KSÜ Mühendislik Bilimleri Dergisi, 28(2), 2025
Araştırma Makalesi

KSU J Eng Sci, 28(2), 2025
Research Article

Kahramanmaras Sutcu Imam University
Journal of Engineering Sciences

Geliş Tarihi : 21.01.2025
Kabul Tarihi :18.03.2025

Received Date : 21.01.2025
Accepted Date : 18.03.2025

MACHINE LEARNING-BASED MODEL DEVELOPMENT IN MOBILE HEALTH APPLICATIONS AND DEVICE-CLOUD INTEGRATION

MOBİL SAĞLIK UYGULAMALARINDA MAKİNE ÖĞRENMESİ TEMELLİ MODEL GELİŞTİRME VE MODELİN CİHAZ-BULUT ENTEGRASYONU

Özge ÇİÇEK[1] (ORCID: 0009-0004-8982-3341)
Sema CANDEMİR[1*] (ORCID: 0000-0001-8619-5619)

[1] Eskisehir Technical University, Artificial Intelligence in Healthcare Laboratory, Computer Engineering Department, Eskisehir, Türkiye

*Sorumlu Yazar / Corresponding Author: Sema CANDEMIR, semacandemir@eskisehir.edu.tr

ABSTRACT

Mobile Health (mHealth) uses mobile devices and wireless technology to support healthcare practices, enabling widespread access to health services. Recent advancements in machine learning (ML) have enhanced healthcare by improving disease diagnosis and monitoring. However, integrating and executing machine learning models, especially those based on image processing and deep learning, on mobile devices can be challenging due to limited processing power and storage capacity. This study describes the steps of developing an ML-based model and its integration into mobile devices and cloud environments. A skin disease predictor using the MobileNet architecture was developed as a use-case mHealth application. Techniques such as transfer learning, data augmentation, and focal loss were employed to enhance model performance. The mHealth model was then integrated into a mobile device and the cloud environment. The on-device model exhibited faster prediction times (average 108.3 ms) compared to the cloud-based model (average 1281.2 ms). While on-device deployment ensured data privacy and offline functionality, the cloud approach provided scalability and easier updates, but at the expense of latency and data security. By providing a comparative analysis, this work demonstrates the feasibility of integrating ML models into mHealth applications, emphasizing the importance of balancing performance, cost, and usability.

Keywords: Mobile health, machine learning, MobileNet, device-cloud integration

ÖZET

Mobil Sağlık (mHealth), mobil cihazlar ve kablosuz teknolojiyi kullanarak sağlık hizmetlerini destekleyen uygulamaları içerir ve sağlık hizmetlerine yaygın erişim sağlar. Makine öğrenimi (ML) alanındaki son gelişmeler, hastalık teşhisi ve takibini iyileştirerek sağlık hizmetlerini geliştirmiştir. Ancak, özellikle görüntü işleme ve derin öğrenmeye dayalı makine öğrenimi modellerini mobil cihazlara entegre etmek ve çalıştırmak, sınırlı işlem gücü ve depolama kapasitesi nedeniyle zor olabilir. Bu çalışma, bir ML tabanlı modelin geliştirilmesi ve mobil cihazlar ile bulut ortamlarına entegrasyonu adımlarını açıklamaktadır. Örnek mHealth uygulaması olarak MobileNet mimarisi kullanılarak bir cilt hastalığı tahmin modeli geliştirilmiştir. Model performansını artırmak için transfer öğrenimi, veri artırma ve focal loss gibi teknikler kullanılmıştır. Eğitilmiş mHealth modeli daha sonra bir mobil cihaza ve bulut ortamına entegre edilmiştir. Cihaz üzerindeki model, bulut tabanlı modele kıyasla daha hızlı tahmin süreleri (cihaz üzerindeki model ortalama 108,3 ms) sergilemiştir (bulut tabanlı model ortalama 1281,2 ms). Cihaz üzerindeki dağıtım, veri gizliliği ve çevrimdışı işlevsellik sağlarken, bulut yaklaşımı ölçeklenebilirlik ve daha kolay güncellemeler sunmuş, ancak gecikme süresi ve veri güvenliği açısından dezavantajlar yaratmıştır. Bu çalışma ML modellerinin mHealth uygulamalarına entegrasyonunun uygulanabilirliğini göstermekte, karşılaştırmalı bir analiz sunmakta ve performans, maliyet ve kullanılabilirlik arasında bir denge kurmanın önemini vurgulamaktadır.

Anahtar Kelimeler: Mobil sağlık, makine öğrenmesi, MobileNet, cihaz-bulut entegrasyonu

ToCite: ÇİÇEK, Ö., & CANDEMİR, S., (2025). MACHINE LEARNING-BASED MODEL DEVELOPMENT IN MOBILE HEALTH APPLICATIONS AND DEVICE-CLOUD INTEGRATION. *Kahramanmaras Sutcu Imam University Journal of Engineering Sciences*, 28(2), 872-882.

INTRODUCTION

With rapid technological advancements and increasing mobile phone usage, the number of health sector mobile applications has grown. These applications enable effective participation of patients and healthcare providers, ultimately improving patient outcomes (Sama et al.,2014). Mobile Health (mHealth) refers to using mobile devices and wireless technology to support medical and public health practices, enabling instant access to health solutions and services, regardless of location. mHealth allows patients and their families to track their own care and customize healthcare services according to their needs, making healthcare applications portable, easily accessible, and personalized (Silva et al.,2015). The widespread usage of mobile devices, low cost of developing mobile applications, and their flexibility have driven rapid growth in healthcare mobile applications. According to the World Health Organization (https://www.who.int/), ~90% of the world's population can benefit from mobile technologies at a relatively low cost. A study by the Health Services Informatics Institute (Molina-Recio et al.,2015) found over 40,000 health and medical applications in Apple stores, increasing to 97,000 when including applications from the Play Store and other platforms. mHealth applications have become the third fastest-growing sector after games and utility programs.

