Applying deep learning to detect melanoma using mobile phone cameras

Degree project report in Computer Engineering

Tobias Alexanderson, Felix Fredin

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Degree project report 2024

Applying deep learning to detect melanoma using mobile phone cameras

Tobias Alexanderson
Felix Fredin

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Applying deep learning to detect melanoma using mobile phone cameras
Tobias Alexanderson
Felix Fredin

Supervisor: Phil Harrison, Chalmers University of Technology
Supervisor: Rasmus Standar, Knightec
Supervisor: Isak Holmdahl, Knightec
Examiner: Jonas Duregård, Department of Computer Science and Engineering

Degree project report 2024
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Applying deep learning to detect melanoma using mobile phone cameras
Tobias Alexanderson
Felix Fredin
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Malignant melanoma, a severe form of skin cancer has seen an increase in incidence, emphasising the need for early detection and diagnosis. This thesis investigates the potential of convolutional neural networks (CNNs) in classifying moles as malignant or benign using images captured from mobile phone cameras. The primary goals were to assess the performance of these models and to develop a mobile application that utilizes the trained models to provide preliminary assessement of potentially cancerous moles.

The ISIC archive was used as a dataset, with over 75 000 images of moles labeled either malignant or benign, to train and validate the models. Through extensive hyperparameter optimization using the Optuna framework, in addition to fine tuning techniques, the EfficientNetV2B3 model achieved a validation accuracy of 86.73%.

A mobile application was developed to integrate the trained model, allowing users to capture images of moles and recieve real time classification results. The application's usability varied due to mixed conditions when taking images, highlighting the importance of optimal conditions. The results indicate that in the right conditions, the mobile application might be able to effectively classify images, serving as a useful tool for early melanoma detection.

This thesis concludes that deep learning models, combined with mobile technology, hold significant promise in improving skin cancer diagnosis in the future.

Acknowledgements

We would like to express our sincere gratitude to Phil Harrison, our supervisor at Chalmers, for providing us invaluable technical advice and support throughout the process.

We also extend our thanks to Isak Holmdahl and Rasmus Standar at Dewire, Knightec, for their administrative support, including organizing scrum sessions and assisting us with the report. Their guidance and support ensured smooth progress of our project.

Special thanks also goes to Dr Dan Fagrell, for providing invaluable images of moles suspected to be cancerous for us to test the application on.

Lastly, we would like to thank our colleagues at Dewire for making this journey enjoyable. Your support and collaboration made the entire process productive while also fun.

Tobias Alexanderson, Felix Fredin, Gothenburg, June 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ADAM	Adaptive Moment Estimation
ANN	Artificial Neural Network
AWS	Amazon Web Services
BN	Batch Normalization
CNN	Convolutional Neural Network
EC2	Amazon Elastic Compute Cloud
EFS	Elastic File System
FN	False Negative
FP	False Positive
GPU	Graphics Processing Unit
Grad-CAM	Gradient-weighted Class Activation Mapping
IDE	Integrated Development Environment
ILSVRC2012	Imagenet Large Scale Visual Recognition Challenge 2012
MAE	Mean Absolute Error
MSE	Mean Squared Error
ReLU	Rectified linear unit activation function
SGD	Stochastic Gradient Descent
SMBO	Sequential Model Based Optimization
TL	Transfer learning
TN	True Negative
TP	True Positive
TPU	Tensor Processing Unit

Contents

List of Figures

List of Tables

+

1

Introduction

1.1 Background

Malignant melanoma is a severe form of skin cancer and the number of cases in Sweden has doubled since the start of the 21st Century [2]. This form of cancer usually develops from moles and common symptoms include increasing size, rashing and changes in colour [2]. Most melanoma cases can be cured, however around 15-20 percent of the cases are incurable and can prove fatal. One of the most important factors that determines the outcome is the time until treatment. If discovered before the cancer can reach the lymph nodes, the estimated survival rate is 99% [3]. Thus, the earlier the malignant melanoma is detected and classified, the better the odds are of curing it.

The use of artificial intelligence in image recognition has been on the rise in recent years, and is today used in many different fields. One of those fields is medicine, e.g. in classifying diseases. This project aims to investigate the possibility of using deep neural networks to classify moles as either malignant or benign using a mobile phone application to reduce the time for diagnosis and to relieve the healthcare system.

