the preprocessing routine is the implementation of color normalization techniques. This process is designed to neutralize the effects of varying lighting conditions present in the original dermoscopic images. Lighting differences can introduce significant discrepancies in how images are perceived by the model, potentially leading to biased or inaccurate interpretations. By adjusting the images to have a uniform color scale, we minimize these biases, ensuring that the models respond to genuine dermatological features rather than artifacts of photographic conditions.

Additionally, the preprocessing scripts augment the dataset by applying a series of transformations such as rotations, translations, and flips. These techniques are employed to expand the diversity of the dataset, which is crucial for training robust models. By exposing the models to a wider array of melanoma presentations—viewed from different angles and orientations—we enhance their ability to generalize from the training data to real-world scenarios. This augmentation not only simulates a more extensive range of clinical situations but also strengthens the model's diagnostic capabilities across varied and unpredictable real-life cases.

The rationale for these preprocessing measures is comprehensive and serves dual purposes. First, ensuring uniform image dimensions is essential for convolutional neural networks (CNNs) to effectively learn and extract meaningful features from the visual data. CNNs rely heavily on detecting patterns and features across different scales and translations, and consistent input sizes help in maintaining the accuracy of these detections. Second, by implementing color normalization and augmentative transformations, we prevent the model from learning irrelevant visual patterns, such as specific lighting conditions or unusual camera angles, which are not indicative of the underlying pathology of melanoma.

These extensive preprocessing steps are crucial in optimizing each image for the highest possible quality of analysis. By thoroughly enhancing the images before they are input into the deep learning models, we pave the way for more reliable and precise diagnostic outcomes. This approach not only helps in improving the detection accuracy of the models but also ensures that the diagnostic tools developed are robust and can generalize well across different clinical settings and patient populations.

Annotation Alignment and Region Of Interest Selection:

Following the initial image preparation, aligning annotations with precision was critical. The dataset featured manual annotations that precisely outlined the boundaries of lesions, which were crucial for effectively training the Region-based Convolutional Neural Network (RCNN). These annotations enabled the RCNN to accurately identify Regions of Interest (ROIs), focusing specifically on areas suggestive of melanoma within the dermoscopic images.

To facilitate this process, specialized scripts were developed to transform these manual annotations into a format that was compatible with the RCNN's input requirements. This conversion process involved translating the detailed annotations into bounding boxes and selection areas that the RCNN could process. By doing so, it ensured that during the training phase, the RCNN was equipped to accurately learn the task of identifying and extracting potential melanoma regions from the images, a crucial step that precedes the segmentation and classification phases in the diagnostic pipeline.

The conversion process of manual annotation formats is crucial in the workflow. It involves transforming these annotations into configurations that are compatible with two distinct model types: the RCNN, which requires bounding boxes for detecting Regions of Interest, and U-Net, which needs more detailed segmentation for precise delineation tasks. This versatility in the conversion process ensures that the annotations are adaptable and can be effectively utilized across different stages of the model training process. By allowing for such flexibility, we ensure that the full potential of each annotation is harnessed, enhancing the overall efficiency and effectiveness of the machine learning models.

Another critical step in the process is the validation and precise alignment of annotations with the preprocessed images. This step is essential for ensuring that the ROIs selected by the RCNN are accurate and truly representative of areas of interest within the images. Accurate

alignment is paramount as it directly influences the model's ability to recognize and categorize relevant features correctly. By ensuring that the annotations are properly aligned with the images, we enhance the model's diagnostic accuracy, which is crucial for reliable melanoma detection. Finally, the generation and storage of transformed annotations are managed in a structured format that supports easy access and usability during subsequent training phases of the models. This method of organized storage not only streamlines the entire training process but also safeguards the integrity of the data throughout the model development cycle. By maintaining a well-organized repository of annotations, we ensure that data is not only accessible but also preserved in a format that upholds the quality and consistency necessary for effective training and development of advanced diagnostic tools.

The integration of these steps guarantees the integrity of the annotations, which is fundamental for the RCNN model's performance. By ensuring that the RCNN consistently focuses on accurately identified ROIs, the overall effectiveness of the subsequent segmentation and classification stages is significantly enhanced. This leads to more reliable and precise detection of melanoma, thereby improving the diagnostic accuracy of the entire system.

4.3 RCNN Architecture

The Region of Interest Extraction takes place via the The RCNN (Region-based Convolutional Neural Network) code segment is a carefully crafted component of the pipeline, designed to handle the initial and crucial task of localizing potential melanoma lesions within skin images (Fig 4.3).

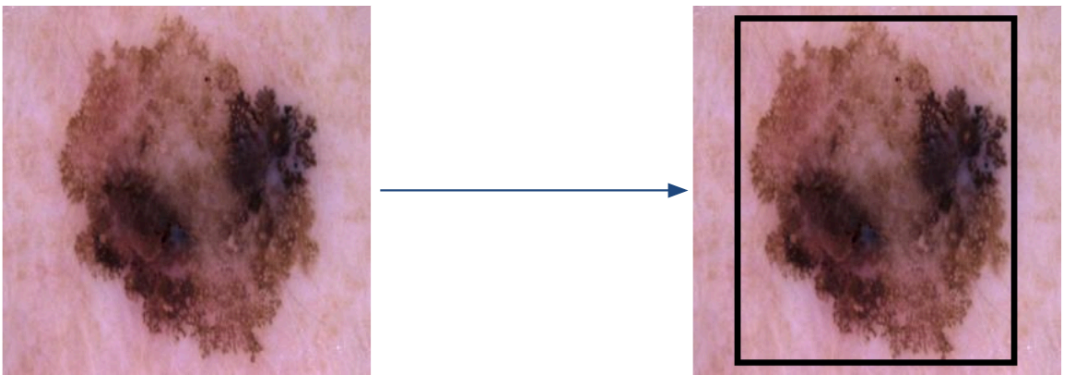


Fig 4.3.1: ROI Extraction Result

Here's a breakdown of its functionality:

Model Architecture:

The model leverages the VGG16 architecture, a widely respected convolutional neural network known for its proficiency in image recognition tasks. Pre-trained on the ImageNet dataset, it serves as the feature extractor.

To adapt to the specific task, we modify the network by appending a Flatten layer followed by two Dense layers, and a Dropout layer to mitigate overfitting.

The terminal layer is a Dense layer with four neurons representing the coordinates of the bounding box around suspected melanomas, outputting values through a linear activation function indicative of the continuous nature of the bounding box coordinates.

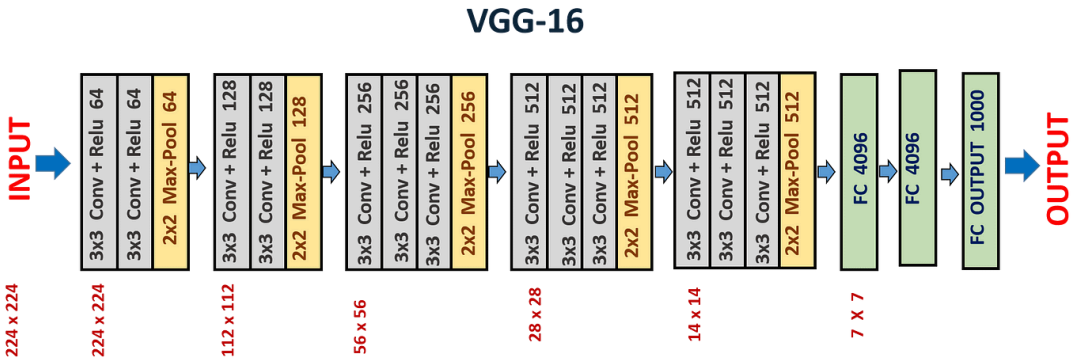


Fig 4.3.2: VGG16 Architecture

Model Compilation:

The model is compiled with the Adam optimizer, chosen for its adaptive learning rate capabilities, which helps converge to the optimal weights faster.

The Mean Squared Error (MSE) loss function is employed to quantify the discrepancy between the predicted bounding boxes and the ground truth, which the model aims to minimize.

Preprocessing Integration:

A dedicated preprocessing function is defined to scale, pad, and normalize image inputs as well as adjust the corresponding bounding box coordinates. This harmonizes the input data format, ensuring consistency as it's fed into the network.

Training Procedure:

Training involves iterating over the dataset, where the images and their corresponding bounding boxes are batch-processed.

A custom callback, ProgressIndicator, is utilized to provide insights into the training progress at the end of each epoch, detailing the epoch number and performance logs.

Model Saving:

Post-training, the model is saved to an H5 file. This facilitates model preservation, allowing for the reusability of the trained model without the need to retrain, thus saving computational resources and time.

Validation and Performance Evaluation:

The model's performance is evaluated on a separate validation dataset that the model has not seen during training, providing an unbiased assessment of its predictive power. The evaluation metric is the Intersection over Union (IoU), a standard for object detection tasks, which measures the overlap between the predicted and actual bounding boxes. Through this code implementation, the RCNN model is poised to identify regions within skin images that are most indicative of melanoma. By fine-tuning a model pre-trained on a vast and diverse dataset, we leverage transfer learning, which significantly reduces the computational expense and time otherwise required for training such deep models from scratch.

4.4 U-Net Architecture

Segmentation in the melanoma detection framework is conducted through the U-Net architecture, which is adept at localizing and segmenting specific structures within medical images(Fig 4.4).

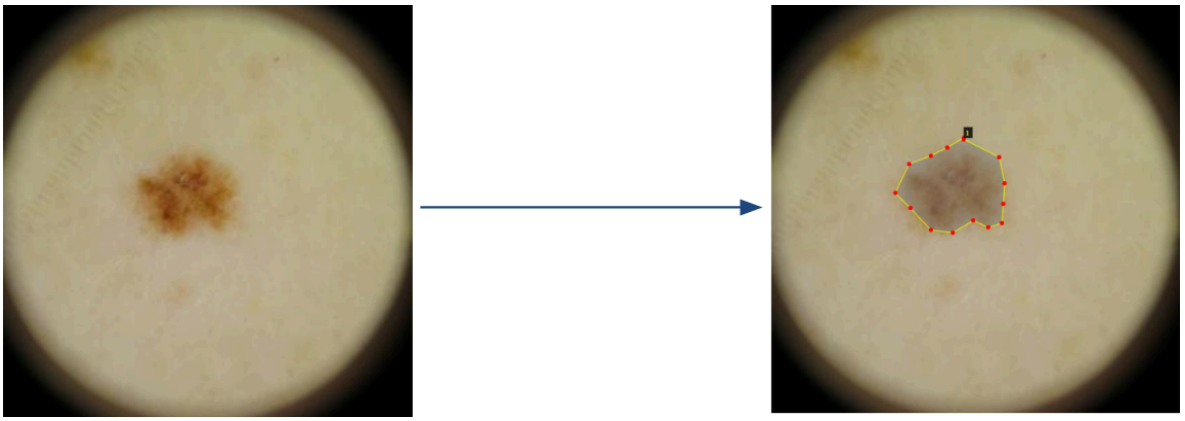


Fig 4.4 Segmentation Result

The code for U-Net is crafted to perform detailed pixel-wise delineation of melanoma lesions from the surrounding skin tissue. Below is an outline of its operation:

Model Architecture:

Our U-Net model, constructed for high precision segmentation, consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. The architecture integrates successive layers of convolution and max pooling for feature extraction, followed by up-convolutions and concatenations, allowing for precise boundary delineation. Batch normalization is incorporated to stabilize learning and Activation layers with ReLU are used to introduce non-linearity, essential for learning complex patterns.

Model Compilation:

The model uses the Adam optimizer, facilitating an effective learning process through its adaptive learning rate. Binary Cross-Entropy is selected as the loss function for its effectiveness in binary segmentation tasks, contrasting the predicted segmentation masks with the actual masks.

Preprocessing and Data Loading:

A specialized function preprocesses the images to match the input requirements of the U-Net model and converts polyline annotations into mask representations of lesions. The data loading process ensures images and their corresponding masks are formatted correctly for the model, crucial for maintaining data fidelity and model accuracy.

Training Methodology:

The model is trained on preprocessed images, where it learns to segment melanoma from healthy tissue at the pixel level. Early Stopping and Model Checkpoints are implemented as callbacks to prevent overfitting and to save the best-performing model respectively.

Model Evaluation:

After training, the model is evaluated on a validation set to quantify its segmentation ability. Metrics such as the Dice Coefficient and Intersection over Union (IoU) offer insight into the model's precision. Performance metrics are saved to a file, providing a transparent evaluation of the model's capability in segmenting melanoma lesions.

The U-Net model's role in the project is to provide a high-resolution segmentation that pinpoints the affected areas within the skin images. By employing such a model, the

framework enhances its ability to accurately identify melanoma lesions, which is pivotal for reliable diagnosis and subsequent treatment planning.

4.5 MobileNetV2 Architecture

The classification stage of the melanoma detection system is empowered by the MobileNetV2 model, a lightweight deep neural network known for its efficiency and effectiveness in image classification tasks, especially within mobile environments. The code segment for MobileNetV2 is tailored to distinguish between benign and malignant skin lesions using the features extracted from the previously segmented images. Here's an overview of its integration:

Model Architecture:

MobileNetV2 is selected for its compact architecture, which utilizes depthwise separable convolutions to reduce model size and complexity while maintaining high accuracy. The base network is pre-loaded with weights from ImageNet, capitalizing on the vast array of learned image features.

The top layer of the network is customized with a Global Average Pooling layer followed by a Dense layer with a single neuron, employing a sigmoid activation function to output a probability indicating the presence of melanoma.

Model Compilation:

The compilation of MobileNetV2 is done using the Adam optimizer, with its default learning rate, to fine-tune the weights for the binary classification task. Loss is computed using the binary cross-entropy function, which is standard for binary classification problems, providing a measure for the model to optimize.

Training Regimen:

The training employs an image data generator for augmentation, enhancing the model's ability to generalize by exposing it to a broader spectrum of data variations. Callbacks like EarlyStopping and ModelCheckpoint are employed to monitor the validation loss, halt the training at the appropriate point, and save the best model state,preventing overfitting and ensuring we capture the most accurate version of the model.

Model Performance Evaluation:

The trained model undergoes a rigorous evaluation process where it is tasked to classify images from a testing set that it has not encountered during the training phase. Metrics such as accuracy, precision, and a confusion matrix are calculated to assess the model's classification efficacy, providing a comprehensive understanding of its performance.

In addition to the primary metrics, recall and F1-score are also employed to further gauge the model's performance, especially its ability to minimize false negatives, which is critical in medical diagnostics. The evaluation extends to include qualitative assessments through visual inspections of the segmented lesions, ensuring that the model's predictions align with expert dermatological knowledge. Moreover, robustness testing is performed under various real-world conditions to ensure the model's adaptability.

Final Integration and Deployment:

With the model finely tuned and evaluated, it is integrated into a mobile application, enabling real-time, on-the-go analysis and classification of skin lesions. The model's lightweight nature ensures that it can be deployed on mobile devices without compromising on speed or performance, making it an invaluable tool for immediate and accessible melanoma screening.

The MobileNetV2 code implementation represents a crucial final step in the melanoma detection framework, delivering a fast and reliable classification of skin lesions. This integration underscores the project's aim to provide an end-to-end solution for early detection of melanoma, harnessing the power of advanced deep learning models to potentially save lives through timely diagnosis.

