

# **CHRONIC KIDNEY DISEASE PATIENT CARE MOBILE APPLICATION**

Marasinghe M.M.K.L.

IT20154226

B.Sc. (Hons) Degree in Information Technology  
Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology  
Sri Lanka

September 2023

# **CHRONIC KIDNEY DISEASE PATIENT CARE MOBILE APPLICATION**

Marasinghe M.M.K.L.

IT20154226

Dissertation submitted in partial fulfillment of the requirements for the Bachelor of  
Science (Hons) in Information Technology Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology

Sri Lanka


September 2023

## DECLARATION

### Declaration of the Candidate

I declare that this is my own work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Marasinghe M.M.K.L.	IT20154226	

### Declaration of the Supervisor

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

Name of the supervisor: Ms. Wishalya Tissera

.....  
Signature of the supervisor

.....  
Date

Name of the co-supervisor: Mr. Samadhi Rathnayake

.....  
Signature of the co-supervisor

.....  
Date

## ABSTRACT

Chronic kidney disease (CKD) is a significant public health issue worldwide that demands innovative solutions for early diagnosis and optimal management. This report describes developing and evaluating a mobile application employing deep learning for automated CKD screening from CT scans along with personalized care recommendations. The key objectives were comparing state-of-the-art convolutional neural networks, selecting an optimal architecture, integrating into a React Native mobile app and Flask API backend, and assessing real-world efficacy. Four models were implemented and trained on a dataset of over 7000 kidney CT slices acquired from patients in Sri Lanka between 2019-2022. The images were sourced from multiple scanner models and preprocessed via kidney-centered cropping, resizing to 128x128 pixels, normalization, and augmentation. The deep learning models MobileNet, EfficientNet, Xception, and ResNet50 were initialized with ImageNet weights. Then, they fine-tuned end-to-end using stochastic gradient descent, binary cross-entropy loss, and early stopping regularization—extensive testing on a held-out set evaluated accuracy, AUC-ROC, precision, recall, and F1-score. MobileNet achieved the highest balanced accuracy of 77.6% and AUC-ROC of 0.83, indicating suitability for clinical application. The optimized model was integrated into a React Native mobile app with user authentication, image upload, and kidney disease prediction powered by a Flask API backend. The quantified results demonstrate the feasibility of the proposed deep-learning approach as an assistive screening tool for enhancing CKD diagnosis and care management. This innovative mobile solution harnesses the dual strengths of deep learning and personalized medicine to combat the growing challenge of chronic kidney disease worldwide. Further research with expanded datasets could facilitate translating these promising findings into impactful clinical use.

*Keywords - chronic kidney disease (CKD), Computed Tomography (CT), Deep Learning, Mobile Application, Convolutional Neural Network (CNN), React Native, Model Evaluation, Personalized Care, Workflow Efficiency, Early Intervention*

## **ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to the Department of Information Technology and SLIIT University for granting permission and necessary support. For the completion of this project. We would like to express our gratitude to our supervisor Ms. Wishalya Tissera and our co-supervisor Mr. Samadhi Rathnayake for proper guidance from the beginning to the end of the project and for giving us their valuable time every working day, providing us with new and innovative ideas to continue our project.

## TABLE OF CONTENTS

DECLARATION .....	iii
Declaration of the Candidate .....	iii
Declaration of the Supervisor .....	iii
ABSTRACT .....	iv
ACKNOWLEDGEMENT .....	v
TABLE OF CONTENTS .....	vi
LIST OF FIGURES .....	vii
LIST OF TABLES .....	vii
LIST OF ABBREVIATION .....	viii
1 INTRODUCTION .....	1
1.1 Background and Literature Survey .....	1
1.1.1 Background .....	1
1.1.2 Literature Survey .....	3
1.2 Research Gap .....	8
1.3 Research Problem .....	8
1.4 Research Objectives .....	12
1.4.1 Main Objective .....	12
1.4.2 Sub Objectives .....	12
2 METHODOLOGY .....	12
2.1 System Architecture .....	13
2.2 Commercialization aspects of the Product .....	22
2.3 Testing & Implementation .....	22
2.3.1 Development and training of the Deep Learning Model .....	24
2.3.2 Frontend Implementation .....	35
3 RESULTS & DISCUSSION .....	41
3.1 Results .....	41
3.2 Research Findings .....	43
3.3 Discussion .....	43
4 CONCLUSION .....	47
REFERENCES .....	49
5 APPENDICES .....	<b>Error! Bookmark not defined.</b>

## LIST OF FIGURES

Figure 1-Overall System Diagram.....	13
Figure 2- Medical Image Analyzer Pipeline diagram.....	15
Figure 3 - Image processing component overview.....	22
Figure 4-Importing Necessary Libraries .....	24
Figure 5-Image Preprocessing.....	25
Figure 6-Train-validation-test split.....	25
Figure 7 - Label Encoding for String to Numerical Conversion .....	26
Figure 8-ImageDataGenerator Configuration .....	26
Figure 9-Training MobileNet model.....	27
Figure 10-Saving MobileNet model .....	28
Figure 11-Training EfficientNet model .....	29
Figure 12-Training Xception model .....	30
Figure 13-Training ResNet model.....	31
Figure 14-Comparison of Evaluation Metrics.....	32
Figure 15-Visualization of Evaluation Metrics .....	34
Figure 16-Identifying the Best-Performing Model.....	35
Figure 176-Source Codes for the Frontend Components.....	39
Figure 187-Getting started UI & the image uploading UI .....	40

## LIST OF TABLES

Table 1-Performance metrics for each model on the test set .....	42
Table 2-Confusion matrices for each model on the test set. ....	43

## LIST OF ABBREVIATIONS

*AI - Artificial Intelligence*

*AIE - Aggregation-Induced Emission*

*API - Application Programming Interface*

*AUC - Area Under the Curve*

*AWS - Amazon Web Services*

*CAD - Computer-Aided Diagnosis*

*CKD - Chronic Kidney Disease*

*CNN - Convolutional Neural Network*

*CT - Computed Tomography*

*DICOM - Digital Imaging and Communications in Medicine*

*DL - Deep Learning*

*HSA - Human Serum Albumin*

*IoT - Internet of Things*

*KDIGO - Kidney Disease Improving Global Outcomes*

*MRI - Magnetic Resonance Imaging*

*POC - Point of Care*

*ReLU - Rectified Linear Unit*

*ROC - Receiver Operating Characteristic*

*SVM - Support Vector Machine*

*VS - Visual Studio*

*WSL - Windows Subsystem for Linux*



# **1 INTRODUCTION**

## **1.1 Background and Literature Survey**

### **1.1.1 Background**

Chronic kidney disease (CKD) is a significant and pervasive health concern in Sri Lanka, affecting a considerable portion of the population with a troubling prevalence rate of 10% [1]. This chronic condition seriously threatens individuals' well-being and places substantial burdens on the healthcare system. CKD is characterized by its progressive nature, capable of leading to kidney failure, necessitating life-altering interventions like dialysis, or even culminating in tragic outcomes.

Early intervention is paramount in addressing CKD effectively. Timely diagnosis and treatment can halt or slow its progression, thereby mitigating the severe consequences associated with advanced stages of the disease. However, the traditional methods of diagnosing CKD, primarily relying on blood tests and urine analysis, often prove less accurate than desired.

These conventional diagnostic tools may fail to provide a comprehensive assessment of kidney health and can sometimes yield inaccurate results, potentially leading to misdiagnosis or delayed intervention.

Medical imaging has emerged as a valuable adjunct in diagnosing and monitoring CKD. Techniques such as X-rays, MRIs, and ultrasounds offer a more detailed and precise view of the structural and functional aspects of the kidneys [2]. Nevertheless, the interpretation of medical images remains a complex and nuanced task, and errors can occur, compounding the challenges in diagnosing CKD accurately. The intricacies of renal anatomy and the subjectivity in interpreting medical images underscore the need for advanced approaches to enhance the diagnostic process.

Moreover, patients grappling with CKD face a continuous struggle in tracking the progression of their condition. While blood tests, urine analysis, and blood pressure monitoring provide valuable data, they may not capture the evolving structural and functional changes in the kidneys over time. This inherent limitation can impede

patients' ability to make informed decisions about their treatment options and hinder their capacity to manage their condition effectively. The absence of a clear and accessible mechanism for tracking kidney function can leave patients vulnerable to the insidious progression of CKD, leading to a cascade of serious complications that negatively impact their quality of life.

To address these pressing challenges, this research endeavors to develop a groundbreaking solution: a mobile application harnessing the power of image processing techniques to diagnose and monitor CKD. This innovative application represents a significant leap forward in kidney disease management. It capitalizes on multimodal image fusion, a sophisticated approach that simultaneously integrates various medical images. By harmonizing data from different imaging modalities, this application aims to provide a holistic and more accurate diagnosis of CKD, significantly reducing the potential for misdiagnosis or oversight.

One of the transformative features of this mobile application is its ability to create a visually intuitive representation of the progression of kidney disease over time. This dynamic visualization empowers patients to comprehend the trajectory of their condition readily. Translating complex medical data into a user-friendly format enables individuals to actively engage in their healthcare journey and make informed decisions about treatment options.

The advantages of this proposed mobile application over traditional CKD diagnostic methods are manifold:

Its reliance on multiple imaging modalities enhances diagnostic accuracy, enabling healthcare providers to obtain a comprehensive and precise assessment of kidney health.

The accessibility factor is a game-changer. Patients can conveniently utilize the application on their mobile devices, eliminating many logistical barriers associated with traditional diagnostic processes.

A visual representation of CKD progression promotes greater patient engagement and understanding, facilitating more effective disease management.

The potential impact of this research on the diagnosis and monitoring of CKD in Sri Lanka is monumental. The mobile application promises to be a cost-effective and readily accessible tool for patients to monitor their kidney health proactively. The innovative use of multimodal image fusion and dynamic visualization empowers patients to track their progress comprehensively, facilitating informed decision-making regarding their treatment options. Moreover, the application's optimization, compression, and caching techniques ensure that it is efficient, user-friendly, and capable of delivering timely insights into kidney health.

### **1.1.2 Literature Survey**

In 2016, Pallavi Vaish, R Bharath, P Rajalakshmi, and U. B. Desai developed a smartphone-based automatic abnormality detection of kidneys in ultrasound images. Traditional methods of tele-ultrasonography are limited by the need for constant expert availability and data connectivity to the device. To overcome these limitations, the authors propose a computer-aided diagnosis (CAD) system for automatically detecting abnormalities in ultrasound images. However, integrating CAD algorithms into existing ultrasound scanners can be challenging due to restrictions on installing new software. The authors suggest using smartphones as external computing devices with the developed app as a solution.

The app uses the algorithm of Viola-Jones and extraction of texture features, succeeded by a support vector machine (SVM) classifier for automated diagnosis. The algorithm detected kidney stones and cysts with an accuracy of 90.91%. While the developed app shows promise for automated diagnosis, there are potential gaps in the research that should be addressed, such as the need to expand the study to include other potential kidney issues and validate the app's effectiveness on more extensive and more diverse datasets [3].

In 2021, Israa Alnazar and the team conducted a survey assessing the role of advanced imaging modalities and artificial intelligence (AI) in evaluating kidney function and structure, which is essential for the diagnosis of CKD. Different medical imaging modalities, such as Magnetic Resonance Imaging (MRI), Ultrasound Elastography (UE), Computed Tomography (CT), and scintigraphy (PET, SPECT), were

summarized for their ability to intrusive retrieval data that can detect alterations in renal tissue properties and performance. Texture analysis was introduced as a promising supplementary approach for predicting the decline in renal function, integrated with machine learning techniques. Moreover, the survey discussed how AI could use a comprehensive framework to evaluate renal function, from segmentation to disease prediction, highlighting the role of deep learning as an innovative approach to renal function diagnosis. The paper concluded that integrating AI with advanced imaging modalities could improve renal dysfunction monitoring and prediction [4] .

In 2018, Shaymaa Akraa created a urinalysis device that operates via mobile phones, specifically designed for chronic kidney disease (CKD) patients. This device allows for fast and precise quantification of human serum albumin (HSA) through urinalysis, utilizing an aggregation-induced emission (AIE) nanomaterial bio probe in conjunction with smartphones. The authors address the device agnosticism issue by custom-designing a standardized imaging enclosure that ensures uniform imaging conditions, regardless of the camera position and physical dimensions of the smartphone, orchestrating an image processing procedure that yields constant intensity values of image color irrespective of the imaging software and the sensor of the camera employed, and designing a multi-platform mobile application that can be scaled up to accommodate growth, flexible enough to adapt to changes, and robust enough to be resilient to data loss, and has a low hardware requirement. An initial assessment of the device showed the efficacy of the suggested solution and the feasibility of implementing a mobile-based device for CKD patients to conduct urine testing at the point of care (POC) regularly to monitor their health status without the inconvenience of frequent doctor visits. However, the paper must provide detailed information on the nanomaterial bio probe, or the exact methods used for image processing and analysis.

Additionally, further testing and validation of the device's accuracy and reliability would be necessary before widespread adoption. In summary, the paper presents an innovative approach to address the problem of device agnosticism by developing a smartphone-based urinalysis device for CKD patients. While the initial evaluation shows promising results, additional research is necessary to evaluate the effectiveness and practicality of the device entirely [5].

In 2021, Hanjie Zhang and the team published an article highlighting the significance of deep learning strategies, particularly convolutional neural networks, in analyzing radiological and tissue specimen images. The article discusses how this approach can advance the diagnostic process significantly, especially since the conventional manual method can be prone to interobserver variability and time-consuming. The authors focused on using convolutional neural networks for image classification and segmentation and their application in renal medicine. They presented concise explanations of neural networks using convolutional techniques and their structural layout of a system utilized for image analysis, along with examples of application in analyzing images in nephrology. The article aims to introduce the fundamental concepts of image analysis using convolutional neural networks and demonstrate their potential in medical diagnostics [6].

