Deep learning pipeline for image classification on mobile phones

Muhammad Muneeb, Samuel F. Feng, and Andreas Henschel

Department of Mathematics and Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology,
Abu Dhabi, UAE

Abstract. This article proposes and documents a machine-learning framework and tutorial for classifying images using mobile phones. Compared to computers, the performance of deep learning model performance degrades when deployed on a mobile phone and requires a systematic approach to find a model that performs optimally on both computers and mobile phones. By following the proposed pipeline, which consists of various computational tools, simple procedural recipes, and technical considerations, one can bring the power of deep learning medical image classification to mobile devices, potentially unlocking new domains of applications. The pipeline is demonstrated on four different publicly available datasets: COVID X-rays, COVID CT scans, leaves, and colorectal cancer. We used two application development frameworks: TensorFlow Lite (real-time testing) and Flutter (digital image testing) to test the proposed pipeline. We found that transferring deep learning models to a mobile phone is limited by hardware and classification accuracy drops. To address this issue, we proposed this pipeline to find an optimized model for mobile phones. Finally, we discuss additional applications and computational concerns related to deploying deep-learning models on phones, including real-time analysis and image preprocessing. We believe the associated documentation and code can help physicians and medical experts develop medical image classification applications for distribution.

Keywords: Image classification, machine learning, medical image classification, mobile phone application, cancer

1 Introduction

Disease diagnosis plays a crucial role in healthcare, and for many conditions, medical image data can aid in diagnosing diseases [1,2,3,4,5,6,7,8]. Recent research has made these diagnostic tools more efficient, primarily through deep-learning methods [9]. However, these methods typically require relatively powerful and expensive computational hardware (e.g. modern GPUs), which may not be available in remote or poor areas of the world that lack modern infrastructure. This study proposes a pipeline for classifying images using mobile phones with a remote computer for model training. Even though a central server is required for training, there are free options [10,11] which are sufficient for training the model.

Due to the widespread availability of mobile phones and applications, tasks such as image classification can now be completed at the point-of-care. The deployment of medical image classification models on mobile phones can lead to several technical issues. For example, the model's performance when trained or tested on a computer degrades significantly when the same model is deployed on a mobile phone. Neural network models easily become large enough for phones' memory, and images captured in real time can degrade classification performance. Furthermore, it is unclear how to build a proper workflow connecting training data, often from different sources with varying image sizes and quality, with a model deployed on a smartphone to aid diagnosis. We highlight these issues (and other) and provide feasible solutions to tackle them. The result is an amalgamation of best practices taken from the modern deep learning and data science landscape, including implementations for parameter reduction (Section 2.3) and data augmentation (Section 2.3), resulting in a cost-effective diagnosis pipeline that can be used for medical image classification aiding expert in the diagnosis of the diseases.

The following section explains the differences and similarities between similar projects and the pipeline proposed in this study.

David C. Wyld et al. (Eds): AIAPP, NLPML, DMA, CRIS, SEC, CoSIT, SIGL - 2022
pp. 01-20, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.120901

In this study, [12], researchers proposed a mobile-based deep learning application for image classification. However, they used Unity and MATLAB for implementation, whereas we used the Android Studio TensorFlow application development template and Python for model training. This study [13] investigated the usability of mobile phones for medical image classification. In this project [14,15], researchers designed an application for skin diseases, with attached hardware for accurate detection. In this paper [16], an artificial intelligence diagnostic system on mobile Android terminals for cholelithiasis disease is proposed. This article [17] discusses the various challenges that arise when deploying a machine learning model on mobile phones. This work [18] employs edge computing on a mobile device with an integrated web server to diagnose and forecast metastasis in histopathology pictures.

Among the existing studies [19,14,20,13,21,22,15,12,23,24], researchers have developed a machine-learning pipeline and shed light on the medical and general images on mobile phone applications. However, they did not provide the source code and applications that can be used for result replication.

Many research papers explain image classification on mobile phones, but the following are the reasons for reproducing existing work.

- The existing papers do not shed light on real-time testing and the concerns that may arise when deploying a machine-learning model on mobile phones. For example, some android mobile phones require models having weights in specific data types (Float32 and Unsigned Int8), and if the model is trained on different data types, then the application crashes.
- The existing papers do not include source code and documentation, which are essential for reproducing the results and developing an application for some other dataset.
- The existing papers developed different pipelines for various images like plants, tissues, and X-ray/CT-Scan classification. However, we proposed a general application that works for any classification problem by including and excluding substeps in the pipeline.
- We compared the model's performance on the computer, mobile phone when pictures were loaded from the camera, and mobile phone in real-time.

We believe there must be a generalized application that can capture images in real time and from mobile galleries and classify them into various categories; for that purpose, we used existing templates. Using such a template assists in classification problems involving birds, flowers, plants, objects, tissues, and lung cancer classification. We also analyzed the performance of the same model on real-time and digital images, which showed that the performance of the model was highly degraded in real-time analysis. Finally, we present a method for improving the model's performance on a mobile phone.

Section 2 provides the technical context and describes the entire seven-step pipeline process. Section 3 demonstrates the implementation of the pipeline on covid-19, plants, and cancer classification as a use case and discusses the model performance and technical considerations. Sections 4 and 5 contain a discussion and conclusion, including a link to all the data and codes to reproduce results.

2 A pipeline for image classification on mobile phones

There are already several contexts in which deep learning or other classification models are deployed on mobile phones, including small-scale applications such as emoji selection from text [25] to larger-scale recommendation systems [26,27] and face detection from camera images [28,29]. Implementing image classification presents many challenges, such as ensuring that the image dimensions are the same for training and testing data. Furthermore, the image background/environment in which the model trained should be the same as in the field; otherwise, the model might be trained

to see spurious details in the image background, leading to incorrect results. One must also ensure that the model is implemented efficiently to fit inside mobile memory, often forcing reductions in the model size that can sacrifice accuracy. Finally, if one wants to leverage publicly available medical image data in a mobile context, the details of implementing best practices are unclear. Our answer to all these considerations is a machine-learning pipeline, illustrated in Figure 1 and described in this section.

A pipeline is only as good as its data, and starting, one must obtain data suitable for model training and identify the end-user's mobile devices.

As is standard in supervised learning, the training data must be labeled; for medical images, this is typically performed by a domain expert [30]. Many such medical image datasets are available in the public domain [31,32], and one should verify labels with the help of a local physician.

2.1 Step 1: Choose different images sizes and generate sub-datasets

The first step is preprocessing, which consists of image rescaling, normalization, and image resizing [33], and organizing the data appropriately for later analysis. The user selects a small number of different image sizes for testing. Extra testing in this first step will help avoid excess model parameters, which might crash the mobile application in a later stage.

In practical applications, we recommend choosing 3-5 different sizes ranging from 30 x 30 to 500 x 500 as sufficient, and these choices may depend on the expected aspect ratios of the training data. During the model validation in step 3 2.3, one of these image sizes will be selected automatically depending on the model performance. Figure 2 shows the subdatasets having different image sizes generated from the original dataset.

2.2 Step 2: Data splitting for validation

It is essential to use stratified-`k`-fold validation for each image size to avoid over-and under-fitting during the training [34]. This ensures that the folds are chosen such that the mean response value is equal across all folds, ultimately decreasing model bias.

Our recommendation is to start with `k=5`, which repeatedly trains models using 80 percent of the original data and uses the other 20 percent to evaluate model performance.

As is typical with cross-validation, one then systematically trains the model on 4 of the 5 folds and uses the held out to assess the model performance, and the results are averaged to estimate overall model performance.

2.3 Step 3: Model Architecture and Training

Image analysis using deep learning methods is a rapidly growing field with many algorithms competing over a wide variety of applications (e.g. LSTM and RCNN) [35]. For medical image classification, Convolutional Neural Networks (CNN) are the most popular [36].