Machine Learning (ML) models analyze medical data, learn distinctive disease patterns, and make predictions for new data. Recently, these models, especially deep learning approaches, have shown great potential in improving healthcare. However, running ML-based models, particularly deep learning models for health applications, on mobile devices is challenging due to limited processing power and storage capacity. Therefore, developing efficient ML models tailored for mobile environments and exploring cloud integration to mitigate device limitations is necessary.

This study outlines the development stages of an ML-based model for mHealth solutions, its integration into both mobile devices and the cloud, and provides a comparison of these environments. The study's contributions are as follows: (i) Given its potential as an mHealth application, the focus is on creating an ML based model to predict skin diseases. The entire process is detailed. The model is trained with a comprehensive dataset to learn distinctive features specific to various skin conditions. (ii) Critical factors in developing a mobile application include the model's high performance, fast feedback, and efficient use of storage. Considering these factors, we chose the MobileNet architecture (Howard et al.,2014) as the base of our model due to its space efficiency, fewer parameters, and sufficient layers for positive classification performance. (iii) Data sensitivity is a key concern in healthcare applications. For this reason, deploying the model directly on a device may be a better option. This study explains how to integrate a trained model into a device. (iv) Deep learning models often require many layers and large datasets for good performance. However, these models can take up a lot of space and may not run efficiently on mobile devices with limited hardware. To address this, integrating the model into the cloud for processing can be a practical alternative. This study also details the process of cloud integration. (v) Finally, the study compares the advantages and disadvantages of on-device versus cloud integration, including a comparison of their response times using the same test data.

The rest of the study is structured as follows: Related Work section provides a brief overview of related studies and highlights the differences between our work and existing literature. Materials and Methods section describes the model development and its integration into both device and cloud environments. Experimental Results and Analysis section presents the experimental results. Conclusion section concludes the study by summarizing our findings and provides comparative insights into the integration approaches of on-device and cloud environments.

RELATED WORK

Many studies have examined mHealth applications. For example, Jung et al. developed an Android-based app to help diabetes patients track their condition (Jung et al.,2014). Similarly, Susanto et al. (Susanto et al.,2022) reviewed smartphone-integrated image recognition systems used in medical areas like dermatology, ophthalmology, nutrition, neurology, respiratory medicine, blood disorders, gynecology, and dentistry. Khan et al. (Khan and Alotaibi, 2020) conducted another review focusing on AI-based big data analysis tools in mHealth applications. They highlighted the use of mobile sensors like cameras, wearable devices, GPS, microphones, and accelerometers to monitor physical activities, track locations, and assess sleep patterns. These technologies significantly improve the effectiveness of mHealth applications.

Several studies have used machine learning to predict skin diseases. Esteva et al. applied Convolutional Neural Networks (CNN) to identify skin cancer from clinical images, achieving over 90% sensitivity and specificity in

distinguishing between malignant and benign lesions (Esteiva et al., 2017). Goceri conducted a comprehensive review (Goceri, 2021) and trained a MobileNet model with a hybrid loss function to classify skin diseases, achieving 94% accuracy. The study also designed a user interface for the model, but did not address how the model could be integrated into devices or the cloud. In a related study, Dai et al. (Dai et al., 2019) integrated a skin disease classification model into a mobile application, testing the model directly on the device. This study emphasized the benefits of running predictions on the device, such as keeping sensitive patient images secure and providing faster feedback. However, they did not optimize the model for storage space, which is crucial for mobile applications. They chose AlexNet model with 8 convolution layers, 61 million parameters, and 250 MB of storage (Han et al.,2015), making it inefficient for mHealth applications.

This study outlines the development of an ML model for mHealth solutions, focusing on its integration into mobile devices and the cloud, and comparing these two environments. Our study differs from similar research in the following ways: (i) We focused on developing a model capable of predicting skin diseases, providing an in-depth development outline. While Dai et al. (Dai et al.,2019) briefly touch on model development and device integration for predicting skin disease, our study explores these aspects more thoroughly. (ii) Critical factors for a mobile application include the model's high performance, fast feedback, and efficient use of storage space. Our study addresses these factors by training a MobileNet model, known for its compact size, fewer parameters, and adequate depth for robust performance. In contrast, Dai et al. (Dai et al.,2019) used an AlexNet model, which is less efficient in terms of storage space. Therefore, our study offers a better-planned solution for mobile implementation compared to Dai et al.'s work. (iii) Our study outlines the integration of the model into both device and cloud environments and compares the performance of integrated models using the same test data. In this regard, the study differs from both Goceri`s (Goceri, 2021) and Dai et al.'s (Dai et al., 2019) works.

