algorithms

MDPI

Article

LungVision: X-ray Imagery Classification for On-Edge Diagnosis Applications

Raghad Aldamani †, Diaa Addeen Abuhani † and Tamer Shanableh *

Computer Science and Engineering Department, American University of Sharjah, Sharjah P.O. Box 26666, United Arab Emirates; g00085944@aus.edu (R.A.); b00086137@aus.edu (D.A.A.)

* Correspondence: tshanableh@aus.edu

† These authors contributed equally to this work.

Abstract: This study presents a comprehensive analysis of utilizing TensorFlow Lite on mobile phones for the on-edge medical diagnosis of lung diseases. This paper focuses on the technical deployment of various deep learning architectures to classify nine respiratory system diseases using X-ray imagery. We propose a simple deep learning architecture that experiments with six different convolutional neural networks. Various quantization techniques are employed to convert the classification models into TensorFlow Lite, including post-classification quantization with floating point 16 bit representation, integer quantization with representative data, and quantization-aware training. This results in a total of 18 models suitable for on-edge deployment for the classification of lung diseases. We then examine the generated models in terms of model size reduction, accuracy, and inference time. Our findings indicate that the quantization-aware training approach demonstrates superior optimization results, achieving an average model size reduction of 75.59%. Among many CNNs, MobileNetV2 exhibited the highest performance-to-size ratio, with an average accuracy loss of 4.1% across all models using the quantization-aware training approach. In terms of inference time, TensorFlow Lite with integer quantization emerged as the most efficient technique, with an average improvement of 1.4 s over other conversion approaches. Our best model, which used EfficientNetB2, achieved an F1-Score of approximately 98.58%, surpassing state-of-the-art performance on the X-ray lung diseases dataset in terms of accuracy, specificity, and sensitivity. The model experienced an F1 loss of around 1% using quantization-aware optimization. The study culminated in the development of a consumer-ready app, with TensorFlow Lite models tailored to mobile devices.

Keywords: medical diagnosis; CNN image classification; model quantization; on-edge image classification

Citation: Aldamani, R.; Abuhani, D.A.; Shanableh, T. LungVision: X-ray Imagery Classification for On-Edge Diagnosis Applications. *Algorithms* **2024**, *17*, 280. https://doi.org/10.3390/a17070280

Academic Editor: Maryam Ravan

Received: 3 June 2024
Revised: 23 June 2024
Accepted: 24 June 2024
Published: 27 June 2024

1. Introduction

Despite the recent technological advancements and growing popularity of mobile devices, they have not yet become widely integrated as essential components in diagnostic imaging [1]. The hesitance to adopt mobile devices in the medical field is usually explained by concerns regarding patients privacy [2], the lack of regulations that ensure the images are visible and manipulated in a clinically acceptable manner [3], and most importantly, whether or not mobile devices can add a value to the diagnostic process in the first place [4]. Recently, multiple studies shed light on the aforementioned privacy [5,6], scalability [7], and usability [8] concerns. The findings of these works showed that it is feasible to utilize mobile devices for medical diagnosis in the near future, provided that extra precaution measures are taken into consideration. For this reason, we believe it is safe to move forward one step ahead and look into the possibility of utilizing neural networks for medical diagnosis on mobile phones.

Generally speaking, deploying neural networks on mobile devices requires reducing the model into a size that is compatible with the operating systems and application size

restrictions of these devices. For instance, Android's Google Play enforces an application size limitation of 200 MB at max [9], while the App Store of IOS has a slightly higher bound of 500 MB [10]. In practice, the application size can influence the user satisfaction, device performance, and storage capacity. As a result, developers aim towards reducing the size of the application as much as possible without compromising on the quality of the application. These consequences explain the fact that the average android application has a size of around 12 MB, while IOS applications have an average size of 35 MB [11], which are far below the maximum application size restrictions enforced by the respective operating systems.

In general, deep learning models suffer from a lack of explainability, leading to an uncertainty in the reliability of real-world systems. This problem is addressed in [12] by adding a reliability estimate to the predictions of the generated models and modelling aleatoric and epistemic uncertainties in the deep learning models. Neural network models tend to have a high number of weights (parameters) that rarely go below a million. Consequently, a trained model is stored along with all its associated weights, resulting in a large model size. This makes it infeasible to have a functional neural network model within your application build. For this reason, many developers opt for having the neural network on an external resource, such as a database or a cloud [13]. However, external resources decrease the security of the diagnosis process, given the possibility of data leakage, privacy breaches, or unintentional third-party access to patients' records. A more reasonable solution is model quantization [14], where the way in which weights are stored is altered to reduce the size of the model, with minimal impact on the overall performance. Quantization can be carried out through storing the weights in smaller formats than the usual float-32, such as float-16, int-8, or even int-4 in some cases. These formats have proven useful and sufficient to maintain a good performance while reducing the size of the model to as much as fourth the original size.

In this paper, we provide a thorough comparison study concerning neural networks' deployment on mobile devices for medical diagnosis. We investigate the use of different TensorFlow Lite quantization techniques, classification architectures, and their impact on user's experience. The X-ray lung diseases dataset [15] is considered as a case study.

2. Literature Review

The task of classifying lung diseases using X-ray images has been a point of discussion for the past decade. Many works have explored the use of convolutional neural networks for the task and achieved reasonably decent results. Cococi et al. [16] used a Slim convolutional neural network to classify pneumonia disease from other diseases using a mobile phone, and achieved an accuracy of 94.4%. The authors show that the accuracy score drops significantly by adding a third class for classification, achieving a score of only 84.66%. Upon quantization using TensorFlow Lite, the authors noticed a drop of 2.43% in the binary classification task and a 0.83% drop in the three classes classification task, without clearly highlighting the form of quantization followed. Muneeb et al. [17] used a CNN model compressed using TensorFlow Lite int-8 quantization to classify six classes from the Chest X-ray14 dataset [18]. The authors reported an accuracy of around 65.82% and highlighted a 6% drop after quantization. Hendrick et al. [19] used GoogleNet to distinguish Tuberculosis disease from others. The proposed methodology achieved an accuracy of 98.39% for the binary problem described. The authors later deployed the model on an IOS device to assess its functionality, but without highlighting the methodology followed. Other works included the use of a pre-trained NASNetMobile to classify pneumonia, which achieved an accuracy score of 97.39% on the testing data [20]. Generally speaking, it can be noticed that only a subset of the lung diseases dataset is usually taken into consideration, with the maximum number of classes used being that which is carried out in [17] using 6 classes, even though the biggest version of the lung diseases dataset contains 9 different classes. Table 1 summarizes the results obtained in previous work in regard to X-ray imagery classification for mobile devices.

Table 1. Summary results of existing work including the number of classes used in each.

Work	Architecture	#Classes Used	F1-Score	Mobile Deployment
Hendrick et al. [19]	GoogleNet	2	98.39	Deployed the model on an iOS device but did not highlight the approach followed
Naskinova et al. [20]	NASNet	2	97.39	Did not deploy the model on a mobile device to ensure feasibility
Cococi et al. [16]	Slim-Net	2	94.40	Used TensorFlow Lite post quantization and reported a 2.43% drop in accuracy
Cococi et al. [16]	Slim-Net	3	84.66	Used TensorFlow Lite post quantization and reported a 0.83% drop in accuracy
Muneeb et al. [17]	Deep CNN	6	65.82	Used TensorFlow Lite integer quantization and reported a 6.0% drop in accuracy

3. Materials and Methods

3.1. Dataset and Pre-Processing

The dataset used in this study is the publicly available X-ray lung diseases dataset. The dataset consists of 9 classes, each representing a respiratory system disease which can be seen in Figure 1.