Convolution is the process of multiplying pixel values by weights and summing them. The first layer of the CNN frequently detects essential characteristics such as horizontal, vertical, and diagonal edges. The first layer's output is then sent to the second layer, which extracts more complex features like corners and edge combinations. Subsequent layers recognize higher-level characteristics such as objects and faces [37]. Based on the activation map of the last convolution layer, the terminal layer outputs a series of confidence ratings (numbers ranging from 0 to 1) that indicate how probable the image belongs to a specific class.

Models are ultimately fit in Keras using `model.fit`. But before training the model, we must address a few key considerations. First, choose the number of parameters or neurons in each layer and the number of layers. If there are too many layers, then there is a possibility that the model

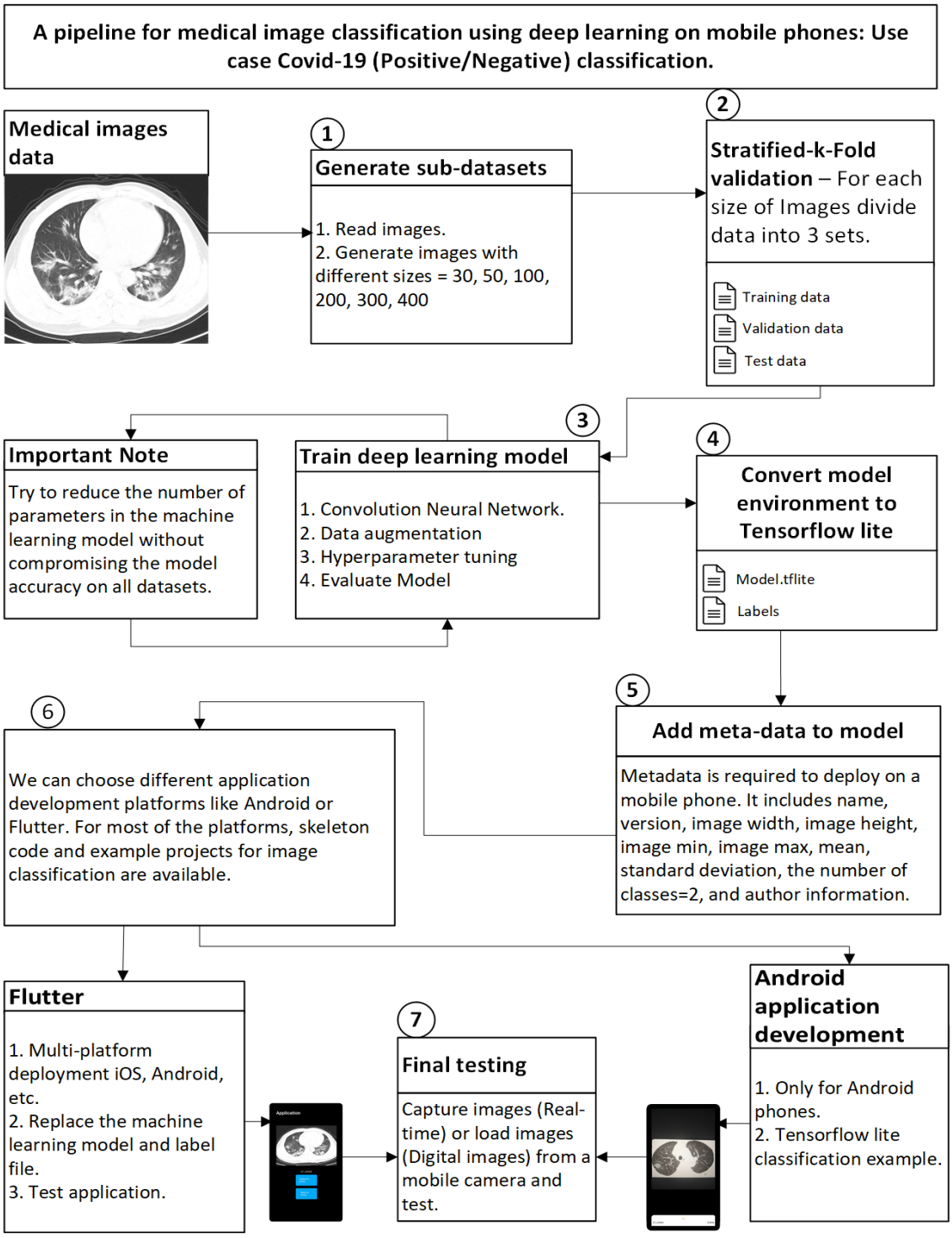


Fig. 1: This diagram shows the overall pipeline for implementing deep learning image classification model for mobile phone devices.

may overfit. If there are too few layers, the model may not learn applicable features. Second, reduce the number of parameters (one could imagine a systematic dropout algorithm that trade-off model size and accuracy) to enable execution within a mobile phone's memory [38], while also minimizing any associated performance penalties. This is another step that is typically adjusted "by hand," and as a starting point, we recommend following the steps taken in our use case below

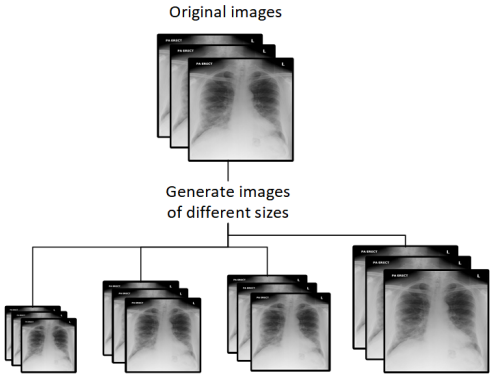


Fig. 2: Choose different images sizes and generate sub-datasets.

in Section 3. Other key considerations before model fitting are included in this subsection and are clearly implemented in the shared code linked below Section 3.

Data augmentation Data augmentation is another key step for maintaining model performance in real-world settings. It artificially stimulates and induces noise and other transformations on the training images when fitting the model [39,40]. In unpredictably noisy applications like ours using potentially multiple mobile phone cameras and hospital settings, such steps are essential.

Parameters for the data augmentation process are controlled via an image train generator and must also be specified. More specifically, when dealing with the medical images captured on mobile phones in real-time, it is essential to consider all options and possibilities with data augmentation, eventually settling on a few different combinations of settings. The key data augmentation parameters in our context are:

- `rescale`: Rescaling (normalizing the pixels values in the image) is the default parameter used in image preprocessing for training the model. Original pictures are in RGB format, with pixel values ranging from 0-255. Such numbers would be too large for the model to cope, resulting in an inflating gradient during the backpropagation phase when training the model. So we multiply the data by `1/255` to change the pixels values between 0-1.
- `rotation range, horizontal flip, and vertical flip`: These parameters randomly rotate and flip training data and should be included because mobile photos can be taken in various rotations. The rotated image is appended with pixels that degrade the classification accuracy in the data augmentation phase. For medical images, the horizontal or vertical flip is fine. However, when the image is rotated, we lose a vast amount of information, depending on the rotation range. One vital point to notice here is that rotated images generated by train generators have appended pixels and can degrade performance. In contrast, images captured by rotating phone cameras do contain all the information.
- `brightness_range`: The brightness range in the train generator increases or decreases the image's color brightness to produce multiple images. When the application is deployed in real-time, there is a possibility that the image's brightness captured from a mobile phone is different from the one on which the model is trained. So, this parameter is compulsory in the data augmentation phase to train the model on images of varying brightness.
- `zoom range, shear range, width shift range, and height shift range`: Zoom range randomly zooms inside pictures, shear range applies shearing transformations to image, width, and height shift change image dimension horizontally and vertically [41]. In a typical image classification task, these parameters can play an essential role in making the model robust. In the case of medical images, it is possible that if these factors are added, the

model's performance will suffer, as evidenced by the findings. In medical images, the difference between the positive and the negative case is subtle. For example, in the dog/cat classification problem, we can distinguish them quickly, but in cases/controls, there is just a white pattern in the image. The transformed image produced using these parameters can convert the positive case into negative and vice versa.