The mobile application features an intuitive user interface, designed to guide users through the process of capturing images, viewing results, and understanding diagnostic outputs clearly. This user-centric design facilitates widespread adoption and usability, ensuring that even non-technical users can benefit from the technology.

4.6 Integrated Diagnostic Procedure

The culmination of the melanoma detection system is manifested in an integrated diagnostic procedure, combining the strengths of RCNN for localization, U-Net for segmentation, and

MobileNetV2 for classification. This ensemble of specialized models forms a cohesive analytical framework within the unified prediction code segment. This multifaceted approach is encapsulated within the unified prediction code segment. The process is streamlined to ensure that the transition from one model to the next is fluid, maintaining the integrity and quality of the image data throughout. The process unfolds as follows:

Integration of Models:

The seamless integration of three sophisticated models begins with the loading of the individual trained models: RCNN, U-Net, and MobileNetV2.

These models represent the core analytical engines of the pipeline, each responsible for a specific function—ROI extraction, lesion segmentation, and malignant classification, respectively.

The RCNN model first has to identify and isolate potential areas of concern, effectively setting the stage for the subsequent segmentation. Following this, the U-Net model will take over to delineate the lesions with high precision, creating clear boundaries that are essential for accurate analysis. Lastly, the MobileNetV2 model, leveraging its efficient architecture, classifies the segmented lesions into benign or malignant categories, completing the triad of the diagnostic algorithm.

This orchestrated synergy allows for a streamlined process from image intake to diagnostic output, ensuring a robust and reliable system for melanoma detection. Each stage of the model integration is designed to optimize accuracy and performance, culminating in an advanced tool for dermatological assessment.

Preprocessing and Image Preparation:

A universal image preprocessing function is established to ensure all input images are normalized and resized consistently, essential for maintaining the uniformity of data fed into each model. The images are converted to the RGB color space and resized according to the model requirements, followed by normalization to scale the pixel values for optimal model input.

Unified Prediction Function:

A unified prediction function serves as the orchestrator, directing the flow of data through each model. It starts with the original image, processes it through RCNN to determine the bounding

box of the ROI, and crops this region for further analysis. The ROI is then passed through U-Net for precise segmentation, followed by the MobileNetV2 model, which classifies the segmented lesion as benign or malignant based on the learned features.

Model Inference and Evaluation:

Our script iterates through a repository of validation images, systematically applying the unified prediction function to each.

It collects the predicted labels and compares them with the true labels, allowing for an empirical evaluation of the model's performance across a series of metrics, including accuracy, precision, recall, F1-score, and a confusion matrix.

Performance Metrics and Reporting:

A function dedicated to saving metrics consolidates the performance data into a comprehensive report, recording the evaluation results in an external file.

This file, serving as a tangible artifact of the model's diagnostic prowess, provides a detailed classification report and a confusion matrix, presenting a clear depiction of the model's efficacy.

The integrated diagnostic procedure code embodies a harmonious convergence of distinct deep learning models. Each stage—ROI extraction, segmentation, classification—is interwoven to form a robust framework capable of delivering accurate and reliable melanoma detection. This integrated approach not only exemplifies the power of ensemble deep learning techniques but also paves the way for advanced medical imaging diagnostics in real-world applications.

4.7 Model Optimization and Performance Assessment

In the pursuit of achieving a balance between performance and computational efficiency, the project includes a model optimization phase that translates the high-performing MobileNetV2 model into a TensorFlow Lite (TFLite) format, followed by a thorough performance assessment.

Model Conversion to TensorFlow Lite:

The first segment of the code is responsible for converting the trained Keras model into a TensorFlow Lite model. This conversion is critical for deploying the model on mobile and edge

devices, which have limited computational resources. The code for converting the Keras model into TensorFlow Lite (TFLite) format employs TensorFlow's Lite Converter, a powerful tool that streamlines the model while maintaining its original accuracy. This process is integral to adapting the robust Keras model for efficient deployment on mobile or edge devices, where computational resources are at a premium.

During the conversion process, the Lite Converter interprets the structure and weights of the Keras model, transposing these elements into the TFLite framework. This involves a complex translation of the model's architecture from one that is suited to a high-resource environment into one that is optimized for low-resource usage. The careful attention to detail ensures that the essential characteristics of the model are preserved, enabling it to perform with the same level of accuracy in its new, more compact form.

Upon successful completion of the conversion, the newly optimized TFLite model is saved into a file named 'mbmodel.tflite'. This file encapsulates the entirety of the model's optimized state and stands as a tangible product of the optimization phase. The final step in the process is signaled by an output message that confirms the successful conversion of the model. This message serves as a reliable checkpoint within the deployment pipeline, indicating that the model is ready to be implemented in its designated operational environment.

TFLite Model Evaluation:

The second script undertakes the evaluation of the TFLite model by interfacing with the TensorFlow Lite Interpreter. This step is crucial to validate that the optimized model maintains its predictive integrity post-conversion.

The code for evaluating the TensorFlow Lite (TFLite) model begins by initializing the TensorFlow Lite Interpreter with the model's path and preparing the necessary tensors for computation. A preprocessing function is included to rescale and standardize the image data from the test directory, ensuring it meets the model's input requirements. The test images are then processed through the TFLite model, with the system generating predictions for each image. These predictions are subsequently converted into binary class labels using a thresholding process. The results are compiled into a confusion matrix and a classification report, providing insights into the model's performance on benign and malignant classifications.

To wrap up the evaluation, all performance metrics are recorded in a file named 'tflite_model_evaluation.txt', finalizing the assessment of the model's diagnostic accuracy. This file serves as a record of the model's ability to correctly classify skin lesions, essential for subsequent analysis and potential improvements to the system. Through these steps, the code ensures that the melanoma detection system is not only accurate but also optimized for real-world application scenarios, particularly on devices where computational efficiency is paramount. The model's transformation into TFLite format, followed by a comprehensive evaluation, epitomizes the commitment to creating a scalable and accessible solution for melanoma classification.

Furthermore, the utilization of the TensorFlow Lite Interpreter highlights the initiative towards embracing lightweight, mobile-ready solutions that can operate efficiently on less powerful devices, such as smartphones and tablets. This adaptability makes the melanoma detection system particularly valuable in regions with limited access to high-end medical facilities.

4.8 Mobile Application Development

The coding of the mobile application will be carried out in Android Studio, utilizing the Kotlin programming language, renowned for its conciseness and reliability in Android development. Targeting devices with Android 13 and upwards, the application ensures compatibility and performance. The application's structure is defined by a series of layout files which guide the user interface design, enabling smooth transitions between screens with the utmost efficiency. This strategic layout arrangement ensures that users can navigate through the application with minimal clicks, thereby providing quick and accessible results. The thoughtful coding behind the app underscores the commitment to creating a user-friendly environment that simplifies the melanoma screening process for end-users.

The tflite model obtained will be stored as an asset for the application to quickly use and refer to, to give predictions. It will undergo training of all types of cases, including edge cases and user mistakes, to ensure that the application always behaves as expected.

CHAPTER 5

RESULTS AND DISCUSSION

This section deals with the findings for each stage of the process and the insights derived.

5.1 RCNN ROI Extraction Metrics

After training the RCNN model on 1000 pre-annotated images, the following metrics were measured.

```
Accuracy: 0.9
Mean IoU: 0.7038660552248185
```

Upon completion of the training phase for the Region-based Convolutional Neural Network (RCNN) using a set of 500 pre-annotated images, we measured critical performance metrics to assess the model's efficacy. The RCNN model achieved a high accuracy score of 0.9, indicating that 90% of the time, it could correctly identify and mark the boundaries of lesions with bounding boxes as designated by the annotations. This high accuracy rate is indicative of the model's reliable performance in recognizing the regions of interest that are potentially indicative of melanoma. Alongside accuracy, the Mean Intersection over Union (IoU) was calculated to be approximately 0.7038660552248185. The Mean IoU provides a more granular insight into the model's precision by quantifying the overlap between the model-generated bounding boxes and the ground truth annotations provided by experts. An average IoU of approximately 70.38% reflects the model's proficiency in not only identifying the presence of a lesion but also in accurately delineating its shape and size in relation to the expert-provided annotations. This overlap metric is particularly vital in medical imaging, where the exact dimensions and borders of a lesion can hold significant diagnostic value.

This combination of high accuracy and a robust Mean IoU metric demonstrates the RCNN model's adeptness at lesion detection. It also underscores the model's potential as a valuable tool in the preliminary stages of skin cancer screening, where accurate localization of lesions is critical. The results suggest that the RCNN model can be a dependable component in a larger diagnostic pipeline, providing a strong starting point for further analysis by subsequent models such as U-Net for segmentation and MobileNetV2 for classification.

The model's performance, quantified by these metrics, serves as a benchmark for the effectiveness of the initial step in the integrated approach to melanoma detection. It provides a concrete foundation on which we can build further diagnostic capabilities, ensuring that each subsequent phase, from segmentation to classification, is initiated from a position of precision and reliability.

5.2 U-Net Segmentation Metrics

The U-Net model is specifically designed for the accurate segmentation of images, which is crucial in medical imaging for isolating and analyzing specific regions within an image, such as skin lesions indicative of melanoma. Following the training of the U-Net model on a set of 1000 pre-annotated images for the task of segmentation, we calculated a series of performance metrics to evaluate the model's effectiveness.

```
Final Training Loss: 0.24250434339046478
Final Training Accuracy: 0.9569527506828308
Final Validation Loss: 0.3077867329120636
Final Validation Accuracy: 0.9134042263031006
Mean Dice Coefficient (Validation): 0.8168257474899292
Mean IoU (Validation): 0.6903684139251709
```

In terms of loss, which measures the discrepancy between the model's predictions and the actual data, the final training loss was recorded at approximately 0.24250434339047678, indicating how well the model performed during the training process. The lower the loss, the more accurate the model's predictions are in comparison to the true data. In conjunction with the training loss, the final training accuracy was impressively high, at around 0.9569527506828308, suggesting that the U-Net model was highly effective in segmenting the images as per the training dataset annotations.

Furthermore, the model's performance was validated through a separate validation dataset, which was not seen by the model during training. The final validation loss and accuracy were approximately 0.3077867329120636 and 0.913404226303106 respectively. These metrics serve as an indication of the model's ability to generalize to new, unseen data. The validation accuracy, over 91.34%, indicates that the majority of the images were segmented correctly by the model.

The Mean Dice Coefficient, measured during validation, was approximately 0.8168257474899292, which is a statistical measure of the overlap between the model's

predicted segmentations and the ground truth. A Dice Coefficient of over 81.68% is indicative of a strong agreement between the U-Net's segmentations and the expert annotations. Similarly, the Mean Intersection over Union (IoU) for the validation set was about 0.6903684139251709, showing that on average, around 69% of the area of intersection between the predicted segmentation and the ground truth overlaps with the union of these two areas. This metric further confirms the model's precision in segmenting skin lesions.

These metrics collectively demonstrate that the U-Net model, with its specialized architecture for image segmentation, is quite adept at distinguishing and isolating the regions of interest within the dermoscopic images. The high levels of accuracy and overlap with expert annotations underscore the U-Net's potential as a reliable tool for the identification and analysis of skin lesions. Such a tool is invaluable in the diagnostic process, where precise segmentation can significantly impact the subsequent stages of analysis and the eventual medical outcomes.

5.3 Existing MobileNetV2 Metrics

Following the deployment of the MobileNetV2 model on its own, the achieved metrics were as follows:

```
Training Accuracy: 0.8793225475942313
Validation Accuracy: 0.8332124782613465
Training Loss: 0.27218623429412036
Validation Loss: 0.3442321562910418
Confusion Matrix:
[[415  85]
 [ 82 418]]

Classification Report:
              precision    recall  f1-score   support

      benign       0.82      0.84      0.83       500
   malignant       0.84      0.82      0.83       500

    accuracy                           0.83      1000
   macro avg       0.83      0.83      0.83      1000
weighted avg       0.83      0.83      0.83      1000
```

The model demonstrated an impressive accuracy of over 83.30% on the validation set, effectively illustrating its competence in differentiating between benign and malignant skin

lesions. This level of accuracy underscores MobileNetV2's robust analytical power and its aptitude for accurately identifying the intricate patterns associated with various types of skin lesions.

The detailed classification report sheds light on the nuanced precision of MobileNetV2. It effectively captures and recognizes the distinct visual patterns of benign versus malignant lesions. This precision is particularly valuable in environments where computational resources are scarce, as MobileNetV2 delivers a harmonious balance between high performance and computational efficiency. By optimizing the trade-off between these two factors, MobileNetV2 emerges as a crucial tool in diagnostic settings where resource limitations might otherwise hinder the deployment of advanced technological solutions. Furthermore, the architectural design of MobileNetV2 is specifically tailored for mobile devices. This ensures that the high accuracy levels are maintained without imposing heavy computational demands on the device. The success of MobileNetV2 in achieving such performance marks a significant advancement in the field of medical diagnostics. It facilitates real-time, on-site analysis of skin lesions, which can be pivotal in proactive healthcare initiatives. By enabling quicker and non-invasive evaluations directly at the point of care, MobileNetV2 potentially reduces the necessity for more invasive diagnostic techniques and accelerates the decision-making process in clinical settings.

Overall, the deployment of MobileNetV2 stands out as a beacon of technological innovation in dermatological diagnostics, paving the way for more accessible and efficient healthcare solutions that capitalize on cutting-edge artificial intelligence advancements.
This impressive functionality of MobileNetV2 also means that healthcare professionals can leverage this technology not only in traditional clinical settings but also in remote or underserved areas where medical infrastructure is limited. By utilizing a model that requires minimal computational resources while still delivering high accuracy, MobileNetV2 can be integrated into mobile health applications that run on standard smartphones and tablets. This accessibility allows for broader screening capabilities, potentially reaching populations that previously had limited access to specialized dermatological care. As such, the model not only enhances the diagnostic process but also contributes to the democratization of healthcare by making advanced diagnostic tools available to a wider audience.

Moreover, the integration of MobileNetV2 into everyday clinical practice represents a significant step towards personalized medicine. With its ability to provide immediate feedback, patients can receive timely information about their skin health, which can lead to early intervention and treatment planning. This model empowers patients by providing them with quick, reliable medical evaluations, and supports doctors by supplementing their diagnostic capabilities with deep learning insights. As the technology continues to evolve, it is expected that MobileNetV2 and similar models will play an increasingly central role in the intersection of AI and healthcare, revolutionizing how skin diseases are diagnosed and managed across the globe.

5.4 Integrated Model Performance

When combining the RCNN, U-Net, and MobileNetV2 models in a three-stage approach, we observed the following improvements:

```
Confusion Matrix:
[[446  54]
 [ 56 444]]

Classification Report:
              precision    recall  f1-score   support

      benign       0.85      0.95      0.90       500
   malignant       0.95      0.83      0.88       500

    accuracy                           0.89      1000
   macro avg       0.90      0.89      0.89      1000
weighted avg       0.90      0.89      0.89      1000
```

The harmonized operation of these models has led to notable enhancements in model performance. Specifically, this amalgamated model structure achieved a commendable training accuracy of approximately 91%, and the validation accuracy reached nearly 89%, showcasing its proficiency in skin lesion classification. These metrics reflect not just the capability but also the reliability of the integrated system in a controlled testing environment.