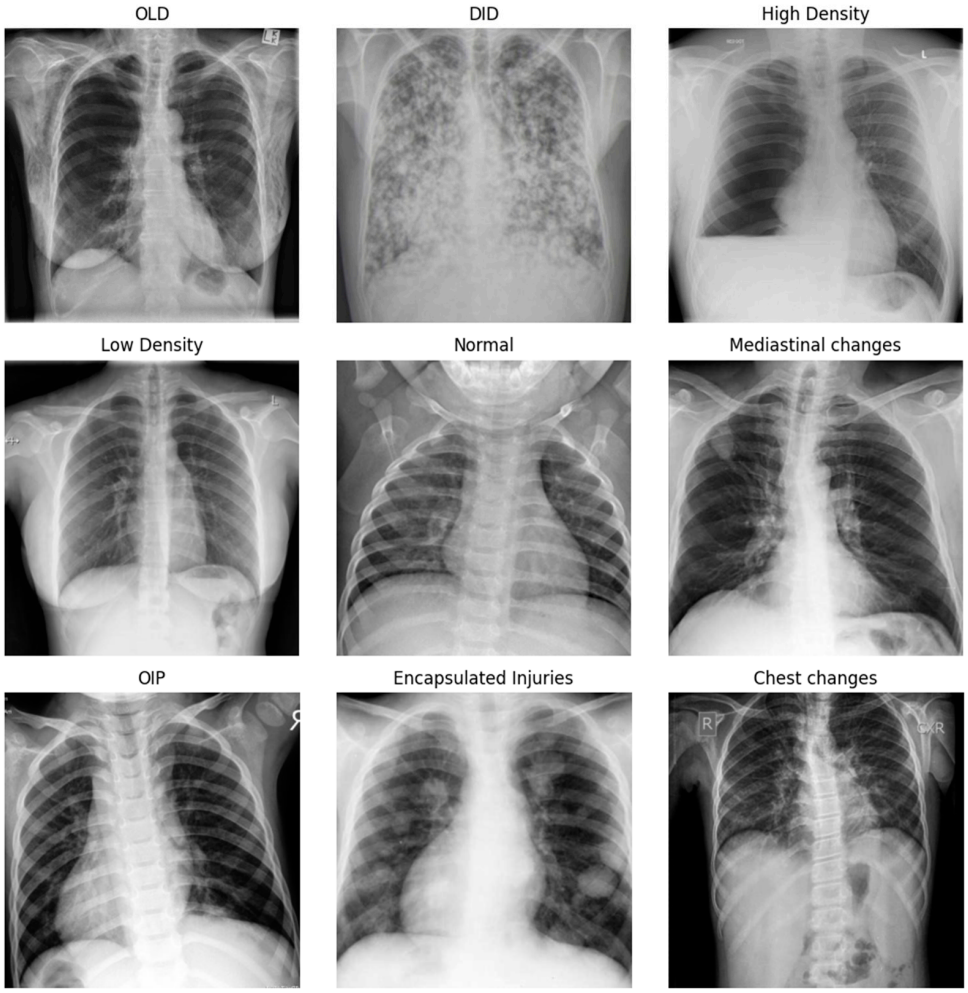


Figure 1. X-ray lung diseases dataset based on the category of the disease.

These diseases include obstructive pulmonary diseases (OLD), degenerative infectious diseases (DID), higher and lower density diseases, encapsulated lesions, mediastinal

changes, chest changes, obstructive inflammatory processes (OIP), in addition to the normal case distributed over 6753 images. The dataset is lightly imbalanced (See Figure 2) and contains images of inconsistent dimensions. For training purposes, all images were resized to a 384 × 384 resolution.

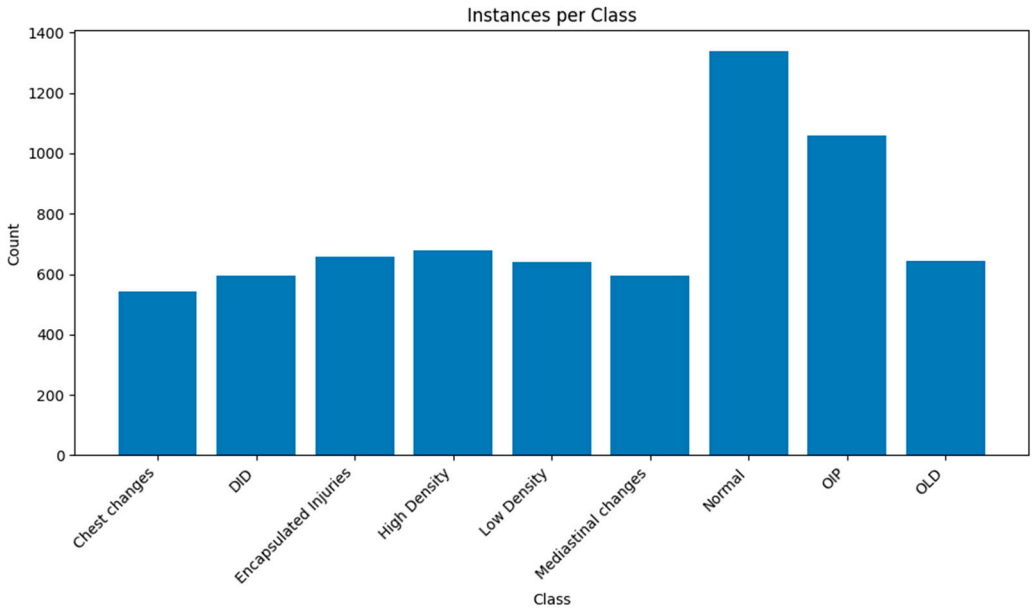


Figure 2. X-ray lung diseases dataset: number of instances per class.

3.2. Neural Network Architectures

Transfer learning [21] is a common idea used to enhance the training performance of deep learning models. Typically, the weights of a pre-trained architecture are used as a starting point to a new training process. In this study, we examine 6 different neural networks suitable for image classification, pre-trained on the ImageNet dataset [22]. Various architectures are examined with the aim of deploying the best performing one on a resource-constrained device; hence, we selected pre-trained neural networks which were approximately 100 MB in size. Figure 3 demonstrates an overview of the proposed methodology.

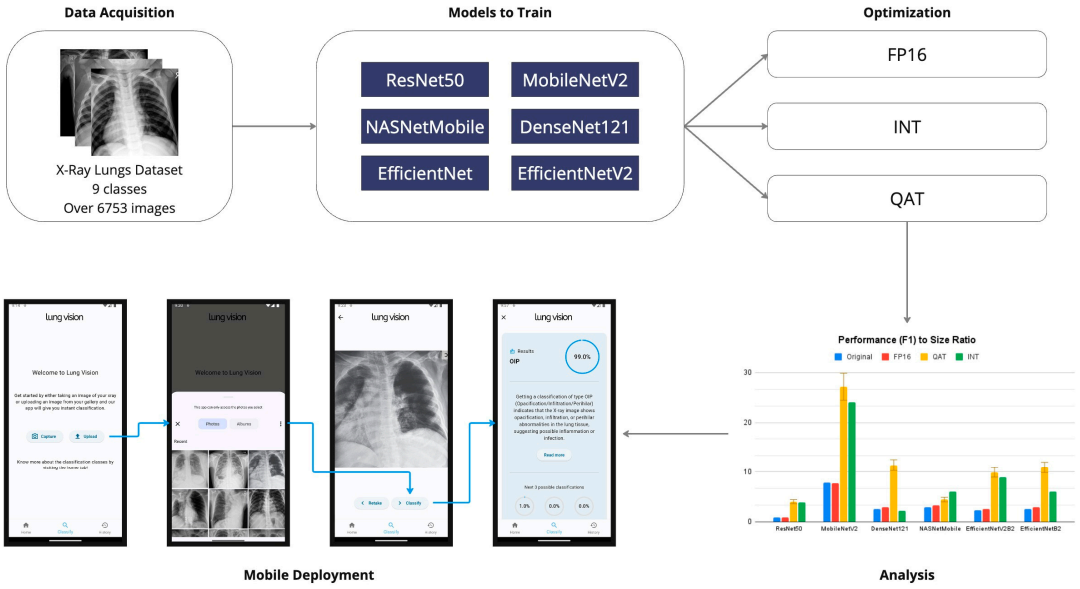


Figure 3. Overview of the proposed methodology.