Table 1 contains our recommendation of 4 train generators and their respective parameter settings in Keras. Figure 3 shows the resulting directory structure after following the above steps.

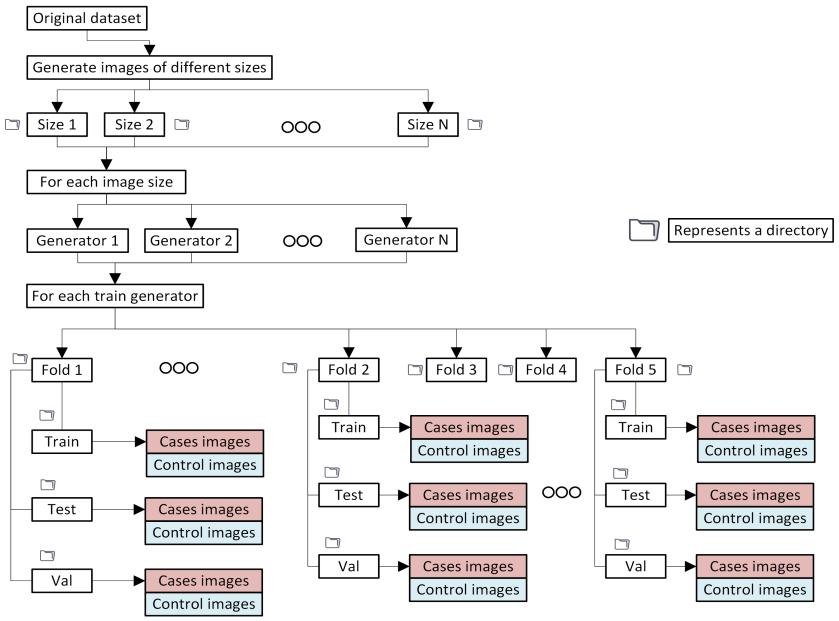


Fig. 3: This diagram shows a directory structure for training the model. Generate images with multiple sizes and make a folder for each size. There can be various train generators for each image size, which do not require a separate folder. For each image size, make one folder for each fold. To use data augmentation or train generator, make three folders (train, test, validation) containing sub-folders representing each category.

Parameter reduction Once one identifies the best model by Stratified-k-fold validation, we must reduce parameters to avoid crashing the mobile application. Hyper-parameter tuning does not, but the number of layers and neurons in each layer affects the model's size. Consider the best model from the previous step has two layers with N filters in the first (convolution layer) and M neurons in the second layer (fully connected layer). Increase filters from 1 to N for the first layer, 1 to M (Neurons) for the second layer, calculate model accuracy and the number of parameters in the model. The number of filters or neurons, the size of the filter, dropout, and strong pooling can be used to reduce the model size.

This step is time-consuming, but once the model with fewer parameters is achieved, it can also be deployed on other devices with low computation power like the Raspberry pi. This model contains a compact form of knowledge learned by the model having high parameters.

Image Generator Parameters	Generator 1	Generator 2	Generator 3	Generator 4
`rescale = 1./255`	✓	✓	✓	✓
`rotation_range = 40`		✓	✓	✓
`brightness_range = [0.2,1.0]`		✓	✓	✓
`horizontal_flip = True`		✓	✓	✓
`vertical_flip = True`		✓	✓	✓
`fill_mode = 'nearest'`		✓	✓	✓
`featurewise_std_normalization = True`			✓	✓
`featurewise_center = True`			✓	✓
`zoom_range = 0.2`				✓
`shear_range = 0.2`				✓
`width_shift_range = 0.2`				✓
`height_shift_range = 0.2`				✓

Table 1: Recommendations for data augmentation settings in Keras image data processing. The first column shows parameter settings, and ✓in subsequent columns denotes inclusion in the 4 recommended image generators.

2.4 Step 4: Convert model environment to TensorFlow Lite

At this step, we have two options: The first is to convert the model to TensorFlow Lite, and the second is to convert the model to TensorFlow Lite and quantize the model (A quantized model executes some or all of the operations on tensors with integers rather than floating-point values) [42].

2.5 Step 5: Specify appropriate metadata

Metadata gives information about the model in addition to its fit weights and architecture. It includes the model name, the input size (image size), and the output size (# of categories), and must be specified. Table 2 gives a starting point for metadata settings. After this step, we have two files: the TensorFlow Lite model and a label text file. One should also verify that the order of categories in the label text file matches the model's prediction order.

name	model's name
version	v1
image width	50
image height	50
image min	0
image max	1
mean	[0]
std	[255]
num_classes	2
author	X

Table 2: Model's metadata parameters and dummy values

2.6 Step 6: Specify appropriate application development platform

Different application development platforms can be used, like a Flutter (`Quantized TensorFlow Lite model`) or Android Studio (`TensorFlow Lite model`). An already developed application template can build deep learning applications in this stage. Explore several options and

decide on the flow of the application. For example, images can be classified in real-time using a camera feed or captured images. This step is related to application development and will not affect the final result.

2.7 Execution and final considerations

At this point, we have a model deployed on potentially multiple mobile phones capable of medical image classification. Select 3 to 5 images from all categories and test the TensorFlow Lite (Quantized/ Unqunatized) model performance on the computer, and that accuracy is the baseline accuracy. For mobile phones, we recommend testing the final model in two ways. The first is to build the application and load those images from the mobile gallery for evaluation (Flutter-based template) using a quantized model. The second option is real-time processing (TensorFlow template) using an unquantized model.

Just like preprocessing is required to train the model on a computer, there is also a preprocessing engine in mobile to preprocess the images, so there can be severe issues when images are tested from the camera feed. The first is the underlying hardware, and the result for the same image using the same model on a computer and mobile phone can yield different results, and there is no solution to mitigate this problem.

The sizes of the images can vary (See figures 4a and 4b), the distance at which the phone should be placed to classify the image can vary, and lastly, the location of images when captured through the phone. If the size of the image varies, then mobile phone distance can be changed such that the frame contains the image. We address each issue separately (See figure 4). Lastly, the background of images can vary (See figures 4c and 4d).

If the model's accuracy when pictures are loaded from the gallery is low, then the model will not perform when pictures are captured and classified in real-time. So, it is recommended to test the model performance on flutter application before shifting it to real-time.

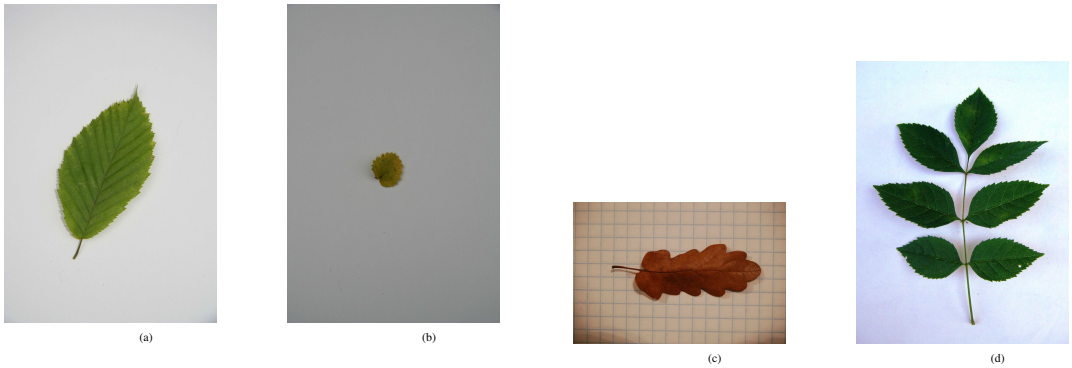


Fig. 4: Panels 4a/4b: Inconsistent images of Hornbeam. Panels 4c/4d: Difference of background of various catagories.

Figure 5 shows how the TensorFlow Lite application template perceives an image.

