Skin Cancer Detection Using Transfer Learning: From System Design to Mobile Deployment

By

Divyeshkumar Kiritbhai Patel
B.E., Gujarat Technological University, 2021

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

We acknowledge and respect the Lək^{w}əŋən (Songhees and X^{w}sepsəm/ Esquimalt) Peoples on whose territory the university stands, and the Lək^{w}əŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

Skin Cancer Detection Using Transfer Learning: From System Design to Mobile Deployment

By

Divyeshkumar Kiritbhai Patel
B.E., Gujarat Technological University, 2021

Supervisory Committee

Prof. Hong-Chuan Yang, Supervisor
(Department of Electrical and Computer Engineering)

Prof. David Capson, Departmental Member
(Department of Electrical and Computer Engineering)

Abstract

Skin cancer is the most common type of cancer. Recognizing the poor availability and reliability of existing solutions for skin cancer detection, we leverage the recent advancements in machine learning models and their on-device capabilities to provide an efficient and handy solution for the early detection of skin cancer. Specifically, we designed a lightweight solution using transfer learning with compact pretrained models to aid in the detection of skin cancer. Our solution can accurately detect malignant skin cancer and identify the five major types of skin cancers from images captured with a handheld dermoscope. We also deploy the solution on an Android mobile device. With our mobile application, general practitioners and remote healthcare workers can make guided referrals to a dermatologist for patients.

Table of Contents

List of Tables

List of Figures

Acronyms

HAM1000: Human Against Machine 10000

CNN: Convolutional neural network

DNN: Deep Neural Network

AI: Artificial intelligence

GPU: Graphics Processing Units

ML: Machine Learning

NPU: Neural Processing Unit

ISDIS: International Society for Digital Imaging of the Skin

ISIC: International Skin Imaging Collaboration

NV: Melanocytic Nevus

MEL: Melanoma

BCC: Basal Cell Carcinoma

AKIEC: Actinic Keratosis

DF: Dermatofibroma

VASC: Vascular Lesions

SCC: Squamous Cell Carcinoma

BKL: Benign Keratosis-Like lesions

UV: Ultraviolet

ReLU: Rectified Linear Unit

ReduceLROnPlateau: Reduce Learning Rate on Plateau

RMSProp: Root Mean Square Propagation

AUC: Area Under the Curve

ROC curve: Receiver Operating Characteristic curve

JEPG: Joint Photographic Experts Group

MB: Mega Bytes

GB: Giga Bytes

MS: Milli Seconds

UI: User Interface

SDK: Software Development Kit

RAM: Random Access Memory

XML: Extensible Markup Language

APK: Android Application Package

Acknowledgements

I want to thank my supervisor, **Prof. Hong-Chuan Yang**, for his valuable guidance and encouragement throughout this project and during my courses. His result-oriented feedback, patience, and expertise have helped a lot in shaping the direction of this project. I am especially grateful for his willingness to let me explore and experiment with different project topics before I finalized this study. This experience greatly broadened my learning.

I also want to thank my classmates for their helpful suggestions and constructive feedback during the development of this project.

1. Introduction

1.1. Background and Motivation

Skin cancer is the most common type of cancer. On average, out of three cancer cases, one of them is a skin cancer in Canada [1]. Over 80,000 cases of skin cancer are diagnosed in Canada each year, more than 5,000 of these are melanoma, the deadliest type of cancer [2]. In 2022, an estimated 330,000 new cases of melanoma were diagnosed worldwide, and almost 60,000 people died from the disease [3]. Early detection can save lives, as when melanoma is diagnosed in its early stages, the five-year survival rate is 99%. However, once it metastasizes, the survival rate drops to around 27% [4].

Despite this urgency, there is a huge shortage of dermatologists. In Canada, fewer than 700 dermatologists serve the entire country, of a population exceeding 40 million [5]. Therefore, there is a long waiting time, in the order of months, to visit a dermatologist. For those living in rural and remote communities, the nearest dermatologist may be hours to days away. Referral from a healthcare worker or a general practitioner can reduce this waiting time, but due to the skin lesion's visual similarity (Figure 1.1), it is challenging for a non-trained healthcare worker to decide whether it is cancerous (malignant) or not (benign).

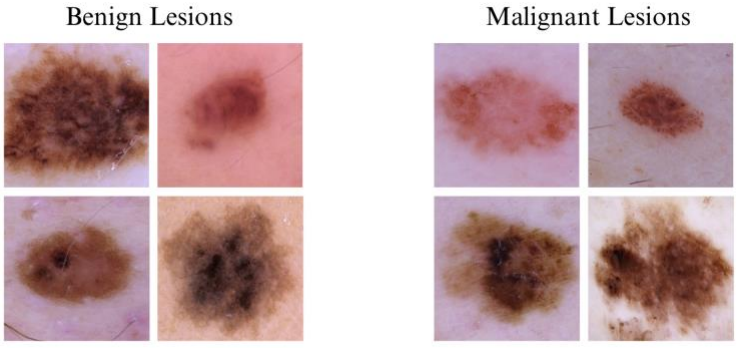


Figure 1.1: Visual similarities between benign vs malignant lesions.

1.2. Literature Review

1.2.1. Existing Research

Several studies have investigated the application of machine learning techniques for skin cancer detection from dermatoscopic images. These works provide the foundation for this project.

Andre Esteva (2017)[13] conducted a study on skin cancer detection using deep neural networks (DNNs). In this paper, 129,450 clinical images, including 3,374 dermoscopy images used for training models directly from images and labels instead of extensive preprocessing, lesion segmentation and extraction of domain-specific visual features before classification for developing a system that matches the classification level of a dermatologist. This experiment required significant resources, but it gave a direction to our project for training models using images and labels without a lot of handcrafted features for preprocessing.

The ABCD (Asymmetry, Border, Colour, and Diameter) rule for skin lesion diagnosis is typically used by experts. So, earlier systems focused on the extraction of these features from skin lesions. The goal was to develop specialized algorithms that extract colour, border features, symmetry, and other diagnostic criteria as inputs for machine learning classifiers. However, every year, more and more samples of dermoscopic images are made available, and machine learning is innovating at record speed, particularly in convolutional neural networks, which can learn patterns directly from images without relying on pre-set rules.

Another study conducted by Breno Krohling (2021) [14] presents the use of metadata and the use of Differential Evolution (DE) to balance the PAD-UFES-20 dataset. They achieved a balanced accuracy of 85% and a recall of 96% with a mobile application connected to a cloud server for skin cancer detection. They mainly focused on developing a system that can identify cancer from the photos taken by mobile devices. Also, the author chose a multiplatform approach with a mobile application and server due to the limitations present in smartphones for on-device machine learning.

Iren Valova (2023) [20] recently presented a similar study of a serverless mobile application for skin cancer detection using transfer learning. They evaluated five custom CNNs and four state-of-the-art pretrained models (Inception v3, ResNet50v2, DenseNet, Xception v2), applying data augmentation and fine-tuning of top layers to address dataset imbalance and improve accuracy. The Inception v3 model achieved the best results of 99.99% train accuracy and 83.49% test accuracy on the classification of 8 categories of skin lesions.

Kekal and Saputri (2023) [19] presented a similar study on detecting melanoma from skin lesions using the 2750 samples from the HAM10000 dataset. The MobileNet model was utilized in this study to classify lesions, focusing mainly on melanoma, nevus, and seborrheic keratosis. The study claims to achieve 88% overall test accuracy, where precision and recall for melanoma detection were 85% and 100% respectively.

Mahmoud (2023) [15] presented a recent study on the transfer learning of 10 different pretrained models and a custom model for melanoma detection using ISIC dermoscopic images, including models like SqueezeNet, GoogLeNet, AlexNet, ResNet-18, ResNet-50, MobileNet-v2, ShuffleNet, NASNet-Mobile, EfficientNetB0, and VGG-19. The dataset comprises 5,106 melanoma cases and 6,343 nevus cases. The study concluded that the proposed custom model achieved 90.85% of classification accuracy with a less complex and lighter model, in comparison to the large pretrained model ResNet-50, which achieved 92.98%.

These studies shaped our approach by showing the benefit of image-only training, a custom model, and transfer learning.

1.2.2. Available Solutions

In 2025, the most downloaded commercial mobile application for skin cancer detection is SkinVision, with 1 million-plus downloads on the Play Store. With this application, users can take pictures of skin lesions using a smartphone camera, and the application uses machine learning algorithms to classify skin lesions. It is a convenient and accessible solution, but the downside of it is that it relies on non-dermoscopic images, which often lack in detail visible in dermoscopic imaging. This raises questions about the reliability of this approach,

and the accuracy heavily depends on camera quality, lighting conditions and how the image is taken.

A recent study conducted in 2022, where 114 patients with potential skin lesions of benign and malignant skin lesions and 7 dermatologists were involved, compared the performance of this application. The performance on 1204 pigmented skin lesions remained below advertised rates with low sensitivity (41.3–83.3%) and specificity (60.0–82.9%) [6]. This shows the inefficiency and unreliability of a smartphone camera-based classification. As shown in Figure 1.2, a huge difference in image quality is visible, which is critical for the accurate prediction of skin cancer.

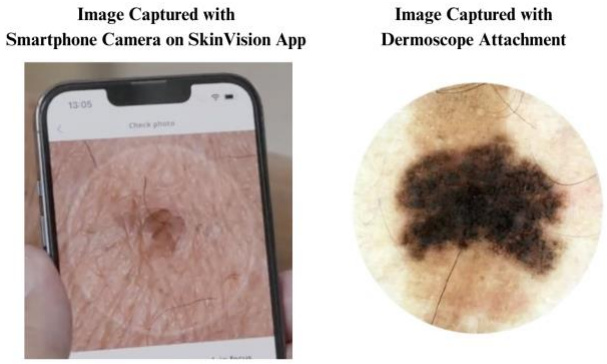


Figure 1.2: Difference between the quality of an image taken from a smartphone camera vs from a dermoscope attachment

In recent years, smartphones have undergone a remarkable evolution. Every year, the capabilities of smartphones have almost doubled compared to the previous year. A few years back, machine learning on smartphones was limited due to less powerful hardware, lower battery capacity and memory size. Most of the AI-powered mobile applications rely on a cloud server for processing, which can lead to delays, and it also raises privacy concerns.

Modern smartphones are now equipped with more powerful processors, a dedicated Neural Processing Unit (NPUs) and improved Graphics Processing Units (GPUs). These advancements allowed on-device machine learning to be possible. Both Android and Apple have their frameworks and libraries that offer ease of model inference, and benefits like reduced latency, enhanced privacy, and offline functionality. Google's Machine Learning (ML) Kit and TensorFlow Lite simplify

the custom model integration, and for Apple, Core ML provides similar functionality.

In this project, we build on these developments and deploy a TensorFlow Lite model in an Android application.

1.3. Project Goal

The goal of this project is to create a reliable solution for local healthcare workers to accurately identify skin cancer using a smartphone. We design and implement a system that can work with close-up dermoscopic images taken by a handheld dermoscope and use an on-device machine learning algorithm to identify whether a skin lesion is benign or malignant and detect the type of skin cancer.

By leveraging the learnings of pretrained models using transfer learning, we develop a system that can identify the skin lesion whether it's benign or malignant accurately, and not just limited to identifying cancer or non-cancer, it can also detect cancer type from 5 major skin cancers.

Compact application size and privacy with on-device inference capabilities are our main considerations in our design. By utilizing the mobile-optimized and lightweight pretrained models, we control the model size that contributes the most to the overall application size. TensorFlow Lite provides a way to run on-device model inference, which also reduces the cost of incorporating the server in the system. By focusing on models with fewer parameters, we ensure that the application will not put much overhead on the user's device and use fewer resources like memory and battery power. As a result, more healthcare providers will benefit from the system, integrating it with their existing mobile devices.

The remainder of the report is organized as follows.

Chapter 2 provides the dataset introduction and preparation strategies. It covers two public datasets (HAM10000 – Human Against Machine 10000 and BCN20000 – Skin Lesions in the wild) that contain images taken from a dermoscope for skin lesions, and different preprocessing techniques like augmentation, data cleaning, data balancing for better generalization, and dataset split for creating different datasets for machine learning models.

Chapter 3 presents the model design and training for our system. It explains transfer learning, model architecture, and contains various experiments we conducted to obtain our best-performing models. The model performance section from this chapter compares the proposed solution with existing solutions discussed in the literature review.

Chapter 4 discusses the overall system architecture and explains the steps taken to create a mobile application, from creating an empty project to deploying it on an Android device.

Eventually, Chapter 5 summarizes the project and looks forward to the future directions in which our models and mobile application could potentially be improved.