Once the classification models are generated, we convert them to TensorFlow Lite to make them suitable for mobile devices. As explained in the next section, we use 3 different quantization solutions for that purpose. Hence, the total number of models generated and examined in this work is 18 (i.e., 6 CNN architectures times 3 quantization solutions). The algorithm followed to generate these models is illustrated in Figure 4.

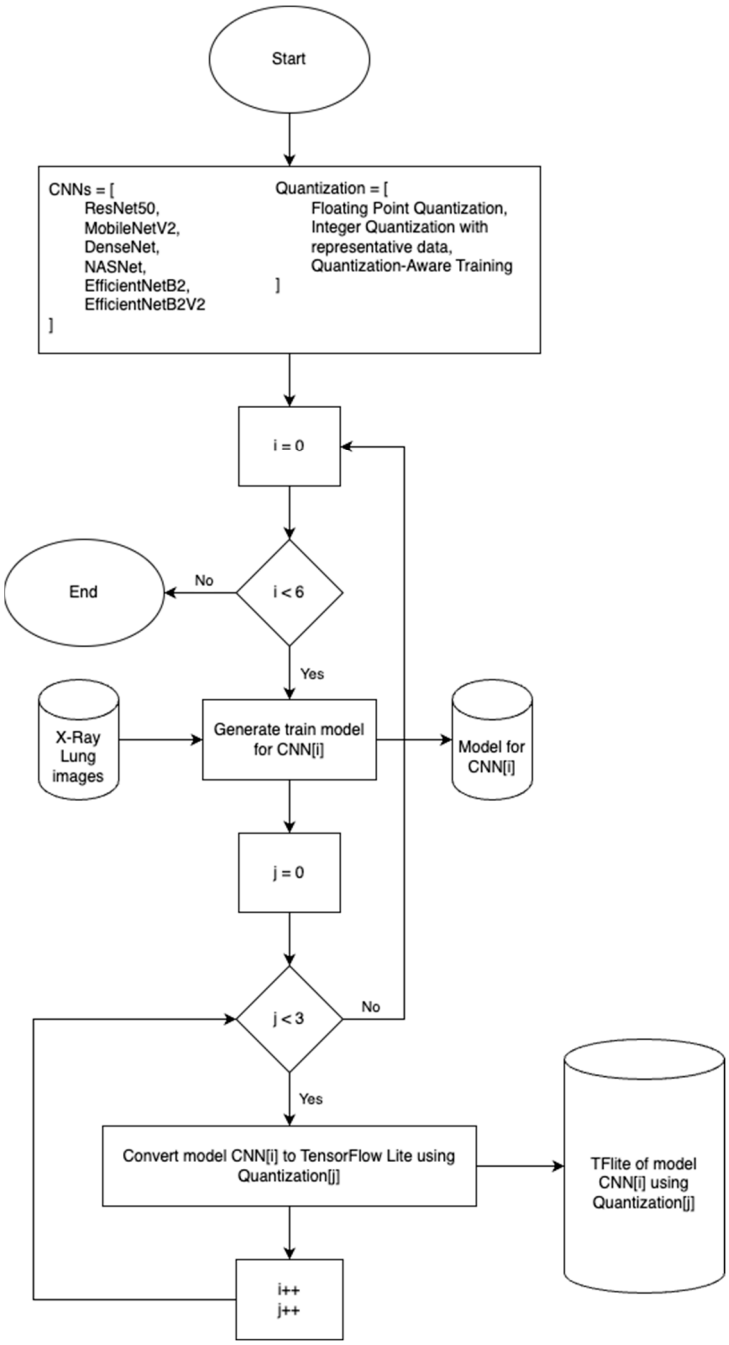


Figure 4. Illustration of the proposed model generation algorithm.

In the rest of this section, we provide a brief description of the CNN architectures used in this work for lung disease classification, and a justification of their use in this work.

The first CNN architecture is the ResNet50 neural network [23], which is a deep convolutional neural network architecture which comprises 50 layers and utilizes residual connections to alleviate the vanishing gradient problem. These residual connections also

enable the training of deeper networks with improved accuracy and faster convergence compared to traditional architectures.

We also experiment with the use of MobileNetV2 [24] for image classification. This network is a lightweight convolutional neural network designed for mobile and embedded vision applications. It utilizes depth wise separable convolutions and inverted residuals to achieve high efficiency and accuracy. MobileNetV2 is well-suited for tasks requiring real-time inference on resource-constrained devices, which is similar to the task at hand in terms of mobile phone deployment constraints.

The third CNN architecture we experiment with is the DenseNet [25], which is a neural network architecture in which each layer is connected to every other layer in a feed-forward fashion. It facilitates feature reuse, encourages feature propagation, and alleviates the vanishing gradient problem. DenseNet achieves state-of-the-art performance on various computer vision tasks with fewer parameters compared to traditional architectures, which may be handy in terms of use for mobile devices.

The forth CNN architecture we experiment with is the NASNet [26], which employs reinforcement learning or evolutionary algorithms to explore vast architectural spaces, resulting in highly efficient and accurate models tailored to specific tasks. NASNet demonstrates superior performance across various domains, showcasing the potential of an automated architecture search in deep learning. In this study, NASNetMobile is used, given the problem we are trying to address, and taking into consideration the small number of parameters the mobile version has when compared to larger versions of NASNet models.

The fifth the CNN architecture is the EfficientNetB2 [27], which is part of the EfficientNet family, featuring a compound scaling method that balances network depth, width, and resolution for improved efficiency and accuracy. It strikes a balance between computational cost and performance, making it suitable for resource-constrained environments while achieving state-of-the-art results in computer vision tasks. EfficientNetB2 leverages a lightweight backbone with efficient building blocks to optimize both inference and training speed without compromising model quality.

The last CNN architecture we experiment with is the EfficientNetB2V2 [28], which is an enhanced version of EfficientNetB2. It further optimizes the architecture of EfficientNet for improved performance and efficiency. It incorporates advancements in network design, regularization techniques, and training methodologies to achieve better accuracy with reduced computational cost. EfficientNetB2V2 retains the balance between model complexity and computational efficiency, making it well-suited for a wide range of computer vision applications on diverse hardware platforms.

3.3. Model Quantization

Once the lung disease models are generated, the next step is to convert them to a representation suitable for mobile devices, as illustrated in Figure 4 above. This can be achieved using model quantization techniques [28], which are used to reduce the size and computational complexity of neural networks to make it suitable for deployment on resource-constrained devices. Model quantization usually involves manipulating the format and number of bits used for representing model weights and biases, or model weights for short. Quantization is important when a fact inference time is required on mobile devices or edge devices in general. Typically, quantization results in a loss of classification accuracy; however, it is subject to the quantization technique followed and the amount of size reduction required. This section provides a brief discussion of the different quantization techniques used in this study, along with the expected impact on the model performance, size, and inference time. TensorFlow Lite framework is used to implement the quantization techniques discussed in this study.