MATERIALS AND METHODS

Dataset

This study uses the HAM10000 dataset (Tschandl et al.,2018) compiled by a dermatology professor from the Medical University of Vienna and Queensland University. The dataset contains 10,000 dermatoscopic images collected from patients over 20 years, along with metadata including age, gender, and cell type. It includes seven types of skin diseases: Melanocytic Nevi (nv), Benign Keratosis-like Lesions (bkl), Dermatofibroma (df), Vascular Lesions (vasc), Actinic Keratoses and Intraepithelial Carcinoma (akiec), Basal Cell Carcinoma (bcc), and Melanoma (mel). Table 1 lists the class labels and the number of samples in each class. Figure 1 shows example images from the dataset.

Table 1. Summary of Skin Disease Classes In The HAM10000 Dataset (Tschandl et al.,2018)

Class Label	Abbreviation	# of Samples	# of Training Samples after Augmentation
Actinic Keratoses and Intraepithelial Carcinoma	akiec	327	5684
Basal Cell Carcinoma	bcc	514	5668
Benign Keratosis-like Lesions	bkl	1,099	5896
Dermatofibroma	df	115	4747
Melanoma	mel	1,113	5886
Melanocytic Nevi	nv	6705	5979
Vascular Lesions	vasc	142	5570

Figure 1. Sample Images From The HAM10000 Dataset (Tschandl et al.,2018)

Model Architecture

The MobileNet architecture (Howard et al.,2014) was selected for model development due to its design for devices with limited computational power and storage capacity. MobileNet uses small kernel layers to minimize the number of parameters and computational costs. Table 2 compares MobileNet with other well-known models, showing its smaller size and fewer parameters, which make it a suitable choice for resource-constrained devices.

Table 2. Comparison Of Well-known Machine Learning Models By Size, Number Of Parameters, And Layers, Emphasizing MobileNet's Suitability For Resource-constrained Devices

Model	Size (MB)	# of Parameters	# of Layers
AlexNet	250	61 M	8
MobileNet	16	4.3 M	55
Vgg-16	528	138.4 M	16
ResNet50	98	25 M	107
Inception V3	92	23.9 M	189
DenseNet201	80	20.2 M	402

Model Training

Developing machine learning algorithms requires comprehensive training data. However, curating datasets for medical problems is particularly challenging due to the difficulty in obtaining expert annotations (Candemir et al.,2021). One way to overcome this challenge is to use pre-trained models on larger datasets from other domains (Weiss et al.,2016). In this study, we used a pre-trained MobileNet model with weights trained on ImageNet (Krizhevsky et al.,2017). Our development dataset includes samples from seven types of skin diseases, with varying numbers of instances for each class, as shown in Table 1. To address this class imbalance, we applied data augmentation to increase the training samples in underrepresented classes. The augmentation methods and their parameter settings are listed in Table 3. The number of training samples for each class after augmentation is listed in Table 1.

Table 3. Data Augmentation Techniques And Their Parameters Used To Address Class Imbalance During Training

Data Augmentation Method	Ratio
Rotation	180 degrees
Zoom	0.1
Height Shift Range	0.1
Width Shift Range	0.1

During model training, we used two types of loss functions: cross-entropy loss and focal loss. Cross entropy loss is commonly used metric for multi-class classification tasks, evaluating the model's predicted probabilities for each class. This loss function increases as the predicted probability deviates from the true label and treats each data point equally within each class. The formula for cross-entropy loss is:

$$L_{CE}(y_i, \hat{y}_i) = -\sum_{i=1}^{N} y_i \, log \; (\hat{y}_i) \tag{1}$$

where N denotes the number of classes to be predicted, y_i is the true value for class i, and $\hat{y}_i$ is the predicted value for class i.

The cross-entropy loss function does not account for the varying difficulty of classifying different classes or individual cases within each class. To address the inter-class and intra-class variability in the dataset, we used Focal Loss (Lin et al.,2017), which is designed to handle challenging cases by giving more weight to samples where the model struggles. This loss function adjusts the model's focus toward harder-to-classify samples, helping it learn from these cases, leading to improved performance across all classes. The formula for focal-loss is:

$$L_{Focal}(y_i, \hat{y}_i) = -\sum_{i=0}^{N} \alpha_i y_i (1-\hat{y}_i)^{\gamma} \, log \; (\hat{y}_i) \tag{2}$$

Where N represents the number of classes to be predicted, y_i represents the true value for class i, . d $\hat{y}_i$ represents the predicted value for class i. The parameters α and γ help balance the importance of classes and control the focus on

hard-to-classify examples, respectively. The implementation details of model development are summarized in Section 4.1.

Integrating Machine Learning Model to Mobile Device

Integrating ML-based mHealth models into mobile devices presents several challenges: (i) The limited computational resources of mobile devices make it difficult to run complex ML models, especially deep learning architectures with many parameters. (ii) The limited storage capacity of mobile devices complicates the deployment of these models, as deep learning models require large storage capacity to store the model's architecture and weights. (iii) Processing and transmitting health data requires data protection to comply with personal data protection laws. Given the sensitivity of health data, ensuring patient privacy is crucial. There are two main approaches to integrating ML models into mobile devices: On-Device Deployment and Cloud Deployment.

On-Device Deployment

In on-device deployment, the model is trained on a high-performance computer. Once trained, the model's architecture and weights are stored on a mobile device, allowing predictions to be performed directly on the device. The approach is illustrated in Figure 2. On-device deployment has several advantages. It ensures the privacy and security of patient data since the data remains on the device, reducing the risk of data breaches. It provides faster prediction feedback as the model runs locally. It also eliminates the need for an internet connection with offline functionality. This approach is also cost-effective compared to cloud deployment, as it avoids recurring cloud service fees. On-device deployment has limitations as well. There is a risk of data loss if the mobile device malfunctions without a backup. Furthermore, if the model requires retraining due to data changes (e.g., data shift) or performance improvements, the application needs to be reloaded on the device, which can be inconvenient for users.