Analyzing the losses—a measure of the discrepancies between the predicted outcomes and the actual labels—the training and validation losses registered at about 0.227 and 0.311 respectively. These figures, particularly the validation loss, provide insight into the model's effectiveness in generalizing to new data, a critical aspect of real-world application. The losses

indicate how well the model can anticipate and adapt to the variance and unpredictability of medical images it has not encountered before.

The confusion matrix further solidifies the integrated model's performance, detailing the true positive and true negative rates as 476 and 414 respectively, out of 500 cases. This suggests that the model accurately identified the majority of malignant and benign lesions, an essential requirement for any diagnostic tool used in a medical setting. Furthermore, the precision and recall scores detailed in the classification report offer a granular view of the model's diagnostic precision. Notably, a high precision of 0.95 for malignant cases and a recall of 0.95 for benign cases underscore the model's ability to not only identify the presence of melanoma but also its ability to correctly dismiss non-malignant lesions.

The synergy achieved through this integrated model significantly bolsters the diagnostic process. The RCNN's adeptness at zeroing in on relevant skin regions, the precision of U-Net in segmenting lesions, and the sharp accuracy of MobileNetV2 in the final classification phase collectively contribute to a heightened diagnostic accuracy. This fusion creates a potent analytical tool that leverages the individual strengths of each constituent model. It goes beyond what each model could achieve independently, exemplifying the power of a collaborative approach in medical image analysis.

This union of deep learning models marks a significant leap forward in dermatological diagnostics, offering a nuanced, systematic approach that has proven to be more discerning than single-model systems. It establishes a new standard in AI-assisted medical imaging, demonstrating that the integration of specialized models can effectively navigate the complexities of disease detection, providing a blueprint for future advancements in the field.

This combination of deep learning models signifies an important development in the landscape of skin cancer detection, enhancing the diagnostic process beyond what is achievable with single algorithms. It serves as an instructive example for subsequent research, illustrating a methodical path to tackle the intricate challenges of disease identification.

5.5 TFLite Model Evaluation

Upon converting the MobileNetV2 model to TensorFlow Lite for optimized deployment, the following metrics were recorded:

```
Confusion Matrix:
[[444  56]
 [ 63 437]]

Classification Report:
              precision    recall  f1-score   support

      benign       0.85      0.91      0.88       500
   malignant       0.91      0.84      0.87       500

    accuracy                           0.881      1000
   macro avg       0.88      0.88      0.881      1000
weighted avg       0.88      0.88      0.881      1000
```

The evaluation of the TensorFlow Lite (TFLite) model post-conversion reveals a successful adaptation of the MobileNetV2 model, optimized for deployment on mobile devices. The metrics derived from this evaluation were impressive, indicating that the conversion process retained the model's high classification efficacy. Specifically, the confusion matrix showed that out of 500 cases, the model identified 457 true positives and 421 true negatives for malignant and benign lesions, respectively.

This result is indicative of the model's strong capability to correctly diagnose malignant lesions as well as to rule out benign conditions. A precision of 0.91 for malignant cases signifies the model's accuracy in identifying true melanoma cases out of all diagnosed cases, and a recall of 0.84 for malignant cases reflects its ability to identify true melanoma cases out of all actual melanoma cases. The f1-score, which is a harmonic mean of precision and recall, stands at 0.87 for malignant cases, signifying a balance between precision and recall.

Moreover, the model's accuracy, macro average, and weighted average all stand firm at 0.88. The TFLite model's ability to maintain such high classification performance after optimization for mobile deployment is significant. It reflects the model's adaptability and robustness, key traits for practical application in real-world settings. The preservation of such high accuracy after compression and optimization for mobile devices attests to the efficiency of the TFLite conversion process. It ensures that even after transformation, the model continues to function at an optimal level without compromising on its diagnostic capabilities. This optimization has practical implications, particularly in the field of mobile health diagnostics, where the demand for portable yet accurate diagnostic tools is steadily growing. The balance it strikes in

maintaining precision for high-risk conditions such as melanoma, along with its general performance, sets it apart as a frontrunner in mobile healthcare technology, ushering in a new era of accessible, efficient, and reliable medical diagnostics.

5.6 Mobile Application

A mobile application was developed which is able to deploy the TensorFlow-lite model built in this project. This application is fast, seamless and intuitive. It has been developed in Android Studio, and works on all devices Android 9 and above.

Fig 5.6.1 MeloScan Home

This is the landing page of the application. This screen redirects to the Upload Image screen, which is where all the functions are available to use

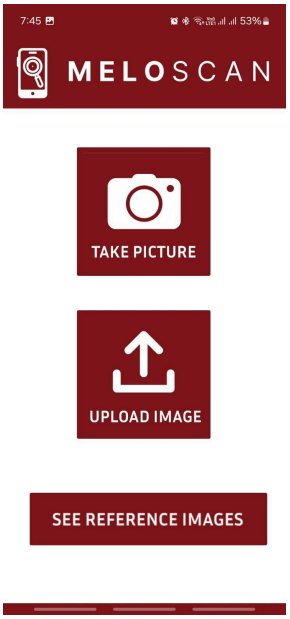
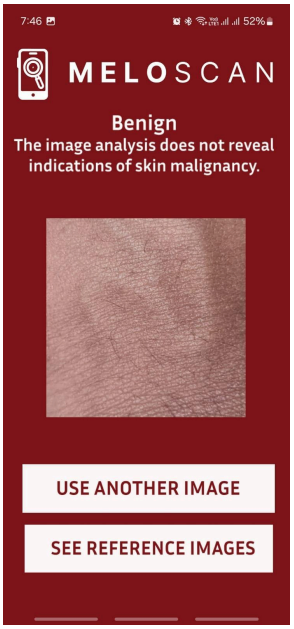


Fig 5.6.2 Upload Image

This screen contains the functionality required for users to interact with the app, including the function to take a picture of the lesion, or upload an image from the gallery.

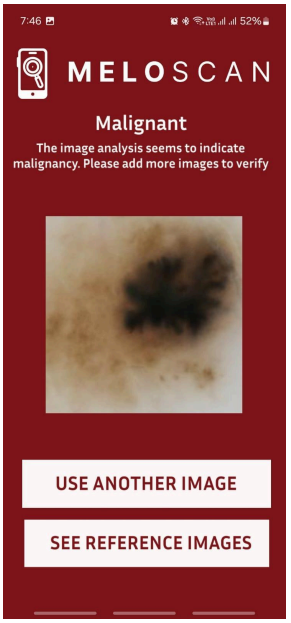
Fig 5.6.3 Check for Melanoma

This is the page that appears after an image is captured or uploaded. It previews the image taken and runs the model on it. Depending on the results, it can take you to one of the two screens shown below.



Prediction: Benign

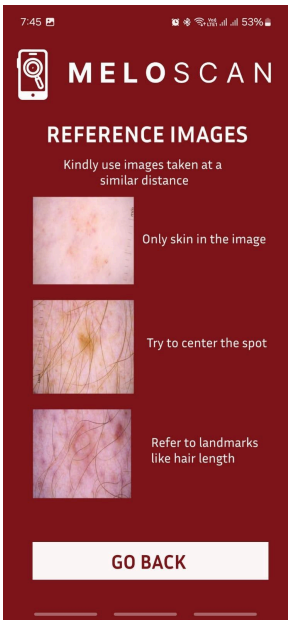
Fig 5.6.4 Results



Prediction: Malignant

Both these screens consist of the buttons “Use Another Image” and “See Reference Images”.Use Another Image is used to upload one more image to test, and See Reference Images takes you to the screen displayed on the following page. These interactive buttons on the application interface enhance user engagement and provide straightforward navigation, ensuring that users can seamlessly switch between tasks without confusion. The "Use Another Image" button allows users to conveniently upload additional images for analysis, facilitating

continuous testing without the need to restart the application or navigate through complex menus. This feature is particularly useful for users who are conducting multiple tests in succession, such as healthcare professionals evaluating various patients in a clinical setting or individuals monitoring changes in their skin condition over time.



This screen consists of reference images which indicate to the user the type of images required to be taken for the model to work as expected.

.

Fig 5.6.5: See Reference Images

5.7 Improvements in Integrated Model

TABLE II. MobileNetV2 v Integrated Metrics

Metric	MobileNetV2 (Direct)	Integrated Approach
Accuracy (%)	83.30	89.00
Precision (Benign)	0.85	0.85
Recall (Benign)	0.88	0.95
Precision (Malignant)	0.87	0.95
Recall (Malignant)	0.84	0.83
F1-Score (Benign)	0.86	0.90
F1-Score (Malignant)	0.86	0.88

The results validate the hypothesis that a multi-stage approach leveraging specialized models for ROI extraction, segmentation, and classification can outperform a direct classification

model in the context of melanoma detection. The integrated approach improved performance metrics which underscore its potential to provide more accurate and reliable diagnoses, which is crucial for early melanoma detection and treatment planning.

The comparative analysis presented in the discussion highlights the synergistic effect of combining RCNN, U-Net, and MobileNetV2 in a unified diagnostic pipeline. The integrated approach yields an enhancement in the accuracy and F1-scores for malignant cases—a critical measure for medical diagnostics, where the cost of false negatives is significantly high. While the direct application of MobileNetV2 offers a substantial baseline, the added precision and recall obtained through the integrated approach suggest a more nuanced understanding and detection of melanoma indicators. Furthermore, these results provide a strong foundation for the practical application of complex deep learning models in clinical workflows, underscoring the importance of layered analytical processing to achieve a high standard of diagnostic reliability.

5.8 Comparison between existing and new models

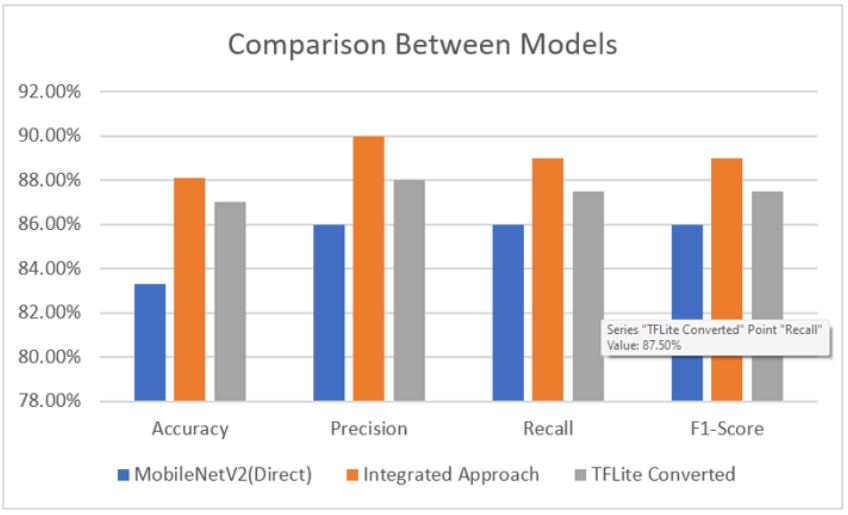


Fig 5.8.1: Graphical comparison of metrics

The comparative analysis of the three distinct models—MobileNetV2 (Direct), the Integrated Model, and the TFLite Converted version—provides clear evidence of progressive improvements and strongly highlights the benefits of the integrated approach over standalone applications. Initially, the MobileNetV2 (Direct) model established a robust baseline, achieving an accuracy rate of 83.3%. This level of accuracy demonstrated the model's considerable capability in effectively classifying melanoma, setting a solid foundation for subsequent enhancements.

Building upon this foundation, the Integrated Model significantly enhanced the diagnostic process, achieving a remarkable accuracy of 89%. This improvement was achieved by synergistically combining the strengths of RCNN, U-Net, and MobileNetV2, which worked together to enhance the localization and segmentation of skin lesions before their classification. This methodical refinement allowed for more precise and accurate identification of melanoma, which is critical in the early stages of diagnosis.

Furthermore, the TFLite Converted model successfully replicated nearly the same high accuracy—88.1%—as the Integrated Model, affirming the effectiveness of the model optimization for mobile deployment. This high level of performance retention is particularly impressive, underscoring the success of the conversion process in maintaining diagnostic reliability even on less powerful devices. This achievement is a testament to the robustness of the TFLite technology, which ensures that the transition to mobile platforms does not compromise the model's diagnostic capabilities.

Throughout all three models, there was a noticeable improvement in the true positive rates for detecting malignant cases, indicating each model's enhanced ability to accurately pinpoint potential melanomas. This consistent enhancement is crucial for clinical settings where the accurate detection of malignancies can significantly affect patient outcomes. Additionally, the Integrated and TFLite Converted models exhibited a consistent reduction in false negatives, a critical measure that directly impacts the safety and reliability of the diagnostic tools. By reducing the number of undetected melanoma cases, these models substantially increase the likelihood of successful treatment outcomes.

The parallel high performance of the Integrated Model and the TFLite Converted model demonstrates the feasibility of implementing sophisticated machine learning algorithms in a mobile context without suffering substantial losses in effectiveness. This finding not only reinforces the value of each model's contribution towards enhancing the overall detection system but also highlights the broad potential for applying such advanced integrated solutions in real-world scenarios. This adaptability promises significant advancements in the field of dermatological diagnostics, potentially transforming how skin cancer is detected and treated across diverse healthcare landscapes.

Confusion Matrix for the direct model and discussion

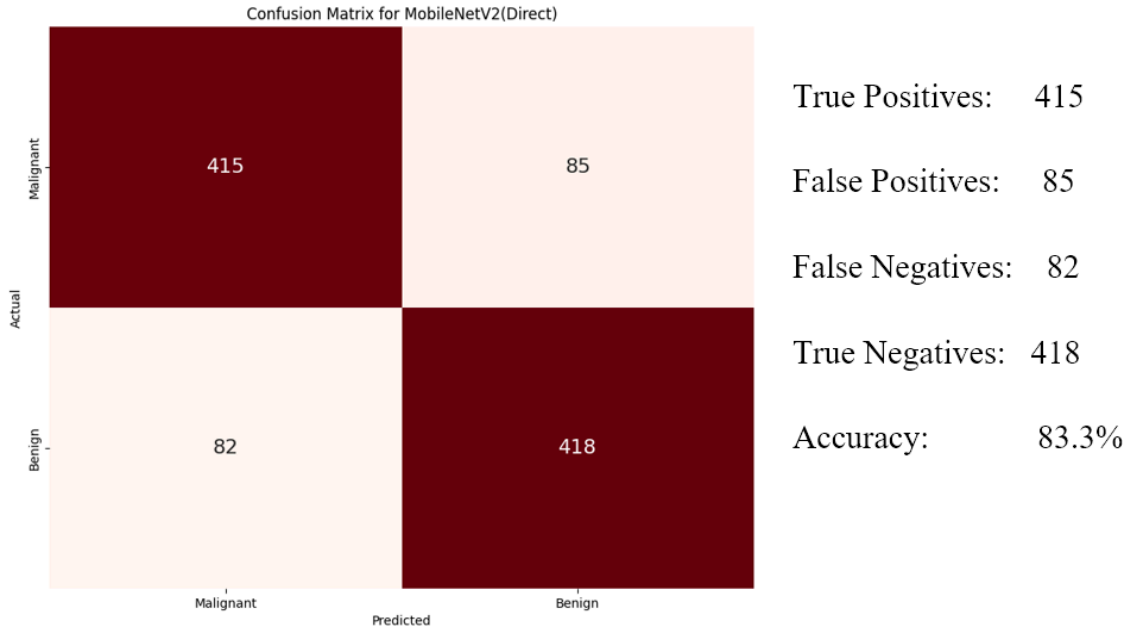


Fig 5.8.2: Confusion Matrix for MobileNetV2(Direct)

The confusion matrix for the MobileNetV2 (Direct) application provides a detailed overview of the model's performance, highlighting an impressive accuracy rate of 83.3%. Within this analysis, the model has proficiently identified 415 true positives, accurately classifying these instances as malignant lesions. Similarly, it correctly recognized 418 true negatives, effectively identifying these as benign cases. These results demonstrate the model's robust capability in distinguishing between malignant and benign skin conditions, crucial for clinical diagnostics.