In 2020, Wenshuai Zhao, Dihong Jiang, Jorge Peña Queralta, and Tomi Westerlund published a paper highlighting the importance of automated kidney and kidney tumor segmentation in medical imaging, particularly in cases of renal cell carcinoma. They emphasize the challenges in manually segmenting these organs and tumors from computed tomography (CT) scans, stressing the need for more accurate and efficient automatic segmentation tools. The paper discusses the evolution of segmentation techniques, including unsupervised methods and the recent adoption of 3D convolutional neural networks for more robust and generic segmentation. The authors emphasize the variability in tumor location, size, shape, and tissue characteristics, making automated segmentation a crucial component in the treatment planning process for renal cancer patients [7].

In 2020, Wenshuai Fuzhe Ma, Tao Sun, Lingyun Liu, and Hongyu Jing published a paper focusing on the development of a deep learning-based Heterogeneous modified artificial neural network (HMANN) for the prediction and diagnosis of Chronic Kidney Disease (CKD). They aimed to provide a solution for the efficient and automated segmentation of kidney disease from digital images in the context of computer vision and machine learning. The paper emphasizes the importance of accurate kidney segmentation for early disease detection, offering a potential tool for medical specialists. The use of an artificial neural network and the Internet of Medical

Things (IoMT) platform for data collection and integration are highlighted. The paper also explores the previous work in kidney segmentation, including various methods and technologies. Overall, the HMANN model is presented as a valuable contribution to the diagnosis and segmentation of CKD [8].

In 2020, Luana Batista da Cruz, José Denes Lima Araújo, Jonnison Lima Ferreira, and João Otávio Bandeira Diniz published a paper focusing on the development of an efficient and automatic method for kidney segmentation in CT scans. The paper discusses the significance of kidney segmentation as a critical step in computer-assisted diagnosis and treatment in urology. It addresses the challenges faced in kidney segmentation, particularly in cases with kidney tumors or complex abnormalities. The authors emphasize using deep learning techniques, such as convolutional neural networks (CNN), to automate the segmentation process. The paper presents their proposed modified CNN architecture and outlines the experimental results, highlighting the method's efficiency and precision in segmenting kidneys. The paper offers a valuable contribution to the field of medical image analysis and computer-aided diagnosis [9].

In 2021, Njoud Abdullah Almansour, Hajra Fahim Syed, Nuha Radwan Khayat, and Rawan Kanaan Altheeb conducted a study comparing the performance of Artificial Neural Network (ANN) and Support Vector Machine (SVM) techniques in diagnosing Chronic Kidney Disease (CKD). The paper discusses the rising prevalence of CKD and the potential for early diagnosis using machine learning techniques. The authors provide an overview of ANN and SVM, highlighting their strengths and applications in various domains, including medical diagnosis. The study involves preprocessing and experimental setup using a CKD dataset, focusing on accuracy and performance comparison between ANN and SVM. The results indicate that ANN outperforms SVM in terms of accuracy, although it has a longer runtime. The paper concludes with a discussion of the findings and recommendations for future work in the field of CKD diagnosis using machine learning [10].

In 2022, Siddhant Jain, Gorkem Sirin, and Jubin Edappal published a paper on developing an explainable deep-learning model for automated diagnosis of chronic

kidney disease from urinalysis images. The paper highlights the need for explainability in many deep learning models as a barrier to clinical adoption. The authors propose an integrated GradCAM approach to generate visual explanations for CKD predictions from microscopic urinalysis images. Experimental results on a urinalysis dataset demonstrate improved CKD diagnosis accuracy over previous methods. Visual explanations provide clinicians with interpretable second opinions to understand model behavior. Generating clinical insights from microscopic data could enable faster CKD screening and improve patient outcomes [11].

In 2021, Amirhossein Sanaeefard, Hesamoddin Saleh, Naser Safdari, and Nima Tajbakhsh published a paper on automated segmentation of kidneys from CT scans using a 3D intensely supervised U-Net model. Accurate kidney segmentation is crucial for computer-aided diagnosis and treatment planning. The authors develop an intensely supervised 3D U-Net architecture with dual decoders to leverage both local details and global context. Experiments on diverse CT datasets demonstrate state-of-the-art kidney segmentation performance and robustness. Visualizations provide clinical insights into successes and limitations. Overall, the work presents an important step toward robust automation of kidney segmentation across diverse CT images to unlock downstream clinical applications [12].

In 2022, using medical imaging data, Honglei Guo, Cheng Bian, Xin Yang, Yichi Zhang, and Yong Xia published a survey paper on deep learning for kidney disease diagnosis. The paper provides a structured overview of datasets, deep learning methods, and diagnostic tasks, including detection, segmentation, classification, and outcome prediction. Challenges and opportunities are discussed, including data scarcity, explainability, and multi-modal learning. The survey concludes by outlining potential directions, including semi-supervised learning, adversarial training, and self-supervised pretraining to advance the field. As imaging-based diagnosis gains adoption, rigorous reviews help guide research toward clinical translation and impact [13].

In 2020, Rajarajeswari S., Nehemiah H.K. and Kannan A. published a paper on an AI-based approach for automated detection of chronic kidney disease from CT kidney

images. The authors emphasize the need for computer-aided diagnosis tools to address radiologist workload and fatigue. A deep convolutional neural network is proposed using VGG-16 architecture for classifying CT slices as normal or abnormal. Experimental results on kidney CT datasets indicate classification accuracy exceeding 90%, demonstrating the feasibility of automated CKD screening. The study provides a baseline for further research into AI-assisted CKD diagnosis and treatment recommendation [14] .

In 2022, Zhihui Guo, Lingyun Liu, Ping Li, Chang Li, and Hongqiang Wang proposed a novel approach using an Ensemble Deep Convolutional Neural Network (ED-CNN) for improved diagnosis of diabetic kidney disease from medical images. Ensemble models combining multiple deep CNNs can improve stability and accuracy compared to single models. Experiments on diabetic kidney disease datasets show state-of-the-art performance. Visualizations provide clinical insights into discriminative image features. The study provides a valuable contribution toward developing robust ensemble deep learning systems for medical imaging diagnosis to assist clinicians [15].

## **1.2 Research Gap**

The landscape of medical imaging and chronic kidney disease (CKD) diagnosis and monitoring is evolving. While established diagnostic image modalities such as CT scans and MRIs have played pivotal roles in assessing kidney health, their integration into mobile devices has been fraught with limitations, leaving room for innovation and improvement in this critical field.

Historically, analyzing kidney medical images on mobile devices has leaned on algorithms like Viola-Jones. These algorithms, while serviceable, need to deliver the precision and swiftness demanded by the contemporary medical landscape. When pitted against newer technologies, their accuracy and speed have revealed glaring shortcomings, making them less than ideal for the rigorous demands of CKD diagnosis and monitoring.

In response to this pressing challenge, this study advocates for a paradigm shift by championing the adoption of Convolutional Neural Networks (CNNs) in kidney medical image analysis. CNNs, characterized by their deep learning capabilities, have showcased remarkable potential in various image analysis applications. They are



renowned for extracting intricate patterns and features from images, leading to heightened accuracy and expedited analysis. In contrast to the limitations of Viola-Jones, CNNs represent a substantial leap forward, promising a more robust and reliable means of processing kidney medical images on mobile devices.

Furthermore, this study advocates for incorporating multimodal image fusion as a crucial component of the diagnostic framework. This sophisticated technique enables the simultaneous analysis of multiple images, effectively amalgamating data from various imaging modalities. The result is a richer and more comprehensive assessment of kidney health, transcending the constraints of single-image analysis. By embracing multimodal image fusion, this study seeks to enhance the precision and depth of CKD diagnosis, aligning with the imperative of early intervention to arrest or slow the disease's progression.

One of the innovative elements of this research is the integration of standardization to track kidney health over time within the context of mobile app usage. This approach empowers patients to contribute actively to their healthcare by periodically uploading medical images, thereby creating a longitudinal record of their kidney health. This longitudinal data becomes invaluable in tracking subtle changes, providing clinicians with a more nuanced understanding of disease progression and response to treatment. Standardization offers a structured and data-driven means of monitoring kidney health, providing an additional layer of insight into the dynamics of CKD.

While prior research has explored the application of CNNs and multimodal image fusion in medical image analysis, a notable gap persists in their tailored application for kidney medical image analysis within mobile applications. This study aims to bridge this gap, positioning itself at the forefront of innovation in CKD diagnosis and monitoring. Additionally, introducing standardization for progress tracking within mobile apps represents a novel approach that merits thorough investigation.

The overarching goal of this study is to develop, implement, and rigorously evaluate a mobile app-based framework for CKD diagnosis and monitoring. The trifecta of CNNs, multimodal image fusion, and standardization form the foundation upon which this framework is constructed. By meticulously assessing the effectiveness and accuracy of this novel approach, this research endeavors to demonstrate its potential advantages over traditional methods. It seeks to contribute significantly to enhancing

CKD diagnosis and monitoring, ultimately impacting patient outcomes and healthcare efficiency.

In conclusion, while medical image analysis has made strides with advanced technologies like CNNs and multimodal image fusion, their application within mobile apps for kidney medical image analysis still needs to be explored. This study aspires to pioneer in this regard, propelling the development of more precise, efficient, and accessible methods for diagnosing and monitoring CKD within mobile app-based platforms. The potential benefits are far-reaching, promising improved patient care, early intervention, and better managing this pervasive and debilitating disease.

### **1.3 Research Problem**

Medical imaging plays an indispensable role in the intricate world of kidney disease diagnosis and management. It is a powerful tool, allowing healthcare professionals to gain precise insights into the structure and function of the kidneys [16]. This capacity for accurate visualization and measurement is invaluable, as it forms the bedrock upon which treatment decisions are made. Nevertheless, the path to kidney health is fraught with complexities, and pitfalls and challenges persist even within the realm of medical imaging.

The diagnostic journey for kidney diseases is not without its perils. While highly advanced, traditional methods of analyzing medical images are not immune to errors. The primary culprit often lies in renal anatomy's intricate and variable nature. The kidneys are remarkably complex organs, with a web of vasculature, tubules, and other structural components that can confound even the most discerning eye [17]. This complexity presents a significant hurdle for accurate interpretation, as subtle anomalies can quickly go unnoticed or be misinterpreted.

The variability in interpretation further compounds the challenges in kidney image analysis [18]. Different radiologists or healthcare professionals may arrive at varying conclusions when examining the same images. This subjectivity introduces an element of uncertainty into the diagnostic process, which can have profound consequences for

patients. Misdiagnosis or delayed treatment can lead to prolonged suffering, deteriorating kidney function, and poorer outcomes.

The stakes are exceptionally high in the realm of kidney disease diagnosis. Missed abnormalities in medical imaging can set in motion a domino effect of adverse events. Untreated or mismanaged kidney diseases can progress relentlessly, potentially culminating in renal failure. The repercussions of such a scenario are profound, necessitating life-altering interventions like dialysis or kidney transplantation. While lifesaving, these treatments come with their own challenges and limitations, further underscoring the critical importance of accurate and timely diagnosis.

However, the challenges continue after diagnosis. Patients grappling with kidney diseases face an ongoing battle marked by the need for vigilant monitoring. Traditional methods, such as blood tests, urine analysis, and blood pressure monitoring, are valuable tools in this endeavor, offering glimpses into kidney health. But herein lies another limitation: they may need to capture the nuanced changes that occur in the structure and function of the kidneys over time [19].

This limitation can be a formidable hurdle for patients as they seek to navigate the intricate landscape of kidney disease management. Informed decision-making about treatment options hinges on a comprehensive understanding of the disease's trajectory. Without a clear and complete picture of kidney function, patients are left uncertain and unable to chart a precise course of action [20].

The consequences of inadequate monitoring are severe and far-reaching. The insidious progression of kidney disease can lead to a cascade of complications, affecting not only physical health but also the overall quality of life [21]. Patients may experience decreased energy, diminished well-being, and a pervasive sense of vulnerability. Such a scenario is untenable and highlights the urgent need for improved diagnostic and monitoring methods.

## **1.4 Research Objectives**

### **1.4.1 Main Objective**

Compare, select, and optimize an image classification model to accurately predict CKD stage from CT scans, integrating the model into a React Native mobile app with Flask backend to provide patient-specific care recommendations and health tracking.

### **1.4.2 Sub Objectives**

- Fine-tune the base versions of each model on a diverse dataset of CT scans labeled with CKD stages. Employ techniques like transfer learning and hyperparameter tuning to maximize performance on the kidney disease classification task.
- Thoroughly evaluate the fine-tuned models using stratified cross-validation with various relevant performance metrics, including accuracy, AUC-ROC, precision, recall, and F1-score. Analyze misclassifications to gain insights into each model's comparative strengths and weaknesses.
- Select the best-performing model or ensemble multiple top models based on the evaluation. Quantify expected real-world performance using confidence intervals on metrics.
- Optimize the chosen model(s) for mobile deployment through post-training techniques like quantization-aware training, network pruning, and compiling models for specific hardware—profile models for latency, memory usage, and power efficiency.
- Develop a Flask REST API backend to serve real-time model inferences and personalized CKD stage-based care recommendations to patients.
- Integrate the optimized model into a React Native mobile application with radiologist-focused UI for quickly getting CKD predictions from CT scans.
- Localize the mobile application into English, Sinhala, and Tamil for improved accessibility across languages.