In this project, a high recall factor is crucial. Recall, often referred to as sensitivity in the diagnostic contexts, measures model's ability to identify all relevant instances correctly. In this case, this translates to accurately identifying as many malignant moles as possible. A high recall factor ensures that fewer cases of potential melanoma are missed, which is paramount to early intervention and treatment. This is because false negatives (i.e. missing the disease when it is present) can lead to worse consequences than false positives (i.e. creating false alarms for disease presence). Although ideally the rate of both false negatives and false positives should be as low as possible.

1.2 Purpose and Goals

The main purpose of this project is to investigate the possibility of lessening the strain on healthcare by providing an additional filter in the form of a preliminary diagnostic tool early on in healthcare. By enabling individuals to assess if they need

medical treatment or not, the amount of time and money spent on unnecessary clinical visits may be reduced.

Simplifying the initial step towards a medical diagnosis could enhance early detection of malignant melanoma. If the preliminary assessment was as straightforward as taking a photograph, more individuals might be inclined to check their moles, potentially leading to earlier identification and treatment, which could save lives by addressing the condition before it progresses.

The project's goal is to develop an application that allows users to photograph a mole and determine if it is benign or malignant. This determination will be facilitated by a model trained on a large dataset of images depicting both benign and malignant melanomas.

1.3 Research questions

The following questions are to be answered in the project:

- What performance is achievable using deep learning models in classifying malignant and benign moles, and how can its accuracy, recall, and precision be further optimized?
- How effective can a mobile camera be in capturing and classifying images of moles, and what factors influence its usability and accuracy in a real-world setting?

1.4 Limitations / Demarcations

The primary objective of the application is to serve as a proof of concept rather than to be integrated into the medical field or released to the public. As such, minimal effort will be directed towards refining the app's interface or user experience. The main goal is to validate the functionality of the underlying model by using it to process real world data. Therefore, as long as the application can effectively capture images, communicate with the model, and accurately display results, it will have fulfilled its intended purpose.

Due to limitations to available computational resources, the longest consecutive training or testing time available on the remote GPU is 24 hours. This will impact the effectiveness of hyper-parameter optimization negatively.

Due to the nature of this project and it not being performed or checked by medical professionals, it is of utmost importance that any recommendations from the app or model are not taken as medical advice.

2

Theory

2.1 Relevant research

The use of image classification to identify skin cancer is becoming increasingly relevant in the medical field. This is suggested by the large amount of peer reviewed papers (>2000), which have been published on the matter since last year. A selection of similar works will be summarized briefly in the upcoming section.

Detection of malignant melanomas using neural networks
A similar project was conducted in 2022 by Simon Länsberg & Anna Manfredsson who investigated the possibility of using neural networks to classify malignant melanoma [25]. After establishing a balanced subset of the ISIC-dataset and trained multiple models on it, they achieved the highest test accuracy, recall and precision of 73%, 63.78% and 78.6% respectively. They also created a mobile application which communicated with the trained model through a server using firebase. Their project validates the possibility of a model to server to mobile application framework, as well as using deep neural networks for classifying melanoma.

Skin Cancer Detection Using Deep Learning—A Review
Maryam Naqvi et al. investigate various models and their performances across different datasets, and also reviews similar skin lesion classification projects [43]. Their findings suggests that an increase in dataset size could increase models' generalization capability. Furthermore, most datasets consists of images where the skin color is white which may lead the model to be biased towards other skin colors.

Skin Lesion Analysis and Cancer Detection Based on Machine/Deep Learning Techniques: A Comprehensive Survey
Mehwish Zafar et al. published a survey [44] in which different approaches to skin cancer detection using AI are compared. The authors describe the frequent problem of noise present in images, which negatively affects models' performances. Noise can range from certain markings, hair, and rulers to border irregularities and poor contrast. Furthermore, the vast majority of images in existing datasets are of light-skinned persons. More dark-skinned samples are necessary to increase accuracy of cancer detection systems. Lastly, based on the reviewed approaches, the authors suggests that CNN-models' are the most effective for this task.

2.2 Malignant Melanoma

Malignant melanoma is a type of skin cancer that originates from melanocytes, cells responsible for producing melanin in the skin [4]. Melanomas may stand out from other moles and therefore may be detectable. Common characteristics include:

- The mole is asymmetrical and of uneven shape.
- The borders of the mole tend to be jagged, as opposed to normal moles having smooth edges.
- Multiple shades and uneven colours being present, often of darker nature.
- The diameter of the mole exceeds 6mm, however smaller moles could be malignant as well.
- The mole may change shape, size or colour over time.