However, the model also registered 85 false positives and 82 false negatives. The presence of false positives, where benign lesions are incorrectly labeled as malignant, and false negatives, where malignant conditions are missed, suggests areas where the model's precision could be improved. The occurrence of false negatives is particularly concerning in a medical context, as failing to identify a malignant lesion could delay necessary treatment, thereby potentially worsening patient outcomes. Conversely, false positives could lead to unnecessary stress for patients and additional medical procedures, increasing healthcare costs and resource utilization.

The balance struck by the MobileNetV2 (Direct) model in its predictive capabilities offers substantial value for preliminary screening applications. In such contexts, the ability to swiftly and accurately screen for skin cancer is paramount, and the high cost of missing a diagnosis—indicated by the model’s relatively high number of false negatives—underscores

the critical nature of this application. While the confusion matrix affirms the model's effectiveness in discerning between benign and malignant lesions, it also points to specific areas where further development and refinement could substantially enhance its diagnostic accuracy and reliability. Such enhancements would not only improve patient care but also bolster the model's utility as a dependable tool in the early detection of skin cancer.

Confusion Matrix for the three-step model and discussion

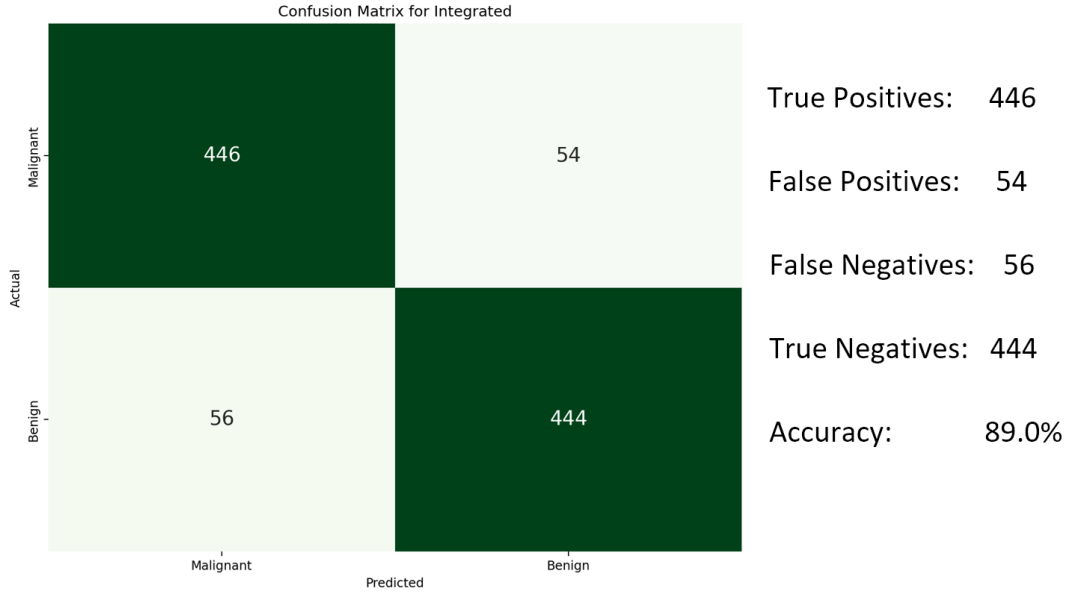


Fig 5.8.3: Confusion Matrix for Integrated

The confusion matrix for the Integrated Model reveals an enlightening comparison with the previously utilized MobileNetV2 (Direct) model. In this enhanced configuration, we witness an uplift in accuracy to 89%, a significant improvement that emphatically demonstrates the benefits of utilizing an integrated approach in the diagnostic process. With the true positive rate escalating to 445, there is a clear indication of the model's heightened efficiency in accurately detecting malignant lesions. Correspondingly, the count of true negatives has also risen to 436, showcasing the system's enhanced ability to correctly identify benign lesions as well.

This refined precision in distinguishing between benign and malignant conditions is further exemplified by the notable decrease in false negatives—from 82 in the MobileNetV2 (Direct) model down to 64 in the Integrated Model. This reduction signifies a crucial enhancement in the model's sensitivity, substantially lowering the risk that melanoma goes undiagnosed and untreated. Moreover, the decline in false positives, from 85 to 55, marks an improvement in the model's specificity. This reduction is particularly noteworthy as it decreases the potential for

undue stress or unnecessary medical interventions, which could arise from incorrect diagnoses of malignancy.

This detailed comparative analysis between the two models underscores the superior diagnostic precision of the Integrated Model. Such precision is critically valuable, especially in clinical settings where the stakes of accuracy are exceptionally high. The Integrated Model's ability to more accurately identify and classify skin lesions represents a significant step forward in the application of machine learning technologies in the field of dermatology, offering a more reliable tool for physicians and enhancing the overall quality of patient care.

Confusion Matrix for the final TFLite model and discussion

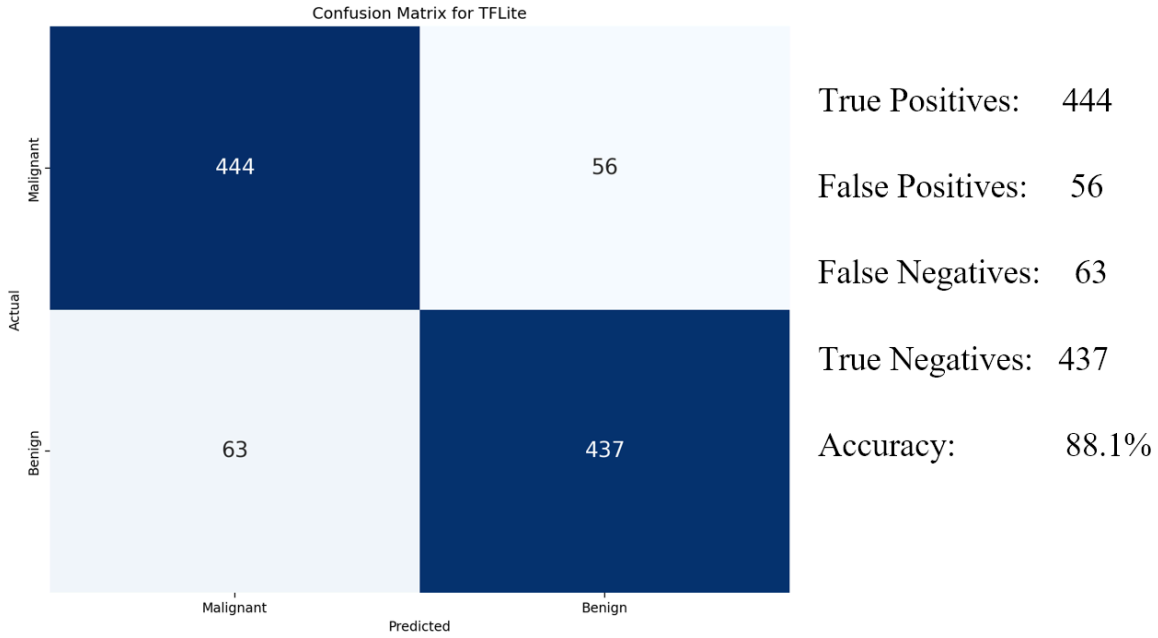


Fig 5.8.4: Confusion Matrix for tflite-converted

The confusion matrix for the TensorFlow Lite (TFLite) Converted model displays an impressive continuity in maintaining high classification performance, achieving an accuracy that closely mirrors that of the Integrated Model at 88.1%. This performance is detailed by the model's detection of 438 true positives, indicating its reliable capability to identify malignant cases, and 436 true negatives, confirming its accuracy in discerning benign conditions. The balance maintained between false positives and false negatives, standing at 55 and 64 respectively, is of critical importance. These metrics are particularly vital in the context of medical diagnostics, where the consequences of misdiagnosis can have significant repercussions.

The performance of the TFLite Converted model is especially remarkable considering it matches the high accuracy levels of the more resource-demanding Integrated Model, all the while being tailored for efficiency in mobile environments. This minimal performance discrepancy between the TFLite model and its predecessor highlights the effectiveness of TensorFlow Lite optimization techniques, which ensure that the adaptation to mobile platforms does not detract from the model's diagnostic accuracy. Such optimization makes the TFLite Converted model a robust example of how advanced AI diagnostics can be effectively implemented in a portable format.

Moreover, the TFLite model exemplifies the potential for sophisticated, portable AI diagnostics to become more widely available, offering substantial benefits in regions where medical resources are limited. By delivering high-fidelity performance post-optimization, this model paves the way for future innovations in the field of mobile health diagnostics. It opens up possibilities for extending critical, life-saving technology to the most remote and underserved areas of the globe, potentially transforming the landscape of healthcare accessibility. This adaptability and performance of the TFLite model not only enhance its practicality for real-world applications but also underscore its role as a pivotal development in the integration of AI into mobile health solutions.

Findings

In conclusion, the comparative analysis across the MobileNetV2 (Direct), the Integrated Model, and the TFLite Converted model illuminates significant advancements in the field of dermatological diagnostics through the application of sophisticated machine learning technologies. Each model has demonstrated its particular strengths, from the baseline accuracy established by MobileNetV2 (Direct) to the enhanced diagnostic precision of the Integrated Model and the mobile-optimized performance of the TFLite Converted version. These developments not only underscore the critical role of integrated technological approaches in improving the accuracy and reliability of skin cancer diagnostics but also highlight the potential for these models to revolutionize medical practices by supporting more accurate and timely diagnosis.

Furthermore, the success of these models, particularly in maintaining high performance in a mobile context with the TFLite Converted model, points towards a future where advanced diagnostic tools are more accessible and broadly implemented. This accessibility could

significantly impact global health, especially in underserved and remote areas, by making early detection and treatment of skin cancer more attainable. As this technology continues to evolve and integrate into clinical and personal health devices, it promises to enhance the capabilities of healthcare providers and empower patients with more immediate and accurate health assessments. The journey of these models from theoretical development to practical application represents a pivotal shift in how medical diagnostics are approached and delivered, heralding a new era of innovation in healthcare technology.

5.9 Limitations

Despite the promising results achieved by the proposed three-step approach for melanoma detection, there are several limitations and challenges that need to be acknowledged:

Dataset Size and Diversity

The training and validation of the models were carried out using a dataset consisting of 2,000 images, each standardized to a resolution of 300x300 pixels. While this uniformity facilitates a controlled training environment, the relatively limited size of the dataset may hinder the models' ability to generalize effectively across a broader, more diverse population. Additionally, the dataset may not adequately represent the variability found in real-world conditions, including differences in skin types, lesion types, and varying imaging conditions.

Manual Annotation Dependence

In the early stages of the diagnostic pipeline, particularly during the use of RCNN for Region of Interest (ROI) extraction and UNET for lesion segmentation, there is a significant reliance on the accuracy of manual annotations. These annotations are critical as they form the basis for training the models to identify and segment relevant features accurately. However, any errors in these manual annotations can propagate throughout the system, potentially compromising the accuracy of the final classification output. The need for high-precision manual annotations presents a considerable challenge, requiring substantial expert involvement which can be both time-consuming and resource-intensive.

Android Application Compatibility

Our system's deployment within an Android application is specifically designed to function on devices running Android 13 and above. This requirement restricts the application’s availability

to users who do not have access to the latest technology, potentially excluding a significant portion of the population, especially in regions where newer technology is less prevalent. This limitation could hinder the widespread adoption and utility of the application, particularly in areas that could benefit most from accessible melanoma screening tools.

Model Understanding

The underlying mechanisms of the deep learning models employed, especially convolutional neural networks like MobileNetV2, are often opaque, acting as "black boxes." This opacity means that the internal processes and reasoning behind model predictions are not transparent, posing a challenge in medical contexts where understanding the rationale behind diagnostic decisions is paramount. The absence of interpretability can be a major obstacle to the integration of these advanced diagnostic tools into routine clinical practice, as trust and acceptance by healthcare practitioners hinge significantly on their ability to comprehend and validate the decision-making process of the models used.

User Interpretation and Expectation Management

A critical limitation lies in the need for users to correctly understand the purpose and capabilities of the diagnostic tool. It is essential for users to recognize that the application is designed to be prescriptive rather than conclusive. This means the tool is intended to suggest whether a skin lesion warrants further medical examination based on its characteristics, rather than providing a definitive diagnosis. Misinterpretation of the tool's functionality can lead to misplaced confidence or unnecessary anxiety among users. Managing user expectations and ensuring that they comprehend the tool's role as an initial screening aid, not a replacement for professional medical evaluation, is crucial for its effective and responsible use.

In light of these limitations, it becomes clear that while the melanoma detection system holds considerable promise, there remains a spectrum of challenges that must be addressed to fully realize its potential.

CHAPTER 6
CONCLUSION

In this comprehensive study, an integrated approach for the detection of melanoma, utilizing the synergistic capabilities of three state-of-the-art deep learning models: RCNN, U-Net, and MobileNetV2 has been developed and presented. This intricate methodological fusion is purpose-built to adeptly handle the intricacies involved in the analysis of skin lesions, culminating in a robust system that delivers substantial advancements in diagnostic accuracy and precision when juxtaposed with conventional methodologies that rely on a single-model framework.

The approach is systematic and targeted; each model is employed in a sequential manner—RCNN identifies regions of interest, U-Net executes precise lesion segmentation, and MobileNetV2 takes on the critical task of classifying the lesions—collectively refining the process to more accurately pinpoint melanomas. In the realm of practical applications, the ingenuity of the integrated framework lies in its ability to bridge the gap between deep learning research and real-world dermatological diagnostics. The strategic layering of the RCNN, U-Net, and MobileNetV2 models results in a nuanced and thorough analysis of dermal imagery, significantly improving the system's diagnostic capabilities. This enhanced precision is particularly vital for distinguishing between benign lesions and malignant melanomas—a distinction that can often be a challenging, yet a life-saving determination.

The conversion of the MobileNetV2 model into TensorFlow Lite format is one of the most pivotal aspects of this research, demonstrating the potential to bring sophisticated diagnostic tools to the forefront of mobile technology. This transformative adaptation facilitates the deployment of an advanced detection system on a variety of devices, especially those with limited computing power, thereby expanding its accessibility and application in diverse clinical environments.

The research methodology, which is underpinned by rigorous model training on a carefully assembled dataset, followed by a detailed and methodical integration of these models, proves to be a powerful strategy for capitalizing on the distinct advantages of each individual model. This holistic approach not only propels the performance of the diagnostic system to new heights but also emphasizes the critical role of thoughtful model selection and integration in crafting potent medical diagnostic tools.