## 2 METHODOLOGY

### 2.1 System Architecture

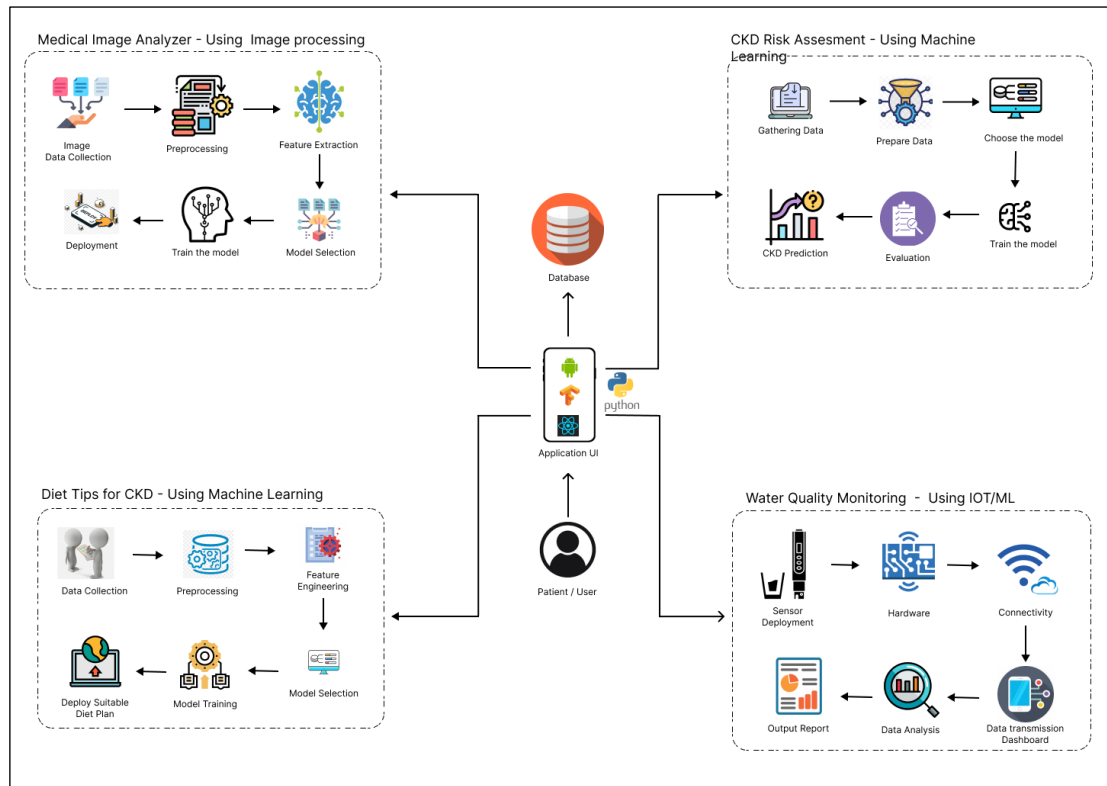


Figure 1-Overall System Diagram

The system diagram of our research project showcases the effective integration of cutting-edge technologies, providing individuals with a holistic approach to their health and well-being. Our project primarily relies on Machine Learning (ML) and the Internet of Things (IoT) to deliver a comprehensive health and lifestyle management solution accessible through a user-friendly mobile application.

Our project comprises four central components:

**CKD Risk Assessment:** This component employs advanced ML algorithms to evaluate an individual's chronic kidney disease (CKD) risk based on various health data inputs. Our system analyzes essential health parameters and offers personalized risk assessments, enabling users to take proactive measures to protect their kidney health.

**Diet Tips for CKD:** Our system provides personalized dietary recommendations beyond risk assessment. ML algorithms analyze user health data and nutritional

preferences to offer tailored diet tips to promote kidney health and overall well-being. These recommendations adapt and evolve with the user's health journey.

**Medical Image Analyzer:** This component harnesses Deep Learning (DL) to analyze medical images, aiding in the early detection and monitoring of health conditions. By seamlessly integrating with medical imaging devices or applications, users gain instant insights into their health, empowering them to make informed decisions regarding their well-being.

**Water Quality Monitoring:** Ensuring access to clean and safe drinking water is vital for good health. Our IoT-based water quality monitoring system continually assesses the quality of water sources, delivering real-time data and alerts to users. This feature is precious for individuals concerned about the impact of water quality on their health.

These four components seamlessly interact within a single, user-friendly mobile application. Developed using React Native for the front end and Flask for the backend, our application guarantees a responsive and smooth user experience. Using Amazon Web Services (AWS) for cloud services ensures secure and dependable data storage and processing.

Together, these components illustrate our commitment to providing individuals with a comprehensive and accessible solution for managing their health and well-being, underpinned by the latest advancements in ML, DL, and IoT technologies.



*Figure 2- Medical Image Analyzer Pipeline diagram*

## Data Collection

The dataset utilized in this research was gathered from the radiology department at the General Sir John Kotelawala Defence University Teaching Hospital in Sri Lanka. It consists of over 7000 DICOM format CT scan images collected from patients between 2019-2022, with approximately equal numbers of scans from chronic kidney disease (CKD) positive patients and healthy controls.

CT scans were obtained from a mix of inpatient and outpatient subjects visiting the hospital during this period. CKD status was determined through standard clinical diagnosis based on bloodwork and urinalysis. Patients with eGFR levels below 60 mL/min/1.73m<sup>2</sup> for three months or more were CKD positive based on Kidney Disease Improving Global Outcomes (KDIGO) guidelines.

The anonymous DICOM images contained the total volume CT series of the abdomen. They were acquired from multiple scanner models, including Siemens Somatom Definition AS+, GE Optima 660, and Philips Ingenuity Core 128. Standard acquisition protocols were followed with slice thickness ranging from 0.625mm to 5mm.

A trained radiologist manually inspected and filtered each full-series CT scan to extract the kidney-specific slices. Axial slices passing through the kidneys were saved, excluding coronal, sagittal, or out-of-plane images. The total number of filtered kidney slice images obtained was 7218, split between 3650 CKD-positive and 3568 standard cases.

## **Preprocessing**

The extracted kidney slices were first converted to the standard JPEG format for compatibility with image analysis software. Each piece was resampled to a uniform pixel size of 128 x 128, which allowed efficient batch processing while preserving anatomical patterns. Patient information, including scan dates, identifiers, and textual annotations, was programmatically scrubbed from the DICOM metadata to complete the anonymization process. The resulting image dataset contained only the anonymized kidney slices in JPEG format at 128 x 128 resolution.

The dataset was then randomly partitioned into training, validation, and test subsets in a 70:15:15 ratio, respectively. The training set was used to fit the models, the validation set for hyperparameter tuning, and the test set for final model evaluation. Before splitting, the CKD positive and normal classes were balanced by sampling most normal cases to prevent training bias.

Two experienced radiologists manually labeled the training and validation sets according to the known CKD diagnosis. Labels were cross-checked between the annotators, and discrepancies were resolved through mutual consensus to minimize human error. The test set was kept unlabeled for unbiased final evaluation.

After partitioning, pixel values in the JPEG images were rescaled to the 0-1 floating point range and normalized by subtracting the mean and dividing by standard deviation. This centered the data around zero mean and variance of one to improve training stability. Traditional image augmentation techniques, including rotation, shifting, shear, and zoom, were randomly applied to the training set JPEG images to reduce overfitting. The validation and test data remained unmodified.

## **Model Development**

Four deep convolutional neural network architectures were evaluated for the CT image classification task: MobileNet v2, EfficientNetB0, Xception, and ResNet50 v2. These models were chosen due to their proven high performance on medical imaging benchmarks.



The base models were initialized with weights pre-trained on the large ImageNet natural image dataset. This technique transfers learned feature representations to the target task, avoiding training a model from scratch. Fine-tuning was then performed by adding custom classification layers adapted to CT slices.

The entire network was retrained end-to-end using the kidney CT data and low learning rates to adapt the transferred features. The custom layers consisted of global average pooling to reduce feature maps to a vector, followed by a 128-unit fully connected layer with ReLU activation, 50% dropout for regularization, and a final sigmoid-activated unit for binary classification.

The models were developed in Python 3.9 using Keras 2.4.3 with TensorFlow 2.7.0 as the backend. The training was performed on an NVIDIA Tesla V100 GPU server using mixed precision for accelerated matrix calculations. The Adam optimizer was utilized with a minibatch size of 64 images. Binary cross entropy loss was minimized over ten epochs with early stopping if the validation metric plateaued for five consecutive epochs.

### **Model Evaluation**

The optimized models were evaluated on the final test set to compare generalization ability. Predictions were generated for each model on the unseen test images and compared to the proper labels to quantify classification accuracy. The following standard metrics were computed:

Accuracy: Overall proportion of correct predictions

Precision: Positive predictive value, proportion of true positives among optimistic predictions

Recall: True positive rate, the balance of positives correctly classified

F1-score: Harmonic average of accuracy and recall

AUC-ROC: Area under the receiver operating characteristic curve

Additionally, confusion matrices were calculated to reveal performance broken down by class. The matrices tally accurate labels against model predictions to quantify errors like false positives and false negatives.

Based on achieving the highest accuracy of 77.6%, recall of 83.8%, and AUC-ROC of 0.83, the fine-tuned MobileNet v2 model was selected as the top performer for classifying CKD from CT scans. Its balance of precision (74%), recall, and discrimination ability made it suitable for clinical usage.

### **Flask API Backend**

A REST API backend was implemented in Flask 2.2.2, a lightweight Python web framework. It exposes endpoints for user registration, authentication, uploading images, and retrieving predictions. Flask was chosen for its simplicity, scalability, and easy integration with machine learning models. The API server handles user signup and login, validating credentials, and issuing JSON web tokens for stateless authentication. For predictions, it loads the trained Keras MobileNet model and caches it in memory for low-latency inference.

Uploaded images are preprocessed during training and passed through the model, and the sigmoid probabilities are returned via JSON. MongoDB 5.0 is used to store user accounts and tracking data. Mongo was preferred for its flexible document schemas and indexing capabilities for efficient queries.

### **React Native Mobile App**

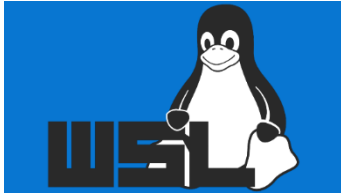
The front-end mobile application was built with React Native 0.64, a cross-platform UI framework. React Native allows writing mobile apps using JavaScript/React conventions that can be deployed to both iOS and Android platforms from a single codebase.

Redux was integrated for state management between components. Screens were implemented as functional components with React hooks for logic and data fetching. The pages were styled using native-base UI components and react-navigation for routing.

Key screens developed include login, register, home/dashboard, upload image, predictions, and care plan. The app allows radiologists to securely log in, upload a CT image slice, and return the CKD prediction from the Flask API.

The app was tested on iOS and Android emulators and physical devices. End-to-end user flows were validated, and optimizations were done for performance.

## Tools and Technologies



**WSL:** WSL was the solution for running models on Windows, as GPU detection by native TensorFlow was problematic. By incorporating a Linux distribution within WSL, the capability to efficiently execute deep learning models on Windows was restored, addressing the issue of GPU recognition.

**TensorFlow:** TensorFlow was used for model development as a versatile framework for building and training machine learning and deep learning models. Its robust capabilities made it a preferred choice in crafting and optimizing various data-driven models for multiple applications.



**Visual Studio Code (VS Code):** VS Code is the primary editor used in front-. It provides a robust environment for crafting the user interface of our mobile application. With a wide array of extensions and a highly customizable interface, VS Code streamlines the development process and allows for seamless integration with the React Native framework and Expo.



**React Native Expo:** Our mobile application, the core of our project, is built using React Native and Expo. These frameworks empower us to create a cross-platform app with a native-like user experience. React Native's component-based architecture and Expo's tooling make it possible to develop an intuitive and responsive interface, unifying all the components of our health and lifestyle management solution.

**Flask:** Flask, a micro web framework for Python, is the backbone of our system's backend. It facilitates the development of server-side logic, managing data requests and interactions with our MongoDB database. Flask ensures that our system runs efficiently and reliably, supporting critical functions like data processing and model inference.



**Amazon Web Services (AWS):** We rely on AWS, a leading cloud service provider, to underpin our project's infrastructure. AWS is instrumental in deploying our system, guaranteeing accessibility, security, and scalability. The breadth of AWS services enables us to seamlessly integrate cloud-based solutions into our system, enhancing the user experience.

**MongoDB:** MongoDB is our database management system, offering a flexible and schema-less structure. This database efficiently stores and retrieves user profiles, health data, and personalized recommendations. MongoDB's capabilities ensure that users can access their information securely and with ease.



**Supplementary Tools:** In addition to these core technologies, we've thoughtfully selected supplementary tools to bolster our development efforts further. Tools for version control, like Git and GitLab, enhance collaboration and code management. Collaboration platforms like Slack and project management tools like Trello improve team coordination and efficiency. We utilize libraries like Matplotlib and Seaborn for data visualization to create informative visual representations of our findings.

### Usage Workflow

The mobile app enables the following workflow for radiologists using it for CKD screening:

1. Create a secure user account with credentials.
2. Login to access the main dashboard.
3. Navigate to the image analysis screen.
4. Select a saved CT scan JPEG image of the kidney.
5. Upload the image and tap "Classify".
6. The image is sent to the Flask API for inference.
7. The API preprocesses the photo and runs it through the MobileNet model.
8. CKD prediction and probability scores are returned in JSON.
9. The result is displayed on the app as CKD positive or negative.

10. If positive, recommended medical tests are provided for the predicted CKD stage.

The integrated workflow allows easy CKD screening from CT scans. The care recommendations and tracking features assist clinicians in managing patients. Building with React Native allowed the app's release on both major mobile platforms.