2. Dataset Preparation

The datasets used in this project are taken from the International Skin Imaging Collaboration (ISIC) Archive. The ISIC is an international initiative aimed at enhancing melanoma diagnosis, sponsored by the International Society for Digital Imaging of the Skin (ISDIS). The ISIC Archive contains the largest publicly available collection of quality-controlled dermoscopic images of skin lesions [7]. Beginning in 2016, ISIC has sponsored annual challenges for skin cancer detection using updated and expert-labelled datasets.

2.1. Datasets Introduction

In this project, the datasets used to train and evaluate machine learning models include HAM10000 and BCN20000, which are presented in the following sections.

2.1.1. HAM10000

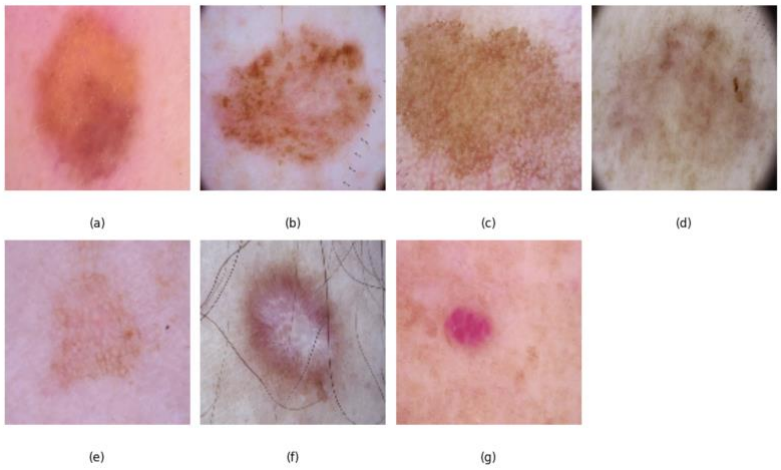
This dataset consists of 10,015 high-quality dermoscopic images. This dataset has served as a training set for the 2018 ISIC challenge. More than 50% of the lesions are confirmed through histopathology, and the rest through follow-up examination, expert consensus, or confirmation by in vivo confocal microscopy[8].

Metadata fields include:

- **lesion_id**: Groups images from the same lesion if taken at different times or angles, and contains a total of 7470 unique lesion IDs.
- **image_id**: A unique identifier of each image.
- **dx**: The confirmed type of skin lesion (**bkl**: Benign keratosis-like lesion, **nv**: Melanocytic nevus, **df**: Dermatofibroma, **mel**: Melanoma, **vasc**: Vascular lesion, **bcc**: Basal cell carcinoma, **akiec**: Actinic keratosis)
- **dx_type**: The method of confirming the lesion type (e.g., histo (Histopathology), follow_up (Follow-up examination), consensus (Expert checkups), or confocal (Laser scanning))
- **age**: Patient's age at the time of image taken.

- **sex**: Patient's gender.
- **localization**: Body location where the lesion is located.

Figure 2.1 below shows an individual sample of each diagnosis type of the HAM10000 dataset.



Here, (a): Melanocytic nevus, (b): Melanoma, (c): Benign keratosis-like lesion, (d): Basal cell carcinoma, (e): Actinic keratosis, (f): Dermatofibroma, (g): Vascular lesions

Figure 2.1: Samples of diagnosis in HAM10000

From our analysis of the HAM10000 dataset (Figure 2.2), it was clearly visible that the dataset is heavily imbalanced, which is a common issue present in most skin cancer datasets. More than 60% of the samples are benign NV (nevus) lesions. The rest of the lesion types, like MEL (melanoma), BCC (basal cell carcinoma), and others, have very few samples. This kind of imbalance can make it harder for the model to learn to detect rare but important cases like melanoma, and the model can be heavily biased towards the class with the majority.

When we looked at the benign versus malignant distribution (Figure 2.2), we saw that about 80% of the lesions are benign, and only 20% are malignant. This shows that the dataset is skewed toward non-cancerous samples, and we needed to address this class imbalance during model training.

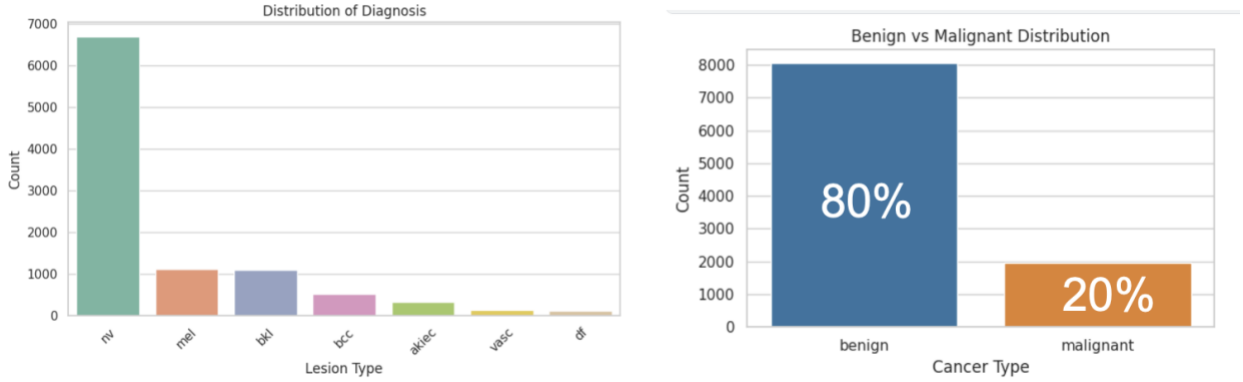


Figure 2.2: Class distribution in HAM10000

2.1.2. BCN20000

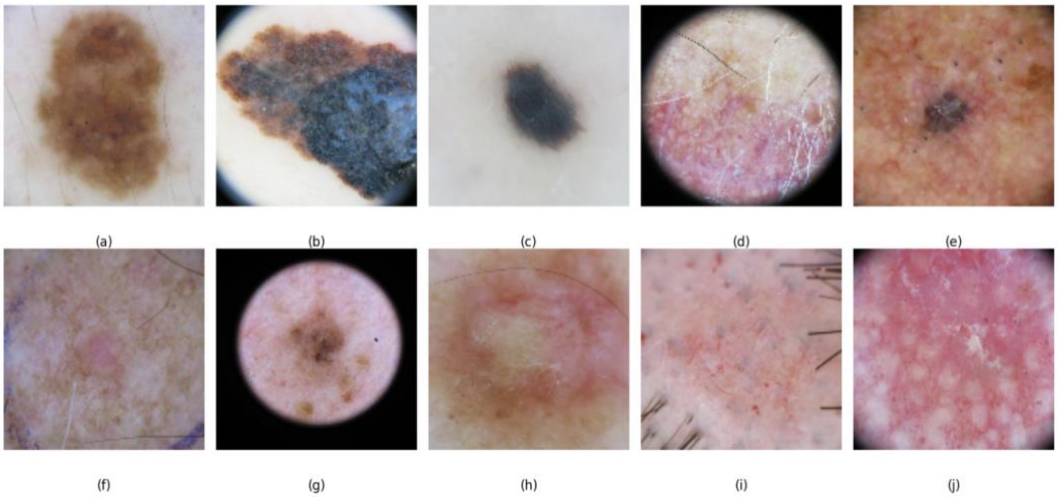
This dataset includes 18,946 high-quality dermoscopic images, which mirror the real-world clinic scenarios. Lesions which are found in hard-to-diagnose locations like nails and mucosa, as well as large lesions which may not fit in the dermoscopy device, are also included in this dataset. This helps bridge the gap between the training data for machine learning models and the day-to-day practice of medical practitioners [9].

Metadata fields include:

- **isic_id:** A unique identifier of each image.
- **attribution:** Name of the image provider.
- **age_approx:** Approximate age of the patient at the time of image taken.
- **anatom_site_general:** Body location of the lesion, includes anterior torso, upper extremities, lower extremities, head/neck, palms/soles, and oral/genitalia.
- **benign_malignant:** Diagnosis of the lesion, whether it is non-cancerous (benign) or cancerous (malignant).
- **diagnosis:** diagnosed cancer type (nevus, melanoma, other, squamous cell carcinoma, solar lentigo, basal cell carcinoma, melanoma metastasis, seborrheic keratosis, actinic keratosis, dermatofibroma, scar, vascular lesion)

- **diagnosis_confirm_type:** Shows how the diagnosis is confirmed, either through histopathology or an expert's general agreement from confocal microscopy.
- **melanocytic:** Indicates whether the lesion originates from melanocytes.
- **sex:** Patient's sex, recorded as male, female, or nan (not recorded).

Figure 2.3 below shows an individual sample of different diagnosis in BCN20000 dataset.



Here, (a) Melanocytic nevus, (b) Melanoma, (c) Melanoma metastasis, (d) Solar lentigo, (e) Basal cell carcinoma, (f) Actinic keratosis, (g) Seborrheic keratosis, (h) Dermatofibroma, (i) Vascular lesions, (j) Squamous cell carcinoma

Figure 2.3: Samples of different diagnosis in BCN20000

In the BCN20000 dataset (Figure 2.4), the data is more evenly spread across different lesion types. While nevus, melanoma, and basal cell carcinoma still have the most samples, other types like seborrheic keratosis, actinic keratosis, and squamous cell carcinoma are also fairly represented.

The benign vs. malignant distribution in this dataset (Figure 2.4) is much more balanced. Here, 47.1% of the lesions are benign, and 46.8% are malignant, which is almost equal. This balance makes BCN20000 a strong dataset for training models that can perform binary classification of the skin lesions more reliably.

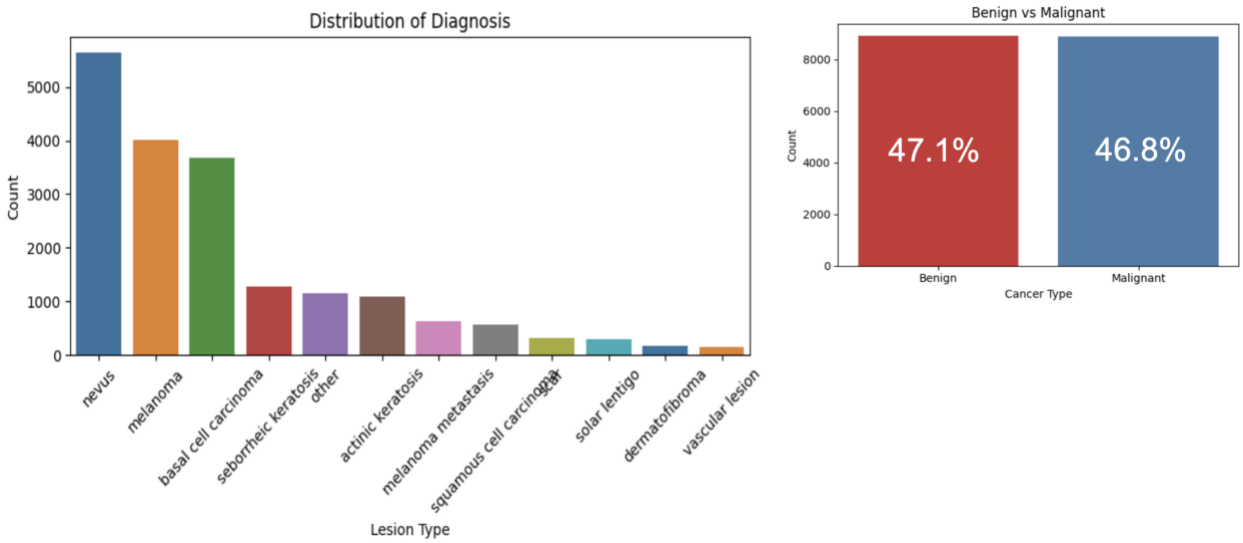


Figure 2.4: Class distribution in BCN20000

2.2. Data Preprocessing and Balancing

2.2.1. Data Cleaning and Formatting

Data preprocessing is an important step before training any model for consistent behaviour. For that, both datasets were analyzed for missing values and formatting issues. In the HAM10000 dataset, both images and their diagnosis labels were present for all the entries. For binary classification, an additional column was created for categorizing diagnosis into benign and malignant, value of 0 for benign and 1 for malignant. Here, the benign category includes nevus, benign keratosis-like lesion, vascular lesions, and dermatofibroma, and the malignant category includes melanoma, basal cell carcinoma, and actinic keratosis.

For BCN20000, rows with missing values for diagnosis were removed, and column ‘diagnosis_1’ contained labels categorizing benign and malignant. In this column, actinic keratosis was labelled as an ‘Indeterminate’. To get a balanced dataset for binary classification, we considered it as benign in the experiments with this dataset.