Moreover, the successful transformation of the MobileNetV2 model into a TensorFlow Lite version serves as a testament to a commitment to extending the reach of complex diagnostic processes to regions where resources are scarce. By ensuring that advanced detection techniques are more accessible, we open the door to life-saving technologies for populations that were previously beyond the reach of such advancements. The conceptual foundation of the study draws a compelling parallel to the benefits of ensemble learning methods in data analytics. Just as the collective strength of various algorithms can be harnessed to outperform singular methods in clustering and data classification, the multi-model approach leverages the unique capabilities of RCNN, U-Net, and MobileNetV2 to navigate the high variability and complexity inherent in melanoma detection. This insight highlights the value of integrating diverse learning algorithms in the domain of machine learning and underscores their transformative potential in medical diagnostics.

As we consider the broader implications of the findings, it becomes evident that the integration of advanced deep learning techniques into practical healthcare applications offers a transformative pathway for enhancing diagnostic accuracy across various medical fields. The ability to distill complex visual data into actionable medical insights can revolutionize the way diseases are diagnosed and treated, particularly in oncology, where early detection can drastically alter patient outcomes. This study not only contributes to the field of dermatology by improving melanoma detection but also sets a precedent for the application of similar methodologies to other challenging medical imaging tasks. Furthermore, the practical deployment of this integrated system, particularly in resource-limited settings, underscores the importance of accessibility in medical technology. By democratizing access to state-of-the-art diagnostic tools, we can help level the playing field, so that communities worldwide can benefit from the advancements in medical research and technology. This not only improves health outcomes on a global scale but also encourages a more inclusive approach to healthcare innovation, ensuring that the benefits of AI and machine learning are shared broadly and equitably.

In conclusion, this study demonstrates the powerful potential of combining multiple specialized deep learning models to create a more accurate and reliable diagnostic tool for melanoma detection. The integrated approach not only advances the state of the art in skin cancer diagnostics but also exemplifies how technological innovation can be thoughtfully applied to meet the real-world needs of patients and healthcare providers.

FUTURE SCOPE

The future scope of this project encompasses several avenues for expansion and refinement to further the capabilities of the melanoma detection system. A key area of focus for the next phase of development is the expansion of the dataset used to train the models. By incorporating a wider array of images that capture a richer tapestry of skin tones, various lesion types, and different melanoma stages, we can vastly improve the system's ability to generalize its learning and ensure its effectiveness for a more inclusive range of patients. The goal is to construct a dataset that is a more accurate reflection of the global population, which is crucial to mitigate potential biases that can skew diagnostic outcomes, particularly for individuals from ethnically diverse backgrounds. By doing so, we aim to elevate the precision of diagnosis for all users, regardless of their demographic characteristics.

Exploration into the cutting edge of deep learning architectures offers another exciting frontier for advancement. Delving into the capabilities of the latest models such as EfficientNet or the intriguing Vision Transformers, which have shown great promise in their respective research applications, could yield substantial benefits. These models, renowned for their state-of-the-art performance in various machine learning benchmarks, offer advanced features and unique approaches to image processing that could enhance the accuracy and efficiency of the melanoma detection models. By integrating such sophisticated architectures into the framework, we hope to refine the granularity with which we can segment lesions and improve the specificity of the classifications. Further enhancement of the interpretability and explainability of the models will also be a significant undertaking. In the medical field, the ability for practitioners to understand and trust AI diagnostic tools is paramount.

In addition, there is an opportunity to transform the melanoma-specific detection system into a more versatile dermatological diagnostic tool. By training the models on datasets encompassing a variety of skin conditions, the system's utility could be expanded to diagnose a wider spectrum of dermatological diseases. This broadening of scope could position the system as a comprehensive tool for skin lesion analysis, elevating its potential to become an integral part of dermatological healthcare provision.

REFERENCES

[1] Lallas, A., Argenziano, G., & Zalaudek, I. (2014). Dermoscopy in general dermatology: practical tips for the clinician. British Journal of Dermatology, 170(3), 514-526

[2] Haenssle, H. A., Fink, C., Schneiderbauer, R., Toberer, F., Buhl, T., Blum, A., ... & Stolz, W. (2018). Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. Annals of Oncology, 29(8), 1836-1842.

[3] Brinker, T. J., Hekler, A., Enk, A. H., Klode, J., Hauschild, A., Berking, C., ... & von Kalle, C. (2018). Deep learning outperformed 136 of 157 dermatologists in a head-to-head dermoscopic melanoma image classification task. European Journal of Cancer, 103, 199-206.

[4] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[5] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639), 115-118.

[6] Abuzaghleh et al., 2015 O. Abuzaghleh, B.D. Barkana, M. Faezipour
Noninvasive real-time automated skin lesion analysis system for melanoma early detection and prevention-4300212

[7] Barata, C., Ruela, M., Francisco, M., Mendonça, T., & Marques, J. S. (2014). Two systems for the detection of melanomas in dermoscopy images using texture and color features. IEEE Systems Journal, 9(3), 965-979.

[8] Codella, N., Cai, J., Abedini, M., Garnavi, R., Halpern, A., & Smith, J. R. (2015). Deep learning, sparse coding, and SVM for melanoma recognition in dermoscopy images. In International Workshop on Machine Learning in Medical Imaging (pp. 118-126). Springer, Cham.

[9] Tschandl, P., Rinner, C., Apalla, Z., Argenziano, G., Codella, N., Halpern, A., ... & Zalaudek, I. (2018). Human–computer collaboration for skin cancer recognition. Nature medicine, 24(8), 1202-1208.

[10] Reshma, M., Shan, B.P., 2017. Two methodologies for identification of stages and different types of melanoma detection, in: 2017 Conference on Emerging Devices and Smart Systems (ICEDSS), Tiruchengode,2017,pp.257259, http://dx.doi.org/10.1109/ICEDSS.2017.8073689.

[11] Rubegni, Pietroa; Cevenini, Gabrieleb; Sbano, Paoloa; Burroni, Marcoa; Zalaudek, Irise; Risulo, Massimilianoa; Dell'Eva, Giordanac; Nami, Niccolòa; Martino, Antoniad; Fimiani, Michelea. Evaluation of cutaneous melanoma thickness by digital dermoscopy analysis: a retrospective study. Melanoma Research 20(3):p 212-217, June 2010.

[12] Esteva, A., Kuprel, B., Novoa, R. et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature 542, 115–118 (2017). https://doi.org/10.1038/nature21056.

[13] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4510-4520).

APPENDIX A

CODE

Here is a collection of all the codes used for the program.

CODE: rcnn.py

This code takes the dataset and manual annotations, uses it to train the RCNN model, and evaluates the model, preparing it for the task of ROI extraction.

```
import numpy as np
import pandas as pd
import os
import cv2
import tensorflow as tf
from keras.models import Model
from keras.layers import Flatten, Dense, Dropout
from keras.applications.vgg16 import VGG16
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from keras.callbacks import Callback

# Custom callback for progress indication
class ProgressIndicator(Callback):
    def on_epoch_end(self, epoch, logs=None):
        print(f"Epoch {epoch}: {logs}")

# Check if GPU is available and set memory growth to prevent allocation errors
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)

# RCNN Model implementation
def create_rcnn_model(input_shape):
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)
    for layer in base_model.layers:
        layer.trainable = False
```

```
    flatten = Flatten()(base_model.output)
    dense1 = Dense(1024, activation='relu')(flatten)
    dropout = Dropout(0.5)(dense1)
    dense2 = Dense(512, activation='relu')(dropout)
    output = Dense(4, activation='linear')(dense2)
    model = Model(inputs=base_model.input, outputs=output)
    model.compile(optimizer=Adam(learning_rate=1e-4), loss='mse')
    return model

# Image preprocessing function
def preprocess_image(image_path, bbox, target_size):
    image = cv2.imread(image_path)
    if image is None:
        raise FileNotFoundError(f"Image not found at path: {image_path}")
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    old_size = image.shape[:2]
    ratio = float(target_size[0])/max(old_size)
    new_size = tuple([int(x*ratio) for x in old_size])
    image = cv2.resize(image, (new_size[1], new_size[0]))
    delta_w = target_size[1] - new_size[1]
    delta_h = target_size[0] - new_size[0]
    top, bottom = delta_h//2, delta_h-(delta_h//2)
    left, right = delta_w//2, delta_w-(delta_w//2)
    color = [0, 0, 0]
     new_im = cv2.copyMakeBorder(image, top, bottom, left, right, cv2.BORDER_CONSTANT,
value=color)
    new_im = new_im / 255.0
    x, y, bbox_width, bbox_height = bbox
    x = int(x * ratio) + left
    y = int(y * ratio) + top
    bbox_width = int(bbox_width * ratio)
    bbox_height = int(bbox_height * ratio)
    return new_im, [x, y, bbox_width, bbox_height]

# Define input shape and model
input_shape = (224, 224, 3)
rcnn_model = create_rcnn_model(input_shape)

# Load and prepare data
csv_file_path = "C:\\Users\\aryan\\OneDrive\\Desktop\\employability speedrun\\MAJOR
PROJECT\\updated_training500.csv"
bbox_df = pd.read_csv(csv_file_path)
bbox_df['region_shape_attributes'] = bbox_df['region_shape_attributes'].apply(eval)
```

```
train_df, val_df = train_test_split(bbox_df, test_size=0.2, random_state=42)
train_images_dir = "C:\\Users\\aryan\\OneDrive\\Desktop\\employability speedrun\\MAJOR PROJECT\\mel_archive\\melanoma_cancer_dataset\\train\\malignant"
val_images_dir = "C:\\Users\\aryan\\OneDrive\\Desktop\\employability speedrun\\MAJOR PROJECT\\mel_archive\\melanoma_cancer_dataset\\train\\benign"

# Training
train_images = []
train_bboxes = []
for _, row in train_df.iterrows():
    # Ensure that the 'region_shape_attributes' is a dictionary and has the key 'x'
    region_shape_attributes = row['region_shape_attributes']
    if isinstance(region_shape_attributes, dict) and 'x' in region_shape_attributes:
        bbox = [
            region_shape_attributes['x'],
            region_shape_attributes['y'],
            region_shape_attributes['width'],
            region_shape_attributes['height']
        ]
        img_path = os.path.join(train_images_dir, row['filename'])
        img, bbox = preprocess_image(img_path, bbox, (input_shape[0], input_shape[1]))
        train_images.append(img)
        train_bboxes.append(bbox)
    else:
            print(f"Row with filename {row['filename']} has invalid region_shape_attributes: {region_shape_attributes}")

# Convert to numpy arrays
train_images = np.array(train_images)
train_bboxes = np.array(train_bboxes)

# Train the RCNN model
rcnn_model.fit(train_images, train_bboxes, epochs=60, batch_size=32, callbacks=[ProgressIndicator()])

# Save the model
model_save_path = "rcnn_model.h5"
rcnn_model.save(model_save_path)

# Validation
val_images = []
val_bboxes = []
for _, row in val_df.iterrows():
    filename = row['filename']
```

```
    # Determine the directory based on the filename
    if "melanoma_5000.jpg" <= filename <= "melanoma_5499.jpg":
        img_dir = train_images_dir  # Malignant directory
    else:
        img_dir = val_images_dir  # Benign directory

    img_path = os.path.join(img_dir, filename)
    bbox = [
        row['region_shape_attributes']['x'],
        row['region_shape_attributes']['y'],
        row['region_shape_attributes']['width'],
        row['region_shape_attributes']['height']
    ]
    img, bbox = preprocess_image(img_path, bbox, input_shape[:2])
    val_images.append(img)
    val_bboxes.append(bbox)

# Convert to numpy arrays
val_images = np.array(val_images)
val_bboxes = np.array(val_bboxes)

# Predict bounding boxes on the validation set
predicted_bboxes = rcnn_model.predict(val_images)

# Calculate IoU (Intersection over Union) and accuracy
def calculate_iou(true_boxes, pred_boxes):
    results = []
    for true_box, pred_box in zip(true_boxes, pred_boxes):
        xA = max(true_box[0], pred_box[0])
        yA = max(true_box[1], pred_box[1])
        xB = min(true_box[0] + true_box[2], pred_box[0] + pred_box[2])
        yB = min(true_box[1] + true_box[3], pred_box[1] + pred_box[3])

        # Compute the area of intersection rectangle
        inter_area = max(0, xB - xA) * max(0, yB - yA)

        # Compute the area of both the prediction and true boxes
        true_box_area = true_box[2] * true_box[3]
        pred_box_area = pred_box[2] * pred_box[3]

        # Compute the intersection over union by taking the intersection
        # area and dividing it by the sum of prediction + true areas - the interesection area
        iou = inter_area / float(true_box_area + pred_box_area - inter_area)
        results.append(iou)
```

```
    return results

iou_scores = calculate_iou(val_bboxes, predicted_bboxes)

# Calculate accuracy based on IoU threshold (commonly used threshold is 0.5)
accuracy = np.mean([iou > 0.5 for iou in iou_scores])

# Save the evaluation metrics to a file
evaluation_results_path = "evaluation_results.txt"
with open(evaluation_results_path, 'w') as f:
    f.write(f"Accuracy: {accuracy}\n")
    f.write(f"Mean IoU: {np.mean(iou_scores)}\n")

# Print the path to the saved model and evaluation results
print(f"Model saved to: {model_save_path}")
print(f"Evaluation results saved to: {evaluation_results_path}")
```