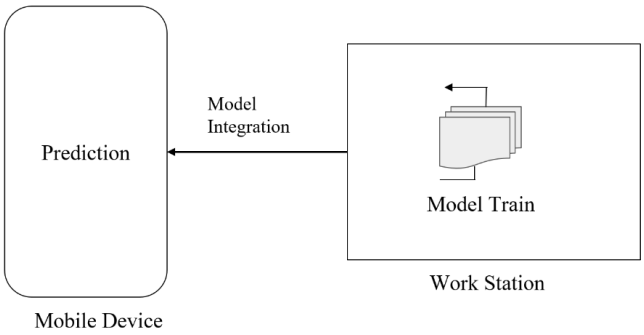

Figure 2. Illustration Of On-device Deployment Where The Model Is Trained On A High-performance Computer And Integrated Into A Mobile Device For Predictions

Cloud Deployment

Cloud deployment operates on a client-server model, where the trained model's architecture and weights are stored on a remote server. The mobile device communicates with the server to perform predictions. The mobile device sends query data to the cloud, the trained model generates predictions for the query data, and then sends the results back to the mobile device. Figure 3 illustrates the cloud deployment.

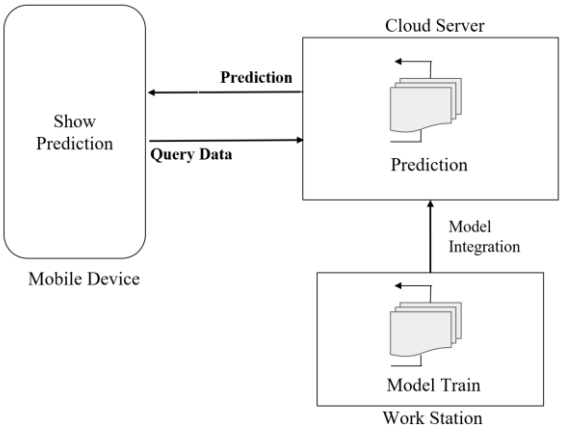

Figure 3. Illustration Of Cloud Deployment. The Prediction Is Carried Out On The Cloud Side, Displaying The Results On The Mobile Device

In cloud deployment, the prediction process is carried out on the cloud side. This eliminates the need for mobile devices to have high-end hardware. This approach also makes data more accessible and easier to manage. With patient consent, mHealth solution providers can access larger datasets, potentially developing advanced AI algorithms for more efficient patient care. Cloud deployment has drawbacks as well. It requires a high-speed network connection for smooth data transfer. Poor connectivity or interruptions can lead to delays, negatively impacting user experience. Additionally, storing data in the cloud incurs ongoing costs, increasing expenses for mHealth solution providers. Privacy and security of patient data are also concerns, as sharing data on the cloud raises the risk of sensitive information leaks. Developers must ensure that data is always encrypted and protected to prevent hacking and unauthorized access.

The advantages and disadvantages of integrating the model into both the device and cloud environments are summarized in Table 4.

Table 4. Comparative Advantages And Disadvantages Of On-device And Cloud-based Machine Learning Model Integration

Method	Advantages	Disadvantages
Device Integration	Ensures privacy and security of sensitive data Offline access Low cost of system requirements Fast access to prediction results	Difficulty in model updates Data loss in case of device failure Limited data storage and data processing hardware
Cloud Integration	Data accessibility and manageability To be independent of the device's data processing and data storage capacity	Network connection requirement Possible delays due to network connection Possibility of privacy breach for sensitive data Cloud system cost Relatively slow access to prediction results

EXPERIMENTAL RESULTS AND ANALYSIS

Implementation Details of the Prediction Model Development

The skin disease prediction model was developed using Python, TensorFlow version 2.0 with the Keras framework, and the scikit-learn library. The dataset was randomly divided into training (60%), validation (20%), and test subsets (20%). To maintain proportional representation of each lesion type due to dataset imbalance (see Table 1), stratification technique was applied during the splitting process. Data augmentation was used on the training data to further address class imbalance (see Table 3). The MobileNet architecture was chosen for its high performance and compact size, making it well-suited for mobile applications. A pre-trained MobileNet was fine-tuned using the training dataset, with optimal hyperparameters determined empirically through the validation data. The training parameters are listed in Table 5. For an efficient training process, we used early stopping and model checkpoints, which regularly save the model weights to capture the best performance at the end of the training process.

Table 5. Training Parameters Used To Develop The MobileNet-based Skin Disease Prediction Model

Parameter	Value
Model Network	MobileNet
Learning Rate	0.001
Batch Size	32
Epochs	15
Loss Functions	Cross Entropy and Focal Loss
Optimizer	Adam [17] $\beta1$ =0.9 and $\beta2$ =0.999
Processing Size	224 x 224

During training process, loss and accuracy were monitored to ensure effective learning and to avoid overfitting. The learning curves for both the training and validation datasets are shown in Figures 4 and 5. Figure 4 presents the learning curves using the cross-entropy loss function, and Figure 5 presents the learning curves using the focal loss function. These graphs show similar trends for both the training and validation subsets, indicating a successful optimization process. During the training process, loss and accuracy were monitored to ensure effective learning and to avoid overfitting. These graphs show similar trends for both datasets, indicating a successful optimization process. However, during the final stage of training, validation accuracy decreases while validation loss increases, which is indicative of onset of overfitting stage. If the training were continued beyond this stage, the model might start

memorizing patterns specific to the training dataset rather than learning generalizable features. To prevent this, we used early stopping technique, which stops the training process when an increase in validation loss is detected. As seen in the graphs, the training process does not continue beyond this point.