Images were first resized using TensorFlow’s tf.image.resize method [22], which by default applies bilinear interpolation, to match the model’s desired input

size of 224×224 pixels. Bilinear interpolation offers a good balance between computational effort and image quality, which is important for efficient model training. While some fine-grained features may be lost when downscaling, 224×224 provides sufficient resolution to capture lesion shape, color, and texture patterns while allowing faster training and compatibility with pretrained models. We also applied model-specific preprocessing functions provided by TensorFlow (e.g., tf.keras.applications.efficientnet_v2.preprocess_input() for EfficientNetV2B0).

2.2.2. Dataset Split

For effective training and testing, we split the dataset into training, test, and validation sets using the Scikit-learn library's stratified sampling technique to maintain class balance. 15% of the total dataset was kept for testing, and from the remaining 85%, another 15% was taken for the validation set. Overall, **our training set was 72.25%, validation was 12.75% and the testing set was 15%.**

2.2.3. Data Augmentation

To reduce overfitting and increase the diversity of the training set, we applied some basic augmentation techniques using TensorFlow. These augmentations were applied randomly during training:

- Random Flip: Image randomly flipped to the left and right.
- Random brightness: Light variations with the maximum delta of 0.2
- Random Contrast: Contrast variations with values 0.8 for lower and 1.2 for upper.
- Random Saturation: Colour variations with values 0.8 and 1.2.
- Random Hue: True Colour (red, green, and blue) variations with value 0.02.

These transformations helped the model generalize better by simulating real-world variations in lighting, orientation, and imaging conditions.

In the beginning, we used Keras ImageDataGenerator for applying augmentations and loading the image data. But epochs were slow due to the

Python-level image loading. For a larger dataset, this approach led to memory overflow and terminated the kernel.

To overcome these issues, we switched to TensorFlow tf.data API for building an input pipeline, which was 2-3 times faster (uses parallel loading) than ImageDataGenerator with NumPy array for images. Also, it resolved memory overflow issues, as it was directly accessing the images from the image location stored in the Data frame. It also provided better flexibility for extending the dataset if needed, as it only required updating the data frame.

2.2.4. Data Balancing

Both the HAM10000 and BCN20000 datasets have significant class imbalance. 60% of the samples in HAM10000 belong to the nevus class. Malignant lesions like melanoma and others were underrepresented.

To balance the classes in HAM10000 and enhance the model's ability to recognize less frequent cancer types, **we combined samples from both HAM10000 and BCN20000**. Before adding data into HAM10000, all the image IDs were compared to ensure no duplicate images were added to the existing dataset. This combined dataset contained a better balance of benign and malignant samples, and more than 1000 samples for each of the five different skin cancer types.

In addition to dataset merging, we also applied below approaches for the imbalanced HAM10000 dataset:

- **Class weights** were used during training to give more importance to minority classes.
- **Stratified batch sampling** to ensure that each training batch contained a mix of classes.

These approaches help reduce the bias toward dominant classes like nevus.

3. Model Design and Training

3.1. Transfer Learning

As our primary goal is to deploy a solution on smartphones, we explored three lightweight pretrained CNN models from the Keras library that have fewer parameters. Here, pretrained models are neural networks trained on large datasets, like ImageNet (a vast visual dataset with over 14 million images) [10], to identify general patterns and extract important features from images. These models are already trained to detect features such as edges, shapes, textures, and even more complex features in deeper layers.

In this project, we explored mainly three pretrained models:

- **MobileNetV2**: A lightweight CNN model for small size, low latency, and low power consumption.
- **EfficientNetV2B0:** A lightweight CNN model that focuses on training speed and parameter efficiency.
- **NasNetMobile:** Specifically optimized for mobile devices to balance performance and resource consumption.

All these models were chosen for high accuracy with low resource usage, which is necessary for mobile or embedded environments. Pretrained models save training time and resources, as training a model from scratch takes a huge amount of time, and it requires a large dataset for better generalization.

These models can be leveraged for our classification task using the technique called Transfer Learning. Here, Transfer learning is a process where we reuse a model that was trained on one task and apply it to a different but related task [11]. In our case, we transferred models' ability to detect features from general images to our skin lesion classification task, by using the pretrained layers to extract features from dermoscopic images, and then training only the new layers (the custom head) we added on top, and a few of the last layers of the base model for classification (benign vs malignant and multiclass classification).

3.2. Model Design

3.2.1. Base Model

We used the above-listed models as a base model by setting include_top = False. This flag removes the final classification layers from the original model (Figure 3.1). In these base Models, the early layers detect simple features like edges and colours, the middle layers learn patterns and shapes, and the deeper layers focus on complex structures such as lesion borders and asymmetry. The custom head was added to this base model for binary and multiclass classification. We set trainable = False initially, so the base model's weights would not be updated during initial training, and then gradually unfreeze some of the last layers with a smaller learning rate so that pretrained weights will not be updated rigorously and lose the advantage of transfer learning.

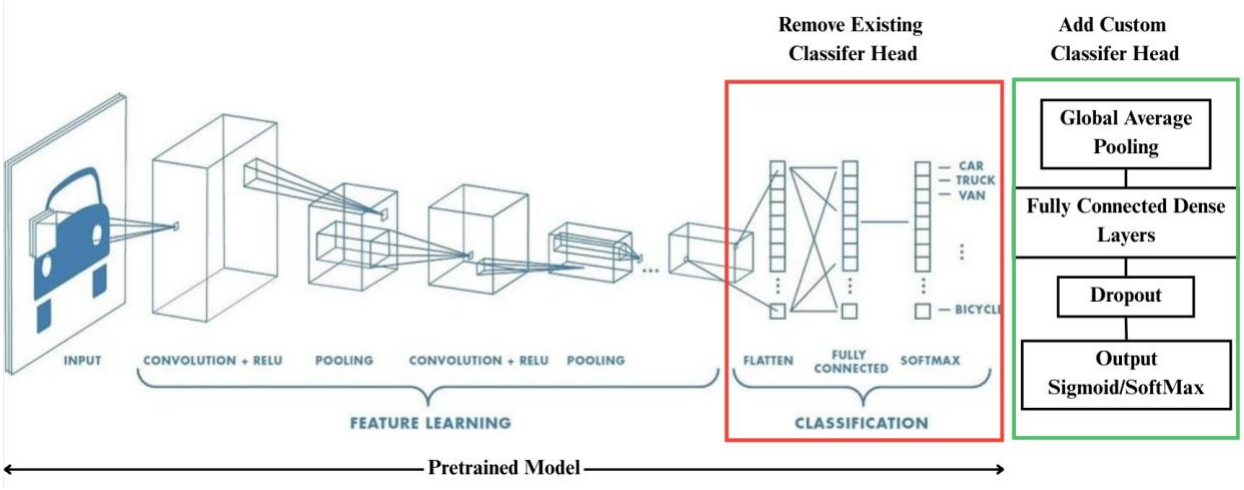


Figure 3.1: Transfer Learning model architecture [12]

3.2.2. Custom Head Structure

Models trained on ImageNet come with a head structure that can classify into 1000 categories. To modify this model to classify benign vs malignant and identify 5 cancer types, a custom head was added in place of the existing head.

Main components of the custom head include the following,

- **Global feature pooling**: It reduces the output of convolutional layers into a compact feature vector by averaging each feature map (GlobalAveragePooling2D).
- **Fully connected dense layers**: Experimented with the number of dense layers for learning patterns and relationships between features.
- **Dropout layer**: This layer randomly turns off the neurons during training to prevent overfitting.
- **Output layer**: This layer was adjusted depending on the classification type, a single neuron with sigmoid activation for binary classification, and a softmax layer for multiclass classification.

We ran several experiments with different head structures with a frozen base model to find the most suitable head for our problem. Figure 3.2 presents some of our experiments with head structure and initial training results of 20 epochs.

Experiment #1(Figure 3.2 (a)): Simple head structure with Global Average Pooling layer to reduce dimensions, one dropout layer with 0.4 dropout rate, and an output layer. The code below presents a custom head. Training for 20 epochs led to 71% train accuracy and 70% validation accuracy. Temporary dips in validation accuracy at epochs 10 and 15 are normal due to data batch variability and dropout in the custom head, which introduce small fluctuations during training.

```
model = tf.keras.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.4),
    layers.Dense(1, activation='sigmoid')
])
```

Experiment #2(Figure 3.2 (b)): 2 dense layers (128 neurons) added with dropout layers (dropout: 0.4) between the pooling layer and the output layer. This head structure achieved 76% train accuracy and 72% validation accuracy.

Experiment #3(Figure 3.2 (c)): Tested with one heavy dense layer with 512 neurons, and 128 neurons for another, added batch normalization and Leaky ReLU as activation function. It achieved 82% and 74% train and validation accuracy, respectively.

Experiment #4(Figure 3.2 (d)): Only one dense layer added with 256 neurons and a dropout layer with a 0.3 dropout rate. It achieved 87% and 75% for training and validation accuracy.

As we can see in Figure 3.2 (c) and (d). Using a complex head structure led to a greater gap between train and validation accuracy. This indicated an overfitting problem. This complex and larger head structure was also experimented with a higher dropout rate of 0.5 to prevent overfitting. Still, it was quickly overfitting in the fine-tuning phase.

From this observation, we decided to use a simple head structure (experiment 1) and a head structure with 2 dense layers and 128 neurons (experiment 2), which has a slightly lighter architecture for our training. After deciding on our head structure, we employed different fine-tuning techniques with some configurations to achieve our best models. In the next chapter, we will see the process of model training and fine-tuning.

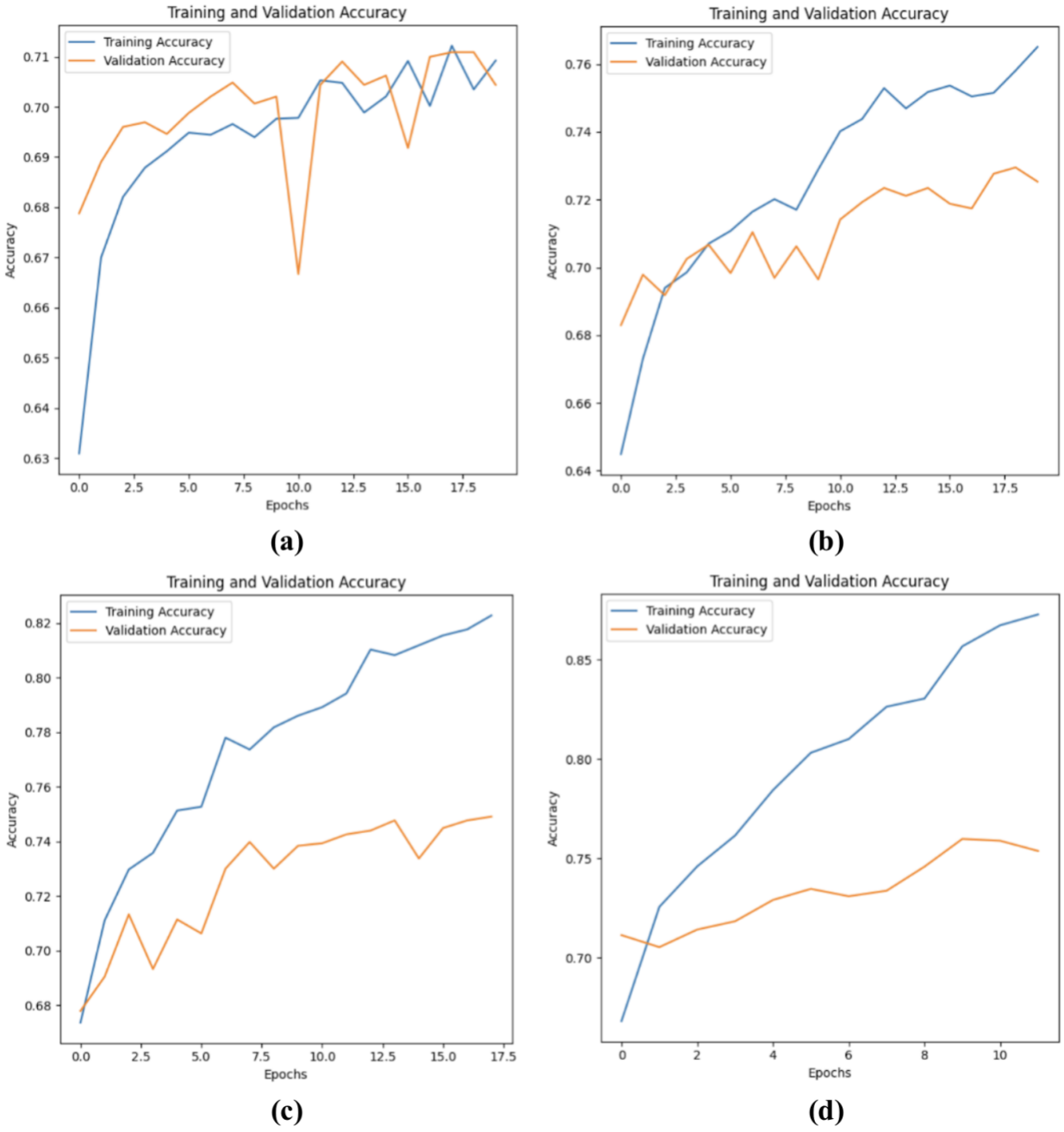


Figure 3.2: Performance of different head structures

3.3. Model Training

While training models, we followed a two-phase approach: initial training with a frozen base model and then fine-tuning with gradually unfreezing the last few

layers. We took inspiration from the standard procedure demonstrated on the TensorFlow website for Fine-tuning[16].