If classification accuracy is not sufficiently high, return to step 2.3 and repeat the process. To enhance performance, modify the picture size, machine learning model (number of neurons and layers), and hyper-parameters. This final step of testing on additional images is essential for robust performance, and even though the model will run without it, we do not recommend skipping it. Figure 6 shows the possibilities in which the proposed pipeline can be used depending on the type of the dataset.

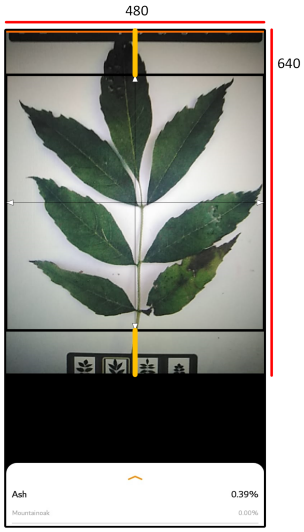


Fig. 5: This diagram shows how real-time picture is perceived by mobile phone. The width of the frame in the TensorFlow Lite template is 480 pixels, and the height is 640 pixels. The captured picture is cropped to 480 by 480, as shown by the black box in the middle. The two yellow lines are equal and show that a particular region is ignored.

3 Use Cases

Medical images have the potential to contribute as a less expensive and more rapid option that can deliver results in minutes instead of days [43]. The added value is not a replacement for the clinical diagnosis of but to rapidly augment physician information with an uncertain and rapidly evolving virus, thereby improving patient care and outcomes.

The system specifications for the following results are: Intel(R) Core(TM) 7-9750H CPU @ 2.60Hz, 16 GB RAM with NVIDIA GeForce RTX 2060 GPU, running Microsoft Windows 10. The development specifications are Cuda compilation tools release 10.0, V10.0.130, Deep Learning framework Keras 2.4.3, Python 3.6.8, and Tensorflow 2.3.1. The mobile phone specifications are a HUAWEI Y7 Prime 2019, Android version 8.1.0, EMUI version 8.2.0, and Model number DUB-LX1.

3.1 Dataset 1: Chest X-ray images

Dataset 1 consisted of images of size (1024,1024,3), which we reduced (Step 1, section 2.1) to $(50, 50, 3)$, $(100, 100, 3)$, $(200, 200, 3)$, and $(300, 300, 3)$. Data splitting for Stratified-5-fold validation (Step 2, section 2.2) resulting in training data (70%), validation data (10%), and test data (20%). The model architecture and training (Step 3, section 2.3) used a CNN with the architecture and hyper-parameters shown in table 9 and 10. This data already contained augmented positive/negative X-ray scan images, so we skipped data augmentation (Section 2.3). Cross-validation, with normal images and augmented images are distributed randomly in the train, validation, and test sets, produced the classification results in Table 3 with image size of 200 yielding the highest accuracy. Parameter reduction (Section 2.3) yielded a model with 4 filters in layer 1 (Convolution layer) and 8 neurons (Fully connected layer) in layer 2, resulting in a reduction in model size from 2,374KB to 24KB. Table 3 shows that the best performance was found with an image size of 200. Taken together with the results from parameter reduction, the model parameters for metadata were finalized and are shown in Table 4.

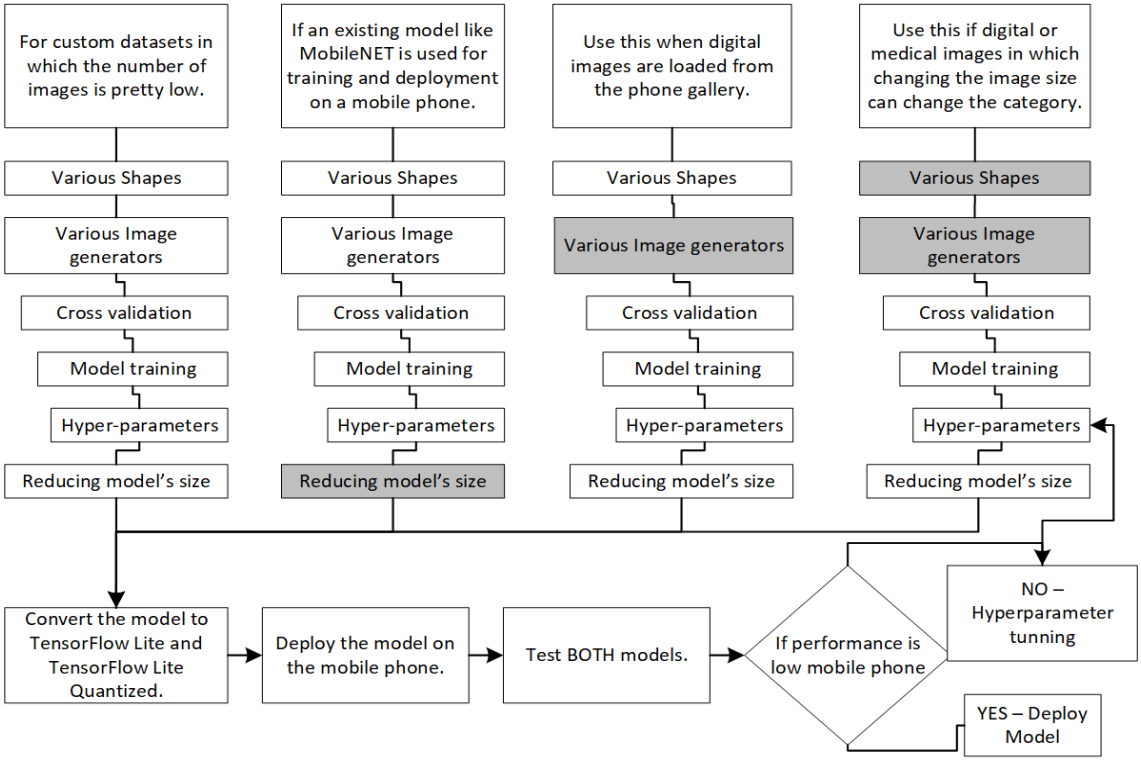


Fig. 6: This figure shows the four possibilities of using the proposed pipeline depending on the type of images. The grayed boxes skip a particular skip step for a particular images type. For custom datasets, all steps are compulsory. If the existing model is used, there is no need for parameter reduction because the model's size is already optimal. When images are loaded from the gallery image generator, digital images will not increase the accuracy. For digital and medical images like CT scans, the image's size is mostly fixed. If the model's performance when tested on a mobile phone is low, tune model architecture and perform hyper-parameter optimization by mutating batch size, number of epochs, and the number of layers in the model.

Dataset 1	
Size	**Accuracy**
50	95.88 (+- 1.80)
100	95.77(+- 1.65)
200	96.54(+- 1.87)
300	59.80(+- 19.41)

Table 3: Dataset 1 [44] cross validation accuracy for different image sizes.

This reduced model was converted to TensorFlow lite (Section, 2.4), and the associated code snippets and scripts can be found in this paper's code repository. These steps involved running the TensorFlow lite converter, generating an appropriate label file with classes, and adding the Table 4 metadata to the TensorFlow Lite model. This resulted in a model with well-defined input size, range of input values, and output (see Table 4). Then, the TensorFlow Lite Android wrapper code generator was used to create platform-specific wrapper code, which efficiently deploys and executes the model code on the mobile phone [45]. Next, we elected to use the Flutter image classification template (Section 2.6), allowing users to capture an image with their camera and pass it from their phone's image gallery application to the model, which then gives the positive or negative prediction.

name	**Dataset 1 Model**
version	**v1**
image width	**200**
image height	**200**
image min	**0**
image max	**1**
mean	**[0]**
std	**[255]**
num_classes	**2**
author	**X**

Table 4: Model's metadata parameters for the best model of dataset 1.