3.3.1. Floating Point Quantization

Floating Point 16 (FP16) quantization involves storing the numerical values using 16-bit floating point format, as opposed to the default 32-bit format. This simple procedure

reduces the memory storage usage and accelerates the computation while having a minimal impact on the overall model's performance, and enables deployment on edge devices such as a mobile phone.

3.3.2. Integer Quantization with Representative Data

Integer quantization using representative data [29] is a method in which a subset of real data samples, known as representative data, is used to simulate model inference during quantization. This approach captures the distribution of activations and weights to ensure that the quantized values accurately represent the model's behavior on unseen data. Integer quantization with representative data helps preserve model accuracy while reducing memory footprint and inference latency, making it suitable for deployment in resource-constrained environments.

3.3.3. Quantization-Aware Training

Quantization-aware training (QAT) [30] is a pre-training technique used to train neural networks while considering the effects of quantization on model performance. During training, QAT simulates the quantization process by applying quantization functions to both activations and weights, allowing the model to adapt to the reduced precision environment. QAT enables the neural network to learn representations that are more robust to quantization, resulting in improved accuracy when deployed on low-bit platforms such as edge devices or specialized hardware.

3.4. Mobile App Deployment

For mobile phone deployment, we developed a fully functional application using the Flutter framework on Android Studio. The main purpose of implementing a complete application is to make sure that our app would be within the constraints of Android's Google Play in terms of size and usability. The application designed takes X-ray images through two ways, by either uploading the X-ray image to the application from the gallery or by taking a picture of the X-ray image using the mobile phone's camera. The integration of the models into the application is straightforward, as Flutter's framework provides multiple packages that can handle various aspects of this integration seamlessly. The deployment typically involves converting and optimizing the trained model into the TensorFlow Lite format, using techniques like Quantization-Aware Training (QAT) to reduce size and enhance performance. However, several challenges are faced during this process. Optimizing model size and inference time is critical due to mobile device constraints and app store size limits. Quantization techniques are essential to address these issues and ensure real-time inference performance. Ensuring the final application size is crucial, as the inclusion of development packages alongside the model can significantly increase the app's total size. The details about the mobile app developed for this research project are available in Appendix A.

3.5. Evaluation Metrics

In general, several metrics can be utilized to evaluate the performance of image classifying. The metrics used in this paper are summarized below.

The first metric is accuracy, which determines an algorithm's ability to make accurate predictions, measured by the ratio of true positive (*TP*) and true negative (*TN*) predictions combined, and divided by the total number of predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

The second metric is specificity, which measures the proportion of true negatives that are correctly identified by the model. It is calculated by dividing the number of true positive predictions (*TP*) by the sum of true positives and false positives (*FP*).

$$Specificity = \frac{TP}{TP + FP} \tag{2}$$

The third metric is sensitivity, which determines an algorithm's capability in recognizing samples, determined by dividing the true positives (TP) by the total of true positives (TP) and false negatives (FN).

$$Sensitivity = \frac{TP}{TP + FN} \tag{3}$$

The fourth metric is the F1-Score, which considers both the sensitivity and specificity described earlier in this paper. It provides a more balanced evaluation to the model performance.

$$F1 = \frac{2 \times Specificity \times Sensitivity}{Specificity + Sensitivity} \tag{4}$$

We also examine the inference time, which refers to the duration it takes for a machine learning model to process input data and generate predictions or outputs. It is a critical metric for assessing the real-time performance and efficiency of models when the speed of prediction may impact the users experience.

4. Experimental Results and Discussion

In this section, we examine the effect of different quantization techniques on the models at hand in terms of the generated model size, model performance and inference time. Following the work reported in the literature, the dataset was split 80–20 for training and testing. All models used were pre-trained on ImageNet and were followed by a simple classification head that consists of a dense layer of 512 nodes, followed by another dense layer for classification utilizing a softmax activation function. The hyperparameters tuned for this experiment include the number of neurons in the final dense layer between [64, 128, 256], the dropout rate between [0.3, 0.5, 0.7], and the learning rate used [0.1, 0.01, 0.001]. The optimal parameter choice was 256, 0.5, and 0.001, respectively, based on hyperparameter tuning. The models were later trained using a categorical cross entropy loss for 15 epochs. An early stopping criterion was used to avoid overfitting with a patience of 3 monitoring accuracy.

Having generated the classification models using the 6 CNN architectures and converting them to TensorFlow Lite using the 3 quantization techniques, it was noticed that the all the quantization techniques of interest reduce the model's size by varying ratios. Additionally, the influence of quantization on the model's performance degradation is based on the combination of CNN architecture and quantization technique used. The impact of quantization on the user's experience, in terms of app inference time, seems to be faster across all models, with some exceptions.

4.1. Model Size

In terms of quantization impact on model sizes, all techniques reduced the sizes of original models by different rations. In general, Quantization-Aware Training (QAT) achieved the highest reduction percentage across all models, with the exception of DenseNet121 with an average reduction ratio of 76.62%. Integer quantization using representative data (INT-Rep) achieved similar percentages, resulting in an average reduction percentage of 75.59% across all models. Floating Point 16 (FP16) optimization seemed to have the least reduction ratio, with an average of 13.84% but with a much higher standard deviation of 6.53%, in comparison to approximately 1.9% and 1.4% for INT-Rep and QAT techniques, respectively. Across all models, MobileNetV2 was the most resilient model to quantize, with an average reduction ratio of 50.14%, whereas NASNet was the easiest to quantize, with an average reduction of 59.14% across all three quantization techniques.

Table 2 provides a more detailed look into the size reduction using different optimization techniques across different CNN models.

Table 2. Impact of quantization techniques on different lung disease model sizes (MB).

Technique/Model	ResNet50	MobileNetV2	DenseNet121	NasNetMobile	EfficientNetB2	EfficientNetV2B2
Original (FP32)	102.4	11.5	33.7	23.9	38.5	42.4
FP16	93.6	11	28.6	18.4	32.1	35.9
INT-REP	24.1	3.2	**7.5**	5.6	9.4	10.6
QAT	**23.8**	**3**	7.6	**5.3**	**8.8**	**9.9**
%Maximum Opt.	76.76%	73.91%	77.74%	77.82%	77.14%	76.65%

4.2. Classification Model Performance

In terms of classification model performance, as expected, the original models without quantization achieved the best performance, with the EfficientNetB2 model achieving a 98.58% F1-Score with 98.60% specificity and 98.59% sensitivity scores. The V2 series of the same model showed a similar performance, with an F1-Score of almost 98%. This is followed by the classification results of ResNet50, which achieved a 96.60% F1-Score.

Comparing our results with existing literature revealed that our proposed solution exceeded the performance of the existing work. It is important to note that the existing work used a subset of the lung disease classes; some used as few as two classes and others used up to six classes. In our work, on the other hand, we have used all nine classes of the dataset. We believe that the main reason behind the low performance of previous works arises from the training methodologies followed. This is expected, as these works used off-the-shelf solutions. Common drawbacks include the lack of the use of transfer learning, severely reducing the image sizes and resulting in a loss of information, and over-training the model, risking overfitting. In this work, we believe that we followed a more appropriate training methodology, by using transfer learning and finetuning the models instead of training from scratch. Additionally, given the detailed features of X-ray images, resizing the images to a reasonable (384×384) resolution ensures that the loss of information is minimal and that it is in line with the inputs expected by the architectures used. Lastly, to avoid overfitting, we utilized an early stopping criteria during the training of our models. The early stopping monitored the model performance on a 10% validation subset of the training dataset, with a patience of five epochs. Additionally, the architecture included a dropout layer before the fully connected dense layer, to further assure that the model does not fit closely to its training data.