3.3.1. Initial Training

In the first phase, a base model was loaded from the Keras library with ImageNet weights and kept frozen using the flag “trainable = false”, and we trained it for 15-25 epochs with the learning rate of 0.0001 until the validation accuracy stabilized. On average, models reached 65-75% validation accuracy in this phase.

This initial phase helped train the final classification layers to learn basic patterns in our skin cancer classification from general features extracted by a pretrained base. Figure 3.2 with head structure experiments shows the results of initial training.

The following configurations were used for training with variations for achieving the best-performing model,

- **Learning Rate**: A Smaller learning rate (0.0001, 0.00001) was used during training to avoid large updates in pretrained ImageNet weights.
- **Reduce Learning Rate on Plateau** (ReduceLROnPlateau): When there was no improvement (decrement) seen for three consecutive epochs in validation loss, it will reduce the learning rate by a factor of 0.5.
- **Optimizer**: Adam and RMSProp optimizers were experimented, where RMSProp showed faster convergence.
- **Early Stopping**: When there was no improvement in validation loss for five consecutive epochs, this parameter was used to stop training automatically to prevent overfitting.
- **Adaptive Batch Size**: Three different batch sizes were experimented with: 16, 32, and 64. Where 32 was mainly used as it was the recommended batch size, but in some cases, when training was stopped early due to non-improvement, changing the batch size to a larger size helped achieve higher validation accuracy and escape local minima.

- **Model Checkpoints**: Validation accuracy was monitored, and when there was an improvement, it saved the model in the local directory to not lose the progress.

3.3.2. Fine-tuning

After initial training, we unfroze the last few frozen layers from the base model in each training cycle of 15-20 epochs or until early stop. In this step, we started with a 10 times smaller learning rate from the initial phase (0.00001). This step increased accuracy by 10-15% on average, which increased overall validation accuracy to 75-90%.

To preserve the learning from ImageNet, a Maximum of 30% of the last layers are unfrozen. For MobileNetV2, we have experimented with unfreezing 25-55 layers out of 154 layers. Similarly, for EfficientNetV2B0, 70-100 layers out of 270 layers, and 250-300 layers out of 769 layers for NasNetMobile.

In total, we trained each model for nearly 50-70 epochs, including the initial and fine-tuning phases. Figure 3.3 below demonstrates the effect of training the last few layers of MobileNetV2, EfficientNetV2B0 and NasNetMobile.

For MobileNetV2, after initial training of 20 epochs, where train and validation accuracy were 78% and 80% respectively, we unfroze the last 55 layers and trained them for more than 20 epochs. The model achieved an accuracy of 97% for training and 87% for validation. Here model started overfitting after 7-8 epochs and was early stopped due to no improvement in validation loss, and best weights were restored from the earlier epoch.

We unfroze 70 layers from EfficientNetV2B0 after initial training. In the initial training, it achieved a similar 79% training and 81% validation accuracy. Accuracy increased to 92% for training and 87% validation before early stopping at epoch 32. After that, another 30 layers, and in total 100 layers unfrozen, resumed training, the model achieved similar 93%, 87% for train and validation in the end as with the 70 unfrozen layers.

Another lightweight and mobile-optimized model in our list was NasNetMobile. We unfroze 250 layers after 13 epochs, and the accuracy graph

shows that the training accuracy spiked to 82.1% from 74%, while the validation accuracy improved by only 2-3%, rising from 78% to 82.1%.

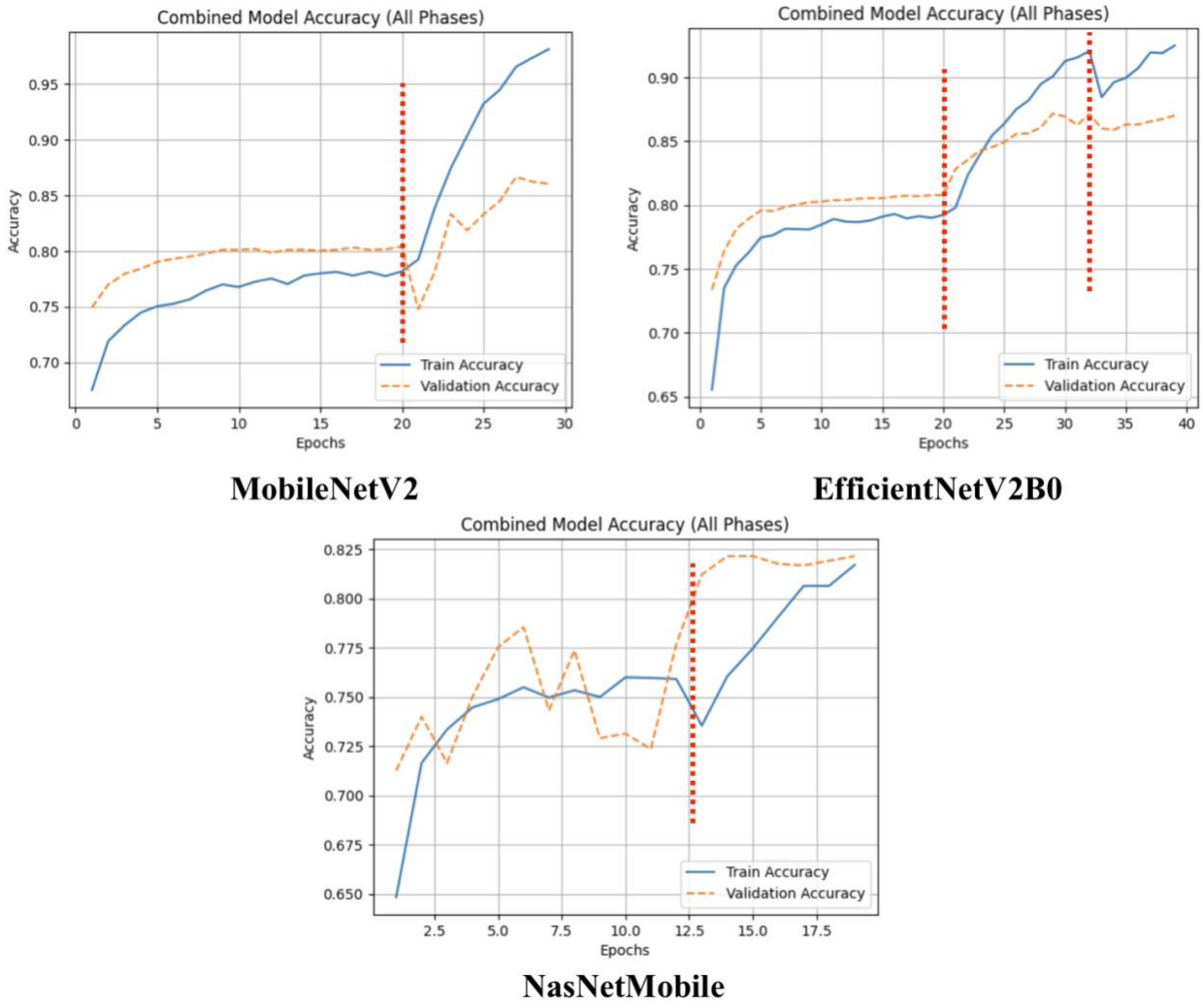


Figure 3.3: Effect of unfreezing and training the last few layers of MobileNetV2, EfficientNetV2B0, and NasNetMobile

The HAM10000 Dataset was used for training these models. We used class weights to reduce model bias towards the majority class, as the dataset was severely imbalanced. Models showed high performance for the benign class but struggled with malignant lesion detection (Figure 3.4).

MobileNetV2 and EfficientNetV2B0 both achieved a similar 87%, and NasNetMobile achieved 81% of test accuracy. However, as highlighted in Figure 3.4, out of 293 malignant samples, 122 samples are misclassified as benign for

MobileNetV2, which is 58% recall. EfficientNetV2B0 showed similar trends with slightly better recall at 61% recall, and NasNetMobile had the lowest recall of 47%.

These results indicate that even though the model learned to classify benign classes efficiently, they were biased due to the class imbalance of an 80% benign and 20% malignant ratio.

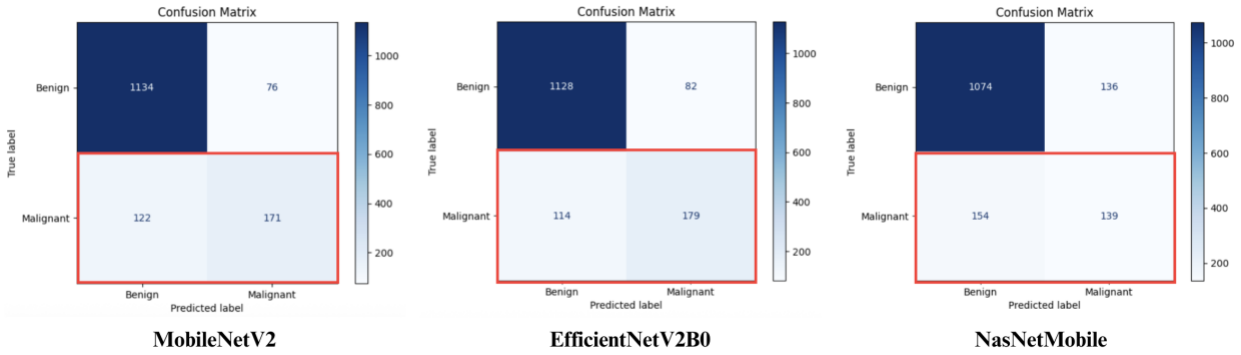


Figure 3.4: Models performance on HAM10000

After observing a poor performance on the imbalanced HAM10000 dataset, we trained our models on BCN20000 for binary classification, which was a larger and almost perfectly balanced dataset. Figure 3.5 shows the performance on the testing set for all the models, MobileNetV2, EfficientNetV2B0, and NasNetMobile.

MobileNetV2 achieved a 0.96 AUC(Area Under the Curve) Score and 90% overall accuracy in 96 total epochs. This time, the model was not biased towards only benign classes due to the perfect class balance between benign and malignant samples, as malignant classification precision was 89% and recall 92%, which was a huge leap from the HAM10000 dataset results. The confusion matrix in Figure 3.5 shows the model's consistency in identifying both benign and malignant lesions, with only a few misclassifications.

EfficientNetV2B0 stopped improving after 62 epochs, the overall accuracy achieved by this model was 87% with an AUC score of 0.92. Although having less accuracy than MobileNetV2, this model showed the fastest convergence in both initial training and fine-tuning. Out of 1256, around 223 benign samples were

classified as malignant (False Positives), and 105 malignant samples were classified as benign (False Negatives).

NasNetMobile showed poor performance on this dataset as well, it achieved 77% of overall test accuracy. Unlike the HAM10000, for this dataset, all the models were strong in identifying the malignant cases.

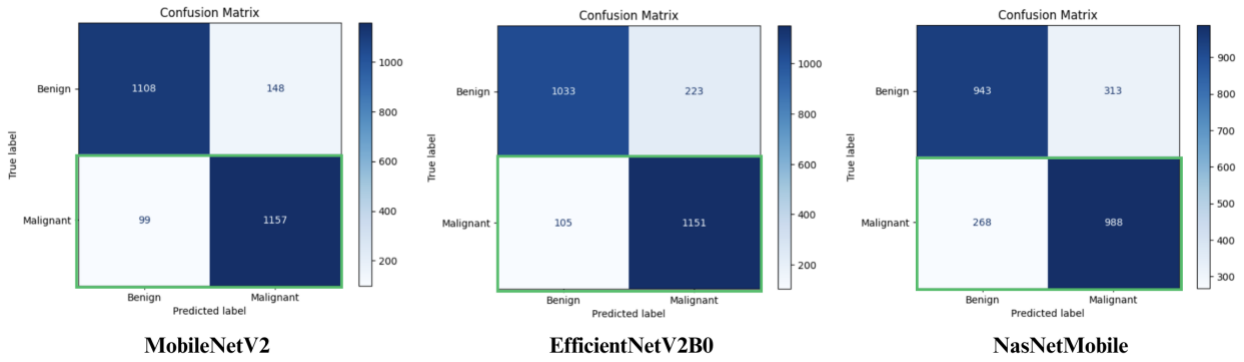


Figure 3.5: Models performance on BCN20000

From the results on both datasets, we found our two strong models for this problem, MobileNetV2 and EfficientNetV2B0, and observed that perfect class balance can make a big impact on the model's generalization capability.