3.2 Dataset 2: Chest CT images

Dataset 2 [46] contained CT-scan images of size $(1024, 1024, 3)$, which we reduced (Step 1, section 2.1) to $(30, 30, 3)$, $(50, 50, 3)$, $(100, 100, 3)$, $(200, 200, 3)$, and $(300, 300, 3)$. The model architecture and training (Step 3, section 2.3) used two CNN models shown in Table 11 (Model 1) and 13 (Model 2) with the data augmentation, and cross-validation to train the model. Table 5 shows the result of classification for dataset 2.

DataSet 2 / Model 1	**Gen1**	**Gen2**	**Gen3**	**Gen4**
Shape=30	63.11(+- 2.98)	59.69(+- 6.21)	61.05(+- 6.48)	59.20(+- 5.11)
Shape=50	63.93(+- 4.33)	57.82(+- 2.84)	59.60(+- 2.80)	57.87(+- 6.51)
Shape=100	**65.82(+- 4.63)**	62.85(+- 4.13)	57.65(+- 2.70)	53.41(+- 6.60)
Shape=200	59.56(+- 3.58)	56.23(+- 7.67)	54.22(+- 5.46)	53.56(+- 3.58)
Shape=300	60.16(+- 9.06)	52.37(+- 6.17)	56.04(+- 6.30)	55.08(+- 2.91)
DataSet 2 / Model 2	**Gen1**	**Gen2**	**Gen3**	**Gen4**
Shape=30	50.95(+- 6.31)	54.89(+- 5.13)	60.68(+- 3.05)	55.77(+- 2.27)
Shape=50	**63.98(+- 1.88)**	60.41(+- 3.96)	58.87(+- 4.76)	56.13(+- 5.77)
Shape=100	59.18(+- 3.80)	55.30(+- 4.00)	56.45(+- 3.53)	55.41(+- 10.60)
Shape=200	62.94(+- 5.49)	53.93(+- 4.03)	56.30(+- 3.41)	50.39(+- 4.00)
Shape=300	60.74(+- 6.91)	50.25(+- 3.97)	51.92(+- 0.81)	53.33(+- 5.81)

Table 5: We used 5-fold validation, 5 different sizes of input images, and 4 generators. If the accuracy is too low, try multiple models.

The best accuracy for dataset 2 was for model 1 with image input size 100, model 1, and train generator 1. Parameter reduction (sub-section, 2.3) reduced the size of the model from 2,374KB to 323KB. Figure 9 (See supplementary material, 6) shows the heatmap of accuracies for each combination of filters and neurons in the first layer and the second layer. Convert model environment to TensorFlow lite version (Step 4, section 2.4).

Specify appropriate metadata (Step 5, section 2.5) adds metadata to the TfLite version of the model. The best accuracy for dataset 2 was for model 1, image size 100, so only change the metadata fields mentioned in Table 6.

name	**Dataset 2 Model**
image width	**100**
image height	**100**

Table 6: Model's metadata parameters for the best model of dataset 2.

After this step, we have a label file and a model with metadata. Specify appropriate application development (Section, 2.6) used flutter image classification template. Examples of the final application at work can be seen in Figures 7a and 7b.

In execution and final considerations (Step 7, section 2.7), we considered the first 5 images from both (CT_COVID and CT_nonCOVID) categories for the final testing on mobile phone, and those images were not part of model training/testing. We tested the model performance on the mobile phone in real-time, and the final accuracy was 0.6, which means the model cannot be deployed, but the performance increased to 0.80 percent when images were loaded from the gallery (Flutter App).

3.3 Dataset 3: Leaves

The dataset [47] consists of five leaves: Ash (24), Beech (28), Hornbeam (30), Mountainoak (20), and Sycamoremaple(20). Dataset is reduced (Step 1, section 2.1) to $(50, 50, 3)$, $(100, 100, 3)$, $(200, 200, 3)$, and $(300, 300, 3)$. The model architecture and training (Step 3, section 2.3) used two CNN models shown in Table 11 (Model 1) and Table 12 (Model 2) with the data augmentation, and cross-validation to train the model. Table 7 shows the result of classification for dataset 3.

DataSet 3	50 - Model 1	100 - Model 2	200 - Model 2	300 - Model 2
Gen1	87.13 (+- 19.63)	86.57 (+- 15.74)	98.04 (+- 2.39)	97.047 (+- 2.41)
Gen2	95.03 (+- 4.87)	91.28 (+- 7.73)	99.04 (+- 1.90)	96.04 (+- 1.97)
Gen3	96.73 (+- 1.64)	94.09 (+- 7.36)	96.04 (+- 3.73)	99.047 (+- 1.90)
Gen4	92.67 (+- 2.94)	91.28 (+- 8.18)	94.14 (+- 5.61)	91.33 (+- 12.92)

Table 7: We used 5-fold validation, 4 different sizes of input images, and 4 generators.

Parameter reduction (sub-section, 2.3) is skipped. Convert model environment to TensorFlow lite version (Step 4, section 2.4) and produce two files: `model.tflite` and `quantizedmodel.tflite`.

Specify appropriate metadata (Step 5, section 2.5) adds metadata to the TfLite version of the model.

After this step, we have a label file and a model with meta data. `model.tflite` is deployed on TensorFlow lite template and `quantizedmodel.tflite` is deployed on Flutter template (Section, 2.6).

In execution and final considerations (Step 7, section 2.7), we considered the first 4 images from all categories for the final testing on mobile phone, and those images were not part of model training/testing. We tested the model performance on the mobile phone when the image size was 50 for generator 3, but the final accuracy was 0.2, which means the model cannot be deployed. At this stage, repeat the process with variation in the model architecture (Step 2, section 2.2) and test the model performance. For shape 224, generator 1, and model 2 (See table 12), the performance increased to 0.75 percent when images were loaded from the gallery (Flutter App). For shape 224, generator 3, and model 2 (See table 12), the performance was 0.75 in real-time (TensorFlow App).

3.4 Dataset 4: Colorectal adenocarcinoma

This is a set of 7180 image patches (9 different categories) from N=50 patients with colorectal adenocarcinoma [48]. The dataset is challenging to train, test, deploy on the phone, and real testing.

Dataset is reduced (Step 1, section 2.1) to $(50, 50, 3)$, $(100, 100, 3)$, $(200, 200, 3)$, and $(300, 300, 3)$. The model architecture and training (Step 3, section 2.3) used one CNN models shown in Table

DataSet 4	50	100	200	300
Gen1	73.19 (+- 6.27)	71.20 (+- 4.11)	18.00 (+- 9.54)	16.80 (+- 8.81)
Gen2	69.19 (+- 10.24)	**75.59 (+- 3.20)**	22.79 (+- 3.91)	22.39 (+- 2.65)
Gen3	68.40 (+- 4.96)	70.00 (+- 4.89)	17.20 (+- 4.11)	20.39 (+- 3.44)
Gen4	70.0 (+- 6.32)	67.20 (+- 7.44)	20.79 (+- 3.24)	22.40 (+- 6.49)

Table 8: Cancer classification result.

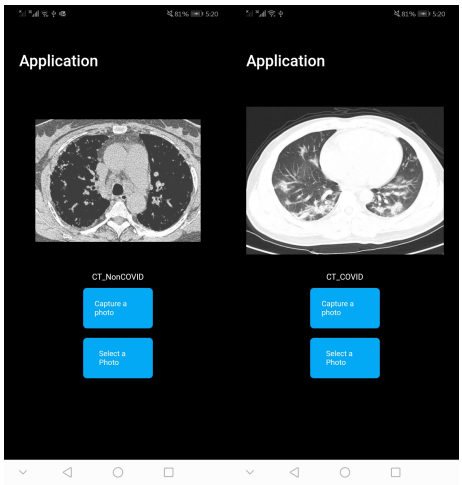
12 (Model 1) with the data augmentation, and cross-validation to train the model. Table 8 shows the result of classification for dataset 4.

Parameter reduction (sub-section, 2.3) is skipped. Convert model environment to TensorFlow lite version (Step 4, section 2.4) and produce two files: `model.tflite` and `quantizedmodel.tflite`.