Table 3 compares this work and previous approaches tackling the same lung disease classification problem. A comparison is carried out in terms of the number of classes used, the F1-Score before quantization, whether quantization has been utilized or not, and whether the trained model has been deployed and tested on a mobile phone or not.

Table 3. Comparison with previous approaches pre-quantization.

Work	# Dataset Classes	F1-Score	Quantization	Deployment
[19], 2019	2	98.39%	No	Yes
[20], 2023	2	97.39%	No	No
[16], 2020	3	94.40%	Yes	Yes
[17], 2022	6	84.66%	Yes	Yes
Proposed solution	9	98.58%	Yes	Yes

What is interesting about our obtained results is that the classification performance of the models optimized using QAT had a moderate impact on the classification accuracy. In fact, aside from the NASNetMobile model—which was severely degraded—the average loss in performance across all other models optimized with QAT was 4.1% in terms of the F1-Score. The EfficientNetV2B2 combined with QAT achieved the best F1-Score, with a result of 97.51%.

In terms of FP16 quantization, the average loss in classification performance was 5.9% across all models. However, QAT models seem to have a higher performance to size ratio

across most models, as illustrated in Figure 5. The performance to size ratio provides insight into the models efficiency. In other words, it shows which model is achieving the highest classification accuracy performance using minimum memory allocation on the device used. The margin of error at the 95% confidence intervals for each model are as follows: ResNet ± 2.750, MobileNet ± 16.523, DenseNet ± 7.023, NasNet ± 2.213, EfficientNetV2 ± 6.397, and EfficientNet ± 6.161. MobileNetV2 model has the best performance to size ratio when compared to other architectures, which perhaps makes it the most optimal choice for deployment on highly resource-constrained devices.

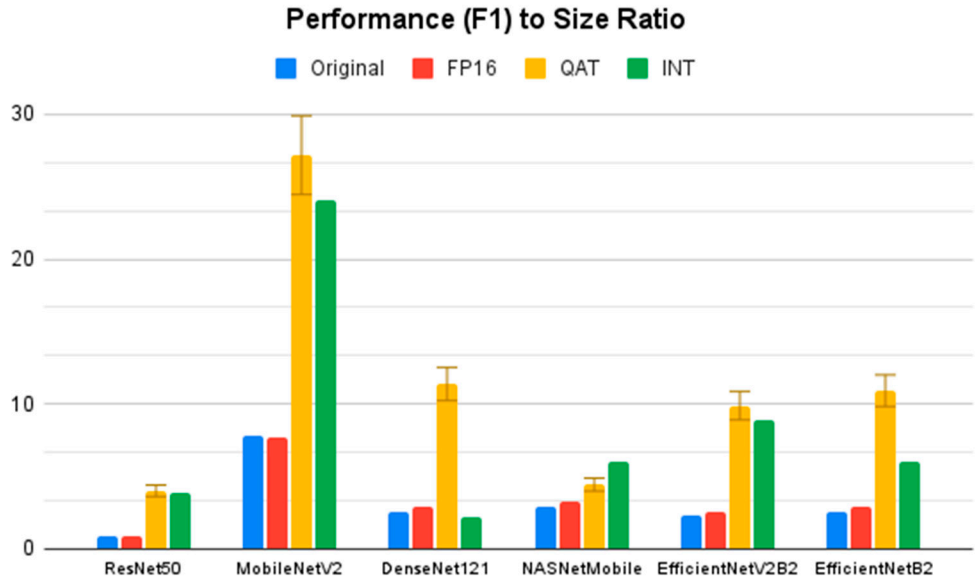


Figure 5. 'Performance to Size Ratio' across all models and using different optimization techniques.

Unlike the previous two approaches, integer quantization using representative data had major impact on the majority of the models in terms of classification performance, resulting in an average loss of around 13.157%. This excludes the DenseNet121 model, which resulted in an outlier readings. It is important to highlight that a model such as EfficientNetV2, which inherently reduces memory consumption [28], maintained a high performance, with a loss of around 4% only.

Table 4 provides a detailed comparison of the results discussed in this section. The table lists the classification results of all generated lung disease classification models using all quantization techniques.

Table 4. Deep learning models' performance summary.

Model	Metric	Quantization Technique			
		FP32	**FP16**	**QAT**	**INT-REP**
ResNet50	Accuracy	96.59	87.50	95.63	92.54
	Specificity	96.81	87.53	95.64	92.76
	Sensitivity	96.59	87.50	95.63	92.54
	F1-Score	96.60	87.49	95.61	92.60
MobileNetV2	Accuracy	90.63	84.67	82.20	77.61
	Specificity	94.27	85.78	83.97	82.96
	Sensitivity	90.63	84.38	82.20	77.61
	F1-Score	90.63	84.76	81.52	77.09
DenseNet121	Accuracy	87.78	59.38	86.61	19.40
	Specificity	88.91	78.47	86.64	29.37
	Sensitivity	88.77	59.38	86.61	19.40
	F1-Score	87.58	60.38	86.46	16.19

Table 4. *Cont.*

Model	Metric	Quantization Technique			
		FP32	**FP16**	**QAT**	**INT-REP**
NasNetMobile	Accuracy	68.67	93.75	24.62	31.34
	Specificity	75.59	95.42	27.90	40.40
	Sensitivity	68.67	93.75	24.63	31.34
	F1-Score	69.16	93.72	23.56	33.54
EfficienetNetV2B2	Accuracy	98.00	**93.75**	**97.52**	**94.03**
	Specificity	98.05	95.42	**97.53**	**95.01**
	Sensitivity	98.00	**93.75**	**97.52**	**94.03**
	F1-Score	97.99	93.72	**97.51**	**94.03**
EfficientNetB2	Accuracy	**98.59**	**93.75**	96.14	56.72
	Specificity	**98.60**	**95.83**	96.45	76.71
	Sensitivity	**98.59**	**93.75**	96.14	56.72
	F1-Score	**98.58**	**93.85**	96.15	56.97

4.3. Inference Time

In terms of model inference time, the only clear trend revealed based on our experiments is that the quantized models have a faster inference time than the original models represented with Float 32 bits (See Table 5). However, the impact of different quantization techniques varied based on the CNN architecture. We note that even though half of the models achieved their best inference time using QAT, the overall best average inference time improvement is achieved using integer quantization with representative data. This technique resulted in an average reduction in inference time of 1.383 s compared to the original model.

Table 5. Impact of quantization techniques on model inference time (in seconds).

Technique/Model	ResNet50	MobileNetV2	DenseNet121	NasNet	EfficientNetB2	EfficientNetV2B2	Avg. Reduction
Original (FP32)	1.230	0.240	1.160	0.520	0.700	0.500	N/A
FP16	1.037	**0.102**	0.607	0.379	0.687	0.505	0.1723
QAT	1.030	0.140	0.850	**0.340**	**0.410**	**0.470**	0.2017
INT-REP	**0.652**	0.193	**0.512**	0.504	0.604	0.519	**1.3830**

Table 5 provides a more detailed view on the inference time achieved by each model.

4.4. Mobile Device Deployment and Testing

To build our app we designed an Android-based mobile application that is user friendly and deployment ready. The application comprises a mobile interface and a backend cloud database, along with an EfficientNetB2V2 classification model quantized using QAT. The database was developed using Cloud Firestore, which serves as an easy yet secure way to store and access data throughout the application. The application is named "LungVision", as it encapsulates the purpose of the app in a simple and clear manner. As mentioned previously, a clear description of the app's functionalities is provided in Appendix A.