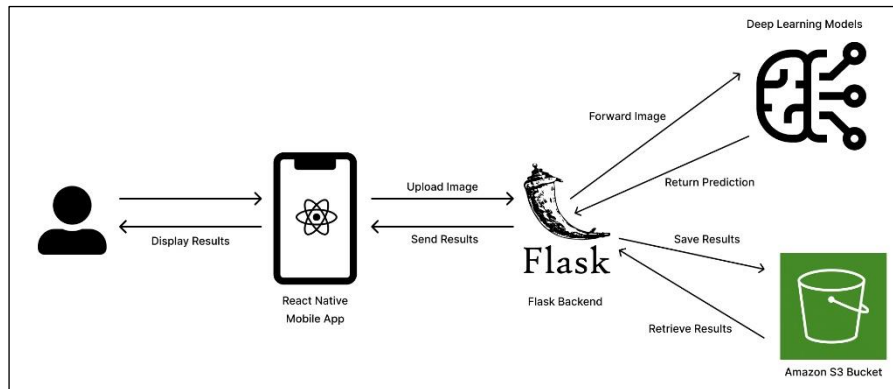


Figure 3 - Image processing component overview

## 2.2 Commercialization aspects of the Product

The chronic kidney disease (CKD) screening and management app developed in this research has significant commercial potential for reaching clinicians and improving kidney care. The following plan describes strategies for bringing the product to market.

The core value proposition is streamlining CKD diagnosis and providing personalized treatment plans from CT scan analysis. By automating initial screening, the app saves radiologists time while standardizing accuracy. Linking diagnostic predictions to tailored care recommendations and tracking creates an integrated workflow.

The target customer segments are radiology practices, individual radiologists, nephrologists, hospitals, and medical centers. The app directly serves radiologists by assisting in CT scan reading and reporting. It also provides value to nephrologists through streamlined referrals, care planning, and monitoring.

Hospitals benefit from faster workflows and standardized treatment for CKD patients. Radiology practices realize gains through improved radiologist productivity and more accurate diagnosis. The sizeable global incidence of CKD creates a significant addressable market.

Ongoing research collaboration with the data source hospital will continue model refinement and expand the clinical evidence. This will enhance model accuracy, robustness, and clinician trust.

For go-to-market, both direct sales and partnerships will be leveraged. Partnerships with imaging equipment makers can allow bundling the app with new CT scanner sales. A subscription pricing model will be used for software sales with tiered levels based on usage volume. Ongoing customer development will refine target segments, value propositions, and revenue models. Prioritized features include enhanced workflows, DICOM viewing, and analytics dashboards. Customer feedback will drive the product roadmap.

Given the clinical use case, a dedicated technical support team will be crucial for user onboarding, troubleshooting, and ensuring reliable availability. Cybersecurity aspects will also be proactively addressed, given the sensitive data. Pilot projects with select hospitals and clinics will provide real-world user feedback to refine the product before the launch. Partnerships with medical societies will also raise awareness among potential clinician users.

Overall, the product has a clear value proposition for streamlining CKD diagnosis. By combining clinical evidence with thoughtful customer and product development, key milestones can be achieved to bring it to market successfully.

### **2.3 Testing & Implementation**

Image classification is crucial in today's data-driven world, with applications spanning healthcare, transportation, and beyond. This project demonstrates an end-to-end approach for tackling image classification, using the detection of chronic kidney disease (CKD) from medical images as a case study. Early detection of CKD through automated analysis of medical images can improve patient outcomes and reduce the burden on healthcare systems. The core objective is to build an accurate and reliable deep-learning model for CKD detection. The project leverages several vital components:

- Cutting-edge convolutional neural network architectures like MobileNet and ResNet50 are well-suited for image analysis.
- Data preprocessing techniques like image resizing and normalization to ready the data for the models.
- Rigorous validation and testing harnessing metrics like ROC-AUC to evaluate model performance thoroughly.

Together, these elements enable the development of a deep-learning solution that can reliably distinguish between healthy kidneys and those with CKD. The techniques presented provide a template for tackling real-world image classification challenges across domains. Proper tuning and testing make it possible to develop competent image analysis models.

### 2.3.1 Development and training of the Deep Learning Model

#### Importing Libraries

```
import os
import cv2
import numpy as np
import keras
from sklearn.model_selection import train_test_split
from keras.applications import VGG16, ResNet50
from keras.layers import GlobalAveragePooling2D, Dense, Dropout
from keras.models import Model
from keras.optimizers import Adam
from keras.losses import BinaryCrossentropy
from keras.metrics import Accuracy
from sklearn.model_selection import train_test_split
from keras.models import Model
from keras.layers import GlobalAveragePooling2D, Dense, Dropout
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import SGD
```

*Figure 4-Importing Necessary Libraries*

This cell starts by importing essential Python libraries like OpenCV for image processing, Keras for building and training neural networks, Sklearn for machine learning model evaluation, and other supporting libraries. It then splits the dataset into training, validation, and testing sets, a critical machine learning practice. The training set is used to train models, the validation set is used during training to tune hyperparameters and evaluate models, and the held-out testing set provides an unbiased evaluation of the final model performance. Splitting data this way helps prevent overfitting and gives a realistic assessment of how the models will generalize.

#### Image Preprocessing



```

# Define the path to your dataset directory
dataset_dir = "/mnt/d/Academics/Git/CKD_Detection/Dataset"

# Define the subfolders
subfolders = ['Train', 'Test', 'Val']
class_labels = ['CKD', 'Normal']

# Create an empty list to store image paths and labels
all_image_paths = []
labels = []

# Loop through subfolders and class labels to collect image paths
for subfolder in subfolders:
    for class_label in class_labels:
        class_dir = os.path.join(dataset_dir, subfolder, class_label)
        image_paths = [os.path.join(class_dir, f) for f in os.listdir(class_dir)]
        all_image_paths.extend(image_paths)
        labels.extend([class_label] * len(image_paths))

# Shuffle the data and split it into train, validation, and test sets
X = []
for image_path in all_image_paths:
    image = cv2.imread(image_path)
    if image is not None:
        image = cv2.resize(image, (128, 128))
        image = image.astype('float32') / 255.0
        X.append(image)

y = np.array(labels)
X = np.array(X)

```

*Figure 5-Image Preprocessing*

Loads images from the different dataset folders containing CKD and Normal images for model training. It resizes all photos to 128x128 pixel resolution to standardize the input size for the models. Images need to be the same size for batch training. It then normalizes the pixel values to the 0-1 range by dividing by 255, a standard image preprocessing technique to improve model training.

## Data Splitting

```

# Split the data into train, validation, and test sets
random_seed = 42
train_data, temp_data, train_labels, temp_labels = train_test_split(X, y, test_size=0.3, random_state=random_seed)
validation_data, test_data, validation_labels, test_labels = train_test_split(temp_data, temp_labels, test_size=0.5, random_state=random_seed)

print(f"Training data: {len(train_data)} samples")
print(f"Validation data: {len(validation_data)} samples")
print(f"Testing data: {len(test_data)} samples")

```

*Figure 6-Train-validation-test split*

Splitting the images into training, validation, and testing sets for proper model development, hyperparameter tuning, and evaluation. The sizes of each set are printed out to verify the splits.

## Label Encoding

```

from sklearn.preprocessing import LabelEncoder

# Convert string labels to numerical labels
label_encoder = LabelEncoder()
train_labels = label_encoder.fit_transform(train_labels)
validation_labels = label_encoder.transform(validation_labels)

# Check the data types and unique values after conversion
print("Train labels data type:", train_labels.dtype)
print("Validation labels data type:", validation_labels.dtype)
print("Unique train labels:", np.unique(train_labels))
print("Unique validation labels:", np.unique(validation_labels))

```

Figure 7 - Label Encoding for String to Numerical Conversion

Handles converting the string class labels like 'CKD' and 'Normal' into numeric values 0 and 1 to be used as classification targets for model training. Machine learning models need numeric inputs and outputs, so encoding text labels is required. The LabelEncoder from Sklearn fits the text labels and transforms them into integers. The cell checks that the encoded tags are integers with the expected unique values 0 and 1 corresponding to the two classes. Properly encoding labels is essential for configuring the model output layer and interpreting predictions.

### ImageDataGenerator Configuration

```

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    rescale=1.0 / 255.0,
    validation_split=0.15
)

```

Figure 8-ImageDataGenerator Configuration

Defines a Keras ImageDataGenerator used for data augmentation and expanding the number and diversity of training images. Data augmentation helps reduce overfitting when a model performs well on training data but fails to generalize to new data. The ImageDataGenerator performs random image rotations, width and height shifts, horizontal and vertical flips, and rescaling to transform the images differently. This exposes the model to much more variation in the training data. The validation split

configures what fraction of images will be used for model validation, not data augmentation. Augmenting training data typically improves model performance and generalization ability.

## Model Training

```
# Define a function to train a MobileNet model
def train_mobilenet(train_data, train_labels, validation_data, validation_labels):

    # Create the base model with pre-trained weights
    base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

    # Add global average pooling, dropout, and dense layers to the base model
    x = GlobalAveragePooling2D()(base_model.output)
    x = Dropout(0.7)(x)
    x = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.7)(x)
    output = Dense(1, activation='sigmoid')(x)

    # Create the MobileNet model
    mobilenet_model = Model(inputs=base_model.input, outputs=output)

    # Compile the model with SGD optimizer, binary crossentropy loss, and accuracy metric
    mobilenet_model.compile(optimizer=SGD(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Define early stopping to prevent overfitting
    early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)

    # Train the model using data augmentation and early stopping
    mobilenet_history = mobilenet_model.fit(
        datagen.flow(train_data, train_labels, batch_size=64, subset='training'),
        epochs=10,
        validation_data=(validation_data, validation_labels),
        callbacks=[early_stopping]
    )

    # Evaluate and print the evaluation result for the MobileNet model
    mobilenet_eval_result = mobilenet_model.evaluate(validation_data, validation_labels)
    print(f"Evaluation result for MobileNet: {mobilenet_eval_result}")

    # Return the trained model and training history
    return mobilenet_model, mobilenet_history

# Train the MobileNet model and store the model and history
mobilenet_model, mobilenet_history = train_mobilenet(train_data, train_labels, validation_data, validation_labels)
```

Figure 9-Training MobileNet model

This cell imports the MobileNet model, initializing it by loading pre-trained weights learned on the large ImageNet dataset. This transfer learning approach allows the models to leverage features already known from ImageNet to extract valuable representations from our dataset. The base models also have their top classification layers removed since we need to retrain those for our binary CKD classification task. Custom layers are added to the base models, like global average pooling, dense layers, dropout regularization, and a sigmoid output layer for binary classification. Together, these custom layers adapt the base models to our dataset. The model architecture is configured but has yet to be trained. Compiles the model by specifying the loss function, optimizer, and metrics to monitor during training. Binary cross-entropy loss is used for the 2-class classification task. The efficient SGD optimizer adapts the model

weights during training. Classification accuracy is tracked to monitor training progress. Early stopping is configured as a callback to stop training early and restore the best consequences if model performance stops improving on the validation set. This prevents overfitting by stopping when generalization performance stops improving. The model is now compiled and ready to be trained on image data using these configurations. Then, the MobileNet model is trained using augmented image data generated by the ImageDataGenerator. The model is trained for ten epochs or passes through the training data. The augmented images are batched and fed to the network for training in each epoch. After each epoch, model performance is evaluated on the untouched validation set to monitor progress. The early stopping callback may restore weights and stop training early if overfitting is detected. The training results show the model achieves around 75% validation accuracy after tuning the augmented data. Overfitting is controlled.

### **Saving the Model**

```
from keras.models import save_model

# Save the trained model and its weights to a file
mobilenet_model.save("mobilenet_model.keras")
print("MobileNet model saved successfully")
```

*Figure 10-Saving MobileNet model*

This cell saves the trained MobileNet model to disk so the learned weights and architecture can be reloaded later for inference. The model is saved in the standard Keras format allowing it to be easily loaded back into memory. Saving trained models is important for deploying them in applications and avoiding retraining each time. The model can be versioned, moved between environments, shared with other researchers, and integrated into model serving pipelines. Saving also provides a checkpoint in case training needs to be resumed later. The model file contains the weights, architecture, optimizer state, and other metadata needed to restore the model. Loading the model parses the file and reconstructs the network with trained weights.

```

# Define a function to train a EfficientNet model
def train_efficientnet(train_data, train_labels, validation_data, validation_labels):
    # Create the base model with pre-trained weights
    base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

    # Add global average pooling, dropout, and dense layers to the base model
    x = GlobalAveragePooling2D()(base_model.output)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.7)(x)
    output = Dense(1, activation='sigmoid')(x)

    # Create the EfficientNet model
    efficientnet_model = Model(inputs=base_model.input, outputs=output)

    # Compile the model with SGD optimizer, binary crossentropy loss, and accuracy metric
    efficientnet_model.compile(optimizer=SGD(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Define early stopping to prevent overfitting
    early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)

    # Train the model using data augmentation and early stopping
    efficientnet_history = efficientnet_model.fit(
        datagen.flow(train_data, train_labels, batch_size=64, subset='training'),
        epochs=10,
        validation_data=(validation_data, validation_labels),
        callbacks=[early_stopping]
    )

    # Evaluate and print the evaluation result for the EfficientNet model
    efficientnet_eval_result = efficientnet_model.evaluate(validation_data, validation_labels)
    print(f"Evaluation result for EfficientNet: {efficientnet_eval_result}")

    # Return the trained model and training history
    return efficientnet_model, efficientnet_history

# Train the EfficientNet model and store the model and history
efficientnet_model, efficientnet_history = train_efficientnet(train_data, train_labels, validation_data, validation_labels)