To compare the performance of transfer learning of pretrained models, a custom model was built and trained on the same dataset.

3-block architecture is used for this custom model, which includes Conv2D layers with ReLU (Rectified Linear Unit) activation, MaxPooling, and Dropout. As shown in Figure 3.6, the convolutional layers return the output feature maps, which go into the Flatten layer. This layer transforms 2D output into a 1D vector. Then this 1D vector was passed to a fully connected Dense layer. This dense layer was added to learn complex patterns from the features extracted by the convolution blocks. To reduce overfitting, a Dropout layer was added, and then, a sigmoid output layer was added for binary classification. We chose a 3-block architecture (instead of a deeper or more complex model) to keep the training and tuning time reasonable. As with this configuration, it took 2 to 3 hours to find the best model with Keras tuner.

For tuning this model according to our problem, we used Keras Tuner with the Random Search strategy. We ran this tuner for 10 trials, each for 20 epochs, to find the best-tuned model. We automated our exploration of different combinations of filter sizes (32–256), dense units (64–256), and learning rates (1e-2, 1e-3, 1e-4) with this approach. We used Adam optimizer with binary cross-entropy as the loss function for this model. This automation saved manual tuning time and helped in avoiding guesswork while designing the custom CNN.

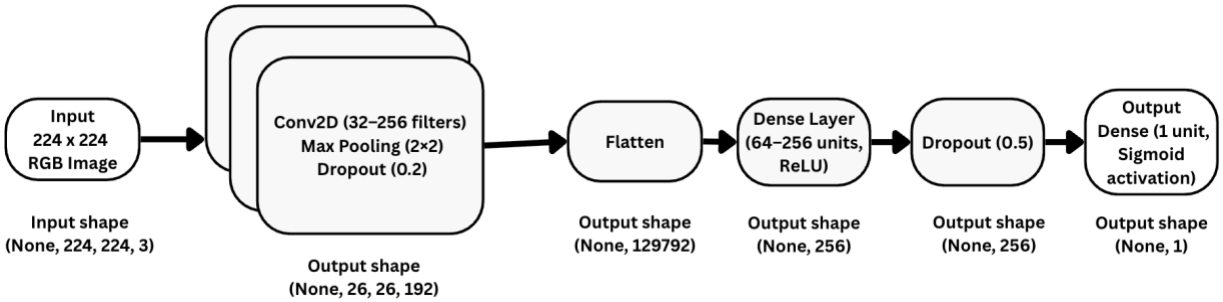


Figure 3.6: Custom CNN model architecture

After finding the best-tuned model from the Keras tuner, we started training that model for 50 epochs with early stopping. Model stopped improving after 16 epochs.

The Custom Model achieved an overall accuracy of 67%, which indicates the inefficiency of custom models as medical applications demand higher accuracy scores. From the confusion matrix, it is visible that the model was better in classifying the malignant class with a recall of 77%, but it also misclassified 531 benign samples as malignant out of 1256 benign samples, which is around 42% of misclassification (Figure 3.7).

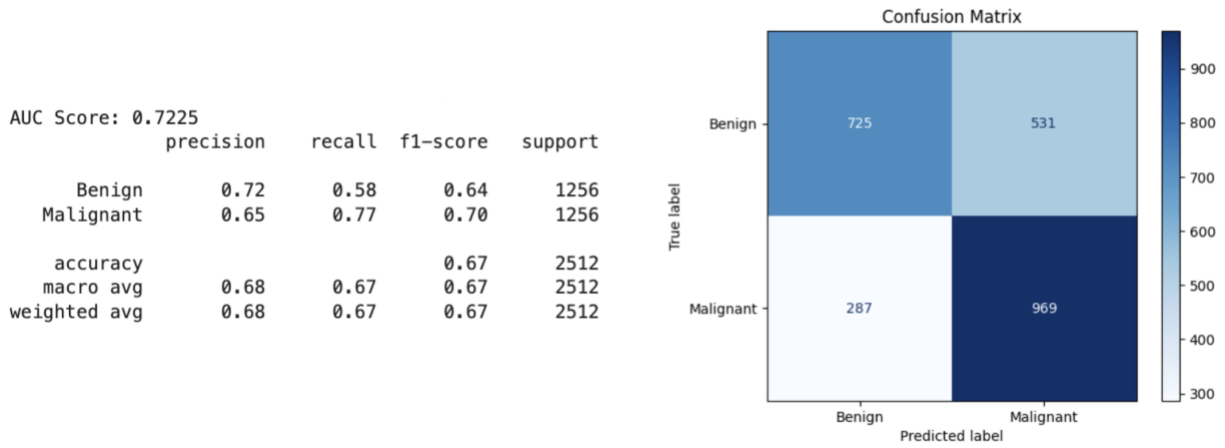


Figure 3.7: Binary classification results of baseline custom CNN model on BCN20000

These results show the power of pretrained models with transfer learning for this image detection task, as our custom CNN model was achieving an overall 67% of accuracy, these pre-trained models were surpassing it in the initial training itself, with 80% validation accuracy. It's also visible that these pretrained models achieve better performance in fewer epochs (generally 30-40).

After achieving 90% on the binary classification problem, **we extended our experiments to Multiclass classification to classify the skin lesions from major cancer types**. We considered 5 categories with more than 1000 samples each, which include Nevus(nv), Melanoma(mel), Benign Keratosis-like Lesion (bkl), Basal Cell Carcinoma(bcc), and Actinic Keratosis(akiec) for multiclass classification experiments.

Here, as shown in Figure 3.8, MobileNetV2 did not perform well, achieving 61% overall test accuracy. Mainly struggled in classifying BKL(Benign Keratosis-Like lesions), whereas EfficientNetV2B0 showed more balanced performance over different classes with 74% of overall classification accuracy.

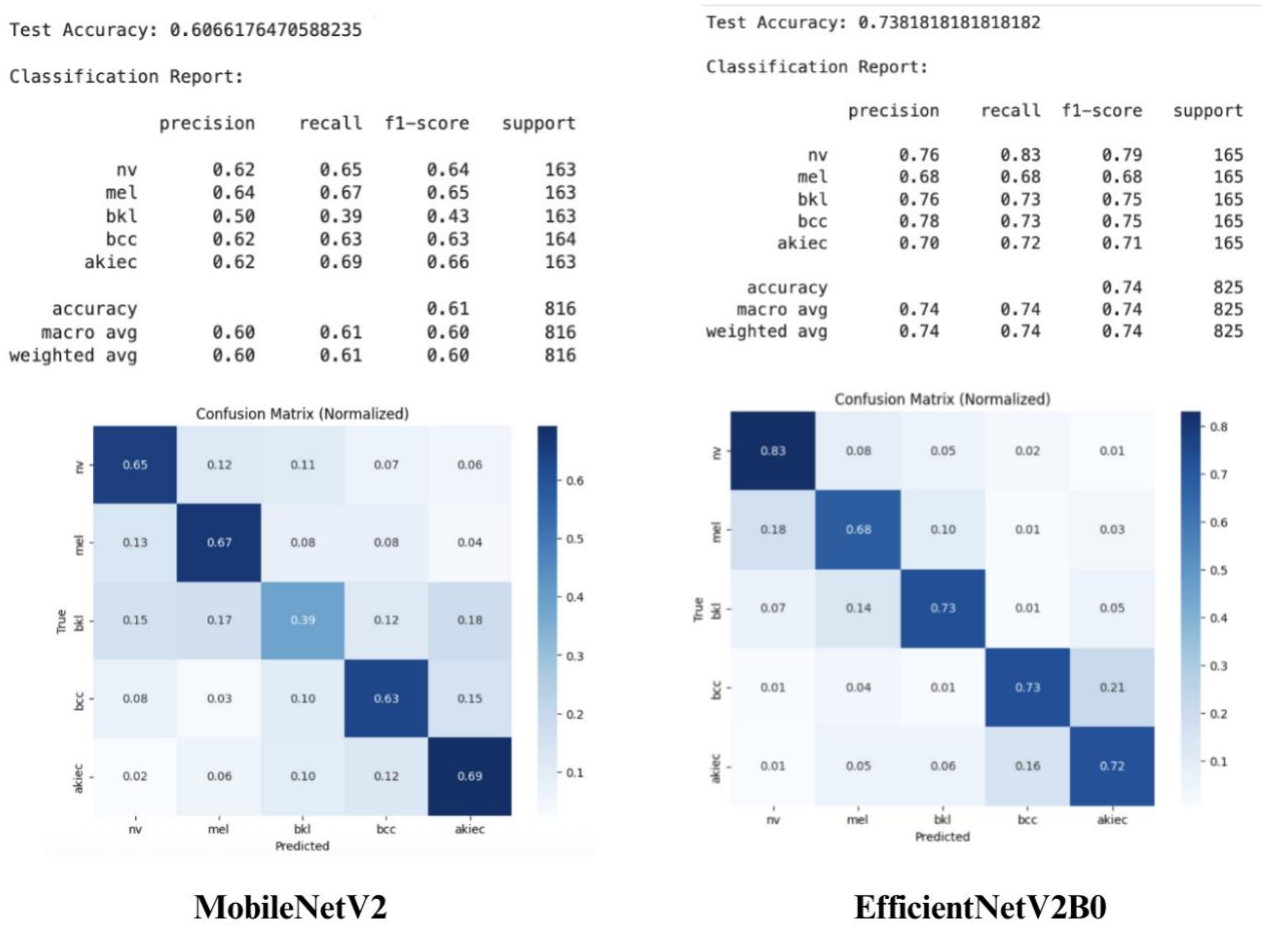


Figure 3.8: Multiclass classification results on BCN2000(MobileNetV2 and EfficientnetV2B0)

To improve test accuracy further for both binary and multiclass classification problems, we combined HAM10000 and BCN20000, because BCN20000 contains lesions which are found in nails and mucosa, as well as large lesions which may not fit in the dermoscopy device, which can affect model's learning capability, and HAM10000 contains consistent and high-quality lesion samples with proper alignment of lesions in the images.

3.4. Model Performance

Figure 3.9 shows the results of MobileNetV2 on the combined dataset. It achieved the highest accuracy and best precision and recall on this combined dataset with an overall 92.56% test accuracy and 0.97 AUC score. This model performed very well on malignant classification, as only 83 benign samples were

misclassified as malignant out of 1209 benign samples. Previously on BCN20000, it misclassified 148 benign samples.

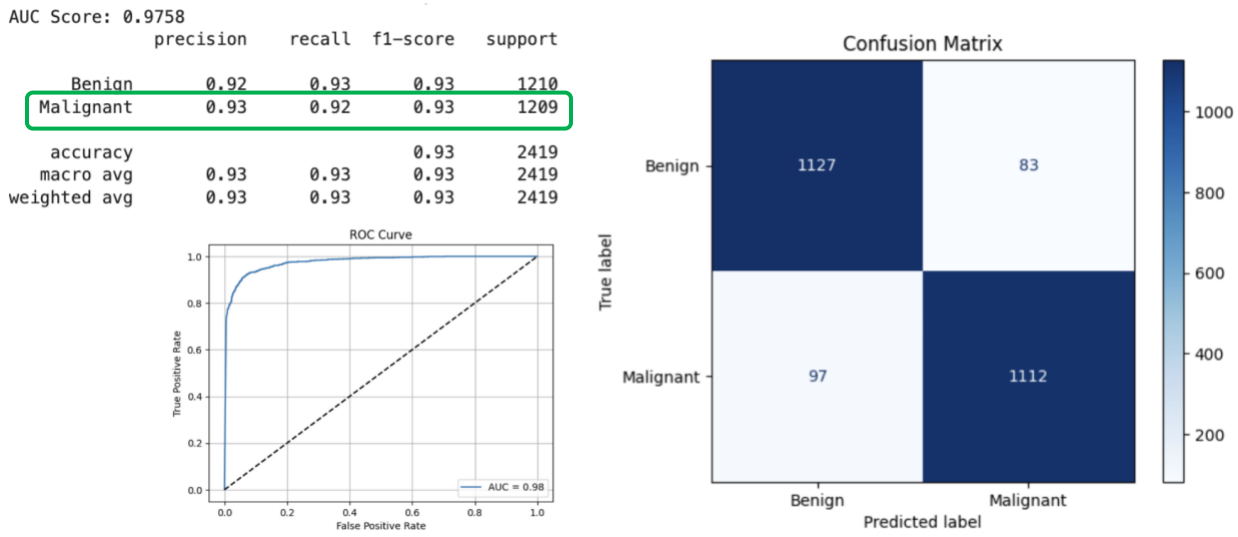


Figure 3.9: Binary classification results on the combined dataset (HAM + BCN)