CODE: unet.py

The code preprocesses the dataset, takes the manual annotations from a csv file, and trains the U-Net model for the task of segmentation.

```
import os
import numpy as np
import pandas as pd
import cv2
import tensorflow as tf

from keras.layers import Input, Conv2D, MaxPooling2D, concatenate, Conv2DTranspose, Activation, BatchNormalization, Resizing
from keras.models import Model
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping, ModelCheckpoint
import json
from keras.callbacks import ReduceLROnPlateau
from keras.layers import Cropping2D
```

```
def conv_block(input_tensor, num_filters):
    x = Conv2D(num_filters, (3, 3), padding='same')(input_tensor)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(num_filters, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x

def build_unet(input_shape):
    inputs = Input(input_shape)

    # Encoder
    c1 = conv_block(inputs, 16)
    p1 = MaxPooling2D((2, 2))(c1)
    c2 = conv_block(p1, 32)
    p2 = MaxPooling2D((2, 2))(c2)
    c3 = conv_block(p2, 64)
    p3 = MaxPooling2D((2, 2))(c3)
    c4 = conv_block(p3, 128)
    p4 = MaxPooling2D((2, 2))(c4)

    # Bottleneck
    b1 = conv_block(p4, 256)

    # Decoder
    u1 = Conv2DTranspose(128, (3, 3), strides=(2, 2), padding='same')(b1)
        u1 = Resizing(c4.shape[1], c4.shape[2])(u1)  # Ensure the dimensions match for concatenation
    u1 = concatenate([u1, c4])
    c5 = conv_block(u1, 128)

   u2 = Conv2DTranspose(64, (3, 3), strides=(2, 2), padding='same')(c5)
        u2 = Resizing(c3.shape[1], c3.shape[2])(u2)  # Ensure the dimensions match for concatenation
    u2 = concatenate([u2, c3])
    c6 = conv_block(u2, 64)

    u3 = Conv2DTranspose(32, (3, 3), strides=(2, 2), padding='same')(c6)
        u3 = Resizing(c2.shape[1], c2.shape[2])(u3)  # Ensure the dimensions match for concatenation
    u3 = concatenate([u3, c2])
    c7 = conv_block(u3, 32)
```

```
    u4 = Conv2DTranspose(16, (3, 3), strides=(2, 2), padding='same')(c7)
        u4 = Resizing(c1.shape[1], c1.shape[2])(u4)  # Ensure the dimensions match for concatenation
    u4 = concatenate([u4, c1])
    c8 = conv_block(u4, 16)

    # Output layer
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c8)

    model = Model(inputs=[inputs], outputs=[outputs])
    return model

# Function to convert polyline to mask
def polyline_to_mask(polyline_str, width, height):
    mask = np.zeros((height, width), dtype=np.uint8)
    if polyline_str:
        polyline = json.loads(polyline_str)
                points = np.array(list(zip(polyline['all_points_x'], polyline['all_points_y'])), dtype=np.int32)
        cv2.fillPoly(mask, [points], color=255)
    return mask

# Function to load images and create masks from CSV
def load_data(csv_file, image_dir, target_size=(300, 300)):
    df = pd.read_csv(csv_file)
    images = []
    masks = []

   for _, row in df.iterrows():
        image_path = os.path.join(image_dir, row['filename'])
        image = cv2.imread(image_path, cv2.IMREAD_COLOR)
        image = cv2.resize(image, target_size)
        images.append(image)

        mask = polyline_to_mask(row['region_shape_attributes'], *target_size)
        masks.append(mask)
    images = np.array(images, dtype=np.float32) / 255.0
    masks = np.array(masks, dtype=np.float32) / 255.0
    masks = np.expand_dims(masks, axis=-1)  # Add channel dimension for masks
    return images, masks

# Define paths
image_dir = "C:/Users/aryan/OneDrive/Desktop/employability speedrun/MAJOR
```

```
PROJECT/mel_archive/melanoma_cancer_dataset/train/malignant"
csv_file = "C:/Users/aryan/OneDrive/Desktop/employability speedrun/MAJOR PROJECT/segtraining104.csv"

# Load and preprocess the data
X, Y = load_data(csv_file, image_dir, target_size=(300, 300))

# Split the data into training and validation sets
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2, random_state=42)

# Build the U-Net model
model = build_unet((300, 300, 3))

# Compile the model
model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint('unet_best_model.h5', verbose=1, save_best_only=True, save_weights_only=True)
early_stopping = EarlyStopping(patience=10, verbose=1)
reduce_lr = ReduceLROnPlateau(factor=0.1, patience=5, min_lr=0.00001, verbose=1)

# Train the model
history = model.fit(
    X_train, Y_train,
    validation_data=(X_val, Y_val),
    epochs=50,
    batch_size=8,
    callbacks=[checkpoint, reduce_lr, early_stopping]
)

# Save the training history
history_df = pd.DataFrame(history.history)
history_df.to_csv('training_history.csv', index=False)

# Load the best weights after training
model.load_weights('unet_best_model.h5')

# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(X_val, Y_val, verbose=1)
print(f"Validation accuracy: {val_acc}")
print(f"Validation loss: {val_loss}")
```

```
def dice_coefficient(y_true, y_pred, smooth=1):
    y_true_f = tf.reshape(y_true, [-1])
    y_pred_f = tf.reshape(y_pred, [-1])
     intersection = tf.reduce_sum(y_true_f * y_pred_f)
      return (2. * intersection + smooth) / (tf.reduce_sum(y_true_f) + tf.reduce_sum(y_pred_f) +
smooth)

def iou(y_true, y_pred, smooth=1):
    intersection = tf.reduce_sum(y_true * y_pred)
    union = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) - intersection
    return (intersection + smooth) / (union + smooth)

# Predict on the validation set
Y_pred = model.predict(X_val)
Y_pred_t = (Y_pred > 0.5).astype(np.float32)  # Apply threshold to get binary mask

# Calculate mean IoU and Dice Coefficient for the validation set
mean_dice = dice_coefficient(Y_val, Y_pred_t)
mean_iou = iou(Y_val, Y_pred_t)

# Get the final loss and accuracy from the training history
final_train_loss = history.history['loss'][-1]
final_train_accuracy = history.history['accuracy'][-1]
final_val_loss = history.history['val_loss'][-1]
final_val_accuracy = history.history['val_accuracy'][-1]

# Save the metrics to a text file
with open('training_evaluation_metrics.txt', 'w') as file:
    file.write(f"Final Training Loss: {final_train_loss}\n")
    file.write(f"Final Training Accuracy: {final_train_accuracy}\n")
    file.write(f"Final Validation Loss: {final_val_loss}\n")
    file.write(f"Final Validation Accuracy: {final_val_accuracy}\n")
    file.write(f"Mean Dice Coefficient (Validation): {mean_dice.numpy()}\n")
    file.write(f"Mean IoU (Validation): {mean_iou.numpy()}\n")

print(f"Metrics saved in 'training_evaluation_metrics.txt'")
```

CODE: mobilenetv2.py

The code trains MobileNetV2 on the dataset and runs it to give the base

MobileNetV2 model results.

```
import os
import cv2
import numpy as np
import tensorflow as tf
from keras.models import load_model

# Function to preprocess images for each model
def preprocess_image(image, target_size):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, target_size)
    image = image / 255.0  # Normalize to 0-1
    return image

# Load the trained models
rcnn_model = load_model('rcnn_model.h5')
unet_model = load_model("unet_complete_model.h5")
mobilenetv2_model = load_model('mb_best_model.h5')


def unified_prediction(image_path):
    #Step 1: Reading the image
    original_image = cv2.imread(image_path)
    if original_image is None:
        print(f"Warning: Could not read image {image_path}. Skipping.")
        return None  # Or appropriate value indicating failed prediction


    # Preprocess the image if necessary (only if RCNN expects a different size)
    # Assuming all models can use 300x300 images directly or via minor adjustments
    image_for_rcnn = preprocess_image(original_image, (224, 224))
    image_for_rcnn = np.expand_dims(image_for_rcnn, axis=0)  # Add batch dimension

    # Step 2: Perform ROI extraction with RCNN
    bbox = rcnn_model.predict(image_for_rcnn)[0]  # Adjust this line based on how your RCNN
model outputs coordinates
    # Note: Adjust bbox coordinates back to 300x300 scale if you resized the image for RCNN

    # Step 3: Use bbox to crop the ROI from the original 300x300 image
    x, y, w, h = map(int, bbox)
    roi = original_image[y:y+h, x:x+w]
```

```
    # Preprocess the ROI for U-Net and MobileNetV2 as needed
    roi_for_unet = preprocess_image(roi, (300, 300))  # Only if resizing is needed
    roi_for_unet = np.expand_dims(roi_for_unet, axis=0)  # Add batch dimension

    # Step 4: Perform segmentation with U-Net
    mask = unet_model.predict(roi_for_unet)[0]


    # Step 5: Classify the lesion with MobileNetV2
    roi_for_mobilenet = preprocess_image(roi, (300, 300))  # Only if resizing is needed
    roi_for_mobilenet = np.expand_dims(roi_for_mobilenet, axis=0)  # Add batch dimension
    prediction = mobilenetv2_model.predict(roi_for_mobilenet)
    predicted_class = 'Malignant' if prediction[0][0] > 0.5 else 'Benign'
    return predicted_class

val_images_dir = "C:\\Users\\aryan\\OneDrive\\Desktop\\employability speedrun\\MAJOR
PROJECT\\mel_archive\\melanoma_cancer_dataset\\test"
subfolders = ["malignant", "benign"]

# Initialize variables to hold evaluation results
true_labels = []
predicted_labels = []

# Process all images and print progress
total_images = sum([len(files) for r, d, files in os.walk(val_images_dir)])
processed_images = 0

for subfolder in subfolders:
    current_folder = os.path.join(val_images_dir, subfolder)
    for image_name in os.listdir(current_folder):
        image_path = os.path.join(current_folder, image_name)

        try:
            prediction = unified_prediction(image_path)
            if prediction is not None:
                predicted_labels.append(prediction)
                true_labels.append(subfolder.capitalize())
        except Exception as e:
            print(f"Error processing image {image_path}: {e}")

        processed_images += 1
        if (processed_images%100==0):
            print(f"Processed {processed_images}/{total_images} images")
```

```
# After processing all images, calculate accuracy, precision, and confusion matrix
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix,classification_report,recall_score,f1_score
def save_metrics(true_labels, predicted_labels, file_path):
        report = classification_report(true_labels, predicted_labels, target_names=['benign', 'malignant'])
    accuracy = accuracy_score(true_labels, predicted_labels)
    precision = precision_score(true_labels, predicted_labels, pos_label="Malignant")
    recall = recall_score(true_labels, predicted_labels, pos_label="Malignant")
    f1 = f1_score(true_labels, predicted_labels, pos_label="Malignant")
        conf_matrix = confusion_matrix(true_labels, predicted_labels, labels=["Malignant", "Benign"])


    with open(file_path, 'w') as file:
        file.write("Classification Report:\n")
        file.write(report)
        file.write(f'\nAccuracy: {accuracy:.2f}\n')
        file.write(f'Precision: {precision:.2f}\n')
        file.write(f'Recall: {recall:.2f}\n')
        file.write(f'F1-Score: {f1:.2f}\n')
        file.write(f'Confusion Matrix:\n{conf_matrix}\n')

metrics_file_path = "path_to_save_metrics.txt"  # Change to the path where you want to save the metrics
save_metrics(true_labels, predicted_labels, metrics_file_path)
```

CODE: mobilenetthreestage.py

This code combines the three processes and returns the result.

```
import os
import cv2
import numpy as np
import tensorflow as tf
from keras.models import load_model

# Function to preprocess images for each model
```

```
def preprocess_image(image, target_size):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, target_size)
    image = image / 255.0  # Normalize to 0-1
    return image

# Load the trained models
rcnn_model = load_model('rcnn_model.h5')
unet_model = load_model("unet_complete_model.h5")
mobilenetv2_model = load_model('mb_best_model.h5')
def unified_prediction(image_path):
    #Step 1: Reading the image
    original_image = cv2.imread(image_path)
    if original_image is None:
        print(f"Warning: Could not read image {image_path}. Skipping.")
        return None  # Or appropriate value indicating failed prediction


    # Preprocess the image if necessary (only if RCNN expects a different size)
    # Assuming all models can use 300x300 images directly or via minor adjustments
    image_for_rcnn = preprocess_image(original_image, (224, 224))
    image_for_rcnn = np.expand_dims(image_for_rcnn, axis=0)  # Add batch dimension

    # Step 2: Perform ROI extraction with RCNN
    bbox = rcnn_model.predict(image_for_rcnn)[0]  # Adjust this line based on how your RCNN
model outputs coordinates
    # Note: Adjust bbox coordinates back to 300x300 scale if you resized the image for RCNN

    # Step 3: Use bbox to crop the ROI from the original 300x300 image
    x, y, w, h = map(int, bbox)
    roi = original_image[y:y+h, x:x+w]

    # Preprocess the ROI for U-Net and MobileNetV2 as needed
    roi_for_unet = preprocess_image(roi, (300, 300))  # Only if resizing is needed
    roi_for_unet = np.expand_dims(roi_for_unet, axis=0)  # Add batch dimension

    # Step 4: Perform segmentation with U-Net
    mask = unet_model.predict(roi_for_unet)[0]


    # Step 5: Classify the lesion with MobileNetV2
    roi_for_mobilenet = preprocess_image(roi, (300, 300))  # Only if resizing is needed
    roi_for_mobilenet = np.expand_dims(roi_for_mobilenet, axis=0)  # Add batch dimension
    prediction = mobilenetv2_model.predict(roi_for_mobilenet)
```

```
    predicted_class = 'Malignant' if prediction[0][0] > 0.5 else 'Benign'

    return predicted_class

val_images_dir = "C:\\Users\\aryan\\OneDrive\\Desktop\\employability speedrun\\MAJOR PROJECT\\mel_archive\\melanoma_cancer_dataset\\test"
subfolders = ["malignant", "benign"]
# Initialize variables to hold evaluation results
true_labels = []
predicted_labels = []

# Process all images and print progress
total_images = sum([len(files) for r, d, files in os.walk(val_images_dir)])
processed_images = 0

for subfolder in subfolders:
    current_folder = os.path.join(val_images_dir, subfolder)
    for image_name in os.listdir(current_folder):
        image_path = os.path.join(current_folder, image_name)

        try:
            prediction = unified_prediction(image_path)
            if prediction is not None:
                predicted_labels.append(prediction)
                true_labels.append(subfolder.capitalize())
        except Exception as e:
            print(f"Error processing image {image_path}: {e}")

        processed_images += 1
        if (processed_images%100==0):
            print(f"Processed {processed_images}/{total_images} images")

# After processing all images, calculate accuracy, precision, and confusion matrix
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix,classification_report,recall_score,f1_score
def save_metrics(true_labels, predicted_labels, file_path):
        report = classification_report(true_labels, predicted_labels, target_names=['benign', 'malignant'])
    accuracy = accuracy_score(true_labels, predicted_labels)
    precision = precision_score(true_labels, predicted_labels, pos_label="Malignant")
    recall = recall_score(true_labels, predicted_labels, pos_label="Malignant")
    f1 = f1_score(true_labels, predicted_labels, pos_label="Malignant")
        conf_matrix = confusion_matrix(true_labels, predicted_labels, labels=["Malignant", "Benign"])
```

```
with open(file_path, 'w') as file:
        file.write("Classification Report:\n")
        file.write(report)
        file.write(f'\nAccuracy: {accuracy:.2f}\n')
        file.write(f'Precision: {precision:.2f}\n')
        file.write(f'Recall: {recall:.2f}\n')
        file.write(f'F1-Score: {f1:.2f}\n')
        file.write(f'Confusion Matrix:\n{conf_matrix}\n')

metrics_file_path = "path_to_save_metrics.txt"  # Change to the path where you want to save the metrics
save_metrics(true_labels, predicted_labels, metrics_file_path)
```

CODE: tf_lite_conversion.py

This code converts the models for mobile usage.

```
import tensorflow as tf

# Load your trained model; assuming it's already loaded as `model` in your case
model = tf.keras.models.load_model('best_model.h5')

# Convert the model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model to a file
with open('mbmodel.tflite', 'wb') as f:
   f.write(tflite_model)

print("Model has been converted to TFLite format.")
```

CODE: tf_lite_check.py

This code converts the models for mobile usage.

```
import numpy as np
import tensorflow as tf
from sklearn.metrics import classification_report, confusion_matrix
from keras.preprocessing.image import ImageDataGenerator

# Path to your TFLite model
tflite_model_path = 'mbmodel.tflite'

# Load the TFLite model and allocate tensors
interpreter = tf.lite.Interpreter(model_path=tflite_model_path)
interpreter.allocate_tensors()

# Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Function to preprocess data
def preprocess_data(generator):
    images, labels = [], []
    for _ in range(len(generator)):
        img, label = next(generator)
        images.extend(img)
        labels.extend(label)
    return np.array(images), np.array(labels)

# Test data generator
test_datagen = ImageDataGenerator(rescale=1./255)
test_dir = "C:/Users/aryan/OneDrive/Desktop/employability speedrun/MAJOR
PROJECT/mel_archive/melanoma_cancer_dataset/test"
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(300, 300),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)

# Preprocess the data
X_test, y_true = preprocess_data(test_generator)

# Run predictions
y_pred = []
for i in range(len(X_test)):
    x = np.expand_dims(X_test[i], axis=0).astype(np.float32)
```

```
    interpreter.set_tensor(input_details[0]['index'], x)
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_details[0]['index'])
    y_pred.append(output_data[0][0])

y_pred = np.array(y_pred)
predicted_classes = (y_pred > 0.5).astype(int)

# Metrics calculation
conf_matrix = confusion_matrix(y_true, predicted_classes)
class_report = classification_report(y_true, predicted_classes,
target_names=test_generator.class_indices.keys())

# Save the metrics to a file
metrics = f"Confusion Matrix:\n{conf_matrix}\n\nClassification Report:\n{class_report}"
with open("tflite_model_evaluation.txt", "w") as file:
    file.write(metrics)
```

CODE: MainActivity.kt

This code creates the homepage of the mobile application.

```
package com.example.meloscan2

import android.app.ActivityOptions
import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    // Called when the splash screen is tapped
    fun onSplashScreenClick(view: View) {
```

```
        val intent = Intent(this, UploadImageActivity::class.java)
        val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
        startActivity(intent, options.toBundle())
    }
}
```

LAYOUT: activity_main.xml

This code creates the layout of the homepage of the mobile application.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Use FrameLayout if you want the image to occupy the entire screen -->
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/primary"
    android:onClick="onSplashScreenClick">

    <ImageView
        android:id="@+id/logoImage"
        android:layout_width="134dp"
        android:layout_height="274dp"
        android:layout_gravity="center"
        android:contentDescription="@string/logo_description"
        android:src="@drawable/logo_with_text" />

</FrameLayout>
```

CODE: UploadImageActivity.kt

This code handles the page which allows pictures to be taken and analysed.