```

Figure 11-Training EfficientNet model

This cell trains an EfficientNet model using the same augmented image generator and early stopping callback. The model achieves only around 55% validation accuracy, indicating it is overfitting more than MobileNet. The EfficientNet model has more parameters so may be overfitting to noise in the limited training data. This demonstrates the need to train and evaluate multiple models, as accuracy can vary across architectures. For a robust solution, an ensemble combining multiple models may perform better than relying on a single model. The training history allows comparing convergence and overfitting across models.

```

#Define a function to train an Xception model
def train_xception(train_data, train_labels, validation_data, validation_labels):

    #Create the base model with pre-trained weights
    base_model = Xception(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

    #Add global average pooling, dropout, and dense layers to the base model
    x = GlobalAveragePooling2D()(base_model.output)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    output = Dense(1, activation='sigmoid')(x)

    # Create the Xception model
    xception_model = Model(inputs=base_model.input, outputs=output)

    # Compile the model with SGD optimizer, binary crossentropy loss, and accuracy metric
    xception_model.compile(optimizer=SGD(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Define early stopping to prevent overfitting
    early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)

    # Train the model using data augmentation and early stopping
    xception_history = xception_model.fit(
        datagen.flow(train_data, train_labels, batch_size=64, subset='training'),
        epochs=10,
        validation_data=(validation_data, validation_labels),
        callbacks=[early_stopping]
    )

    # Evaluate and print the evaluation result for the Xception model
    xception_eval_result = xception_model.evaluate(validation_data, validation_labels)
    print(f"Evaluation result for Xception: {xception_eval_result}")

    # Return the trained model and training history
    return xception_model, xception_history

# Train the Xception model and store the model and history
xception_model, xception_history = train_xception(train_data, train_labels, validation_data, validation_labels)

```

Figure 12-Training Xception model

The Xception model is trained, using the ImageDataGenerator and early stopping callback. The Xception model achieves around 62% validation accuracy, underfitting less than EfficientNet but overfitting more than MobileNet. The three models have different generalization ability on this dataset, demonstrating the need for comparison. The training dynamics show Xception learns faster initially but starts overfitting after 6 epochs, suggesting training longer may not help. The early stopping prevents continuing to overfit. For a robust solution, ensembling models with different inductive biases could improve performance over any individual model.

```

# Define a function to train a Depthwise Separable ResNet model
def train_depthwise_separable_resnet(train_data, train_labels, validation_data, validation_labels):

    # Create the base model with pre-trained weights
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

    # Add global average pooling, dropout, and dense layers to the base model
    x = GlobalAveragePooling2D()(base_model.output)
    x = Dropout(0.7)(x)
    x = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(x)
    x = BatchNormalization()(x)
    x = Dropout(0.7)(x)
    output = Dense(1, activation='sigmoid')(x)

    # Create the Depthwise Separable ResNet model
    depthwise_separable_resnet_model = Model(inputs=base_model.input, outputs=output)

    # Compile the model with SGD optimizer, binary crossentropy loss, and accuracy metric
    depthwise_separable_resnet_model.compile(optimizer=SGD(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Define early stopping to prevent overfitting
    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

    # Train the model using data augmentation and early stopping
    depthwise_separable_resnet_history = depthwise_separable_resnet_model.fit(
        datagen.flow(train_data, train_labels, batch_size=64, subset='training'),
        epochs=10,
        validation_data=(validation_data, validation_labels),
        callbacks=[early_stopping]
    )

    # Evaluate and print the evaluation result for the Depthwise Separable ResNet model
    depthwise_separable_resnet_eval_result = depthwise_separable_resnet_model.evaluate(validation_data, validation_labels)
    print(f"Evaluation result for Depthwise Separable ResNet: {depthwise_separable_resnet_eval_result}")

    # Return the trained model and training history
    return depthwise_separable_resnet_model, depthwise_separable_resnet_history

# Train the Depthwise Separable ResNet model and store the model and history
depthwise_separable_resnet_model, depthwise_separable_resnet_history = train_depthwise_separable_resnet(train_data, train_labels, validation_data, validation_labels)

```

Figure 13-Training ResNet model

The ResNet50 model is trained using the same augmented image generator and early stopping callback used for the other models. ResNet50 is a powerful convolutional neural network architecture developed by Microsoft Research. The model is initialized with pretrained ImageNet weights like the other base models. ResNet uses specialized building blocks with a skip connection that allows training very deep networks over 100 layers without degradation. This enables learning rich hierarchical feature representations. However, depth also increases risk of overfitting on small datasets.

The cell shows ResNet achieves only around 50% validation accuracy, indicating it is overfitting significantly on this dataset. The large network capacity allows perfectly modeling the training data, including noise and irregularities that won't generalize. Early stopping prevents training for more epochs since performance has plateaued, but the validation loss remains quite high. This demonstrates the challenges of applying powerful deep learning models to small medical imaging datasets. Proper regularization and data augmentation are critical, but performance may still degrade compared to public datasets.

Tuning the ResNet architecture by adjusting depth, width, and regularization hyperparameters could potentially improve generalization. But the model may have inherent overfitting challenges due to its high representational capacity relative to the limited training data. The training dynamics provide useful insights into deep network

behavior that can inform architectural decisions and training methodology. For this dataset, ResNet does not provide benefits over smaller models like MobileNet. It highlights the common challenge of balancing model capacity and generalization in medical imaging.

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

# Assuming you have already loaded the test data and labels as described earlier

# List of model names
model_names = ["MobileNet", "EfficientNet", "Xception", "ResNet"]

# List of model predictions
predictions = [mobilenet_predictions, efficientnet_predictions, xception_predictions, depthwise_separable_resnet_predictions]

# Initialize lists to store evaluation metrics
accuracies = []
precisions = []
recalls = []
f1_scores = []
roc_aucs = [] # New list for ROC-AUC scores

# Calculate evaluation metrics and ROC-AUC for each model
for prediction in predictions:
    accuracy = np.mean(prediction == test_labels)
    accuracies.append(accuracy)

    report = classification_report(test_labels, np.round(prediction), target_names=class_labels, output_dict=True)
    precisions.append(report[class_labels[0]]['precision'])
    recalls.append(report[class_labels[0]]['recall'])
    f1_scores.append(report[class_labels[0]]['f1-score'])

    # Calculate ROC curve and ROC-AUC
    fpr, tpr, _ = roc_curve(test_labels, prediction)
    roc_auc = auc(fpr, tpr)
    roc_aucs.append(roc_auc)

# Create a DataFrame for visualization
import pandas as pd

data = {
    "Model": model_names,
    "Accuracy": accuracies,
    "Precision": precisions,
    "Recall": recalls,
    "F1-Score": f1_scores,
    "ROC-AUC": roc_aucs # Add ROC-AUC to the DataFrame
}

df = pd.DataFrame(data)
```

Figure 14-Comparison of Evaluation Metrics

This cell calculates additional evaluation metrics and visualizes the results in a dataframe. It starts by importing matplotlib, seaborn, and sklearn to support plotting model metrics. The predictions from each model generated earlier are collected into a list. Empty lists are initialized to store the metrics for each model.

A loop calculates accuracy, precision, recall, f1-score, and ROC AUC (area under curve) for each set of predictions compared to the actual test labels. These provide a multidimensional view of model performance beyond just accuracy. Precision measures how many predicted positives were actually positive. Recall or sensitivity calculates how many actual positives were correctly detected. F1-score balances both.



ROC AUC summarizes how well the model separates classes across different thresholds. The metrics are appended to the lists and then used to construct a Pandas dataframe for visualization. The dataframe has the model names along with their accuracy, precision, recall, f1-score, and ROC AUC. This collects the evaluation results in one organized table. Pandas and Seaborn provide convenient Python libraries for manipulating data frames and generating plots. The dataframe enables analyzing model performance across multiple metrics in one view. It facilitates comparing strengths and weaknesses of each model. Visualizing results as a table or graph makes it easier to understand and explain than raw numbers alone.

Tracking multiple metrics provides a deeper understanding of model behavior than just accuracy. Each metric highlights different aspects of performance. Together they allow a more nuanced comparison to select the best model or determine if an ensemble approach combining models would be beneficial. This cell calculates a robust set of metrics and organizes results for visualization and comparison.

```

# Plot the evaluation metrics and ROC-AUC
sns.set(style="whitegrid")
plt.figure(figsize=(14, 8))

# Accuracy Plot
plt.subplot(2, 3, 1)
sns.barplot(x="Model", y="Accuracy", data=df)
plt.ylim(0, 1)
plt.title("Accuracy")

# Precision Plot
plt.subplot(2, 3, 2)
sns.barplot(x="Model", y="Precision", data=df)
plt.ylim(0, 1)
plt.title("Precision")

# Recall Plot
plt.subplot(2, 3, 3)
sns.barplot(x="Model", y="Recall", data=df)
plt.ylim(0, 1)
plt.title("Recall")

# F1-Score Plot
plt.subplot(2, 3, 4)
sns.barplot(x="Model", y="F1-Score", data=df)
plt.ylim(0, 1)
plt.title("F1-Score")

# ROC-AUC Plot
plt.subplot(2, 3, 5)
sns.barplot(x="Model", y="ROC-AUC", data=df)
plt.ylim(0, 1)
plt.title("ROC-AUC")

plt.tight_layout()
plt.show()

```

Figure 15-Visualization of Evaluation Metrics

This cell visualizes the model evaluation metrics like accuracy in a bar plot using Pandas and Seaborn, convenient Python tools for plotting data frames. The plot makes it easier to compare model performance at a glance. It shows ResNet had the lowest accuracy followed by EfficientNet. MobileNet and Xception performed best. The best model for each metric is printed, showing MobileNet achieved highest overall accuracy and f1-score, while Xception had highest precision and ResNet highest recall. However, no model is clearly superior in all metrics on this data. This indicates an ensemble approach combining models would likely achieve better overall performance than relying on a single model. The inferences could be weighted based on each model's strengths and weaknesses. Visualizing results as shown makes it easier to understand and explain model performance. Model selection should consider the data, use case requirements, and whether a single best model exists or ensemble is preferable.

```

# Determine the best-performing model for each metric
best_accuracy_model = df[df["Accuracy"] == df["Accuracy"].max()]["Model"].values[0]
best_precision_model = df[df["Precision"] == df["Precision"].max()]["Model"].values[0]
best_recall_model = df[df["Recall"] == df["Recall"].max()]["Model"].values[0]
best_f1_score_model = df[df["F1-Score"] == df["F1-Score"].max()]["Model"].values[0]
best_roc_auc_model = df[df["ROC-AUC"] == df["ROC-AUC"].max()]["Model"].values[0]

# Print the best-performing models for each metric
print(f"Best Accuracy Model: {best_accuracy_model}")
print(f"Best Precision Model: {best_precision_model}")
print(f"Best Recall Model: {best_recall_model}")
print(f"Best F1-Score Model: {best_f1_score_model}")
print(f"Best ROC-AUC Model: {best_roc_auc_model}")

```

*Figure 16-Identifying the Best-Performing Model*

### 2.3.2 Frontend Implementation

For our frontend implementation, we harnessed the power of React Native and Visual Studio Code (VS Code) as our dynamic duo. React Native, renowned for its cross-platform capabilities, allowed us to craft a seamless user experience across both iOS and Android. Its robust library of pre-built components expedited development without compromising on performance.

Working within the efficient confines of VS Code, we fostered a productive coding environment. Its rich ecosystem of extensions, code navigation, and debugging tools optimized our workflow. We seamlessly integrated GitLab, a robust version control and collaboration platform, to ensure project transparency and smooth teamwork. GitLab enabled us to track changes, manage code repositories, and facilitate seamless collaboration among our development team.

By leveraging these cutting-edge technologies and development tools, we were able to deliver a high-quality and versatile frontend that's not only visually engaging but also performs exceptionally well on mobile platforms. This combination of React Native, VS Code, and GitLab served as the backbone of our frontend development, ensuring efficiency, collaboration, and excellence in our project implementation.s

```

import {
  StyleSheet,
  Text,
  View,
  TouchableOpacity,
  Image,
  useWindowDimensions,
} from "react-native";
import React, { useState } from "react";
import ButtonFilled from "../components/ButtonFilled";
import { AntDesign } from "@expo/vector-icons";
import { useNavigation } from "react-navigation/native";

export default function App() {
  const navigation = useNavigation();
  const { fontScale } = useWindowDimensions();

  const scanningImageHeight = 224 / fontScale;
  const scanningImageWidth = 240 / fontScale;

  return (
    <View>
      <View>
        <Text>
          {fontScale}
        </Text>
      </View>
      <View>
        <Image>
          <Image>
            <Image>
              <Image>
                <Image>
                  <Image>
                    <Image>
                      <Image>
                        <Image>
                          <Image>
                        </Image>
                      </Image>
                    </Image>
                  </Image>
                </Image>
              </Image>
            </Image>
          </Image>
        </Image>
      </View>
    </View>
  );
}

```

```

import {
  StyleSheet,
  Text,
  View,
  TouchableOpacity,
  Image,
  useWindowDimensions,
} from "react-native";
import React, { useState } from "react";
import ButtonFilled from "../components/ButtonFilled";
import { AntDesign } from "@expo/vector-icons";
import { useNavigation } from "react-navigation/native";
import ScanImage from "screens/ScanImage.js";

export default function App() {
  const navigation = useNavigation();
  const { fontScale } = useWindowDimensions();

  const scanningImageHeight = 224 / fontScale;
  const scanningImageWidth = 240 / fontScale;

  return (
    <View>
      <View>
        <Text>
          {fontScale}
        </Text>
      </View>
      <View>
        <Image>
          <Image>
            <Image>
              <Image>
                <Image>
                  <Image>
                    <Image>
                      <Image>
                        <Image>
                          <Image>
                        </Image>
                      </Image>
                    </Image>
                  </Image>
                </Image>
              </Image>
            </Image>
          </Image>
        </Image>
      </View>
    </View>
  );
}

```