Model	Accuracy	Precision	Recall	F1-Score	Model Size (MB)	Parameters
Custom CNN	82.76 %	83.58 %	82.76 %	82.66 %	401.27	100.3M
MobileNetV2	92.56 %	92.56 %	92.56 %	92.56 %	18.54	3.5M
EfficientNetV2B0	91.57 %	91.58 %	91.57 %	91.57 %	33.93	7.2M
NasNetMobile	87.43 %	87.44 %	87.43 %	87.43 %	21.05	5.3M
InceptionV3	89.67 %	89.70 %	89.67 %	89.66 %	138.57	23.9M
EfficientNetB7	90.33 %	90.35 %	90.33 %	90.33 %	377.9	66.7M

Table 1: Performance comparison of different models on the combined dataset

Table 1 above presents the performance comparison of 6 different models on the combined dataset. Apart from 3 lightweight models (MobileNetV2, EfficientNetV2B0, and NasNetMobile), being the largest model, the Custom Model achieved the least test accuracy of 82%, which shows the efficiency of pretrained models. Furthermore, for comparison, a Bigger pretrained model like InceptionV3 and EfficientNetB7 were also trained on the same dataset, and they did not outperform the smaller models, indicating that increased model size and complexity did not always yield a performance gain, while requiring substantially

more computational resources. Figure 3.10 below shows the ROC curve for all 6 models. Both MobileNetV2 and EfficientNetV2B0 cover the maximum area under the curve. Both performed similarly in terms of test accuracy, but MobileNetV2 is the optimal choice for deployment in the mobile application because of its low computational cost and its smaller model size.

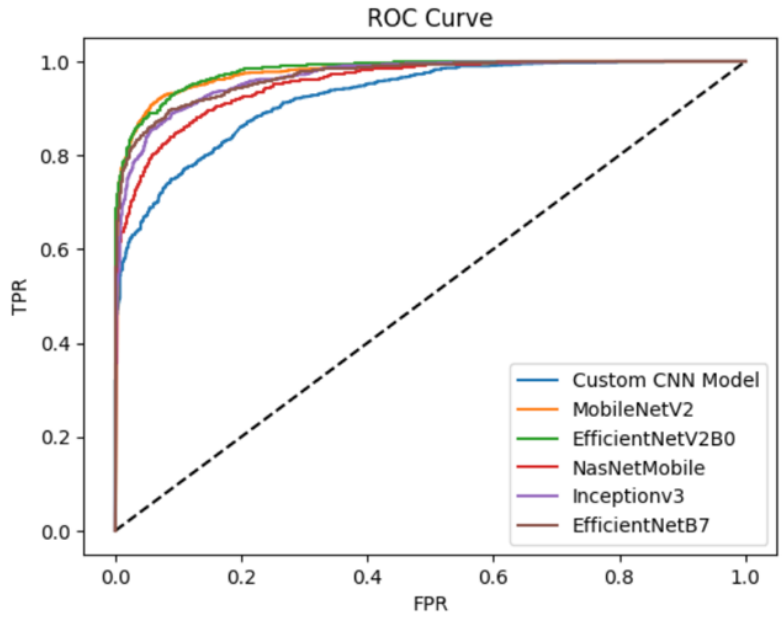


Figure 3.10: ROC curve of 6 different models

As we achieved the best performance of MobileNet on the combined dataset for binary classification, the same dataset was used for multiclass classification and for class balance, we took only 1000 samples for each class as per the results of the previous experiment with BCN20000. Here, the best performing model was EfficientNetV2B0 for the 5-class classification, with an overall test accuracy of 88.6% and a macro average F1-score of 0.89 (Figure 3.11).

Other models like MobileNetV2 achieved 69% test accuracy, while NasNetMobile achieved 59% test accuracy. A larger model, such as InceptionV3, had an overfitting problem on this dataset, achieving 99% train accuracy and 70% test accuracy.

The confusion matrix of EfficientNetV2B0 (Figure 3.11) showed consistent predictions across all classes with relatively low confusion between melanoma and

nevus, which are typically harder to distinguish. Additionally, it was noticeable that some overlap was present between melanocytic (nv, mel, bkl) and keratinocytic (bcc, akiec) lesions, due to the visual similarities between classes. In Figure 3.11, the bounding boxes show how these groups are misclassified internally. The multi-class ROC curve shows the model's efficiency in distinguishing between the 5 different skin cancer classes.

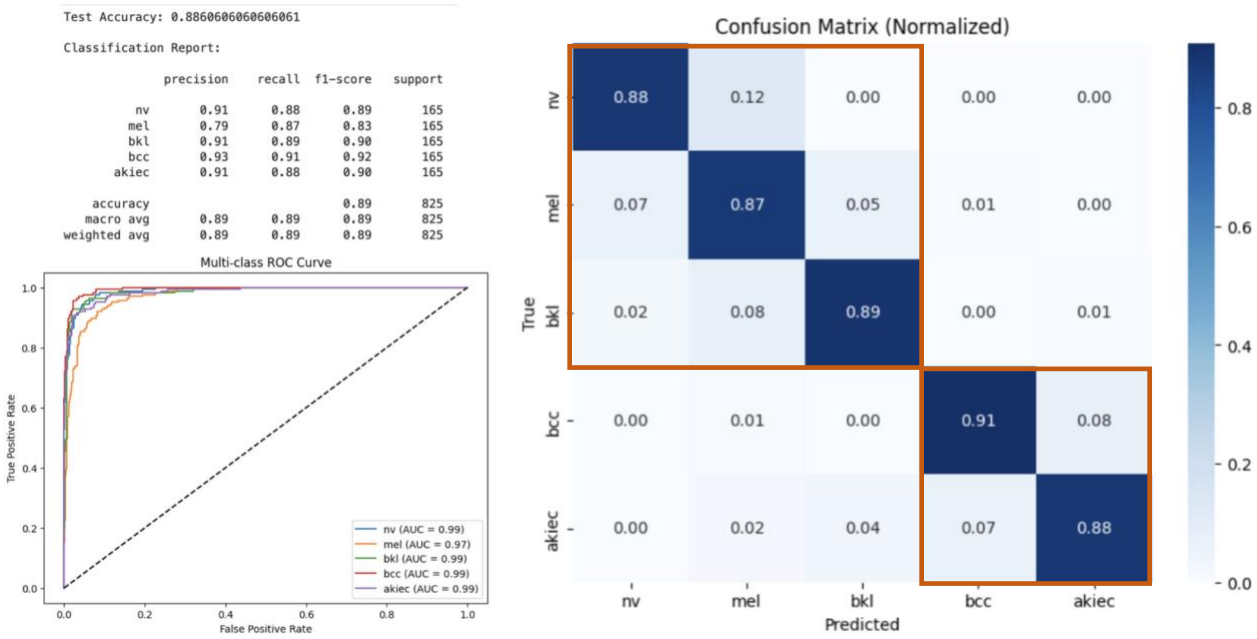


Figure 3.11: Multiclass classification results on the combined dataset (EfficientNetV2B0)

Author	Classification	Model Used	Methodology	Accuracy
Esteva (2017) [13]	Benign vs Malignant	Inception-v3	Transfer Learning	72.10%
Breno Krohling (2021) [14]	Benign vs Malignant	ResNet-50	Transfer Learning	85.00%
Kekal and Saputri (2023) [19]	Melanoma, Nevus, and Seborrheic Keratosis	MobileNet	Transfer Learning	88.00%
Mahmoud (2023) [15]	Melanoma vs Nevus	ResNet-50	Transfer Learning	92.98%
Mahmoud (2023) [15]	Melanoma vs Nevus	Custom CNN	Training from Scratch	90.85%
Our Proposed	**Benign vs Malignant**	**MobileNetV2**	**Transfer Learning**	**92.56%**
Valova [20]	8-Class	Inception-v3	Transfer Learning	83.49%
Our Proposed	**5-Class**	**EfficientNetV2B0**	**Transfer Learning**	**88.60%**

Table 2: Comparative analysis of accuracy from previously published papers

Here, Table 2 above presents the comparison of our proposed solution with the existing research papers. In most cases of binary classification, our proposed

solution performed better and was almost similar to the ResNet-50, despite being 5 times smaller in size. For Multiclass classification, author Valova achieved 83.49% of accuracy for an 8-class problem. In our solution, to obtain consistent performance over different classes on an imbalanced dataset, we narrowed down to a 5-class problem and achieved 88.60% accuracy with a much smaller model than Inception-v3. Where some of the above solutions need a server to perform classification, the smaller size of both of our models allowed us to implement on-device classification effectively for smartphones.

To deploy these models on mobile devices, we converted them to TensorFlow Lite format. TensorFlow Lite is a lightweight version of TensorFlow, which is optimized for on-device machine learning. We used the TFLiteConverter.from_keras_model() method to convert the model.

This conversion reduced the size of the model by almost 50%, as the MobileNetV2 Keras model, with the size of 18.54 MB, was reduced to 9.59 MB, and the EfficientNetV2B0, with 33.93 to 24.13 MB. Both models had no performance degradation after the conversion. With the default optimizer during conversion, we further reduced the model size to 2.69 MB and 6.77 MB for MobileNetV2 and EfficientNetV2B0 models, respectively. Although these models showed a little performance degradation of 0.58% for MobileNet and 1.45% for EfficientNetV2B0. As the chosen models were already smaller in size without optimization, we chose the models with no performance degradation for our application.

To mimic the different camera quality, we tested our converted models with low-quality images. As shown in Figure 3.9, we employed 3 strategies to generate low-quality images that mimic different conditions of smartphone-captured images. At first, we reduced the image size to (112 x 112), which is half the original input size (224 x 224) on which the model was trained (Figure 3.9 (b)). In JPEG quality reduction (c), we reduced the image quality by a compression ratio of 10/100. In the third strategy, we applied a blur effect with a Gaussian kernel of (5, 5).

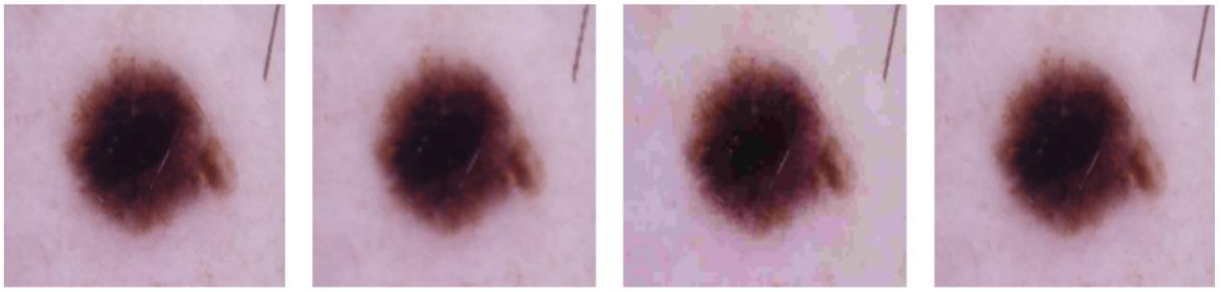


Figure 3.12: Strategies for generating low-quality images

As per the results of binary classification presented in Table 3 below, the model performed well on classifying cancer on different types of low-quality images, which shows the TensorFlow Lite Model's ability and robustness of the model against different camera quality and smartphone categories. On the lowest quality images, which were from JPEG compression, a 10 times reduction from the original quality, the model achieved 85% classification accuracy.

Image Quality	Accuracy	Benign Precision	Malignant Precision	Benign Recall	Malignant Recall
Original	92%	92%	93%	93%	92%
Resize (50% reduction)	87%	91%	83%	81%	92%
JPEG Compression (90% reduction)	85%	82%	89%	90%	80%
Blur	91%	92%	90%	90%	93%

Table 3: Binary classification results on different low-quality images

Table 4 shows the results of multiclass classification using the EfficientNetV2B0 TensorFlow Lite model on different image qualities. It showed a degradation of 22% in accuracy for severely compressed JPEG images, as the model achieved 67% of overall accuracy. Model on images with a blur effect and resized images (50% low quality) showed a reasonable accuracy of 82% and 70% respectively.