Specify appropriate metadata (Step 5, section 2.5) adds metadata to the TfLite version of the model.

After this step, we have a label file and a model with meta data. `model.tflite` is deployed on TensorFlow lite template and `quantizedmodel.tflite` is deployed on Flutter template (Section, 2.6).

In execution and final considerations (Step 7, section 2.7), we considered the first 4 images from all categories for the final testing on mobile phone, and those images were not part of model training/testing. We tested the model performance on the mobile phone in real-time, and the final accuracy was 0.2, which means the model cannot be deployed. We increased the image size to 200, and the performance increased to 0.56 percent when images were loaded from the gallery (Flutter App). One point to notice here is the test accuracy was 0.99 on the computer.



(a) (b)

Fig. 7: Screenshot of application 1 7a 7b for covid detection using CT scan, deployed on android phone. Panels 7a/7b: Covid negative/positive Chest CT-scans, respectively.

The following paragraph elaborates the time to execute the pipeline.

For reading images (Step 1, 2.1), write a script, and it may take an average time of 20 - 30 minutes, depending on the way the dataset is stored. Below we calculated the time to train the model for images of various sizes for dataset 2, containing about 744 images. Time to train machine learning model (Step 3, 2.3) was 10, 17, 50, 180, and 600 minutes (total time 14 hours) for images having dimensions 30, 50, 100, 200, and 300. The parameter reduction step (Step 4, 2.4)

is time-consuming, and for one model having two layers, it took about 1 to 2 days. Converting the model to TensorFlow lite and adding metadata (Step 5, 2.5) takes about 5 minutes. Modifying the image classification template (Step 6, 2.6) for a specific dataset will take about 30 minutes. Building and deploying the application will take about 6 hours for one phone. So the total time for running the pipeline for one dataset is about 3 days.

4 Discussion

This section contains the limitations and future directions of the proposed pipeline. There are a few limitations associated with the proposed approach. For example, we considered only android phones (a specific vendor) running a particular version of the Android operating system. There is a high probability that the final classification performance would be identical for phones running the Android operating system due to the same android operating system, but for iPhone or Raspberry Pi, the results may vary. Such applications can also be developed for the iPhone using a different application development framework, one of the future directions for the proposed framework.

5 Conclusion

This study proposed a pipeline for deploying a deep learning model for medical image classification on mobile phones. The scope of the solution is not only limited to covid-19, but we can also use it for breast cancer or any other medical dataset, making the mobile phone a diagnostic tool for medical images classification. Complex models and other high-end application development skills can also lead to image segmentation. It is essential to highlight the usability and application of the proposed pipeline. Imagine traveling in a deep forest, which has insects and plants that can cause rashes. If someone suffers from skin allergies from a plant, they cannot call a doctor or find any medical assistance. Nevertheless, having a mobile phone application can tell which medicine is appropriate for a particular injury. Such applications can empower doctors in clinical settings where they may require knowledge from other sources to diagnose some diseases better. It will also give access to the public to use that application because there are about 4.3 billion people who use mobile phones.

6 Supplementary information

The documentation associated with the manuscript is available at the following link. https://github.com/MuhammadMuneeb007/A-pipeline-for-image-classification-using-deep-learning-on-mobile-phones

The code segments associated with the documentation are available at the following link. https://muhammadmuneeb007.github.io/A-pipeline-for-image-classification-using-deep-learning-on-mobile-phones/Find%20a%20dataset.html#directory-form

The files, associated applications, and directories are available at the following link. https://1drv.ms/u/s!AlFVll05llt7gwheV2i4SN3rba13?e=My0QKq

This section contains the material referenced in the section 3.

Model 1 architecture for dataset 1	
Layers	Parameters
Layer 1 - Con2D	30 Filters * (kernel size = (3,3))
Layer 2 - MaxPool2D	(pool size = (2,2))
Reshape	-
Layer 3 - FullyConnected	(50 Neurons)
Relu	-
Layer 4 - FullyConnected	(2 Neurons)
Softmax	-

Table 9: Model 1 architecture for dataset 1.

Model's Hyper-parameters	
Hyper-parameters	Value
Batch size	10
Epochs	50
Validation size	10%
Optimizer	SGD
Loss	categorical/binary_crossentropy
Metrics	Accuracy

Table 10: Hyper-parameters for all dataset 1 and 2.

Model 1 architecture for dataset 2	
Layers	Parameters
Layer 1 - Con2D	32 Filters * (kernel size = (3,3))
Layer 2 - MaxPool2D	(pool size = (2,2))
Reshape	-
Layer 3 - FullyConnected	(128 Neurons)
Relu	-
Layer 4 - FullyConnected	(2 Neurons)
Softmax	-

Table 11: Model 1 architecture for dataset 2.

Model 2 architecture for dataset 3	
Layers	Parameters
Layer 1 - Con2D	32 Filters * (kernel size = (3,3))
Layer 2 - MaxPool2D	(pool size = (2,2))
Layer 3 - Con2D	32 Filters * (kernel size = (3,3))
Layer 4 - MaxPool2D	(pool size = (2,2))
Layer 5 - Con2D	32 Filters * (kernel size = (3,3))
Layer 6 - MaxPool2D	(pool size = (2,2))
Layer 7 - Con2D	64 Filters * (kernel size = (3,3))
Layer 8 - MaxPool2D	(pool size = (2,2))
Reshape	-
Layer 9 - FullyConnected	(128 Neurons)
Relu	-
Layer 10 - FullyConnected	(50 Neurons)
Relu	-
Layer 11 - FullyConnected	(20 Neurons)
Relu	-
Layer 12 - FullyConnected	(Categories Neurons)
Softmax	-

Table 12: Model 2 architecture for dataset 3 and 4. We increased the number of convolutional layers to extract the information because the model 13 did not work.

Model 2 architecture for dataset 2	
Layers	Parameters
Layer 1 - Con2D	32 Filters * (kernel size = (3,3))
Layer 2 - MaxPool2D	(pool size = (2,2))
Layer 3 - Con2D	64 Filters * (kernel size = (3,3))
Layer 4 - MaxPool2D	(pool size = (2,2))
Reshape	-
Layer 5 - FullyConnected	(256 Neurons)
Relu	-
Layer 6 - FullyConnected	(128 Neurons)
Relu	-
Softmax	-

Table 13: Model 2 architecture for dataset 2.

Acknowledgments

This publication is based upon work supported by the Khalifa University of Science and Technology under Award No. CIRA-2019-050 to SFF.

References

1. S. Mitra and B. U. Shankar, “Medical image analysis for cancer management in natural computing framework,” *Information Sciences*, vol. 306, pp. 111–131, Jun. 2015. [Online]. Available: https://doi.org/10.1016/j.ins.2015.02.015
2. P. D. Velusamy and P. Karandharaj, “Medical image processing schemes for cancer detection: A survey,” in *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*. IEEE, Mar. 2014. [Online]. Available: https://doi.org/10.1109/icgccee.2014.6922267

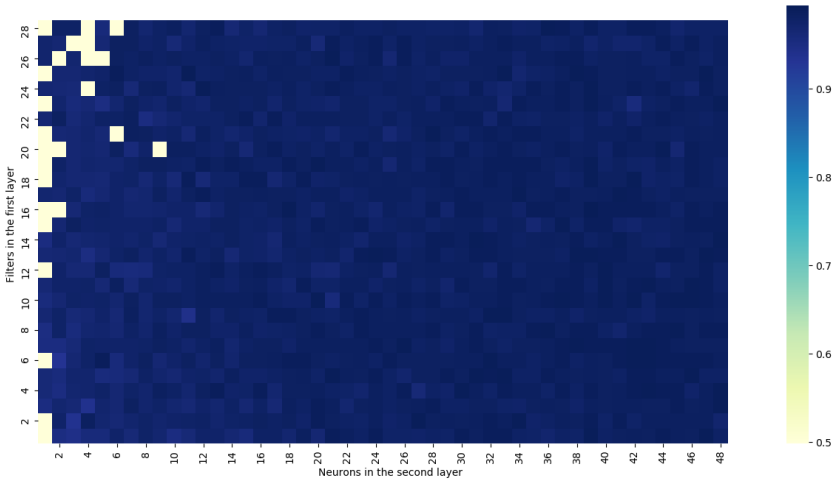


Fig. 8: Heatmap of accuracies. Y-axis and X-axis represent the number of neurons in the first layer and the second layer.