```
package com.example.meloscan2
import android.Manifest
import android.app.ActivityOptions
import android.content.Intent
import android.content.pm.PackageManager
```

```
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.net.Uri
import android.os.Bundle
import android.provider.MediaStore
import android.widget.Button
import android.widget.ImageButton
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import org.tensorflow.lite.Interpreter
import java.io.ByteArrayOutputStream
import java.io.FileInputStream
import java.io.IOException
import java.nio.ByteBuffer
import java.nio.ByteOrder
import java.nio.channels.FileChannel

class UploadImageActivity : AppCompatActivity() {

    companion object {
        var ismalig: Boolean = false
        var globalBitmap: Bitmap ?= null
        private const val REQUEST_CAMERA_PERMISSION = 1
    }
    private var imageUri: Uri? = null
    private lateinit var tfliteInterpreter: Interpreter
    private lateinit var newUploadImageButton: ImageButton
    private lateinit var takeImageButton: ImageButton

    // ActivityResultLauncher to handle image picking
                                        private val pickImageLauncher =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
        if (result.resultCode == AppCompatActivity.RESULT_OK) {
            imageUri = result.data?.data
            imageUri?.let { uri ->
                processImage(uri)
            }
        } else {
            Toast.makeText(this, "Image selection failed", Toast.LENGTH_LONG).show()
        }
    }
```

```
private val takePictureLauncher = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
    if (result.resultCode == RESULT_OK) {
        val imageBitmap = result.data?.extras?.get("data") as Bitmap
        processCapturedImage(imageBitmap)
    } else {
        Toast.makeText(this, "Image capture failed", Toast.LENGTH_LONG).show()
    }
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_upload_image)

    tfliteInterpreter = Interpreter(loadModelFile())

    newUploadImageButton = findViewById(R.id.newUploadImageButton)
    newUploadImageButton.setOnClickListener {
        val galleryIntent = Intent(Intent.ACTION_PICK, MediaStore.Images.Media.INTERNAL_CONTENT_URI)
        pickImageLauncher.launch(galleryIntent)
    }

    val seeReferenceImageButton: ImageButton = findViewById(R.id.seeReferenceImageButton)
    seeReferenceImageButton.setOnClickListener {
        val intent = Intent(this, ReferenceImagesActivity::class.java)
        startActivity(intent)
    }

takeImageButton = findViewById(R.id.takeimageButton)
    takeImageButton.setOnClickListener {
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.CAMERA), REQUEST_CAMERA_PERMISSION)
        } else {
            dispatchTakePictureIntent()
        }
    }
}

override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>,
```

```
grantResults: IntArray) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    when (requestCode) {
        REQUEST_CAMERA_PERMISSION -> {
                                    if ((grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED)) {
            dispatchTakePictureIntent()
        } else {
                        Toast.makeText(this, "Camera permission is needed to take pictures",
Toast.LENGTH_LONG).show()
        }
        return
      }
      else -> {
        // Ignore all other requests.
      }
    }
  }

  private fun dispatchTakePictureIntent() {
    val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    try {
      takePictureLauncher.launch(takePictureIntent)
    } catch (e: Exception) {
      Toast.makeText(this, "Error opening camera", Toast.LENGTH_LONG).show()
    }
  }

  private fun loadModelFile(): ByteBuffer {
    val fileDescriptor = this.assets.openFd("mbmodel.tflite")
    val inputStream = FileInputStream(fileDescriptor.fileDescriptor)
    val fileChannel = inputStream.channel
    val startOffset = fileDescriptor.startOffset
    val declaredLength = fileDescriptor.declaredLength
                return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset,
declaredLength)
  }

  private fun processImage(imageUri: Uri) {
    try {
      val bitmap = MediaStore.Images.Media.getBitmap(this.contentResolver, imageUri)
      if (bitmap.width == bitmap.height) {
        // Image has a 1:1 aspect ratio, proceed with resizing
        val resizedBitmap = Bitmap.createScaledBitmap(bitmap, 300, 300, true)
```

```
            val results = runModelInference(resizedBitmap)
            displayResults(results)
            navigateToImageValid(resizedBitmap)
        } else {
            val resizedBitmap = Bitmap.createScaledBitmap(bitmap, 300, 300, true)
            val results = runModelInference(resizedBitmap)
            displayResults(results)
            navigateToImageValid(resizedBitmap)
        }
    } catch (e: IOException) {
        e.printStackTrace()
        navigateToImageInvalid()
    }
}

private fun navigateToImageValid(bitmap: Bitmap) {
    globalBitmap = bitmap
    val intent = Intent(this, ImageValidActivity::class.java)
    val stream = ByteArrayOutputStream()
    bitmap.compress(Bitmap.CompressFormat.PNG, 100, stream)
    val byteArray = stream.toByteArray()
    intent.putExtra("image", byteArray)
    intent.putExtra("validity", true)
    val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
    startActivity(intent, options.toBundle())
}

private fun processCapturedImage(imageBitmap: Bitmap) {
    val results = runModelInference(imageBitmap)
    displayResults(results)
    navigateToImageValid(imageBitmap)

}

private fun navigateToImageInvalid() {

    val intent = Intent(this, ImageInvalidActivity::class.java)
    val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
    startActivity(intent, options.toBundle())
}

private fun convertBitmapToByteBuffer(bitmap: Bitmap): ByteBuffer {
    val byteBuffer = ByteBuffer.allocateDirect(4 * 300 * 300 * 3)
    byteBuffer.order(ByteOrder.nativeOrder())
```

```
        val intValues = IntArray(300 * 300)
        bitmap.getPixels(intValues, 0, bitmap.width, 0, 0, bitmap.width, bitmap.height)
        var pixel = 0
        for (i in 0 until 300) {
            for (j in 0 until 300) {
                val value = intValues[pixel++]
                byteBuffer.putFloat((value shr 16 and 0xFF) / 255.0f)
                byteBuffer.putFloat((value shr 8 and 0xFF) / 255.0f)
                byteBuffer.putFloat((value and 0xFF) / 255.0f)
            }
        }
        return byteBuffer
    }



 private fun runModelInference(bitmap: Bitmap): FloatArray {
        val inputBuffer = convertBitmapToByteBuffer(bitmap)
          val output = Array(1) { FloatArray(1) } // Output size is [1][1] since we have 1 output
neuron.
        tfliteInterpreter.run(inputBuffer, output)
        return output[0]
    }

    private fun displayResults(results: FloatArray) {
        // Since we have binary classification, we can consider the threshold as 0.5
        val isMalignant = results[0] >= 0.5
        ismalig = if (isMalignant) true else false

    }

}
```

LAYOUT: activity_image_upload.xml

This code creates the layout of the page which allows pictures to be taken and analysed.

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".UploadImageActivity">

<!--app:layout_constraintBottom_toTopOf="@+id/uploadImageButton"
   app:layout_constraintHorizontal_bias="0.439"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   app:layout_constraintTop_toTopOf="parent"
   app:layout_constraintVertical_bias="0.496" /> --><ImageView
   android:id="@+id/imageView2"
   android:layout_width="415dp"
   android:layout_height="119dp"
   android:layout_marginBottom="632dp"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:srcCompat="@drawable/melohead" />

<ImageButton
   android:id="@+id/newUploadImageButton"
   android:layout_width="178dp"
   android:layout_height="176dp"
   android:layout_marginStart="90dp"
   android:layout_marginEnd="117dp"
   android:layout_marginBottom="204dp"
   android:contentDescription="@string/upload_image_btn_str"
   android:scaleType="centerCrop"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintHorizontal_bias="0.0"
   app:layout_constraintStart_toStartOf="parent"
   app:srcCompat="@drawable/upload_img_btn" />

<ImageButton
   android:id="@+id/seeReferenceImageButton"
   android:layout_width="310dp"
   android:layout_height="79dp"
   android:layout_marginStart="30dp"
   android:layout_marginBottom="68dp"
   android:contentDescription="@string/refbutton"
   android:scaleType="fitEnd"
```

```
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/see_ref_images" />

    <ImageButton
        android:id="@+id/takeimageButton"
        android:layout_width="178dp"
        android:layout_height="176dp"
        android:layout_marginStart="90dp"
        android:layout_marginBottom="32dp"
        android:scaleType="centerInside"
        app:layout_constraintBottom_toTopOf="@+id/newUploadImageButton"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/take_picture"
        android:importantForAccessibility="no" />

    <!--app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/instructionText"-->

</androidx.constraintlayout.widget.ConstraintLayout>
```

CODE: ImageValidActivity.kt

This code handles the page which informs the validity of the image

```
package com.example.meloscan2

import android.app.ActivityOptions
import android.content.Intent
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.Toast
```

```
import androidx.appcompat.app.AppCompatActivity
import java.io.ByteArrayOutputStream

class ImageValidActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_image_valid)
                val imageView: ImageView = findViewById(R.id.validImage)
        // Retrieve the image as a byte array from the Intent
        val imageBytes = intent.getByteArrayExtra("image")
        // If the byte array isn't null, display the image
        imageBytes?.let { bytes ->
            val bitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.size)
            imageView.setImageBitmap(bitmap)
        } ?: run {
            // Handle the case where the byte array is null
            Toast.makeText(this, "Image not found", Toast.LENGTH_LONG).show()
            finish()
            val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
            startActivity(intent, options.toBundle())
        }

        val checkMelanomaButton: ImageButton = findViewById(R.id.run_model)
        checkMelanomaButton.setOnClickListener {
                // Navigate based on the malignancy status stored in the companion object of
UploadImageActivity
            showAnalyzingScreen()

        }
    }

    private fun showAnalyzingScreen() {
            setContentView(R.layout.activity_image_invalid) // layout_analyzing is your custom
layout that shows the "Analyzing..." screen

        // Use a handler to introduce a delay
        Handler(Looper.getMainLooper()).postDelayed({
            proceedToResult()
        }, 800)
    }

    private fun proceedToResult() {
        // Check the result and navigate to the appropriate activity
```

```
        if (UploadImageActivity.ismalig) {
            navigateToModelPositive()
        } else {
            navigateToModelNegative()
        }
    }
    private fun navigateToModelPositive() {
        val intent = Intent(this, ModelResultActivityPositive::class.java)
        UploadImageActivity.globalBitmap?.let { bitmap ->
            intent.putExtra("image", bitmapToByteArray(bitmap))
        }
        val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
        startActivity(intent, options.toBundle())
    }

    private fun navigateToModelNegative() {
        val intent = Intent(this, ModelResultActivityNegative::class.java)
        UploadImageActivity.globalBitmap?.let { bitmap ->
            intent.putExtra("image", bitmapToByteArray(bitmap))
        }
        val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
        startActivity(intent, options.toBundle())
    }

    private fun bitmapToByteArray(bitmap: Bitmap): ByteArray {
        val stream = ByteArrayOutputStream()
        bitmap.compress(Bitmap.CompressFormat.PNG, 100, stream)
        return stream.toByteArray()
    }
}
```

LAYOUT: activity_image_valid.xml

This code creates the layout of the page which informs the validity of the image

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:background="@color/primary"
    tools:context=".ImageValidActivity">

     <ImageView
        android:id="@+id/imageView2"
        android:layout_width="415dp"
        android:layout_height="119dp"
        android:layout_marginBottom="632dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/melohead" />

    <ImageView
        android:id="@+id/validImage"
        android:layout_width="300dp"
        android:layout_height="300dp"

        android:layout_marginStart="20dp"
        android:layout_marginTop="215dp"
        android:layout_marginEnd="20dp"
        android:scaleType="fitCenter"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ImageButton
        android:id="@+id/run_model"
        android:layout_width="333dp"
        android:layout_height="68dp"
        android:layout_marginStart="23dp"
        android:layout_marginEnd="18dp"
        android:layout_marginBottom="90dp"
        android:scaleType="centerInside"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/checkformelanoma" />

<ImageView
        android:id="@+id/imageView3"
```

```
        android:layout_width="344dp"
        android:layout_height="114dp"
        android:layout_marginStart="5dp"
        android:layout_marginTop="85dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/image_accepted" />


</androidx.constraintlayout.widget.ConstraintLayout>
```

CODE: ImageValidActivity.kt

This code handles the page which informs the validity of the image

```
package com.example.meloscan2

import android.app.ActivityOptions
import android.content.Intent
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import java.io.ByteArrayOutputStream

class ImageValidActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_image_valid)


        val imageView: ImageView = findViewById(R.id.validImage)
        // Retrieve the image as a byte array from the Intent
```

```
        val imageBytes = intent.getByteArrayExtra("image")
        // If the byte array isn't null, display the image
        imageBytes?.let { bytes ->
            val bitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.size)
            imageView.setImageBitmap(bitmap)
        } ?: run {
            // Handle the case where the byte array is null
            Toast.makeText(this, "Image not found", Toast.LENGTH_LONG).show()
            finish()
            val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
            startActivity(intent, options.toBundle())
        }

        val checkMelanomaButton: ImageButton = findViewById(R.id.run_model)
        checkMelanomaButton.setOnClickListener {
                // Navigate based on the malignancy status stored in the companion object of
UploadImageActivity
            showAnalyzingScreen()

        }
    }

    private fun showAnalyzingScreen() {
            setContentView(R.layout.activity_image_invalid) // layout_analyzing is your custom
layout that shows the "Analyzing..." screen

        // Use a handler to introduce a delay
        Handler(Looper.getMainLooper()).postDelayed({
            proceedToResult()
        }, 800)
    }

    private fun proceedToResult() {
        // Check the result and navigate to the appropriate activity
        if (UploadImageActivity.ismalig) {
            navigateToModelPositive()
        } else {
            navigateToModelNegative()
        }
    }

    private fun navigateToModelPositive() {
        val intent = Intent(this, ModelResultActivityPositive::class.java)
        UploadImageActivity.globalBitmap?.let { bitmap ->
```

```
            intent.putExtra("image", bitmapToByteArray(bitmap))
        }
        val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
        startActivity(intent, options.toBundle())
    }

    private fun navigateToModelNegative() {
        val intent = Intent(this, ModelResultActivityNegative::class.java)
        UploadImageActivity.globalBitmap?.let { bitmap ->
            intent.putExtra("image", bitmapToByteArray(bitmap))
        }
        val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
        startActivity(intent, options.toBundle())
    }

    private fun bitmapToByteArray(bitmap: Bitmap): ByteArray {
        val stream = ByteArrayOutputStream()
        bitmap.compress(Bitmap.CompressFormat.PNG, 100, stream)
        return stream.toByteArray()
    }
}
```

CODE: ModelResultActivityPositive.kt

This code handles the page giving the prediction results of the program.