Discerning between a regular mole and a malignant mole may not be straightforward even if there are a multitude of symptoms present.

2.3 Performance of trained doctors

There are several steps in diagnosing malignant melanoma, with the first one being a skin exam to check for any inconsistencies among birthmarks and moles [27]. If the moles look suspicious, they are further examined using microscopes to check for cancerous cells.

The diagnostic accuracy for doctors with more than ten years of experience working with skin cancer was measured to be 80% [26]. This study was conducted over 25 years ago, so that number may have changed, but it is still far from 100% which is what should be striven for. There were less experienced registrars who also had their diagnostic accuracy measured, Their experience ranged from one to five years, and their accuracy ranged between 56% and 62%. Worth noting is that for all of the tested groups the specificity was higher than the sensitivity. Specificity and sensitivity can be compared to precision and recall, and in an application like this, sensitivity/recall is arguably more important than the former. In Table 2.1 these performance metrics are visualized. Recall and precision are explained further in section 2.6.

Category	**10 Years of experience**	**3-5 Years of experience**	**1-2 Years of experience**
Diagnostic accuracy (%)	80	62	56
Specificity (%)	99	97	98
Sensitivity (%)	91	90	79
Positive predictive value (%)	86	67	67

Table 2.1: Comparison of diagnostic performance metrics across different levels of experience.

2.4 Deep learning

The following section provide insight into the relevant theory about deep learning.

2.4.1 Neural networks

Neural networks diverge from conventional programming by not executing explicitly defined tasks. Instead, they learn autonomously by analyzing data, mimicking the inner workings of the human brain [30]. Neural networks consist of a vast interconnected network of neurons which are organized in different layers. A neuron has the ability to receive incoming data and generate output signals to subsequent neurons. The grade of influence of which one neuron affects another is called weight. Weights are positive or negative numerical numbers which are multiplied with the input signal and dictating the strength of a bond between neurons. Higher numbers equal a stronger relation, therefore increasing the chances of the interconnected neurons triggering. If the weight is zero, it will not activate subsequent neurons. Thus, a neuron does not always transfer data. An activation function is assessed to more precisely dictate what signals should be passed on through the network [31, 48]. This phenomena is similar to how threshold potential works for synapses in the brain.

To make a neural network usable it must be trained for a certain task or environment. This process can be referred to as the learning process. The learning process typically involves training under supervised learning conditions, where each instance of data is labeled. The network learns by iterating through the labeled data and adjusting its internal weights to minimize discrepancies between its predictions and the actual labels. The goal is to enforce connections which lead to more accurate predictions and weaken those that do not.

To adjust the weight within the network the difference between the prediction and the label must be quantified. This is achieved through various forms of loss functions tailored to the task. Subsequently, a mechanism called back-propagation calculates the gradient of the loss with respect to the weight of a connection. This gradient is calculated throughout the entire network, hence starting off at the output layer is crucial as subsequent gradients can be computed by propagating errors backward through the network. This effectively reduces the computational power needed to optimize the model. Each gradient indicate how much each weight contributes to the overall error in the network's predictions. Lastly the weights are updated through an algorithm called gradient descent, which moves the weights towards a direction that minimizes the total error [36, 48].

2.4.2 Convolutional neural network

A problem with fully connected neural networks where all neurons connect to every neuron in subsequent layers, is that they do not consider the spatial structure of the image. Thus, if the objects are located at different positions across images, a fully connected network may encounter challenges in accurately classifying them

[5]. Fully connected neural networks are for these reasons not suitable for image classification. CNNs, on the other hand, can handle these issues. They work by considering different spatial features at different hierarchical levels, from simple features such as edges and blobs, to more complex features formed by combining these simple features. Multiple layers of convolution and pooling (detailed below) are used to extract the features from the data and fully connected layers at the end of the network are then used for classification [49, 7].

The convolution layers are responsible for most of the computing. The objective of these layers is to scan the input image for features. To detect features, a matrix consisting of learned weights is applied to scan part of the image. This matrix is more commonly known as a filter or a feature detector. The filter calculates the dot-product between the images' pixel values and the matrix and stores the calculated value in an output array called a feature map. The higher the similarity between the image and the filter, the larger the dot-product will be. Lastly, the feature detector is moved to scan a new portion of the image. This process is repeated until the feature detector has swept across the entire image [7, 53]. The result is an array consisting of values describing the similarity between the image and a predefined feature. To achieve non-linearity a rectified linear unit function (ReLu) is applied to the output array. This transforms all negative values to 0, but maintains positive values as they were before [7, 53].