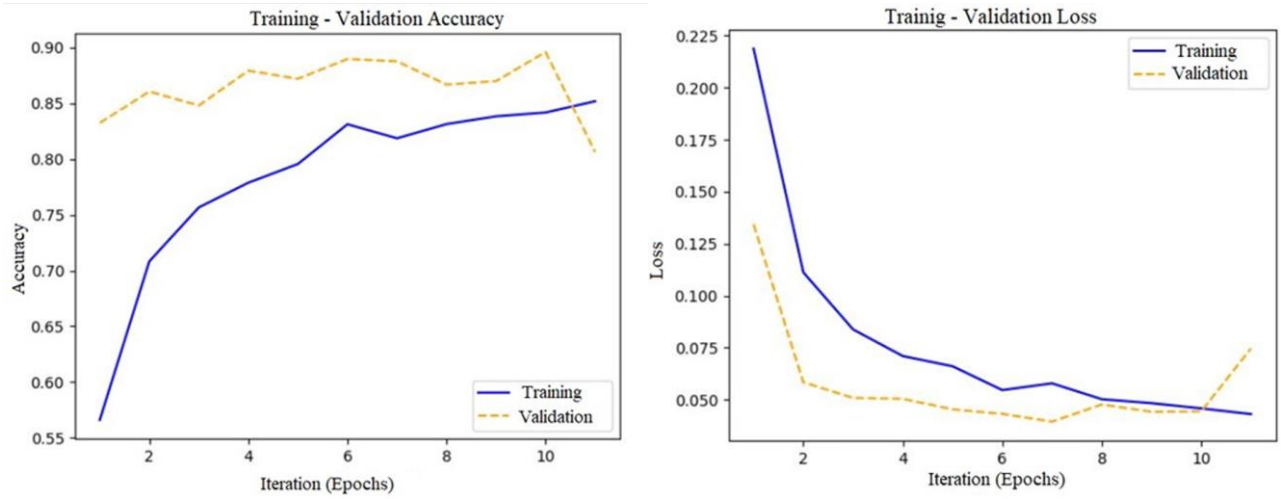

Figure 4. Learning Curves During The Training Process using Cross Entropy Loss Function. The Left Plot Shows The Accuracy Over Epochs For Both The Training And Validation Dataset. The Right Plot Illustrates The Corresponding Loss Curves

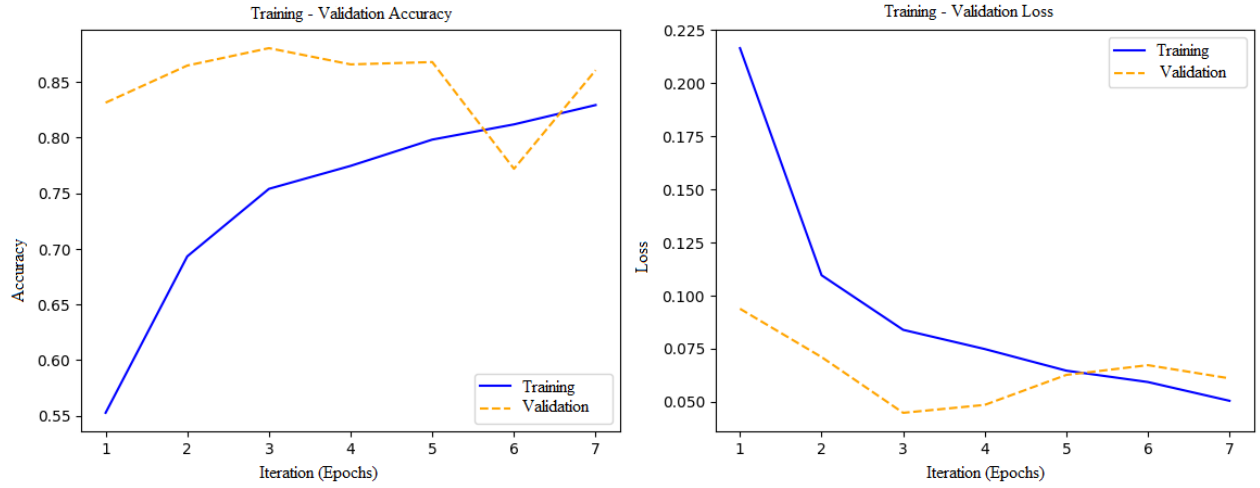

Figure 5. Learning Curves During The Training Process using Focal Loss Function. The Left Plot Shows The Accuracy Over Epochs For Both The Training And Validation Dataset. The Right Plot Illustrates The Corresponding Loss Curves

The performance of the skin disease prediction model was evaluated on the test data. The results are reported for each class with accuracy, F1 score, recall, and precision in Table 6 for the cross-entropy loss function and in Table 7 for the focal loss function.