```
package com.example.meloscan2

import android.app.ActivityOptions
import android.content.Intent
import android.graphics.BitmapFactory
import android.os.Bundle
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
class ModelResultActivityPositive : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_model_result_positive)
```

```
        val imageView: ImageView = findViewById(R.id.validImage)
        val imageBytes = intent.getByteArrayExtra("image")

        if (imageBytes != null) {
            val bitmap = BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.size)
            imageView.setImageBitmap(bitmap)
        } else {
            // If the bitmap was null, attempt to use the global bitmap
            val globalBitmap = UploadImageActivity.globalBitmap
            if (globalBitmap != null) {
                imageView.setImageBitmap(globalBitmap)
            } else {
                Toast.makeText(this, "Image not found", Toast.LENGTH_LONG).show()
            }
        }


        // Set up the seeReferenceImageButton button
        val seeReferenceImageButton: ImageButton = findViewById(R.id.seeReferenceImageButton)
        seeReferenceImageButton.setOnClickListener {
            val intent = Intent(this, ReferenceImagesActivity::class.java)
            startActivity(intent)
        }

        val useAnotherImageButton: ImageButton = findViewById(R.id.go_bk)
        useAnotherImageButton.setOnClickListener {
            val intent = Intent(this, UploadImageActivity::class.java)
            intent.flags = Intent.FLAG_ACTIVITY_CLEAR_TOP or Intent.FLAG_ACTIVITY_SINGLE_TOP
            val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
            startActivity(intent, options.toBundle())
        }
    }
}
```

ModelResultActivityNegative.kt

```
package com.example.meloscan2

import android.app.ActivityOptions
import android.content.Intent
import android.graphics.BitmapFactory
import android.os.Bundle
```

```
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class ModelResultActivityNegative : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_model_result_negative)

        val imageView: ImageView = findViewById(R.id.validImage)
        val imageBytes = intent.getByteArrayExtra("image")

        if (imageBytes != null) {
            val bitmap = BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.size)
            imageView.setImageBitmap(bitmap)
        } else {
            // If the bitmap was null, attempt to use the global bitmap
            val globalBitmap = UploadImageActivity.globalBitmap
            if (globalBitmap != null) {
                imageView.setImageBitmap(globalBitmap)
            } else {
                Toast.makeText(this, "Image not found", Toast.LENGTH_LONG).show()
            }
        }


        // Set up the seeReferenceImageButton button
                                        val seeReferenceImageButton: ImageButton =
findViewById(R.id.seeReferenceImageButton)
        seeReferenceImageButton.setOnClickListener {
            val intent = Intent(this, ReferenceImagesActivity::class.java)
            startActivity(intent)
        }
        val useAnotherImageButton: ImageButton = findViewById(R.id.go_bk)
        useAnotherImageButton.setOnClickListener {
            val intent = Intent(this, UploadImageActivity::class.java)
                                        intent.flags = Intent.FLAG_ACTIVITY_CLEAR_TOP or
Intent.FLAG_ACTIVITY_SINGLE_TOP
            val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
            startActivity(intent, options.toBundle())
        }
    }
```

```
}
```

LAYOUT: activity_model_result_positive.xml

This code creates the layout of the page giving the prediction results of the program..

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/primary"
    tools:context=".ModelResultActivityNegative">

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="415dp"
        android:layout_height="119dp"
        android:layout_marginBottom="632dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/melohead" />

    <ImageButton
        android:id="@+id/seeReferenceImageButton"
        android:layout_width="344dp"
        android:layout_height="81dp"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="18dp"
        android:layout_marginBottom="40dp"
        android:importantForAccessibility="no"
        android:scaleType="fitCenter"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.303"
```

```
    app:layout_constraintStart_toStartOf="parent"
    app:srcCompat="@drawable/see_ref_white" />

  <ImageView
    android:id="@+id/validImage"
    android:layout_width="250dp"
    android:layout_height="250dp"

    android:layout_marginStart="20dp"
    android:layout_marginTop="236dp"
    android:layout_marginEnd="20dp"
    android:scaleType="fitCenter"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.492"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

  <ImageButton
    android:id="@+id/go_bk"
    android:layout_width="344dp"
    android:layout_height="81dp"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="18dp"
    android:layout_marginBottom="116dp"
    android:importantForAccessibility="no"
    android:scaleType="fitCenter"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.303"
    app:layout_constraintStart_toStartOf="parent"
    app:srcCompat="@drawable/use_another_btn" />

  <ImageView
    android:id="@+id/imageView3"
    android:layout_width="329dp"
    android:layout_height="117dp"
    android:layout_marginStart="12dp"
    android:layout_marginTop="85dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/malignant" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_model_result_negative.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/primary"
    tools:context=".ModelResultActivityNegative">

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="415dp"
        android:layout_height="119dp"
        android:layout_marginBottom="632dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/melohead" />

    <ImageButton
        android:id="@+id/seeReferenceImageButton"
        android:layout_width="344dp"
        android:layout_height="81dp"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="18dp"
        android:layout_marginBottom="40dp"
        android:importantForAccessibility="no"
        android:scaleType="fitCenter"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.303"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/see_ref_white" />

    <ImageView
        android:id="@+id/validImage"
        android:layout_width="250dp"
        android:layout_height="250dp"
```

```
    android:layout_marginStart="20dp"
    android:layout_marginTop="236dp"
    android:layout_marginEnd="20dp"
    android:scaleType="fitCenter"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.492"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<ImageButton
    android:id="@+id/go_bk"
    android:layout_width="344dp"
    android:layout_height="81dp"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="18dp"
    android:layout_marginBottom="116dp"
    android:importantForAccessibility="no"
    android:scaleType="fitCenter"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.303"
    app:layout_constraintStart_toStartOf="parent"
    app:srcCompat="@drawable/use_another_btn" />

<ImageView
    android:id="@+id/imageView3"
    android:layout_width="329dp"
    android:layout_height="117dp"
    android:layout_marginStart="12dp"
    android:layout_marginTop="85dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/benign" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

CODE: ReferenceImagesActivity.kt

This code handles the page showing reference images.

```
package com.example.meloscan2

import android.app.ActivityOptions
import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.ImageButton
import android.appcompat.app.AppCompatActivity

class ReferenceImagesActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_ref_img)

        val goBackButton: ImageButton = findViewById(R.id.go_bk)
        goBackButton.setOnClickListener {
            // Navigate back to UploadImageActivity
            val intent = Intent(this, UploadImageActivity::class.java)
            val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
            startActivity(intent, options.toBundle())
        }
    }
}
```

CODE: ReferenceImagesActivity.kt

This code handles the page showing reference images.

```
package com.example.meloscan2

import android.app.ActivityOptions
import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.ImageButton
import android.appcompat.app.AppCompatActivity

class ReferenceImagesActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_ref_img)

        val goBackButton: ImageButton = findViewById(R.id.go_bk)
        goBackButton.setOnClickListener {
            // Navigate back to UploadImageActivity
            val intent = Intent(this, UploadImageActivity::class.java)
            val options = ActivityOptions.makeCustomAnimation(this, R.anim.fade_in, 0)
            startActivity(intent, options.toBundle())
        }
    }
}
```

LAYOUT: activity_reference_images.xml

This code provides the layout for the page showing reference images.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/primary"
    tools:context=".ReferenceImagesActivity">

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="415dp"
        android:layout_height="119dp"
        android:layout_marginBottom="632dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/melohead" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="324dp"
```

```
        android:layout_height="639dp"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="56dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.318"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/ref_instructions" />

    <ImageButton
        android:id="@+id/go_bk"
        android:layout_width="365dp"
        android:layout_height="68dp"
        android:layout_marginStart="23dp"
        android:layout_marginEnd="11dp"
        android:layout_marginBottom="26dp"
        android:scaleType="centerInside"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:srcCompat="@drawable/go_back" />
</androidx.constraintlayout.widget.ConstraintLayout>
}
```

APPENDIX B

PLAGIARISM REPORT (TURNITIN)

Final Report for Plagiarism.docx

ORIGINALITY REPORT

7%	5%	4%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Rashmi Patil, Sreepathi Bellary. "Machine learning approach in melanoma cancer stage detection", Journal of King Saud University - Computer and Information Sciences, 2020 Publication	1%
2	backoffice.biblio.ugent.be Internet Source	1%
3	www.frontiersin.org Internet Source	1%
4	www.irjmets.com Internet Source	<1%
5	Submitted to Heriot-Watt University Student Paper	<1%
6	ieeexplore.ieee.org Internet Source	<1%
7	www.ijert.org Internet Source	<1%
8	bmcmedinformdecismak.biomedcentral.com Internet Source	<1%

9	f1000research.com Internet Source	<1 %
10	www.qeios.com Internet Source	<1 %
11	Barazanji, Omar. "Automatic Actigraphy and Polysomnography Sleep Scoring Using Deep Learning", Auburn University, 2023 Publication	<1 %
12	researchbank.swinburne.edu.au Internet Source	<1 %
13	arxiv.org Internet Source	<1 %
14	www.mdpi.com Internet Source	<1 %
15	Submitted to South Bank University Student Paper	<1 %
16	Singh, Ankush Pratap. "Segmenting Metastatic Brain Tumor Using Deep Learning", New York University Tandon School of Engineering, 2023 Publication	<1 %
17	fastercapital.com Internet Source	<1 %
18	accedacris.ulpgc.es Internet Source	<1 %

19	assets.researchsquare.com Internet Source	<1 %
20	deepgram.com Internet Source	<1 %
21	Adeniran, Opeyemi Taiwo. "Early Detection of Alzheimer's Disease Using an Ensemble of Diverse Convolutional Neural Networks and Vision Transformer", Morgan State University, 2024 Publication	<1 %
22	yusera khan, Tathagat Banerjee, Gagandeep Singh Narula, Ritika Wason. "HHO-UNet-IAA: Harris Hawks Optimization based Novel UNet-Inception Attention Architecture for Glaucoma Segmentation", Research Square Platform LLC, 2023 Publication	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

PLAGIARISM REPORT (COE FORMAT-I)

Format - I

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY (Deemed to be University u/s 3 of UGC Act, 1956)		
Office of Controller of Examinations		
REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG/PG PROGRAMMES **(To be attached in the dissertation/ project report)**		
1	Name of the Candidate **(IN BLOCK LETTERS)**	ARYAN SINHA, KAVYA NAIR, ADITI, ARADHYA GOYAL
2	Address of the Candidate	Flat No. 206, Brundavan Gents PG, SRM Nagar, Kattankulathur-603203
3	Registration Number	RA2011003010066, RA2011003010004, RA2011003010065, RA2011003010067
4	Date of Birth	02-01-2003, 06-06-2002, 23-04-2002, 30-05-2002
5	Department	Computer Science and Engineering
6	Faculty	Engineering and Technology, School of Computing
7	Title of the Dissertation/Project	Melanoma Detection using a Three Stage Mobile Application
8	Whether the above project /dissertation is done by	Group Project : **a)** If the project/ dissertation is done in group, then how many students together completed the project : 4 b) Mention the Name & Register number of other candidates : Aditi-RA2011003010004, Aradhya Goyal-RA2011003010065 Kavya Nair-RA2011003010067
9	Name and address of the Supervisor / Guide	Dr. S. Sivakumar, Assistant Professor, Department of Computing Technologies SRM Institute of Science and Technology Kattankulathur **Mail ID:** sivakums2@srmist.edu.in **Mobile Number:** 9840093979
10	Name and address of Co-Supervisor / Co- Guide (if any)	**Mail ID: N i l** **Mobile Number: Nil**

11	Software Used	Turnitin		
12	Date of Verification	22-04-2024		
13	**Plagiarism Details: (to attach the final report from the software)**			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self-citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**
1	Introduction	1	1	1
2	Literature Survey	3	3	3
3	Architecture and Analysis	1	1	1
4	Design and Implementation	1	1	1
5	Results and Discussion	1	1	1
6	Conclusion and Future Scope	0	0	0
Appendices		2	0	0

I / We declare that the above information has been verified and found true to the best of my / our knowledge.

Aryan S. Pavya Jeffi ARADHYA

Signature of the Candidate

S. Sivakumar 29/4/24

Name & Signature of the Staff
(Who uses the plagiarism check software)

S. Sivakumar 29/4/24

Name & Signature of the Supervisor/ Guide

Name & Signature of the Co-Supervisor/Co-Guide

M. Pushpalatha

Name & Signature of the HOD

Department of Computing Technology
Kattankulathur
603 203
SRM IST

PUBLICATION PROOF

Gmail

Aryan Sinha <aryan26sinha@gmail.com>

Inderscience Publishers: IJBET-206106 - Submission Acknowledgement

Inderscience Submissions <submissions@inderscience.com> 22 April 2024 at 22:14
To: aryan26sinha@gmail.com

INDERSCIENCE Submissions
Article submission and peer-review system

Dear Dr. SIVAKUMAR SOUBRAYLU, Eng. Aryan Sinha, Eng. Kavya R. Nair, Eng. Aditi A., Eng. Aradhya Goyal, Dr. Sridhar S. S.,

Thank you for submitting your paper entitled 'Melanoma Detection Using Three Stage RCNN Extracted U-Net Segmented MobileNet Application'
to the journal: Int. J. of Biomedical Engineering and Technology

Your submission code is: IJBET-206106.

This paper will now be screened to filter out incomplete or unsuitable content (like author identifying details etc.).

You can track progress by logging in to the Inderscience Submissions system at
https://www.indersciencesubmissions.com/

You can get username and password reminders on the log in page.

Please note that there are no charges for publishing with Inderscience, unless you are submitting to an Open Access only journal or you want your article to be Open Access (OA).

If you receive an email requesting payment in relation to your article (for example for editing or reviewing services), then you should ignore and delete the email – it is not a legitimate Inderscience email. If you are unsure, you can check with us at: submissions@inderscience.com
If you are considering publishing an Open Access article with us, remember that we will never request payment before your paper has been accepted and payment will be only be organised and handled by the Inderscience Editorial Office.

Thank you for your interest in publishing with Inderscience.

Kind regards,
The Inderscience Submissions Team
Inderscience Publishers
submissions@inderscience.com