In terms of inference time on the mobile app, which generally varies across different quantization methods and models as shown in Figure 6, we randomly selected and classified five images per class from the testing split. The model achieved an F1-Score of 91.31%, with an accuracy of 88.89%, as well as 99.18% specificity and a 100% sensitivity score. While the test data on the phone is small, it is nevertheless reasonably good enough to show that the application is capable of distinguishing different respiratory system diseases. Figure 7 shows the confusion matrix obtained using the mobile phone.

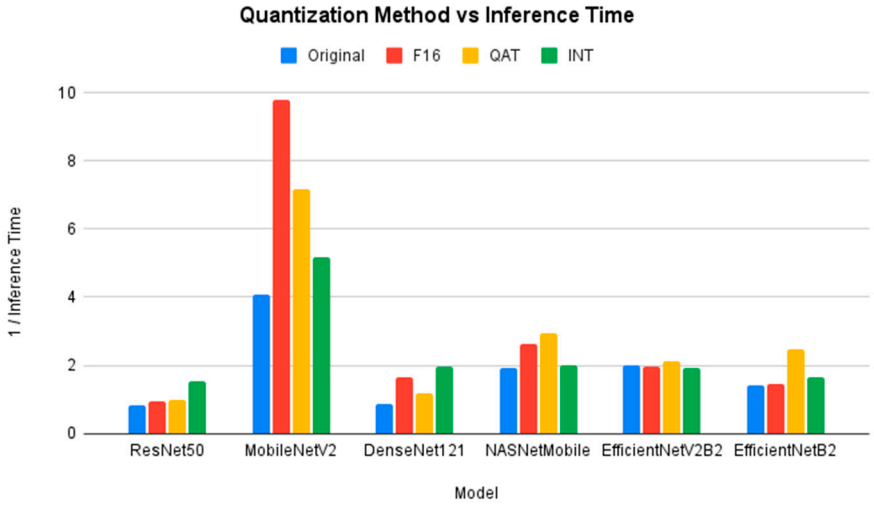


Figure 6. The measured inference time across different models.

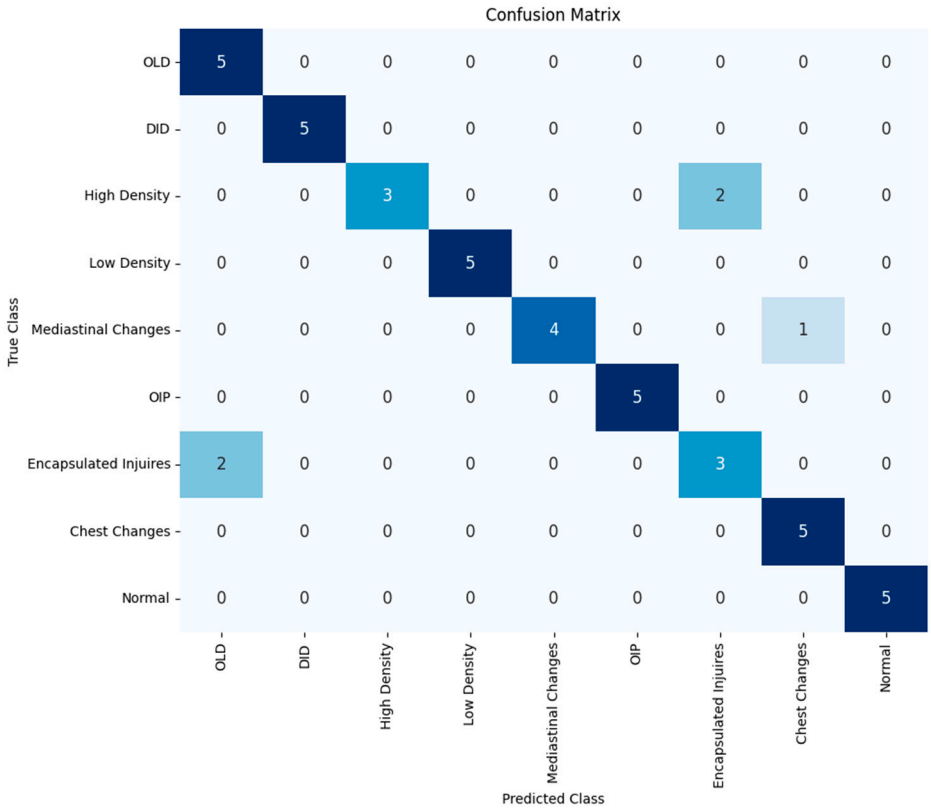


Figure 7. The confusion matrix of the on-app deployed model, with five images randomly selected per class. The names of the classes are listed in the figure.

5. Conclusions

In this work, we conducted a comparative study to examine the feasibility of using mobile phones for on-edge medical diagnosis. We focused on the technical aspect of the deployment by training different neural networks to classify nine respiratory system diseases using X-ray imagery. We achieved an F1-Score of around 98.58% using the EfficientNetB2 model, which beats the state-of-the-art in terms of performance and the classes taken into consideration on the X-ray lung diseases dataset. Later, we analyzed three different optimization techniques, namely, Floating Point 16 (FP16), Quantization-Aware Training (QAT), and integer quantization using representative data (INT). The influence of the three techniques was studied in terms of model size, performance, and inference time. We found

that QAT quantization achieves the highest optimization average, with a 75.59% reduction in size and a maximum optimization of 77.82%. In terms of performance, we highlight that MobileNetV2 achieves the highest performance to size ratio using QAT, with an average loss of as little as 4.1% across all models. Then, we tested the models' inference time, and we highlighted that INT quantization rapidly outperforms other techniques, with an average inference time improvement of 1.3830 s. Finally, we illustrated the functionality of the proposed approach using a consumer-ready application, developed to match the constraints of a mobile phone running an Android operating system and in line with Google Play regulations.

The size of a typical CNN employed in the current approach varies depending on the specific architecture used. In this study, six different CNN architectures were evaluated: a custom model (102.4 MB), MobileNetV2 (11.5 MB), NASNetMobile (23.9 MB), EfficientNetB2 (38.5 MB), EfficientNetB2V2 (42.4 MB), and DenseNet (33.7 MB). While models like MobileNetV2 are relatively small and suitable for deployment without further optimization, larger models such as EfficientNetB2 and custom models require size reduction. Utilizing TensorFlow Lite and quantization methods significantly reduce these model's size while maintaining performance, making them more suitable for deployment on standard smartphones. However, the quantization process, while effective in reducing model size and improving inference time, can lead to a slight decrease in model performance. This trade-off between size, speed, and accuracy requires careful consideration, especially in critical applications such as medical diagnosis. To measure the robustness to low-quality X-ray images, we conducted experiments by taking photos of existing X-ray images using the mobile phone camera. We noticed that the detection accuracy is lower in the case of low-quality/noisy images, which may require relying on image enhancements and noise cancellation before feeding the image to the model. This represents a future area of study that we believe is necessary for practical use and industrial deployment.