Table 6. Evaluation Results Of The Skin Disease Prediction Model Trained With Cross-entropy Loss Function, Evaluated On The Test Dataset

Class	Accuracy	F1 Score	Recall	Precision
akiec	0.5926	0.5926	0.5926	0.5926
bcc	0.6562	0.7241	0.6562	0.8077
bkl	0.6076	0.6038	0.6076	0.6000
df	0.4286	0.5000	0.4286	0.6000
mel	0.2439	0.3509	0.2439	0.625
vasc	0.5833	0.7368	0.5833	1.000
Average	**0.5847**	**0.6381**	**0.5847**	**0.7375**

Table 7. Evaluation Results Of The Skin Disease Prediction Model Trained With Focal Loss Function, Evaluated On The Test Dataset

Class	Accuracy	F1 Score	Recall	Precision
akiec	0.6296	0.3846	0.3704	0.4000
bcc	0.625	0.6102	0.5806	0.6429
bkl	0.5570	0.6713	0.6000	0.7619
df	0.5714	0.6667	0.5714	0.8000
mel	0.4146	0.4675	0.4286	0.5143
vasc	0.5833	0.8421	0.7273	1.000
Average	**0.6222**	**0.6570**	**0.6082**	**0.7223**

When analyzing the overall performance, we observe that the choice of loss function impacts the model`s ability to classify different skin disease types. Compared to the performance of cross-entropy, the focal loss function enhances the prediction of less frequent and harder-to-classify classes, leading to an increase in overall accuracy from 0.5847 to 0.6222 (+3.75%) and an improvement in the overall F1-score from 0.6381 to 0.6570. Recall also improved from 0.5847 to 0.6082 (+2.35%), while precision decreased slightly from 0.7375 to 0.7223 (-1.53%). The overall results suggest that focal loss prioritizes harder-to-classify cases and less frequent classes.

There is a noticeable difference in performance for underrepresented disease classes. For instance, Dermatofibroma, one of the least frequent classes in the dataset, had 14.29% increase in accuracy (0.4286 to 0.5714). The F1- score increased from 0.5000 to 0.6667, indicating a better balance between precision and recall. Recall improved from 0.4286 to 0.5714, suggesting that the model identified more true positives. Melanoma, a critical yet challenging class due to high intra-class variability and its visual similarity to benign lesions, had a notable performance increase with focal loss. Accuracy improved by 17.08% (0.2439 to 0.4146), the F1-score increased from 0.3509 to 0.4675, and recall increased from 0.2439 to 0.4286, indicating that focal loss helped the model identify more melanoma cases. The improved performance suggests that focal loss effectively addresses class imbalance by giving additional weight to harder examples, helping the model recognize subtle distinctions in melanoma lesions. Vascular lesions also benefited from focal loss, with F1-score increasing from 0.7368 to 0.8421 and recall improving from 0.5833 to 0.7273 (+14.39%). These improvements suggest that focal loss was beneficial for handling less frequent and harder-to-classify skin diseases. While focal loss generally improves classification for less frequent classes, some performance trade-offs were observed. Performance losses were observed for Basal Cell Carcinoma and Actinic Keratoses. Melanoma also had very low accuracy with both loss functions. This could be due to melanoma's high intra-class variation and visual similarity to other skin lesions, making it difficult for the model to learn distinguishing features.

Overall, focal loss demonstrates a better ability to handle imbalanced classes by improving recall and F1-score, particularly for underrepresented skin disease types.

Implementation Details of the On-Device Deployment

In this part of the study, the developed model is integrated into a mobile device. The device used in this study is Samsung Galaxy S23, containing Qualcomm Snapdragon 8 Gen 2 processor with 8GB of RAM and 128 GB of internal storage. The operating system in the mobile device is Android 13, which is compatible with machine learning libraries. The mobile device also includes high-resolution Dynamic AMOLED 2X display and supports 5G connectivity for an optimal environment for both processing and user interaction. The integration of this model into such a powerful smartphone enables real-time, on-device inference, contributing to both speed and privacy.

The weights and architecture of the trained model were saved in a .h5 file format specific to Tensorflow-Keras, which is approximately 40 MB in size. For app development, two cross-platform frameworks were used: Flutter and React Native.

Flutter, supported by Google, is an open-source platform that offers a versatile interface for rapid development using the Dart language. Dart, also developed by Google, is designed for mobile app development and operates on both iOS and Android devices. The trained model was converted to a TensorFlow Lite model using TFLiteConverter. TensorFlow Lite is a framework within TensorFlow designed for developing lightweight and high-performance ML models for mobile and embedded devices. This conversion reduced the model size from 40 MB to 12 MB. To integrate the model into a Flutter application, the TensorFlow Lite Flutter plugin was utilized. The TensorFlow Lite model, along with the labels corresponding to skin disease classes, was added to the mobile application project. Once

the model integration was completed, it allows users to upload images and generate predictions using the trained model's architecture and weights stored on the device (See Figure 6).

We also used React Native with the TensorFlow.js React Native adapter. These platforms enable the model to run on both iOS and Android devices. The .h file, which contains the model architecture and weights, was converted to the TensorFlow.js format. This conversion divided the large weight file into smaller 5 MB parts. To further reduce the model size, quantization techniques were applied by decreasing the precision from 32-bit to 16-bit. The reduced model was then divided into two files: the model.json file, which contains the model's architecture, and the *.bin file, which stores the model's weights as binary format. These files were then added to the mobile application project using TensorFlow's React Native adapter. Once the model is uploaded, the users can upload a query image, and the model predicts the skin disease.