Image Quality	Accuracy	F1-Score NV	F1-Score MEL	F1-Score BKL	F1-Score BCC	F1-Score AKIEC
Original	89%	90%	84%	91%	92%	89%

Resize (50% reduction)	70%	76%	65%	54%	79%	71%
JPEG Compression (90% reduction)	67%	72%	57%	67%	67%	72%
Blur	82%	85%	78%	77%	89%	82%

Table 4: Multiclass classification results on different low-quality images

These benchmark experiments on different levels of low-quality images prove that models are not losing performance by a huge degree when image quality reduces.

4. Mobile Application Deployment

4.1. System Architecture

We aim at a fully offline skin lesion detection tool designed for smartphone (Android) devices. Figure 4.1 presents our system architecture, and the following are the steps of our system workflow,

- User launches our mobile application and captures an image using the mobile camera with a handheld dermoscopic attachment or imports from the gallery if having a standalone portable dermoscope.
- The captured image is then preprocessed on-device (Resized, normalized, and converted to a byte buffer) and passed to the TensorFlow Lite model for prediction.
- Based on the model's output, the application displays the result whether it's benign or malignant with a confidence score.
- In parallel, the model also uses a multiclass classification model to identify 5 different types of cancer, and the application shows the cancer type with a confidence score.

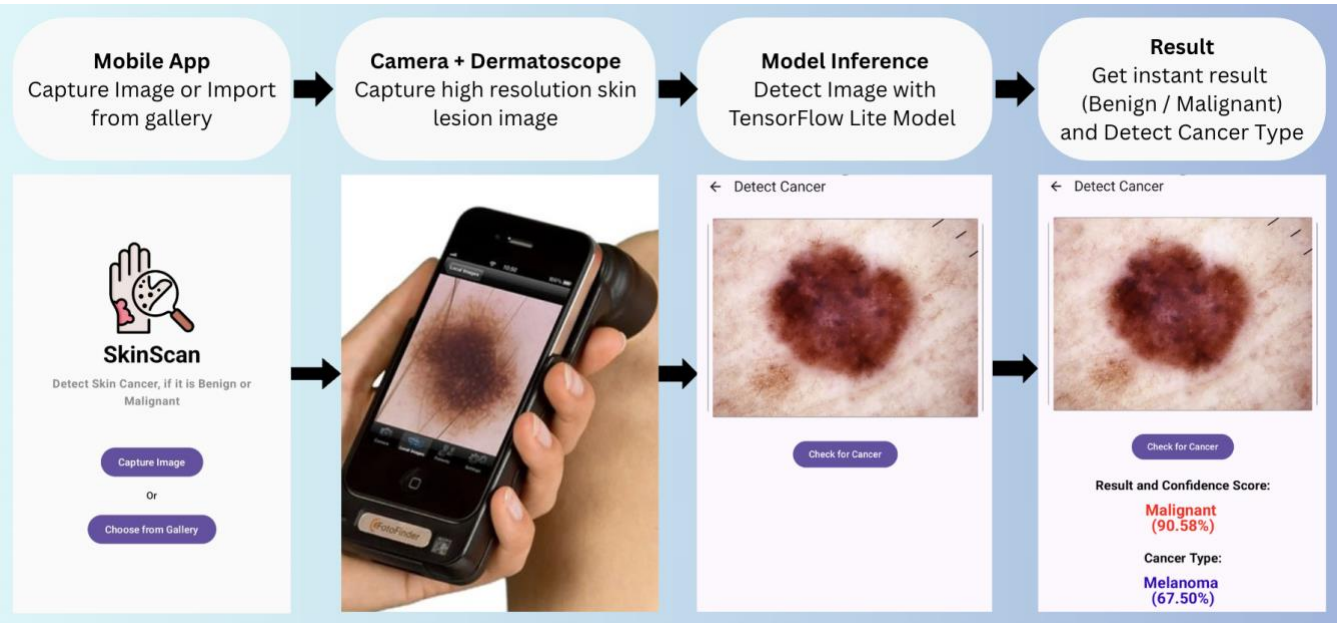


Figure 4.1: System architecture of skin cancer detection using a smartphone application and a dermoscopic attachment.

As discussed earlier in the literature review section, an important part of our system is the handheld dermoscope attachment. Figure 4.2 shows some examples of currently available dermoscopic attachments in the market with their prices.

Figure 4.2: Some of the available options for handheld dermoscopes with prices.

4.2. Features and Implementation

In this project, we develop an Android application named SkinScan to offer a user-friendly offline solution for skin cancer detection. This application is designed for mid-range devices.

The desired features of the application include:

- A fast and secure mobile application that can work efficiently on mid-range devices.
- Standard input interface to connect with different third-party handheld Dermoscopes.
- Scalable architecture, allowing for easy addition of new functionalities.
- Offline classification capabilities for the application to work without internet as a requirement.
- Easy to understand the flow of the application for non-technical users.

The following steps were taken to develop the application for Android:

Step 1: Create an Application *and Set Up User Interfaces (UIs)*

The first step was to create a new project from templates by clicking on File > New > New Project. From available templates for “Phone and Tablet” application development, choose an empty activity. In the next dialog, name the application (SkinScan) > Choose the minimum SDK version (the lowest Android version on which the application can be installed) > Finish. After successfully creating an empty application, we added the TensorFlow Lite library to the Gradle build file. The library was sourced from the Maven repository[21].

To create user interfaces, the modern Jetpack Compose framework was used, where we can define a block of UI as a Compose function in Kotlin files. Here, for the Home Screen (Figure 4.3), the Column block contained all the elements. The modifier attribute was used to modify the structure of the block. This included the text blocks for the name of the application and description, and two button components: the “Capture Image” button to start the camera for capturing an image, and the “Choose from Gallery” button to open the gallery for images. Android provides “ActivityResultsContracts”, a collection of standard activity call contracts. From these, the “TakePicturePreview()” method was used to open the capture camera image screen, and the “GetContent()” method was used to open the gallery.

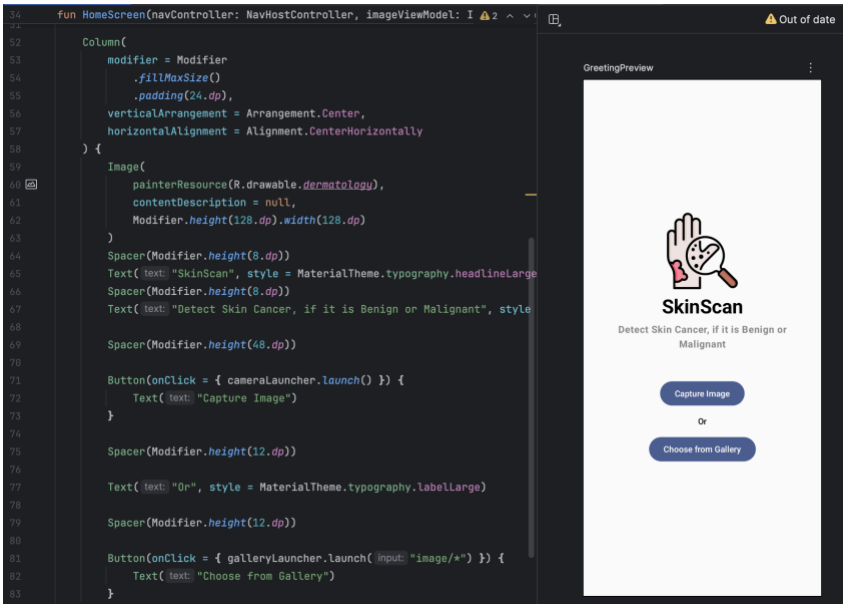


Figure 4.3: Build a simple user interface for the home page

Similarly, a second screen was developed that shows the captured or selected image, and a "Detect Cancer" button was added to run model inference on it.

Step 2: Add TensorFlow Lite Models

We created a folder named “assets” inside the “app/src/main” folder to put our models (Figure 4.4). The Assets folder is used to store raw and static files that can be accessed easily using the AssetManager Class. This folder allows us to package these files directly within the Android Application Package (APK), which means these models will be available upon application installation and no need to download them separately.

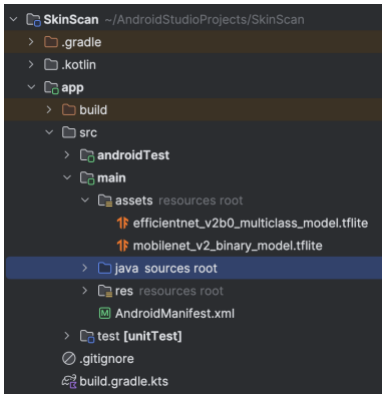


Figure 4.4: TensorFlow Lite models placement in the assets folder

Step 3: Image Processing for Model Input

According to the input requirements for both of our models (MobileNetV2 and EfficientV2B0), the image is resized to 224 x 224 Pixels in Height and width using the scale() method from Bitmap [23] (Figure 4.5). By default, this method applies bilinear interpolation. This matches the bilinear resizing used in the training pipeline, ensuring consistency between training and inference. A same-sized byte buffer was created to store the normalized value of each pixel, as the interpreter of the model requires input as a byte buffer. Here, value 1 represents batch size (single image), value 224 * 224 is the size of the buffer, value 3 is for the number of color channels, here red, green, and blue, and value 4 represents the size of each color channel (4 bytes = 32-bit).

```
val resized = bitmap.scale( width: 224,  height: 224)
val buffer = ByteBuffer.allocateDirect( capacity: 1 * 224 * 224 * 3 * 4).apply {
    order(ByteOrder.nativeOrder())
}
```

Figure 4.5: Image resize and byte buffer creation

For MobileNetV2, each pixel value was normalized between [-1, 1], and no normalization was applied for the input of EfficientNetV2B0, as per the model requirement. Each pixel value was then converted to a float value and placed into the byte buffer. (Figure 4.6)

```
for (y in 0 ≤  until  < 224) {
    for (x in 0 ≤  until  < 224) {
        val pixel = resized[x, y]
        if (isMbNet) {
            buffer.putFloat((Color.red(pixel) - 127.5f) / 127.5f)
            buffer.putFloat((Color.green(pixel) - 127.5f) / 127.5f)
            buffer.putFloat((Color.blue(pixel) - 127.5f) / 127.5f)
        } else {
            buffer.putFloat(Color.red(pixel).toFloat())
            buffer.putFloat(Color.green(pixel).toFloat())
            buffer.putFloat(Color.blue(pixel).toFloat())
        }
    }
}
```

Figure 4.6: Pixel value normalization based on the model

Step 4: Load Models and Run Interference

AssetManager gives read-only access to the files stored in it. Using the openFd() function, we created a file descriptor, FileInputStream, which reads the file, and uses this file descriptor as a parameter to open the file for reading. Function loadModelFile() (Figure 4.7) returns MappedByteBuffer, which is a byte buffer whose content is a memory-mapped region of the file (model). Passing this byte buffer (model) as a parameter to the interpreter, which is provided by the TensorFlow Lite library, creates an object of the interpreter, on which we can run model inference.

```
private fun loadModelFile(context: Context, modelPath: String): MappedByteBuffer {
    val fileDescriptor = context.assets.openFd(modelPath)
    val inputStream = FileInputStream(fileDescriptor.fileDescriptor)
    return inputStream.channel.map(
        FileChannel.MapMode.READ_ONLY, fileDescriptor.startOffset, fileDescriptor.declaredLength
    )
}
private val interpreterMbNet by lazy {
    val model = loadModelFile(application, modelPath: "mobilenet_v2_binary_model.tflite")
    Interpreter(model)
}
private val interpreterEffNet by lazy {
    val model = loadModelFile(application, modelPath: "efficientnet_v2b0_multiclass_model.tflite")
    Interpreter(model)
}
```

Figure 4.7: Load model and create interpreter

Here, Figure 4.8 shows the model inference with the MobileNetV2 model. The input was generated by the preprocessImage() function (*step 3*). For the output, we created an array of size 1 containing a float array of size 1 that can hold a single floating-point number, as required by the interpreter. Now we ran the inference using the interpreter.run() method, which runs the model and stores the predicted score in the output array. If the score is higher than 0.5, the input lesion is malignant, else benign.

```
val inputMbNet = preprocessImage(bitmap, isMbNet: true)
val outputMbNet = Array( size: 1) { FloatArray( size: 1) }

interpreterMbNet.run(inputMbNet, outputMbNet)

val score = outputMbNet[0][0]
val labelMbNet = if (score > 0.5f) "Malignant" else "Benign"
val confidenceMbNet = if (labelMbNet == "Malignant") score else (1 - score)
```

Figure 4.8: Model inference on MobileNetV2

With a similar approach, we ran inference on EfficientNetV2B0. The inner array size of the output was taken as 5, as the model returns five prediction scores for five classes. The maximum of these predictions was taken as the final predicted class, using maxByOrNull{} lambda function, which returns the index of the largest value in the array. We mapped this index to our 5 class names to show it on the result screen (Figure 4.9).

```
val inputEffNet = preprocessImage(bitmap,  isMbNet: false)
val outputEffNet = Array( size: 1) { FloatArray( size: 5) }
interpreterEffNet.run(inputEffNet, outputEffNet)
val probabilitiesEffNet = outputEffNet[0]

val predictedClass =
    probabilitiesEffNet.indices.maxByOrNull { probabilitiesEffNet[it] } ?: -1
val labelEffNet = when (predictedClass) {
    0 -> "Nevus"
    1 -> "Melanoma"
    2 -> "Benign Keratosis-Like lesions"
    3 -> "Basal Cell Carcinoma"
    4 -> "Actinic Keratosis"
    else -> "Unknown"
}
val confidenceEffNet = probabilitiesEffNet[predictedClass]
```

Figure 4.9: Model inference on EfficientNetV2B0

Step 5: Testing the Application on the Images from the Test Set

It is necessary to see if the application works as expected. We used a virtual emulator device (Pixel 8 Pro) for this testing. 20 images of each category were added to the application storage from the original test set for testing gallery input and model predictions. Figure 4.10 shows the results of the application on a randomly chosen image from 100 total test images. On-device inference performed similarly to the original Keras models, by making most of the correct predictions, with some degree of misclassification (Last screenshot of Figure 4.10).