Author Contributions: Conceptualization, R.A., D.A.A. and T.S.; methodology, R.A., D.A.A. and T.S.; software, R.A. and D.A.A.; validation, R.A., D.A.A. and T.S.; formal analysis, R.A. and D.A.A.; investigation, R.A. and D.A.A.; resources, R.A. and D.A.A.; data curation, R.A. and D.A.A.; writing—original draft preparation, R.A., D.A.A. and T.S.; writing—review and editing, R.A., D.A.A. and T.S.; supervision, T.S.; project administration, T.S.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

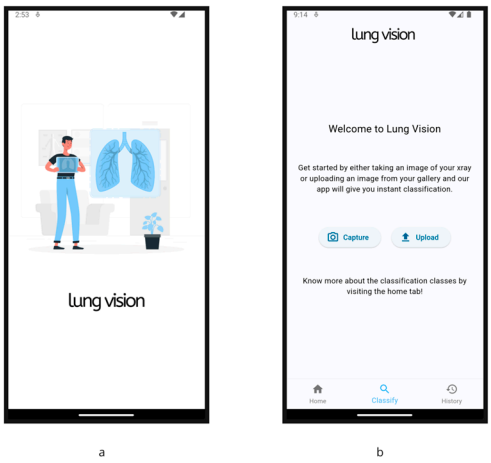
Data Availability Statement: The data used in this research are publicly available and cited where relevant. Other supporting material is available upon request.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

The mobile application, developed for the implementation of X-ray-based lung disease classification, consists of three main activities with a total release size of around 100 MB, which is in line with the constrains highlighted earlier by the Google Play Store. The application starts up with a splash screen displaying the app name. This screen lasts for 3 s before it shows the main screen of the application, as seen in Figure A1. The user can then navigate through the app using the bottom navigation bar, as illustrated in Figure A1.

Additionally, on the home screen of the app, we showcase all the possible classification classes with additional information, such as keywords displayed as tags, disease definition, possible symptoms, recommended actions, and our model's classification confidence, as seen in Figure A2.



a b

Figure A1. (**a**) Splash Screen, (**b**) Main Screen.

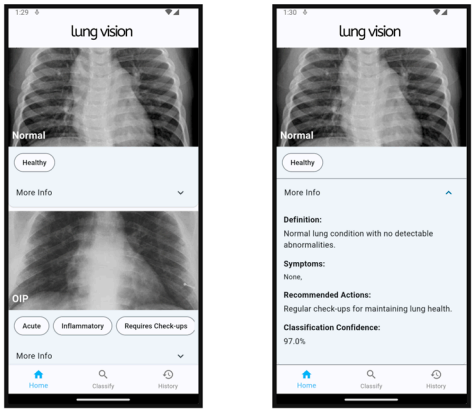


Figure A2. Home Screen.

The main screen of the app is the classification screen, which allows for capturing an image using the camera or selecting an image from the phone's gallery. Subsequently, the app previews the selected image, where it allows for the re-selection of the image or the continuation of the classification process. Once confirmed, the app uses the proposed TensorFlow Lite model to perform the classification task. Figure A3 displays a walk-through of the mentioned classification process. Alongside displaying the classification report, this screen also utilizes Cloud Firestore for permanent storage of the classification results. This feature is used for scalability purposes, which allows for the expansion into a multi-user app in the future should the need arise.

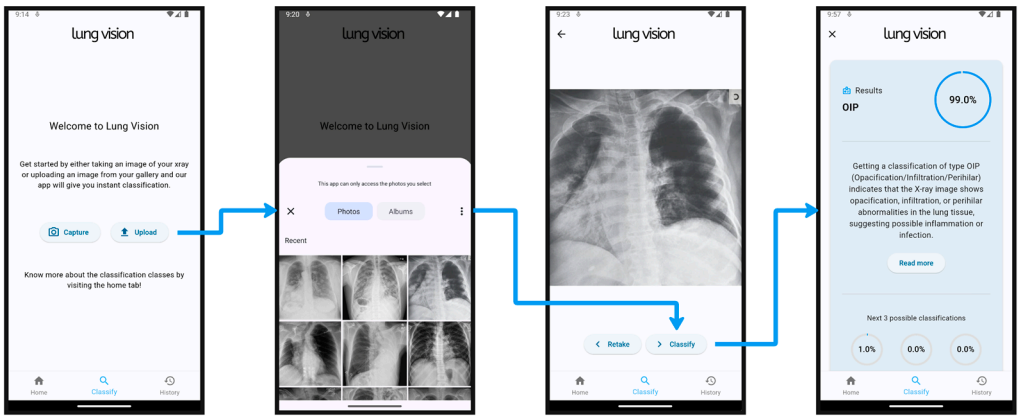


Figure A3. Classification process walk-through.

The app also allows the user to display the history of all previous classification reports. This is accomplished by fetching all records from the Cloud Firestore and displaying them based on the most recent classification result. This activity also allows for the deletion of all records, the deletion of a specific record, and reviewing the full report.

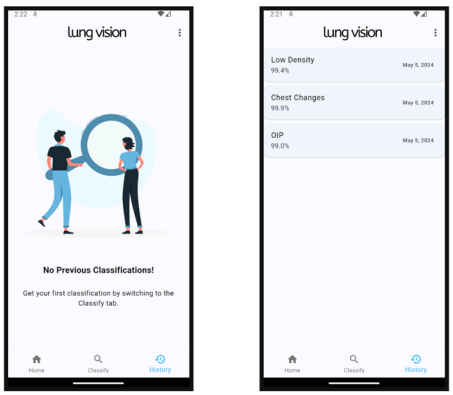


Figure A4. (**a**) History Screen with no reports, (**b**) History Screen with summary results.

Clicking on a record will display the full classification report. Downloading the report is achieved by clicking on the menu found in the top right corner of the screen and clicking the download option. The report will be saved as an image in the phone's gallery. Figure A5 showcases the process of viewing and downloading a report.

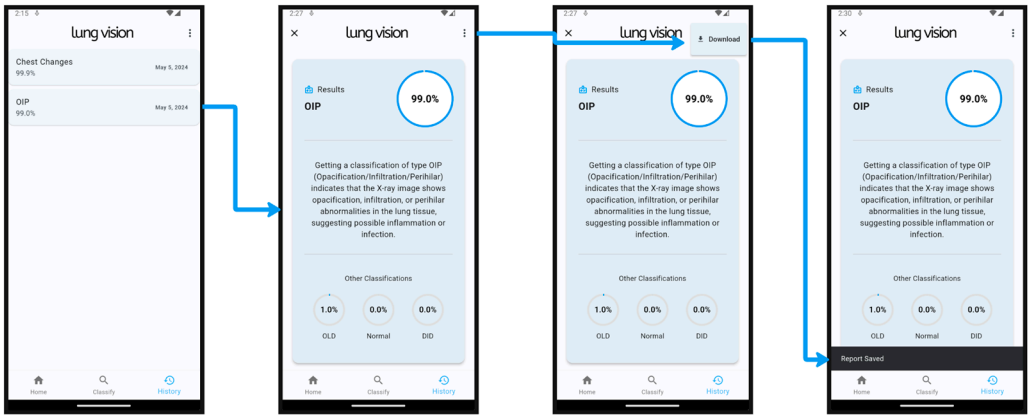


Figure A5. The process of viewing and downloading a report.

To delete a single report, the user can simply swipe the desired record from right to left. This action will prompt a deletion confirmation dialog, where it either confirms the deletion and deletes the record, or cancels the action. Figure A6 illustrates a single record deletion process.

In the case of mass deletion of all reports, the user can click on the menu placed at the top right corner of the app screen. Clicking the menu will display a deletion option that once clicked shows a deletion confirmation dialog. If deletion is confirmed, all records will be deleted, otherwise nothing will happen. Figure A7 displays the mentioned steps.

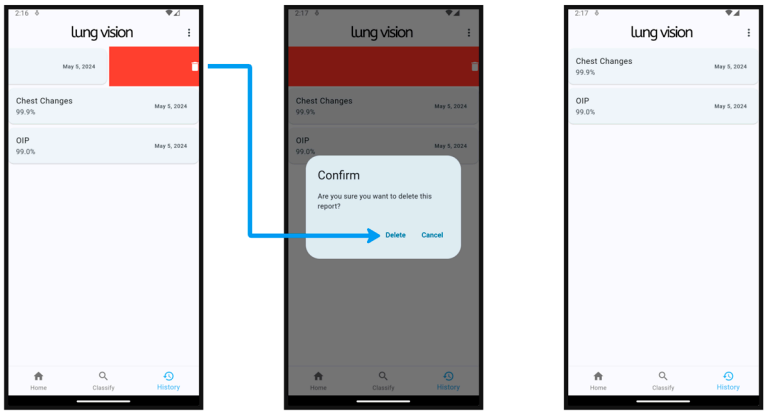


Figure A6. Single report deletion process.