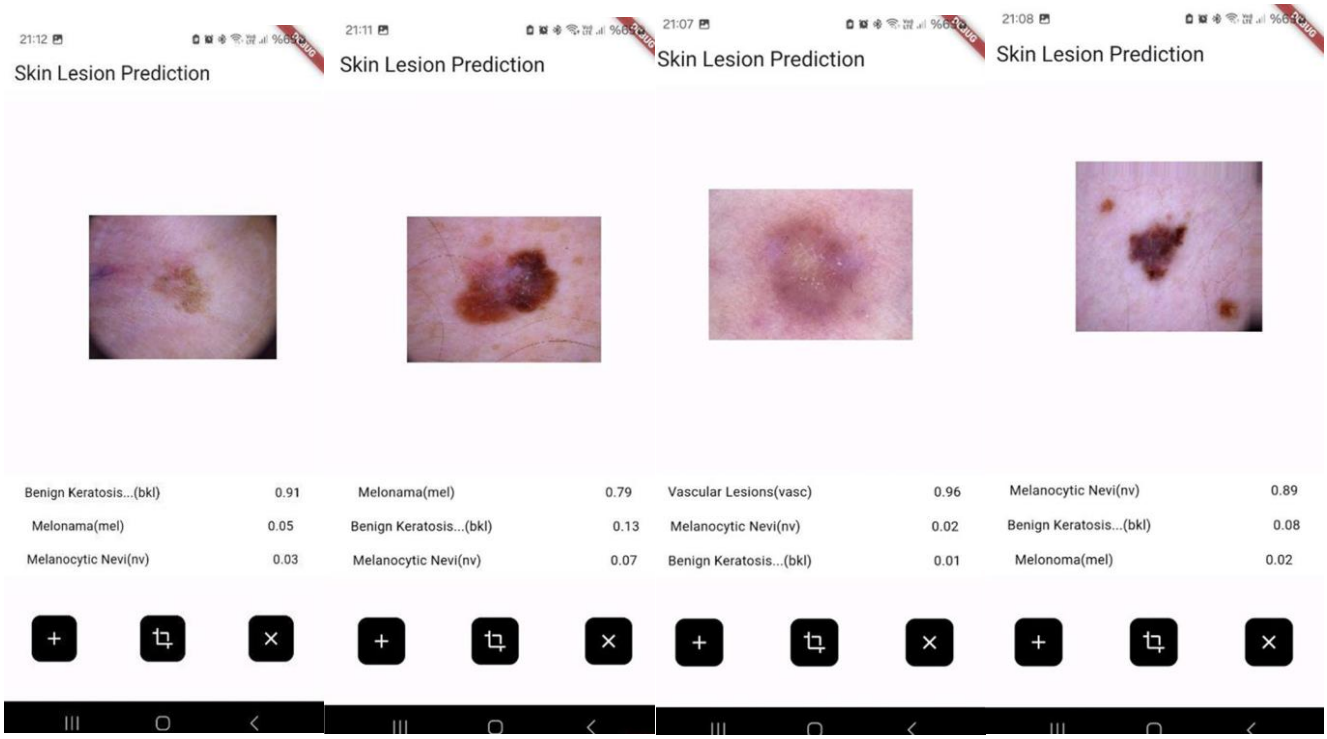

Figure 6. User Interface Of The Mobile Application Showing The Prediction Results For Different Skin Disease Images. The Integrated Model Runs Directly On The Device Using TensorFlow Lite For The Flutter Application

Implementation Details of the Cloud Deployment

In this part of the study, the developed model is integrated into a cloud environment. The cloud infrastructure used in the study is based on Google Cloud Platform, which contains GPU-enabled virtual machines equipped with NVIDIA A100 Tensor Core GPUs. Deployment is managed through a Docker container which simplifies packaging, deployment, and execution of the model in a consistent and efficient manner.

To integrate the model into the cloud, it was saved in the SavedModel format, which is compatible with TensorFlow Serving. TensorFlow Serving is an open-source tool designed for deploying TensorFlow models in production environments. It supports RESTful APIs to allow mobile apps to interact with the model. With a RESTAPI, users can upload photos through the app interface. They can either select photos from the gallery or take new ones using the camera. These photos are then converted into JSON data and sent to the model's REST API, which processes the query and returns predictions as an API response. The SavedModel format stores the model's architecture, weights, variables, and training parameters. The model is then placed in a directory accessible to TensorFlow Serving to start the deployment process. The deployment uses Docker, a containerization platform that simplifies application development and deployment. TensorFlow Serving is pulled from the Docker cloud environment using the command docker pull tensorflow/serving, making it ready for use. TensorFlow Serving is then initialized with the code shown in Figure 7. To confirm the model is running, a request is sent to port 8501. If the model responds with the status 'AVAILABLE' it is ready for deployment. The model accepts input in JSON format and processes lesion photos as image matrices. The pre-trained model classifies and predicts skin lesions using these image matrices.

```
docker run -t --rm -p 8501:8501
-v "path/to/saved_model:/models/saved_model"
-e MODEL_NAME=saved_model tensorflow/serving
```

Figure 7. TensorFlow Serving Initialization Command Used For Deploying The Model In A Cloud Environment

Comparison of On-device and Cloud Integration

The trained model has been integrated into both the device and the cloud environments. The feedback speeds were compared using the same test data. Prediction results were obtained from 100 test images. The average prediction times are as shown in Table 8.