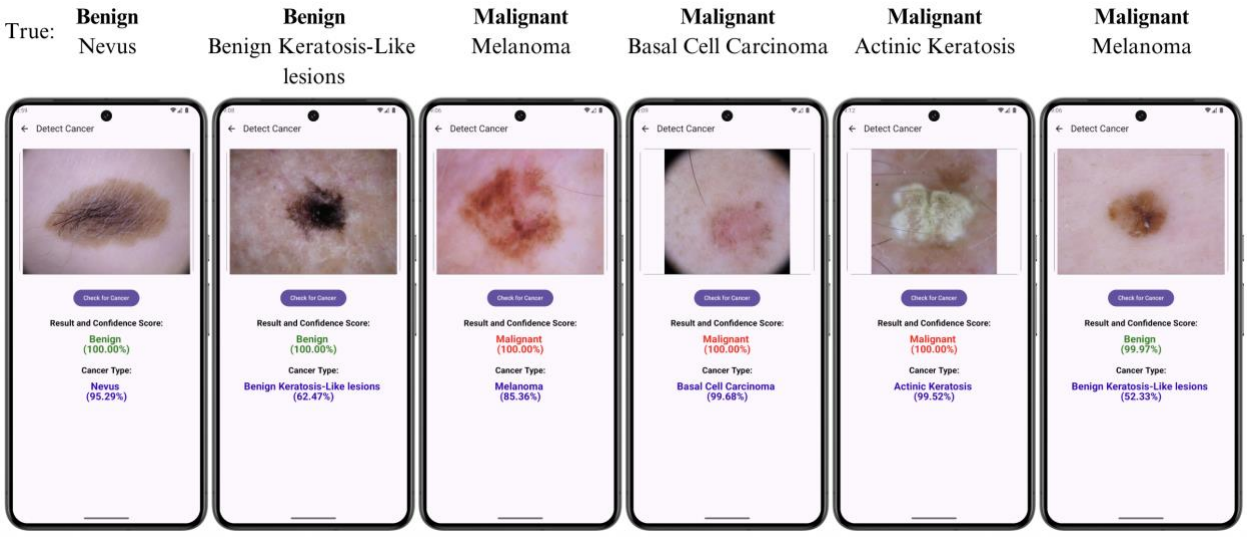


Figure 4.10: True labels vs application predictions with confidence score

While dealing with the mobile application, we also need to monitor resource usage. Figure 4.11 shows the RAM(Random Access Memory) usage graph on model inference for this application. Initially application uses 53 MB of RAM, and when we initiated model inference, it spiked to 200 to 260 MB of RAM usage. As modern Android devices often have 4 GB or more of RAM, 250 MB of usage can be considered a reasonable amount for a single application.

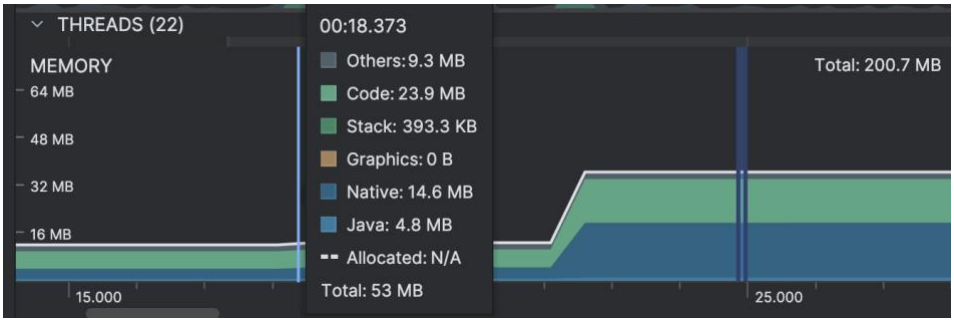


Figure 4.11: RAM usage statistics of the application using the profiler tool

Another considerable metric is how much time the application takes to make on-device predictions. After pressing the detect cancer button, our application shows prediction results in between 70 - 300 MS (milliseconds). For individual models, MobileNetV2 takes around 16 – 90 MS, and EfficientNetV2B0 takes 30 – 180 MS of time to predict skin lesions.

Step 6: Create a Signed APK to Deploy on the Device

Signed APK is the production application that can be shared with end users or can be uploaded to the Play Store as a signed APK bundle. This process obfuscates the application files to prevent reverse engineering and make it more secure for deployment. Also, it reduces the overall application size by not including the unused packages and libraries that may have been added during development.

To create APK/signed app bundle, we needed to follow certain steps, process is as follows: Build > Generate signed app bundle or apk > Create or choose from existing Key Store (This file contains application sign key, we can generate it inside the Android Studio and store it securely outside the application in computer, we need to provide password, and personal details about the author of

the application) > Next > Choose release option from debug or release versions > Finish. This creates the release version of the application with the .apk format.

Size of the APK: 55.9 MB, after installation on the device, the application takes 75 MB of storage space.

5. Conclusion and Future Work

Our smartphone-based skin cancer detection system is a lightweight, budget-friendly solution that enables remote and rural healthcare workers to detect skin cancer early and provide accurate referrals to dermatologists. Our smartphone-based system demonstrated the ability to bring deep learning powered diagnosis to low-resource settings. With this project, we have demonstrated the effectiveness of using transfer learning with lightweight pretrained models for skin cancer classification. Models like MobileNetV2 and EfficientNetV2B0, with transfer learning, significantly outperformed the custom CNN model, achieving 92.56% and 91.57% test accuracy, respectively, compared to 82.76% for the custom model.

Transfer learning was not only effective but also reduced training time in comparison to training the model from scratch, by achieving optimal performance in 50-70 epochs on average. Class balance using the combined dataset proved effective as it improved model recall and F1-scores for underrepresented classes and helped avoid model bias.

Future Work

- Train models on larger datasets such as ISIC 2020 (33,000+ samples) to improve generalization and address current dataset limitations.
- Extend multiclass classification from 5 classes to cover all 7 diagnostic categories so that the system can cover vascular lesions, squamous cell carcinoma, etc.
- Explore IOS deployment to reach Apple’s device users.
- Testing the mobile application for negative and unexpected scenarios and resolving issues that arise during testing.
- Adding more information about detected cancer for guidance, and providing features like getting the nearest dermatologist list and appointments for a full-fledged production-level application.

Bibliography

[1] Government of Canada. (2023). Skin Cancer. https://www.canada.ca/en/public-health/services/sun-safety/skin-cancer.html

[2] Canadian Skin Cancer Foundation. (n.d.). Skin Cancer Facts. https://www.canadianskincancerfoundation.com/skin-cancer/

[3] World Health Organization. (n.d.). *Skin cancer*. International Agency for Research on Cancer (IARC). https://www.iarc.who.int/cancer-type/skin-cancer/

[4] Skin Cancer Foundation. (2025). Skin Cancer Facts & Statistics. https://www.skincancer.org/skin-cancer-information/skin-cancer-facts/

[5] The Weather Network. (2024) News: Canada's Dermatologist shortage. https://www.theweathernetwork.com/en/news/lifestyle/health/canadas-dermatologist-shortage-may-lead-to-undiagnosed-skin-cancers

[6] Jahn, A. S., Navarini, A. A., Cerminara, S. E., Kostner, L., Huber, S. M., Kunz, M., Maul, J.-T., Dummer, R., Sommer, S., Neuner, A. D., Levesque, M. P., Cheng, P. F., & Maul, L. V. (2022). Over-detection of melanoma-suspect lesions by a CE-certified smartphone app: Performance in comparison to dermatologists, 2D and 3D convolutional neural networks in a prospective data set of 1204 pigmented skin lesions involving patients' perception. *Cancers*, *14*(15), 3829. https://pmc.ncbi.nlm.nih.gov/articles/PMC9367531/

[7] ISIC Challenge. ISIC Challenge. (n.d.). Welcome to the ISIC challenge. International Skin Imaging Collaboration. https://challenge.isic-archive.com/

[8] Tschandl, P., Rosendahl, C. & Kittler, H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions (2018). https://academictorrents.com/details/dc3188ee1ce7e2d2254113111b406c484101ba65

[9] Hernández-Pérez, C., Combalia, M., Podlipnik, S., Codella, N. C. F., Rotemberg, V., Halpern, A. C., Reiter, O., Carrera, C., Barreiro, A., Helba, B., Puig, S., Vilaplana, V., & Malvehy, J. (2024). BCN20000: Dermoscopic lesions in the wild. Scientific Data, 11, Article 641. https://www.nature.com/articles/s41597-024-03387-w

[10] Wikipedia contributors. (2024, July 28). ImageNet. In Wikipedia. https://en.wikipedia.org/wiki/ImageNet

[11] IBM. (2024). Transfer learning. IBM. https://www.ibm.com/think/topics/transfer-learning

[12] Hussain, Mahbub & Bird, Jordan & Resende Faria, Diego. (2018). A Study on CNN Transfer Learning for Image Classification. https://www.researchgate.net/publication/325803364_A_Study_on_CNN_Transfer_Learning_for_Image_Classification

[13] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*. https://www.nature.com/articles/nature21056

[14] Krohling, B., Castro, P. B. C., Pacheco, A. G. C., & Krohling, R. A. (2021). A smartphone-based application for skin cancer classification using deep learning with clinical images and lesion information. https://arxiv.org/pdf/2104.14353

[15] Mahmoud, H., Omer, O.A., Ragab, S., Esmaiel, H., Abdel-Nasser, M. (2024). Classifying melanoma in ISIC dermoscopic images using efficient Convolutional Neural Networks and deep transfer learning. Traitement du Signal, Vol. 41, No. 2, pp. 679-691. https://doi.org/10.18280/ts.410211

[16] TensorFlow. (2024). *Transfer learning and fine-tuning*. TensorFlow. https://www.tensorflow.org/tutorials/images/transfer_learning

[17] International Skin Imaging Collaboration (ISIC). (2019). ISIC 2019: Skin lesion analysis towards melanoma detection [Dataset]. ISIC Archive. https://challenge.isic-archive.com/data/#2019

[18] Codella, N. C. F., Gutman, D., Celebi, M. E., Helba, B., Marchetti, M. A., Dusza, S. W., ... Halpern, A. (2018). Skin lesion analysis toward melanoma detection: A challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), hosted by the International Skin Imaging Collaboration (ISIC) [Dataset]. arXiv preprint arXiv:1710.05006. https://arxiv.org/abs/1710.05006

[19] Kekal, Harming & Saputri, Daniati. (2023). Optimization of Melanoma Skin Cancer Detection with the Convolutional Neural Network. Journal Medical Informatics Technology. 23-28. 10.37034/medinftech.v1i1.5. https://medinftech.org/index.php/medinftech/article/view/10/12

[20] Valova, I., Dinh, P., & Gueorguieva, N. (2023). Mobile application for skin cancer classification using deep learning. https://jmids.avestia.com/2023/003.html

[21] TensorFlow Lite (2025). *tensorflow-lite* [Software artifact]. Maven Repository https://mvnrepository.com/artifact/org.tensorflow/tensorflow-lite

[22] TensorFlow. (n.d.). tf.image.resize. TensorFlow API documentation. https://www.tensorflow.org/api_docs/python/tf/image/resize

[23] Google. (n.d.). Bitmap.scale(int, int, boolean) [Android graphics API]. In Android Developers. https://developer.android.com/reference/kotlin/androidx/core/graphics/package-summary#(android.graphics.Bitmap).scale(kotlin.Int,kotlin.Int,kotlin.Boolean)