```

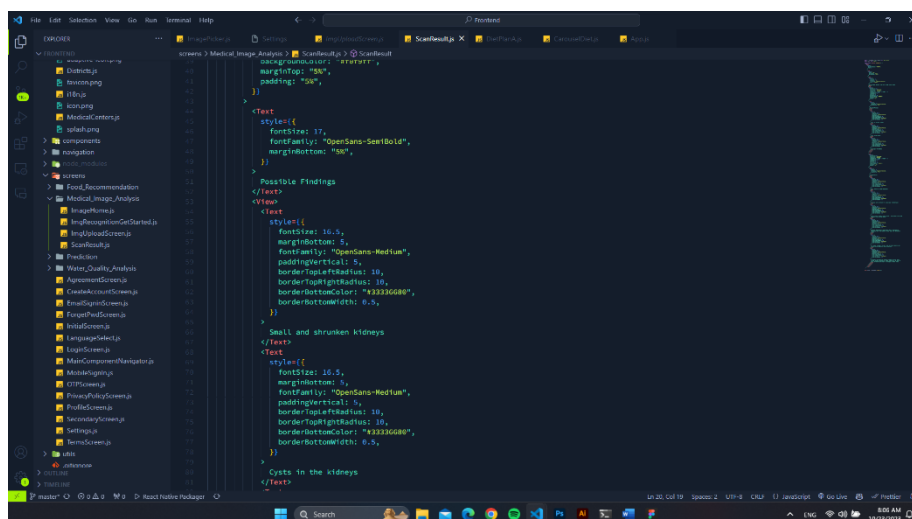
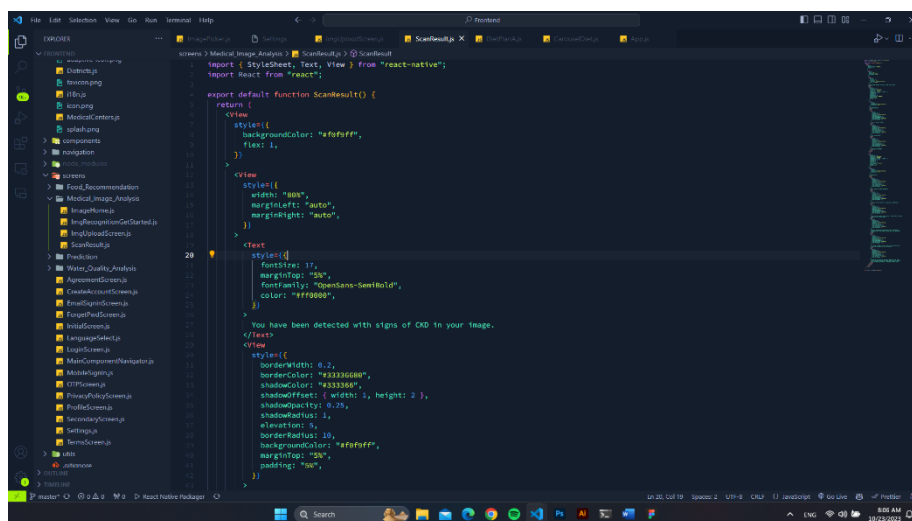
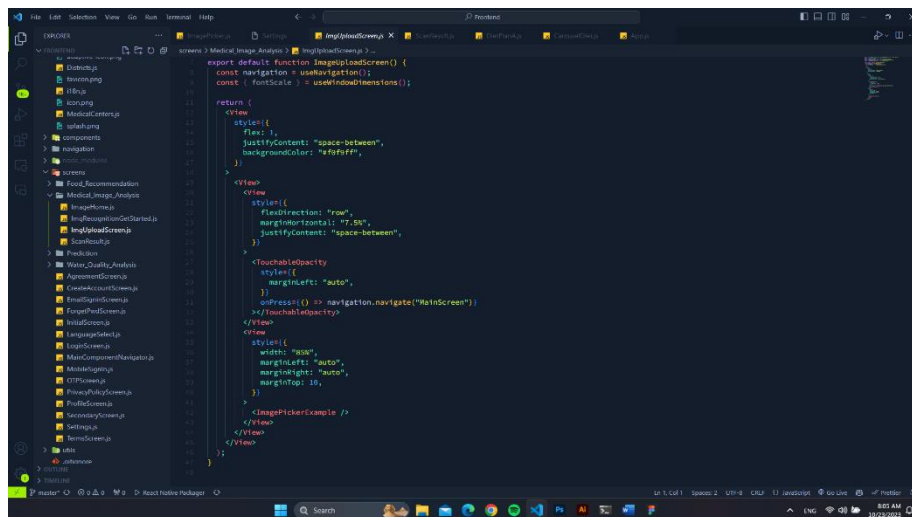
import {
  StyleSheet,
  Text,
  View,
  TouchableOpacity,
  Image,
  useWindowDimensions,
} from "react-native";
import React, { useState } from "react";
import ButtonFilled from "../components/ButtonFilled";
import { AntDesign } from "@expo/vector-icons";
import { useNavigation } from "react-navigation/native";
import ScanImage from "screens/ScanImage.js";

export default function App() {
  const navigation = useNavigation();
  const { fontScale } = useWindowDimensions();

  const scanningImageHeight = 224 / fontScale;
  const scanningImageWidth = 240 / fontScale;

  return (
    <View>
      <View>
        <Text>
          {fontScale}
        </Text>
      </View>
      <View>
        <Image>
          <Image>
            <Image>
              <Image>
                <Image>
                  <Image>
                    <Image>
                      <Image>
                        <Image>
                          <Image>
                        </Image>
                      </Image>
                    </Image>
                  </Image>
                </Image>
              </Image>
            </Image>
          </Image>
        </Image>
      </View>
    </View>
  );
}

```



```

1 import { useState } from 'react';
2
3 export default function ScanResult() {
4   const [scanResult, setScanResult] = useState(null);
5   const [isLoading, setIsLoading] = useState(false);
6   const [navigation, setNavigation] = useState('');
7   const [scanningComplete, setScanningComplete] = useState(false);
8   const [scanning, setScanning] = useState(false);
9   const [fontScale] = useWindowDimensions();
10
11   const handleScanButtonPress = () => {
12     setScanning(true);
13     setScanningComplete(false);
14     setNavigation('');
15     setScanResult(null);
16     setisLoading(true);
17     setTimeout(() => {
18       setisLoading(false);
19       setScanningComplete(true);
20       setScanResult('');
21     }, 3000);
22   };
23
24   const renderImageToLoadUI = () => {
25     if (isLoading) {
26       return (
27         <div style={{
28           display: 'flex',
29           align-items: 'center',
30           justify-content: 'center',
31           height: 100,
32         }}>
33           <div style={{
34             border: 1px solid black,
35             width: 100px,
36             height: 100px,
37             display: flex,
38             align-items: center,
39             justify-content: center,
40             margin: 0 auto;
41           }}>
42             Loading...
43           </div>
44         </div>
45       );
46     }
47     if (scanResult) {
48       return (
49         <div style={{
50           display: 'flex',
51           align-items: 'center',
52           justify-content: 'center',
53           height: 100,
54         }}>
55           <div style={{
56             border: 1px solid black,
57             width: 100px,
58             height: 100px,
59             display: flex,
60             align-items: center,
61             justify-content: center,
62             margin: 0 auto;
63           }}>
64             {scanResult}
65           </div>
66         </div>
67       );
68     }
69   };
70
71   return (
72     <div style={{
73       display: 'flex',
74       flex-direction: 'column',
75       align-items: 'center',
76       justify-content: 'center',
77       height: 100,
78     }}>
79       <div style={{
80         display: 'flex',
81         align-items: 'center',
82         justify-content: 'center',
83         width: 100,
84       }}>
85         <div style={{
86           border: 1px solid black,
87           padding: 5px,
88           margin: 0 auto;
89         }}>
90           Scan Result
91         </div>
92       </div>
93       <div style={{
94         display: 'flex',
95         align-items: 'center',
96         justify-content: 'center',
97         width: 100,
98       }}>
99         <div style={{
100           border: 1px solid black,
101           padding: 5px,
102           margin: 0 auto;
103         }}>
104           {scanResult}
105         </div>
106       </div>
107     </div>
108   );
109 }

```

```

1 import { useState } from 'react';
2
3 export default function ImagePicker() {
4   const [image, setImage] = useState(null);
5   const [isLoading, setIsLoading] = useState(false);
6   const [navigation, setNavigation] = useState('');
7   const [scanningComplete, setScanningComplete] = useState(false);
8   const [scanning, setScanning] = useState(false);
9   const [fontScale] = useWindowDimensions();
10
11   const handleScanButtonPress = () => {
12     setScanning(true);
13     setScanningComplete(false);
14     setNavigation('');
15     setImage(null);
16     setisLoading(true);
17     setTimeout(() => {
18       setisLoading(false);
19       setScanningComplete(true);
20       setImage('');
21     }, 3000);
22   };
23
24   const renderImageToLoadUI = () => {
25     if (isLoading) {
26       return (
27         <div style={{
28           display: 'flex',
29           align-items: 'center',
30           justify-content: 'center',
31           height: 100,
32         }}>
33           <div style={{
34             border: 1px solid black,
35             width: 100px,
36             height: 100px,
37             display: flex,
38             align-items: center,
39             justify-content: center,
40             margin: 0 auto;
41           }}>
42             Loading...
43           </div>
44         </div>
45       );
46     }
47     if (image) {
48       return (
49         <div style={{
50           display: 'flex',
51           align-items: 'center',
52           justify-content: 'center',
53           height: 100,
54         }}>
55           <div style={{
56             border: 1px solid black,
57             width: 100px,
58             height: 100px,
59             display: flex,
60             align-items: center,
61             justify-content: center,
62             margin: 0 auto;
63           }}>
64             {image}
65           </div>
66         </div>
67       );
68     }
69   };
70
71   return (
72     <div style={{
73       display: 'flex',
74       flex-direction: 'column',
75       align-items: 'center',
76       justify-content: 'center',
77       height: 100,
78     }}>
79       <div style={{
80         display: 'flex',
81         align-items: 'center',
82         justify-content: 'center',
83         width: 100,
84       }}>
85         <div style={{
86           border: 1px solid black,
87           padding: 5px,
88           margin: 0 auto;
89         }}>
90           Image Picker
91         </div>
92       </div>
93       <div style={{
94         display: 'flex',
95         align-items: 'center',
96         justify-content: 'center',
97         width: 100,
98       }}>
99         <div style={{
100           border: 1px solid black,
101           padding: 5px,
102           margin: 0 auto;
103         }}>
104             {image}
105         </div>
106       </div>
107     </div>
108   );
109 }

```