To reduce computational complexity a pooling layer is added to each feature map generated by the convolution layer. The pooling layer works similarly to the convolutional layer in the sense that it scans part of the input to generate a new condensed output. What sets them apart is that this filter does not consist of any weights, it only aggregates the input and stores it in a new down-sampled output array. There are multiple ways of aggregating, but the most common is max-pooling. Max-pooling stores the largest value of each scan in the new array [49, 7].

Fully connected layers are used to connect every neuron from the final flattened convolutional or pooling layer to the output layer. These layers integrate the features learned by the convolutional layers and produce the final class predictions, thus completing the model.

2.4.3 Hyperparameters

When training a model, there are certain settings that can be adjusted to influence the training process, consequently affecting the model's performance. These settings are called **hyperparameters**. Certain hyperparameters might be more impactful than others, which becomes apparent during the training process when using tools like Optuna (see section 2.7). Common hyperparameters to adjust include (but is not limited to):

Epochs The number of epochs describes how many iterations of the dataset that the model will be exposed to. One epoch is the equivalent to going through the entire dataset used for training once. After an epoch is complete, the model adjusts

internal weights accordingly. The more epochs a model is trained for, the better it will acclimatize to the training data. However this might introduce the risk of overfitting, which is why the technique early stopping is usually a good idea to include while training the mode. This is explained further in Section 2.4.4.

Batch size The batch size dictates the number of samples, such as images, processed within each back-propagation iteration. Increasing the batch size means gradients are calculated based on a larger sample, potentially enhancing the model's ability to generalize and perform better on unseen data. Bigger batch size also leads to faster training due to fewer updates to the weights. However, too large batch sizes severely affects the memory usage of processor or GPU. Furthermore, Ibrahem Kandel & Mauro Castelli suggests in their paper [39] that too large batch sizes may lead to worse performance, and when fine-tuning a model it is generally better to use a smaller batch size along with a lower learning rate.

Batch normalization When models undergo training there are changes in distribution of each layer's inputs as preceding layers change. This phenomena is called internal covariate shift, and can cause a model to adjust its parameters more slowly to keep up with all the changes. This entails lower learning rates, as higher learning rates leads to instability and difficulties in convergence. To allow for higher learning rates the technique batch normalization was introduced. BN addresses internal covariate shift by normalizing the inputs to a layer using a subset of the training set, also known as a mini-batch. This improves training speed and efficiency while at the same time working as a regularizer [37].

Learning rate Learning rate is the hyperparameter that influences how fast the model can adjust its internal parameters. After each epoch or batch, the model proposes changes to its weights. The learning rate then specifies how to handle these updates to weights. A learning rate of 0.001 will scale down the adjustments to a thousandth of the proposed magnitude.

Learning rate can be crucial in the sense that a learning rate too high might cause the model to overshoot the optimal parameters, which could lead to it never converging. A lower learning rate makes the updates of the weights more cautious, but will make the model take longer to train and converge at the best internal weights. A learning rate schedule can be implemented to start off with a relatively high learning rate to approach better weights than what was initialized. After a set amount of epochs, the learning rate is then reduced to make sure the weights can converge [40].

Dropout Layers Dropout layers works by randomly deactivating nodes within the neural network during training. By doing this the network cannot reliably form specific connections and therefore the frequency of co-adaptations is also reduced. Dropout layers has been shown to reduce overfitting as well as increasing the models performance in multiple different neural network tasks [54]. To adjust the frequency of dropout layers, the **dropout rate** is adjusted.

Regularizers
To decrease models tendency of overfitting, regularizers can be used. The two most

prevalent ones are called l1 and l2, they both work by adding penalties to the loss function. What separates l1 to l2 is how this is achieved. The l1 regularizer works by penalizing the loss function (see 2.4.3.1) proportional the absolute value of the models weights. This leads to sparse weight vectors, meaning some of them become zero which can be beneficial when dealing with high dimensional data. L2 works by penalizing the loss proportional to the square value of the models weights, which unlike l1, leads to the desired outcome of reducing complexity without inducing sparsity [47].