3. S. Bauer, R. Wiest, L.-P. Nolte, and M. Reyes, "A survey of MRI-based medical image analysis for brain tumor studies," *Physics in Medicine and Biology*, vol. 58, no. 13, pp. R97–R129, Jun. 2013. [Online]. Available: https://doi.org/10.1088/0031-9155/58/13/r97
4. W. Lin, T. Tong, Q. Gao, D. Guo, X. Du, Y. Yang, G. Guo, M. Xiao, M. Du, and X. Q. and, "Convolutional neural networks-based MRI image analysis for the alzheimer's disease prediction from mild cognitive impairment," *Frontiers in Neuroscience*, vol. 12, Nov. 2018. [Online]. Available: https://doi.org/10.3389/fnins.2018.00777
5. J. Islam and Y. Zhang, "Brain MRI analysis for alzheimer's disease diagnosis using an ensemble system of deep convolutional neural networks," *Brain Informatics*, vol. 5, no. 2, May 2018. [Online]. Available: https://doi.org/10.1186/s40708-018-0080-3
6. S. Bhattacharya, P. K. R. Maddikunta, Q.-V. Pham, T. R. Gadekallu, S. R. K. S, C. L. Chowdhary, M. Alazab, and M. J. Piran, "Deep learning and medical image processing for coronavirus (COVID-19) pandemic: A survey," *Sustainable Cities and Society*, vol. 65, p. 102589, Feb. 2021. [Online]. Available: https://doi.org/10.1016/j.scs.2020.102589
7. S. Liang, H. Liu, Y. Gu, X. Guo, H. Li, L. Li, Z. Wu, M. Liu, and L. Tao, "Fast automated detection of COVID-19 from medical images using convolutional neural networks," *Communications Biology*, vol. 4, no. 1, Jan. 2021. [Online]. Available: https://doi.org/10.1038/s42003-020-01535-7
8. T. A. Soomro, L. Zheng, A. J. Afifi, A. Ali, M. Yin, and J. Gao, "Artificial intelligence (AI) for medical imaging to combat coronavirus disease (COVID-19): a detailed review with direction for future research," *Artificial Intelligence Review*, Apr. 2021. [Online]. Available: https://doi.org/10.1007/s10462-021-09985-z
9. G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, Dec. 2017. [Online]. Available: https://doi.org/10.1016/j.media.2017.07.005
10. Reference, "Welcome to colaboratory - colaboratory," https://colab.research.google.com/notebooks/intro.ipynb, (Accessed on 08/16/2021).
11. ——, "Microsoft azure notebooks," https://notebooks.azure.com/, (Accessed on 08/16/2021).
12. T. V. Dittimi and C. Y. Suen, "Mobile phone based ensemble classification of deep learned feature for medical image analysis," *IETE Technical Review*, vol. 37, no. 2, pp. 157–168, Feb. 2019. [Online]. Available: https://doi.org/10.1080/02564602.2019.1576550
13. B. Hunt, A. J. Ruiz, and B. W. Pogue, "Smartphone-based imaging systems for medical applications: a critical review," *Journal of Biomedical Optics*, vol. 26, no. 04, Apr. 2021. [Online]. Available: https://doi.org/10.1117/1.jbo.26.4.040902
14. N.-M. Cheung, V. Pomponiu, D. Toan, and H. Nejati, "Mobile image analysis for medical applications," *SPIE Newsroom*, Jul. 2015. [Online]. Available: https://doi.org/10.1117/2.1201506.005997
15. A. Karargyris, O. Karargyris, and A. Pantelopoulos, "DERMA/care: An advanced image-processing mobile application for monitoring skin cancer," in *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*. IEEE, Nov. 2012. [Online]. Available: https://doi.org/10.1109/ictai.2012.180

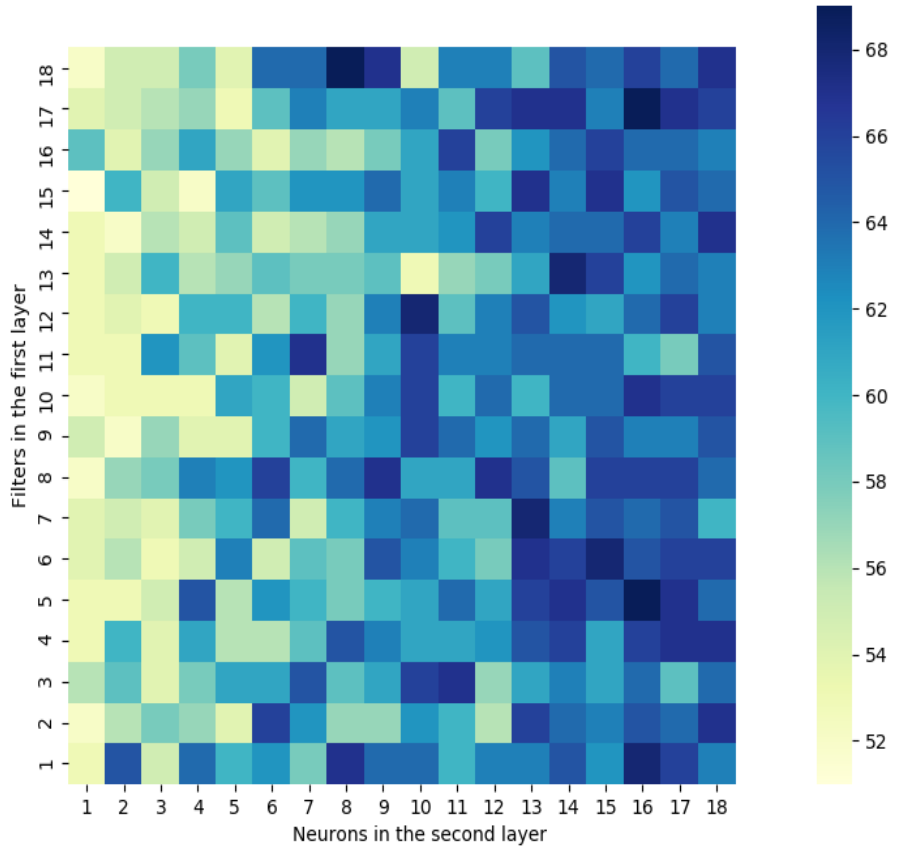


Fig. 9: Heatmap of accuracies. Y-axis and X-axis represent the number of neurons in the first layer and the second layer.