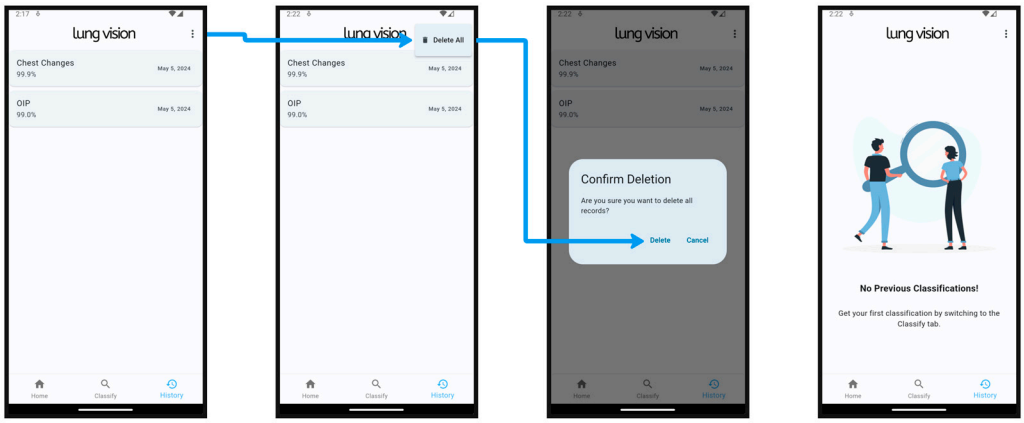


Figure A7. The process of deleting all reports.

References

1. Kufel, J.; Bargieł, K.; Koźlik, M.; Czogalik, Ł.; Dudek, P.; Jaworski, A.; Magiera, M.; Bartnikowska, W.; Cebula, M.; Nawrat, Z.; et al. Usability of mobile solutions intended for diagnostic images—A systematic review. *Healthcare* **2022**, *10*, 2040. [CrossRef] [PubMed]
2. Flaherty, J.L. Digital diagnosis: Privacy and the regulation of mobile phone health applications. *Am. J. Law Med.* **2014**, *40*, 416. [PubMed]
3. Hirschorn, D.S.; Choudhri, A.F.; Shih, G.; Kim, W. Use of mobile devices for medical imaging. *J. Am. Coll. Radiol.* **2014**, *11*, 1277–1285. [CrossRef] [PubMed]
4. Venson, J.E.; Bevilacqua, F.; Berni, J.; Onuki, F.; Maciel, A. Diagnostic concordance between mobile interfaces and conventional workstations for emergency imaging assessment. *Int. J. Med. Inform.* **2018**, *113*, 1–8. [CrossRef] [PubMed]
5. Tovino, S.A. Privacy and security issues with mobile health research applications. *J. Law Med. Ethics* **2020**, *48*, 154–158. [CrossRef] [PubMed]
6. Benjumea, J.; Ropero, J.; Rivera-Romero, O.; Dorronzoro-Zubiete, E.; Carrasco, A. Privacy assessment in mobile health apps: Scoping review. *JMIR mHealth uHealth* **2020**, *8*, e18868. [CrossRef] [PubMed]
7. Sahin, V.H.; Oztel, I.; Yolcu Oztel, G. Human monkeypox classification from skin lesion images with deep pre-trained network using mobile application. *J. Med. Syst.* **2022**, *46*, 79. [CrossRef] [PubMed]
8. Badiauzzaman, I.S.M. Assessment of the usage of mobile applications (APPS) in medical imaging among medical imaging students. *Int. J. Allied Health Sci.* **2018**, *2*, 347–356.
9. Google. About Android App Bundles | Android Developers—Developer.android.com. 2024. Available online: https://developer.android.com/guide/app-bundle#size_restrictions (accessed on 9 April 2024).
10. Apple. Maximum Build File Sizes-Reference-App Store Connect-Help-Apple Developer—developer.apple.com. 2024. Available online: https://developer.apple.com/help/app-store-connect/reference/maximum-build-file-sizes/ (accessed on 9 April 2024).
11. Suri, B.; Taneja, S.; Bhanot, I.; Sharma, H.; Raj, A. Cross-Platform Empirical Analysis of Mobile Application Development Frameworks: Kotlin, React Native and Flutter. In Proceedings of the 4th International Conference on Information Management & Machine Intelligence, Jaipur, India, 23–24 December 2022; pp. 1–6.
12. Seckler, H.; Metzler, R. Bayesian deep learning for error estimation in the analysis of anomalous diffusion. *Nat. Commun.* **2022**, *13*, 6717. [CrossRef]

13. Eshratifar, A.E.; Pedram, M. Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment. In Proceedings of the 2018 on Great Lakes Symposium on VLSI, Chicago, IL, USA, 23–25 May 2018; pp. 111–116.
14. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M.W.; Keutzer, K. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*; Chapman and Hall/CRC: London, UK, 2022; pp. 291–326.
15. Feltrin, F. X-ray Lung Diseases Images (9 Classes). *Kaggle*. February 2023. Available online: https://www.kaggle.com/datasets/fernando2rad/x-ray-lung-diseases-images-9-classes (accessed on 6 April 2024).
16. Cococi, A.; Felea, I.; Armanda, D.; Dogaru, R. Pneumonia detection on chest X-ray images using convolutional neural networks designed for resource constrained environments. In Proceedings of the 2020 International Conference on e-Health and Bioengineering (EHB), Iasi, Romania, 29–30 October 2020; pp. 1–4.
17. Muneeb, M.; Feng, S.F.; Henschel, A. Deep learning pipeline for image classification on mobile phones. *arXiv* **2022**, arXiv:2206.00105.
18. Guan, Q.; Huang, Y. Multi-label chest X-ray image classification via category-wise residual attention learning. *Pattern Recognit. Lett.* **2020**, *130*, 259–266. [CrossRef]
19. Hendrick, H.; Wang, Z.-H.; Chen, H.-I.; Chang, P.-L.; Jong, G.-J. IOS Mobile APP for Tuberculosis Detection Based on Chest X-Ray Image. In Proceedings of the 2019 2nd International Conference on Applied Information Technology and Innovation (ICAITI), Denpasar, Indonesia, 21–22 September 2019; pp. 122–125.
20. Naskinova, I. Transfer learning with NASNet-Mobile for Pneumonia X-ray classification. *Asian-Eur. J. Math.* **2023**, *16*, 2250240. [CrossRef]
21. Weiss, K.; Khoshgoftaar, T.M.; Wang, D. A survey of transfer learning. *J. Big Data* **2016**, *3*, 1–40. [CrossRef]
22. Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255.
23. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
24. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
25. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
26. Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q.V. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 8697–8710.
27. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning (PMLR), Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
28. Tan, M.; Le, Q. Efficientnetv2: Smaller models and faster training. *arXiv* **2021**, arXiv:2104.00298.
29. Elthakeb, A.T.; Pilligundla, P.; Mireshghallah, F.; Cloninger, A.; Esmaeilzadeh, H. Divide and conquer: Leveraging intermediate feature representations for quantized training of neural networks. In Proceedings of the International Conference on Machine Learning (PMLR), Virtual, 13–18 July 2020; pp. 2880–2891.
30. Park, E.; Yoo, S.; Vajda, P. Value-aware quantization for training and inference of neural networks. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 580–595.