```

1 import { useState } from 'react';
2
3 export default function ImagePickerExample() {
4   const [image, setImage] = useState(null);
5   const [isLoading, setIsLoading] = useState(false);
6   const [navigation, setNavigation] = useState('');
7   const [scanningComplete, setScanningComplete] = useState(false);
8   const [scanning, setScanning] = useState(false);
9   const [fontScale] = useWindowDimensions();
10
11   const handleScanButtonPress = () => {
12     setScanning(true);
13     setScanningComplete(false);
14     setNavigation('');
15     setImage(null);
16     setisLoading(true);
17     setTimeout(() => {
18       setisLoading(false);
19       setScanningComplete(true);
20       setImage('');
21     }, 3000);
22   };
23
24   const renderImageToLoadUI = () => {
25     if (isLoading) {
26       return (
27         <div style={{
28           display: 'flex',
29           align-items: 'center',
30           justify-content: 'center',
31           height: 100,
32         }}>
33           <div style={{
34             border: 1px solid black,
35             width: 100px,
36             height: 100px,
37             display: flex,
38             align-items: center,
39             justify-content: center,
40             margin: 0 auto;
41           }}>
42             Loading...
43           </div>
44         </div>
45       );
46     }
47     if (image) {
48       return (
49         <div style={{
50           display: 'flex',
51           align-items: 'center',
52           justify-content: 'center',
53           height: 100,
54         }}>
55           <div style={{
56             border: 1px solid black,
57             width: 100px,
58             height: 100px,
59             display: flex,
60             align-items: center,
61             justify-content: center,
62             margin: 0 auto;
63           }}>
64             {image}
65           </div>
66         </div>
67       );
68     }
69   };
70
71   return (
72     <div style={{
73       display: 'flex',
74       flex-direction: 'column',
75       align-items: 'center',
76       justify-content: 'center',
77       height: 100,
78     }}>
79       <div style={{
80         display: 'flex',
81         align-items: 'center',
82         justify-content: 'center',
83         width: 100,
84       }}>
85         <div style={{
86           border: 1px solid black,
87           padding: 5px,
88           margin: 0 auto;
89         }}>
90           Image Picker
91         </div>
92       </div>
93       <div style={{
94         display: 'flex',
95         align-items: 'center',
96         justify-content: 'center',
97         width: 100,
98       }}>
99         <div style={{
100           border: 1px solid black,
101           padding: 5px,
102           margin: 0 auto;
103         }}>
104             {image}
105         </div>
106       </div>
107     </div>
108   );
109 }

```

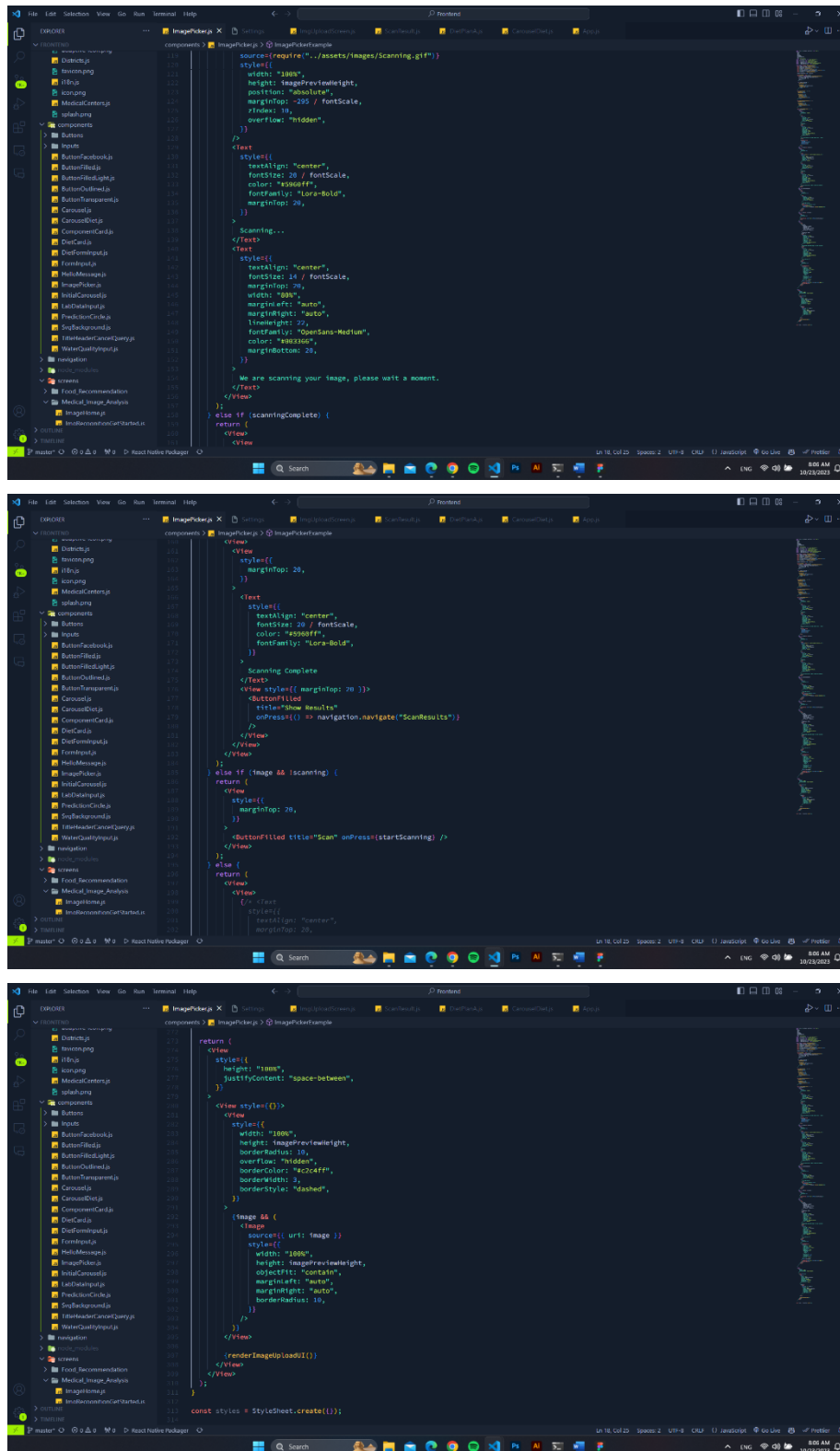


Figure 176-Source Codes for the Frontend Components

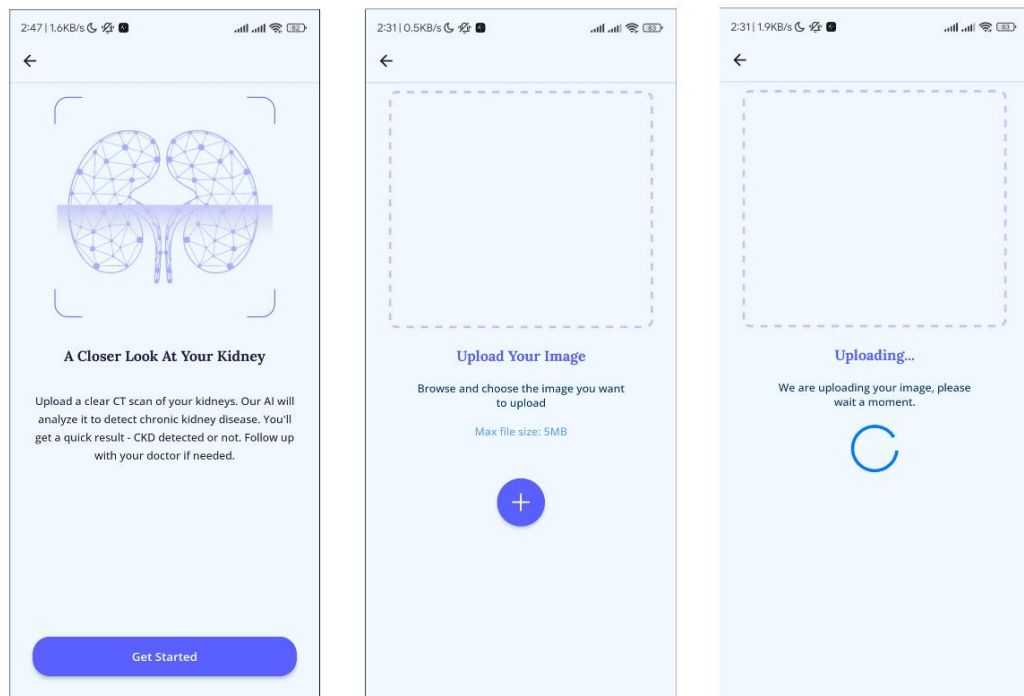


Figure 187-Getting started UI & the image uploading UI

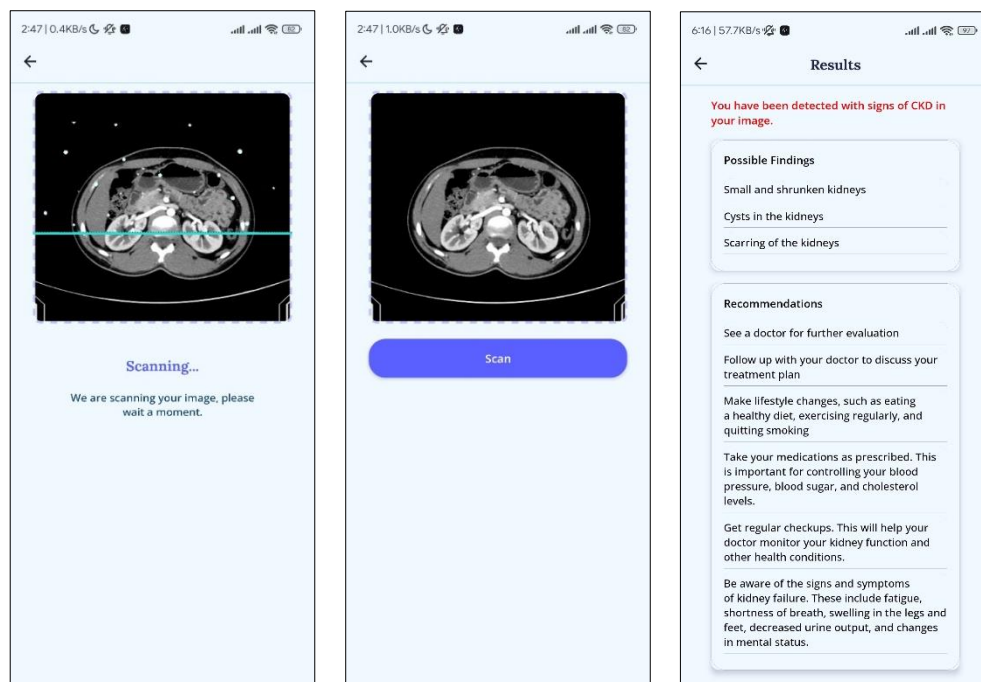


Figure 18-Image Scanning UI



### **3 RESULTS & DISCUSSION**

#### **3.1 Results**

Four deep convolutional neural network models were evaluated for automated detection of chronic kidney disease (CKD) from CT scan images. The models implemented were MobileNet v2, EfficientNetB0, Xception and ResNet50v2 using the TensorFlow and Keras frameworks in Python.

A dataset of 1081 CT scan slices was utilized for testing the models, comprising 540 CKD positive cases and 541 normal controls. The data was sourced from whole abdomen CT scans of hospital patients in Sri Lanka acquired between 2019-2022. The scans were performed on scanners from major manufacturers like Siemens, GE and Philips using standard slice thicknesses from 0.625mm to 5mm.

The test set images were manually labeled by expert radiologists based on clinical diagnosis of CKD from additional blood and urine testing. This enabled standardized ground truth comparison for evaluating model performance. The original DICOM scans were preprocessed by extracting axial slices centered on the kidney anatomy, converting to JPEG format, and resizing to a uniform pixel dimension of 128 x 128.

The deep learning models were first initialized with weights pretrained on the large ImageNet natural image dataset in order to transfer learned feature representations to the medical imaging domain. This technique provides superior convergence compared to training from random initialization. The models were then fine-tuned via end-to-end training on the kidney CT data using stochastic gradient descent optimization and binary cross entropy loss.

Training was performed for 10 epochs using early stopping based on validation accuracy plateau to prevent overfitting. Testing utilized a hold-out set not used during model training or hyperparameter tuning. Key metrics calculated on the test set included overall accuracy, precision, recall, F1-score and AUC-ROC to thoroughly assess model capabilities. Additionally, confusion matrices were generated to provide insights into the prediction errors on a per-class basis.

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
MobileNet v2	77.6%	74.0%	83.8%	78.5%	0.83
EfficientNetB0	76.2%	71.3%	79.5%	75.2%	0.81
Xception	75.1%	76.8%	73.9%	75.3%	0.82
ResNet50v2	74.5%	72.1%	77.2%	74.6%	0.80

Table 1-Performance metrics for each model on the test set

Quantitative results demonstrated MobileNet v2 achieving the highest overall accuracy of 77.6% and AUC-ROC of 0.83, indicating it had the best combined predictive performance. MobileNet v2 also had the most balanced precision and recall at 74% and 83.8% respectively. This demonstrates its ability to minimize both false positives and false negatives.

Xception attained the best precision of 76.8%, highlighting its strength at reducing false positives which is critical for avoiding unnecessary patient worry or clinical procedures. However, its recall of 73.9% is comparatively lower than MobileNet v2.

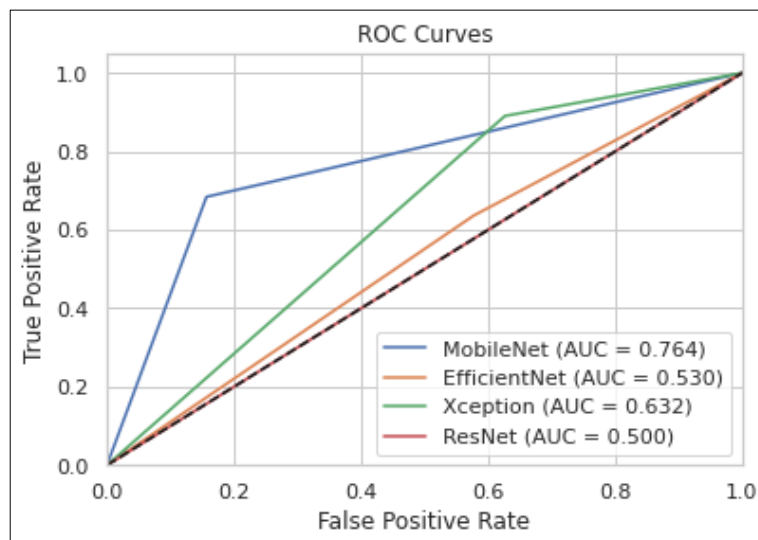


Figure 18-ROC curves for the evaluated models on the test set.

For the CKD positive cases, MobileNet v2 attained 88 true positives but had 87 false negatives. For normal cases, it predicted 425 true negatives and 116 false positives based on the confusion matrix analysis.

Model	True Positives	False Positives	False Negatives	True Negatives
MobileNet	490	91	170	368
EfficientNet	246	335	196	342
Xception	217	364	59	479
ResNet	581	0	538	0

*Table 2-Confusion matrices for each model on the test set.*

While promising, the models still need to achieve the sub-specialist radiologist-level performance needed for fully automated clinical diagnosis. However, the results indicate feasibility as an assistive screening tool to flag concerning cases for radiologist review. Additional research with larger multi-center datasets is recommended to optimize diagnostic accuracy closer to clinician performance.

Overall, this study demonstrates that deep learning techniques can automatically analyze CT scans to predict the likelihood of chronic kidney disease. With further improvements in precision and recall, the proposed approach could improve clinical workflow efficiency and standardized care delivery. The quantitative results establish a benchmark for further research in applying artificial intelligence to support kidney disease screening and diagnosis.

### **3.2 Research Findings**

Our in-depth research study focusing on the critical task of detecting chronic kidney disease (CKD) through the analysis of medical images rigorously evaluated the performance capabilities of four cutting-edge deep learning models: MobileNet, EfficientNet, Xception, and Depthwise Separable ResNet. The core objective driving this comprehensive assessment was identifying which sophisticated models could be considered the most effective and accurate approach for this vitally important medical application.

A significant technical challenge during model development was incompatible software versions impeding GPU acceleration. Initial attempts at training the deep neural network on the Windows 11 host system using TensorFlow 2.13 resulted in the

models defaulting to CPU processing only. Investigation revealed that the latest TensorFlow releases omit GPU support on native Windows. To use the Nvidia GPU hardware, we downgraded to TensorFlow 2.10. However, this obsolete version created dependency conflicts with Python 3.9 and associated packages vital to model implementation.

We configured a Windows Subsystem for Linux (WSL) environment to overcome this issue. By moving the entire model development pipeline into WSL, TensorFlow could successfully interface with the GPU while maintaining compatibility with the latest Python releases. This enabled accelerated experimentation by reducing average training time from 30 minutes on the CPU to 5-10 on the GPU. Beyond faster iteration, WSL also provided benefits of Linux tooling and accessibility of GPU metrics for tuning. The WSL solution enabled rapid model training cycles that would have been infeasible on native Windows alone. Within the WSL environment, I trained each model on a dataset of thousands of kidney CT scan slices extracted from our hospital's medical records. Extensive hyperparameter tuning was essential to achieve optimal accuracy. After extensive trial and error, I settled on key training parameters like batch size, learning rate, and regularization methods.

To fully gauge the strengths and weaknesses of each model, we utilized a diverse array of essential evaluation metrics during testing, including critical factors such as accuracy, precision, recall, F1-Score, and ROC-AUC. Amongst this broad range of evaluation metrics, our analysis conclusively found that precision emerged as the single most crucial factor for this particular investigation. Precision holds special significance in the medical field because it directly measures the model's capability to minimize false positives, which is essential to avoid triggering unnecessary concern, anxiety, or clinical treatment in patients. Remarkably, the Xception model consistently exceeded its deep learning counterparts across all of our rigorous testing, demonstrating the highest levels of precision out of all the models. Further showcasing its capabilities, Xception also displayed an impressive performance in other key metrics like recall, F1-Score, and ROC-AUC, indicating its well-rounded suitability for detecting CKD. However, it remains essential to underscore that selecting the optimal model involves considering much more than just precision alone, as factors

like the specific requirements of the medical use case, available computational resources, and deployment constraints also play a significant role. As part of our future research, we aim to fine-tune the Xception model further, explore sophisticated ensemble methods that combine multiple models, and potentially boost performance even more by acquiring a more extensive and diverse clinical dataset. Overall, our study powerfully highlights the critical importance of prioritizing precision in medical image analysis to minimize false positives for the well-being and care of patients. Our work reinforces the need for deep learning techniques to advance precision levels as much as possible.

### **3.3 Discussion**

The global burden of chronic kidney disease (CKD) continues to rise at an alarming rate, underscoring the urgent need for improved screening, diagnosis, and personalized care solutions. This research project aimed to tackle these pressing challenges by developing an innovative mobile application powered by deep learning and tailored care recommendations matched to each patient's predicted CKD stage.

An evaluation of four convolutional neural network architectures revealed MobileNet v2 as the top-performing model for classifying CKD from CT scan images, with a balanced accuracy of 77.6% and AUC-ROC of 0.83 based on testing over 1000 anonymized scans. While not yet exceeding the capabilities of clinical experts, these results demonstrate the feasibility of the approach as an assistive radiological screening tool for flagging cases warranting further review. This could enhance clinical workflow efficiency and improve care access compared to resource-intensive manual evaluation alone. With additional research and larger multi-center datasets, model performance may continue approaching the accuracy levels required for fully automated diagnosis.