Table 8. Comparison of Prediction Feedback Speeds

Method	Prediction Feedback Time
Device-Integrated Model	108.3 ms
Cloud-Integrated Model	1281.2 ms

The results show that the prediction response speed of the device-integrated model is approximately ten times faster than that of the cloud-integrated model. This indicates that for mHealth applications requiring frequent prediction queries, cloud services may be less efficient. Factors such as network speed and server traffic load could further slow cloud-based predictions, potentially leading to data loss and reduced application efficiency. Although the fast feedback of the device-integrated model is advantageous, certain steps are needed before integration, such as reducing the model size to fit on the device. In this study, a relatively compact architecture like MobileNet was used. However, some applications may require larger and more complex models. As model depth increases, resulting in more parameters and larger sizes, integration into the device could burden its hardware and processing capacity.

CONCLUSIONS

This study presents the development of a deep learning-based model for mHealth applications, focusing on its integration into both mobile devices and the cloud environment, and comparing these deployment methods. The focus was on predicting skin diseases, given the model's practical application and usability. The MobileNet architecture was selected for model development due to its compatibility with devices that have limited computational power and storage capacity. The model was trained on a large, publicly available dataset. To improve performance during training, transfer learning, data augmentation, and focal loss techniques were applied.

We integrated the trained model into both a mobile device and a cloud environment. Our comparison showed that on-device integration for mHealth applications provides offline functionality, faster prediction feedback, and lower costs. However, it has challenges such as limited processing power and storage capacity, which make it difficult to run and maintain complex machine learning models. Additionally, issues such as potential data loss due to device failure and the difficulty of updating the model can reduce the application's usability. The cloud integration mitigates hardware limitations and allows for easier model updates. However, it requires a constant internet connection, potentially cause delays in prediction feedback. Additionally, sending sensitive patient data to the cloud increases the risk of data leakage. Regular payments for storing data in the cloud also add extra costs, especially for widely used mHealth applications.

In conclusion, developing ML-based mHealth solutions requires careful consideration of the benefits and drawbacks of on-device and cloud deployment. Balancing these factors is crucial for designing an effective mHealth application.

ACKNOWLEDGEMENTS

This research is partially supported by the TUBITAK-BIDEB 2232 International Fellowship program under the grant number 121C085.

This research is largely based on the work conducted by Özge Çiçek in her master's thesis titled *"Mobil Sağlık Uygulamalarında Makine Öğrenmesi Temelli Model Geliştirme ve Modelin Cihaz-Bulut Dağıtımı"* (Cicek, 2024)

REFERENCES

Candemir, S., Nguyen, X. V., Folio, L. R., & Prevedello, L. M. (2021). Training strategies for radiology deep learning models in data-limited scenarios. *Radiology: Artificial Intelligence*, *3*(6), e210014.

Çiçek, Ö. (2024). Mobil sağlık uygulamalarında makine öğrenmesi temelli model geliştirme ve modelin cihaz-bulut Dağıtımı, *Master's Thesis,* Eskisehir Technical University, Türkiye.

Dai, X., Spasić, I., Meyer, B., Chapman, S., & Andres, F. (2019). Machine learning on mobile: An on-device inference app for skin cancer detection. *In IEEE 4th Int. Conf. on Fog and Mobile Edge Computing*, pp. 301 305.

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

Goceri, E. (2021). Diagnosis of skin diseases in the era of deep learning and mobile technology., 134, 104458. *Computers in Biology and Medicine*,134, 104458.

Han, S., Mao, H., Dally, W.J. (2015). Deep compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding . *arXiv preprint* arXiv:1510.00149.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint* arXiv:1704.04861.

Jung EY, Kim J, Chung KY, Park DK. (2014). Mobile healthcare application with EMR interoperability for diabetes patients. *Cluster Comput* 17(3):871–880.

Khan, Z., Alotaibi, S. (2020). Applications of artificial intelligence and big data analytics in m-health: A healthcare system perspective. *Journal Of Healthcare Engineering*, 8894694. https://doi.org/10.1155/2020/8894694.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84-90.

Lin, T. Y., Goyal, P., Girshick, R., He, K., Dollar, P. (2017). Focal Loss for Dense Object Detection. *In Proceedings of the IEEE International Conference on Computer Vision* (pp. 2980-2988)

Molina-Recio, G., García-Hernández, L., Castilla-Melero, A., Palomo-Romero, J. M., Molina-Luque, R., Sánchez-Muñoz, A. A., & Salas-Morera, L. (2015). Impact of health apps in health and computer science publications. A systematic review from 2010 to 2014. *In Bioinformatics and Biomedical Engineering*, Granada, Spain, April 15-17, 2015. Proceedings, Part II 3 (pp. 24-34).

Sama, P. R., Eapen, Z. J., Weinfurt, K. P., Shah, B. R., & Schulman, K. A. (2014). An evaluation of mobile health application tools. *JMIR mHealth and uHealth*, 2(2), e3088.

Silva, B. M., Rodrigues, J. J., de la Torre Díez, I., López-Coronado, M., & Saleem, K. (2015). Mobile-health: A review of current state in 2015. *Journal of biomedical informatics*, 56, 265-272.

Susanto, A., Winarto, H., Fahira, A., Abdurrohman, H., Muharram, A., Widitha, U., Efirianti, G., George, Y. Tjoa, K. (2022). Building an artificial intelligence-powered medical image recognition smartphone application: What medical practitioners need to know. *Informatics In Medicine Unlocked*, 32:101017.

Tschandl, P., Rosendahl, C. & Kittler, H. (2018). The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Sci. Data*, 5, 180161 doi:10.1038/sdata.2018.161.

Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big data*, 3, 1-40.