16. S. Pang, S. Wang, A. Rodríguez-Patón, P. Li, and X. Wang, "An artificial intelligent diagnostic system on mobile android terminals for cholelithiasis by lightweight convolutional neural network," *PLOS ONE*, vol. 14, no. 9, p. e0221720, Sep. 2019. [Online]. Available: https://doi.org/10.1371/journal.pone.0221720
17. M. Curiel and L. Flórez-Valencia, "Challenges in processing medical images in mobile devices," in *Trends and Advancements of Image Processing and Its Applications*. Springer International Publishing, Nov. 2021, pp. 31–51. [Online]. Available: https://doi.org/10.1007/978-3-030-75945-2_2
18. A. Johny and K. N. Madhusoodanan, "Edge computing using embedded webserver with mobile device for diagnosis and prediction of metastasis in histopathological images," *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, Nov. 2021. [Online]. Available: https://doi.org/10.1007/s44196-021-00040-x
19. C. Morikawa, M. Kobayashi, M. Satoh, Y. Kuroda, T. Inomata, H. Matsuo, T. Miura, and M. Hilaga, "Image and video processing on mobile devices: a survey," *The Visual Computer*, vol. 37, no. 12, pp. 2931–2949, Jun. 2021. [Online]. Available: https://doi.org/10.1007/s00371-021-02200-8
20. R. Rajendran and J. Rajendiran, "Image analysis using smartphones for medical applications: A survey," pp. 275–290, Jul. 2019. [Online]. Available: https://doi.org/10.1002/9781119439004.ch12
21. J. K. Carroll, A. Moorhead, R. Bond, W. G. LeBlanc, R. J. Petrella, and K. Fiscella, "Who uses mobile phone health apps and does use matter? a secondary data analytics approach," *Journal of Medical Internet Research*, vol. 19, no. 4, p. e125, Apr. 2017. [Online]. Available: https://doi.org/10.2196/jmir.5604
22. Z. F. khan and S. R. Alotaibi, "Applications of artificial intelligence and big data analytics in m-health: A healthcare system perspective," *Journal of Healthcare Engineering*, vol. 2020, pp. 1–15, Sep. 2020. [Online]. Available: https://doi.org/10.1155/2020/8894694

23. M. Straczkiewicz, P. James, and J.-P. Onnela, "A systematic review of smartphone-based human activity recognition methods for health research," *npj Digital Medicine*, vol. 4, no. 1, Oct. 2021. [Online]. Available: https://doi.org/10.1038/s41746-021-00514-4
24. K. Nwe, M. E. Larsen, N. Nelissen, and D. C.-W. Wong, "Medical mobile app classification using the national institute for health and care excellence evidence standards framework for digital health technologies: Interrater reliability study," *Journal of Medical Internet Research*, vol. 22, no. 6, p. e17457, Jun. 2020. [Online]. Available: https://doi.org/10.2196/17457
25. Q. Bai, Q. Dan, Z. Mu, and M. Yang, "A systematic review of emoji: Current research and future perspectives," *Frontiers in Psychology*, vol. 10, Oct. 2019. [Online]. Available: https://doi.org/10.3389/fpsyg.2019.02221
26. R. C. Jisha, J. M. Amrita, A. R. Vijay, and G. S. Indhu, "Mobile app recommendation system using machine learning classification," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, 2020, pp. 940–943.
27. K. Zhu, Z. Liu, L. Zhang, and X. Gu, "A mobile application recommendation framework by exploiting personal preference with constraints," *Mobile Information Systems*, vol. 2017, pp. 1–9, 2017. [Online]. Available: https://doi.org/10.1155/2017/4542326
28. K. Choi, K.-A. Toh, and H. Byun, "Realtime training on mobile devices for face recognition applications," *Pattern Recognition*, vol. 44, no. 2, pp. 386–400, Feb. 2011. [Online]. Available: https://doi.org/10.1016/j.patcog.2010.08.009
29. B. Ríos-Sánchez, D. C. da Silva, N. Martín-Yuste, and C. Sánchez-Ávila, "Deep learning for face recognition on mobile devices," *IET Biometrics*, vol. 9, no. 3, pp. 109–117, Feb. 2020. [Online]. Available: https://doi.org/10.1049/iet-bmt.2019.0093
30. D. M. Hedderich and S. B. Eickhoff, "Machine learning for psychiatry: getting doctors at the black box?" *Molecular Psychiatry*, vol. 26, no. 1, pp. 23–25, Nov. 2020. [Online]. Available: https://doi.org/10.1038/s41380-020-00931-z
31. Reference, "aylward.org - open-access medical image repositories," https://www.aylward.org/notes/open-access-medical-image-repositories, (Accessed on 08/18/2021).
32. ——, "Smir - sicas medical image repository," https://www.smir.ch/, (Accessed on 08/18/2021).
33. S. S. Nath, G. Mishra, J. Kar, S. Chakraborty, and N. Dey, "A survey of image classification methods and techniques," in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. IEEE, Jul. 2014. [Online]. Available: https://doi.org/10.1109/iccicct.2014.6993023
34. B. G. Marcot and A. M. Hanea, "What is an optimal value of k in k-fold cross-validation in discrete bayesian network analysis?" *Computational Statistics*, vol. 36, no. 3, pp. 2009–2031, Jun. 2020. [Online]. Available: https://doi.org/10.1007/s00180-020-00999-9
35. Y. Aslam and S. N, "A review of deep learning approaches for image analysis," in *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, 2019, pp. 709–714.
36. S. S. Yadav and S. M. Jadhav, "Deep convolutional neural network based medical image classification for disease diagnosis," *Journal of Big Data*, vol. 6, no. 1, Dec. 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0276-2
37. F. Chollet, *Deep Learning with Python*, 1st ed. USA: Manning Publications Co., 2017.
38. X. Xia, E. Shihab, Y. Kamei, D. Lo, and X. Wang, "Predicting crashing releases of mobile applications," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, Sep. 2016. [Online]. Available: https://doi.org/10.1145/2961111.2962606
39. C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, Jul. 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0
40. L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *ArXiv*, vol. abs/1712.04621, 2017.
41. Reference, "Building powerful image classification models using very little data," https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html, (Accessed on 08/22/2021).
42. Y. Deng, "Deep learning on mobile devices: a review," in *Mobile Multimedia/Image Processing, Security, and Applications 2019*, S. S. Agaian, S. P. DelMarco, and V. K. Asari, Eds. SPIE, May 2019. [Online]. Available: https://doi.org/10.1117/12.2518469
43. Reference, "Plain radiograph/x-ray - insideradiology," https://www.insideradiology.com.au/plain-radiograph-x-ray/#:~:text=It%20usually%20takes%20less%20than,about%2C%20and%20your%20general%20health., (Accessed on 06/02/2021).
44. A. M. Alqudah, "Augmented covid-19 x-ray images dataset," 2020. [Online]. Available: https://data.mendeley.com/datasets/2fxz4px6d8/3
45. Reference, "Adding metadata to tensorflow lite models," https://www.tensorflow.org/lite/convert/metadata, (Accessed on 09/06/2021).
46. J. Zhao, Y. Zhang, X. He, and P. Xie, "Covid-ct-dataset: a ct scan dataset about covid-19," *arXiv preprint arXiv:2003.13865*, 2020.
47. Reference, "Leaves," https://ftp.cngb.org/pub/gigadb/pub/10.5524/100001_101000/100251/Leaf_outlines.zip, (Accessed on 03/12/2022).

48. ——, “100,000 histological images of human colorectal cancer and healthy tissue — zenodo,” https://zenodo.org /record/1214456#.YhxvROhBzIU, (Accessed on 03/09/2022).

Authors

Muhammad Muneeb obtained his M.Sc. in Computer Science from the Khalifa University, Abu Dhabi, UAE. I am currently working as a research associate in the same institute under the supervision of Dr. Samuel. I like to work on inter-discipline problems. Have interests in algorithms, automation, genetics, medical image analysis, and optimization.

Samuel F. Feng obtained his PhD in Applied and Computational Mathematics from Princeton University (2012), his MA from Princeton University (2009), and his BA from Rice University (2007). From 2012 to 2014 he was a Ruth L. Kirschstein NRSA Postdoctoral Fellow at the Princeton Neuroscience Institute, working on stochastic models of decision making in psychology and neuroscience. In 2014 he joined the Department of Mathematics at Khalifa University, Abu Dhabi, UAE, where is currently an assistant professor.

Andreas Henschel earned his M.Sc. and Ph.D. in Computer Science from the Technical University of Dresden (Germany), in 2002 and 2008, respectively. He joined Masdar Institute as a Postdoctoral researcher in 2009. In 2011, he became Assistant Professor at Masdar Institute, UAE. He spent one year at the Massachusetts Institute of Technology MIT), USA as a Visiting Scholar.