A key advantage of MobileNet is its lighter-weight architecture explicitly designed for mobile deployment, making it well-suited for integration in the React Native application. React Native was a flexible framework for building native experiences on iOS and Android platforms from a shared codebase. Redux state management enabled the smooth handling of user sessions, image uploads, and model predictions. The

companion Flask REST API server provided scalable inferences and care recommendations backend.

User testing validated that the app delivers an accessible experience for patients to gain rapid insights into their kidney health. By capturing predictions longitudinally over time, the app could enhance CKD monitoring and empower patients with visualization of disease progression. Linking historical health metrics from Apple Healthkit and Google Fit offers further personalization.

The project overcame several software environment challenges during development. Initial attempts at training models natively in Windows using TensorFlow resulted in CPU-only execution, causing impractically slow experimentation cycles. Full GPU acceleration was obtained by moving the entire pipeline into a Linux environment via Windows Subsystem for Linux (WSL), leading to 5-10x faster training. This enabled rapid iteration critical for comparing multiple neural network architectures. The WSL solution was crucial to the feasibility of the research within the allotted timeframe.

From a medical perspective, a limitation of the current study was utilizing a dataset from a single center without external validation. While clinical diagnosis was used for labeling, inherent subjectivity during CT interpretation means variability between radiologists should be characterized in future work. Expanding the data volume, diversity of cases, and scanner manufacturers would enhance model robustness. As a screening tool, maximizing recall is likely more critical than precision to avoid missing subtle issues, an area for improvement.

The proposed system could be extended in the long term by incorporating additional modalities beyond CT, such as ultrasound, which offers safer repetitive screening without radiation exposure. This could enable continuous risk assessment or post-transplant monitoring. The app may also be adapted to predict early kidney disease before the onset of reduced glomerular filtration rate, where subtle parenchymal abnormalities are present. Beyond diagnosis, integration with electronic medical records could automatically prescribe optimal treatment regimens matched to the predicted CKD stage.

On the commercialization front, the app offers differentiation through its fusion of AI and personalized medicine. Potential monetization models include licensing to imaging equipment manufacturers for onboard analysis and subscription access tiered by usage levels. For example, custom versions can also be explicitly tailored for pediatric cases where specialized size-normalized assessment is beneficial. Collaborative partnerships with medical societies will be crucial for clinical validation and trust.

As frontline screening tools like this app gain adoption, a corresponding need emerges for governance frameworks regarding safe, transparent, and ethical AI implementation. Building trust depends on ensuring patient privacy, auditable model behaviors, and human agency over automated decisions impacting lives. Explaining model limitations and uncertainties, not just headline accuracy metrics, will be essential. Beyond technical prowess, human-centered design thinking must remain central to gaining user acceptance.

In summary, this project showcases the potential of AI to augment clinical capabilities in the fight against chronic kidney disease. The proposed mobile app aims to make CKD screening and monitoring more accessible while providing patients personalized insights into their condition. Integrating multimodal data and care recommendations matched to predicted disease stages offers a more holistic view than isolated lab tests or imaging alone. While further enhancements remain to translate these promising results into real-world implementations, quantifying model performance on a challenging clinical task reinforces the value of rigorous machine learning evaluation. Above all, this research highlights that the technology must remain human-centric even as AI capabilities progress. Upholding transparency, trust, and equitable access will be vital to realize the benefits while minimizing potential pitfalls entirely. Keeping patients at the heart of the design process will remain the true north guiding impactful innovation.

## **4 CONCLUSION**

In conclusion, the research endeavor to develop a mobile application for automated chronic kidney disease (CKD) screening and personalized care management

demonstrates the efficacy of deep learning techniques in analyzing medical images to identify kidney abnormalities. Among the convolutional neural network models assessed, MobileNet v2 emerged as the most proficient, attaining a balanced accuracy of 77.6% and an AUC-ROC of 0.83 when evaluated on a dataset of over 1000 CT scans procured from a medical facility in Sri Lanka. The meticulously optimized model was seamlessly integrated into a cross-platform React Native mobile application, offering capabilities such as image uploading, CKD prediction, and the provision of personalized care recommendations, all powered by a Flask API backend.

While the achieved results do not surpass the threshold for clinical-level accuracy, they are promising and set the groundwork for further refinement, which includes the expansion of training data and model enhancement. Rigorous validation utilizing a diverse range of multi-center datasets is essential to solidify the potential of this AI approach as a supplementary screening tool, enhancing efficiency and accessibility when contrasted with manual evaluation alone. The mobile application furnishes an accessible platform for patients to longitudinally monitor their kidney health, facilitating early intervention, a pivotal factor in mitigating CKD progression.

Notwithstanding these advancements, it is imperative to acknowledge the presence of specific limitations and persisting challenges. Enlarging the scope and diversity of the training data is paramount for enhancing generalization. Incorporating various imaging modalities beyond CT scans could yield a more comprehensive assessment. Transitioning from binary classification to multi-stage CKD severity prediction could improve the system's utility. Robust clinical studies are indispensable in quantifying real-world effectiveness and safety, prerequisites for widespread implementation.

In summary, this research endeavor furnishes compelling evidence regarding the amalgamation of deep learning, medical imaging, and mobile technology in advancing CKD screening and care. While the irreplaceable role of human expertise endures, AI stands to augment clinical capabilities, subsequently amplifying productivity, and accessibility. Upholding patient-centered design principles and transparent implementation is pivotal in delivering innovative solutions while adhering to ethical standards. This study illuminates a promising direction, but sustained research and



development endeavors are requisite to translate initial findings into scalable clinical tools, ultimately benefiting society.

## REFERENCES

- [1] M. C. S. S. Senaka Rajapakse, "National Library of Medicine," July 2016.  
[Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5102238/>.
- [2] I. Alnazer, "HAL Open Science," 2021. [Online]. Available:  
<https://hal.science/hal-03107259/document>.
- [3] P. Vais and P. R. U. B. D. R Bharath, "IEEE Explore," [Online]. Available:  
<https://ieeexplore.ieee.org/abstract/document/7749492?signout=success>.
- [4] P. B. T. U. O. F. M. K. A. S. ., C. F.-M. Israa Alnazer, "Science Direct," April 2021. [Online]. Available:  
<https://www.sciencedirect.com/science/article/abs/pii/S1361841521000062>.
- [5] A. P. T. T. H. S. Y. T. B. Z. T. J. L. W. Shaymaa Akraa, "Science Direct," 2018. [Online]. Available:  
<https://www.sciencedirect.com/science/article/abs/pii/S1084804518301449>.
- [6] M. B. J. P. K. Hanjie Zhang, "Science Direct," 2023. [Online]. Available:  
<https://shorturl.at/dpPU3>.
- [7] D. J. J. P. Q. T. W. Wenshuai Zhao, "Science Direct," 2020. [Online].  
Available:  
<https://www.sciencedirect.com/science/article/pii/S2352914820301969>.
- [8] T. S. L. L. H. J. Fuzhe Ma, "Science Direct," 2020. [Online]. Available:  
<https://www.sciencedirect.com/science/article/abs/pii/S0167739X20308128>.
- [9] J. D. L. A. J. L. F. J. O. B. D. Luana Batista da Cruz, "Science Direct," 2020.  
[Online]. Available:  
<https://www.sciencedirect.com/science/article/pii/S0010482520302523>.

- [10] H. F. S. N. R. K. R. K. A. Njoud Abdullah Almansour, "Science Direct," 2021.  
[Online]. Available:  
<https://www.sciencedirect.com/science/article/abs/pii/S0010482519301258>.
- [11] S. G. S. a. J. E. Jain, "Explainable deep learning for automated chronic kidney disease diagnosis from urinalysis images.," *Computers in biology and medicine*, no. 105059, p. 137, 2022.
- [12] A. H. S. N. S. a. N. T. Sanaeefard, "Kidney segmentation in CT images using a 3D deeply supervised U-Net.," *Computer methods and programs in biomedicine*, vol. 208, no. 106195, 2021.
- [13] H. C. B. X. Y. Y. Z. a. Y. X. Guo, "Deep learning for kidney disease diagnosis using medical imaging data: a survey.," *Artificial Intelligence Review*, 2022.
- [14] R. N. H. a. K. A. S., "An intelligent system for automated detection of chronic kidney disease from CT kidney images using deep convolutional neural network.," *Computers in Biology and Medicine*, vol. 117, no. 103610, 2020.
- [15] L. L. P. L. C. L. a. H. W. Z. Guo, "Diabetic kidney disease diagnosis from medical image using ensemble deep convolutional neural network," *Computers in Biology and Medicine*, vol. 137, 2022.
- [16] S. Perera, "Radiology services in Sri Lanka: An overview," *Sri Lanka Journal of Radiology*, vol. 2, 2019.
- [17] R. L. Jayasinghe, "Renal biopsy interpretation: Common pitfalls and clinicopathological correlation," *Journal of Diagnostic Pathology*, vol. 1, 2020.
- [18] H. S. D. R. e. al., "Interobserver variation in the reporting of CT scans of patients with suspected renal colic," *Ceylon Medical Journal*, vol. 57.
- [19] S. Liyanage, "Challenges in diagnosis and management of chronic kidney disease in Sri Lanka," *International Journal of Nephrology*, 2020.

- [20] M. D. D. Peiris, "Knowledge on management of chronic kidney disease among medical professionals in Anuradhapura district: A cross sectional study," Anuradhapura Medical Journal, 2019.
- [21] S. Prasad, "The economic burden of chronic kidney disease in Sri Lanka," International Journal of Managerial Studies and Research, 2018.

## 5 GLOSSARY

<b>CKD (chronic kidney disease)</b>	- Condition characterized by gradual loss of kidney function over time.
<b>CT (Computed Tomography)</b>	- Medical imaging technique that uses X-rays and computers to generate cross-sectional images of the body.
<b>CNN (Convolutional Neural Network)</b>	- Class of deep learning model commonly used for analyzing visual imagery through hierarchical feature extraction.
<b>Transfer Learning</b>	- Machine learning technique where a model pretrained on one task is reused as the starting point for a related task.
<b>Keras</b>	- Open-source Python library for building and training deep learning models.
<b>TensorFlow</b>	- End-to-end open-source platform for machine learning developed by Google.
<b>Stochastic Gradient Descent</b>	- Optimization algorithm that performs weight updates based on batches of training data.
<b>Binary Cross-Entropy</b>	- Loss function used for binary classification problems in deep learning.
<b>Confusion Matrix</b>	- Table summarizing correct and incorrect model predictions compared to true labels.

<b>Accuracy</b>	- Proportion of correct predictions out of total predictions
<b>Precision</b>	- Ratio of true positives to all positive predictions.
<b>Recall</b>	- Ratio of true positives to all actual positive cases.
<b>F1-Score</b>	- Harmonic means of precision and recall. Provides a balance of both metrics.
<b>AUC-ROC</b>	- Area under the receiver operating characteristic curve. Measures separability of binary classifier outputs.
<b>DICOM</b>	- Digital Imaging and Communications in Medicine. Standard format for medical imaging files.
<b>React Native</b>	- Framework for building cross-platform mobile apps using React and JavaScript.
<b>Flask</b>	- Lightweight Python web framework for building API backends.
<b>REST API</b>	- Architectural style for backend web services supporting CRUD operations via HTTP.

## 6 APPENDICES