2.4.3.1 Loss functions

Loss function is the hyperparameter responsible for determining the error of the predictions. It takes the models predictions and compares them to the actual labels, and then in some way calculates how far off the predictions were. Some regular loss functions include MSE, MAE, and binary cross-entropy [41].

MSE calculates the average of the squared errors between the predicted values ($\hat{y}_i$) and the actual values (y_i), and is calculated using Equation 2.1.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.1}$$

Where n is the number of samples.

MAE calculates the average absolute errors between the predicted values ($\hat{y}_i$) and the actual values (y_i), and is calculated using Equation 2.2.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{2.2}$$

Where n is the number of samples.

Binary cross-entropy or log-loss is a loss function which measures the difference of the models' probabilistic prediction and the correct label. It penalizes models for confident incorrect answers, but at the same time, when the model makes a good prediction which is correct the loss approaches zero. Binary cross-entropy is a common loss function in binary classification tasks. Equation 2.3 shows how to calculate the log-loss.

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{2.3}$$

Where:

- $L(y, \hat{y})$ is the binary logarithmic loss.
- N is the total number of data points.

- y_i is the true binary label (0 or 1) for the i-th data point.
- $\hat{y}_i$ is the predicted probability that the i-th data point belongs to the positive class (i.e., class 1).

2.4.3.2 Optimizers

Optimizers aim to either minimize or maximize the objective function, depending on which task is at hand. In this projects case, it is to minimize the loss which is quantified by the loss function. To minimize the loss, a technique called gradient descent is used. It works by calculating the gradient of the loss function with respect to the model's weights. From this, the direction of the highest rate of change is attained. The gradient will point directly uphill, which is why it's negated to instead travel downwards and minimize the loss [36]. Based on the gradient, the model's parameters are adjusted to improve performance. The degree of which the parameters are adjusted is dependent on the learning rate. What differs from different optimizers are adjustments on how the gradient is calculated. The traditional gradient descent computes the gradient based on the entire training dataset, and is calculated using Equation 2.4.

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}(w \cdot x_i - y_i)^2 \tag{2.4}$$

Where:

- $L(w)$ is the loss function, which quantifies the discrepancy between the model's predictions and the actual target values (labels).
- w represents the model parameters that the optimization algorithm seeks to adjust in order to minimize the loss function.
- N is the total number of data points in the dataset.
- x_i represents the feature vector of the i-th data point.
- y_i represents the true output (label) associated with the i-th data point.

This results in slow computations since the gradient will depend on the entire training set. In contrast, **SGD** addresses this inefficiency by iteratively computing the gradient based on small subsets of the training data. By randomly sampling mini-batches from the dataset, SGD achieves faster updates and can handle large datasets more efficiently. The SGD update rule is expressed as in Equation 2.5.

$$w_{t+1} = w_t - \eta \nabla L(w_t) \tag{2.5}$$

Where:

- w_t represents the model parameters at time step t.
- η is the learning rate.
- $\nabla L(w_t)$ is the gradient of the loss function with respect to the parameters at time step t.

This iterative approach enables SGD to navigate through the parameter space more efficiently, although it introduces stochasticity and noise into the optimization process.

Like SGD, the optimizer **ADAM** aims to minimize the loss function by iteratively updating the model parameters. However, it adapts the learning rates for each parameter individually based on estimates of first and second moments of the gradients. Fundamentally, Adam maintains two moving averages: the first moment (the mean) and the second moment (the uncentered variance) of the gradients. These moving averages are computed using exponential decay rates, typically denoted as β_1 and β_2, which control the decay rates of these moving averages. Adams adaptive learning rates and momentum allows it to converge faster and more reliably than traditional SGD, especially for problems with sparse gradients or noisy data [51].

2.4.3.3 Activation functions

Activation functions help determine the output of neurons in the neural network by deciding if a neuron should fire or not [34]. Being able to choose when a neuron activates essentially adds non-linearity to the model. There are several use cases for different activation functions, like removing any negative data, or normalizing everything into a specific range. Some common activation functions are the following:

ReLU: This function works by returning zero for any negative input and itself for positive inputs. Equation 2.6 illustrates the mathematical formula.

$$\text{ReLU}(x) = \max(0, x) \tag{2.6}$$

The **Sigmoid** function keeps the value between 0 and 1, thus achieving a probability score, by compressing the values into the range. The function is shown in Equation 2.7.

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

Tanh function works similarly to sigmoid, but it normalizes the values into the range of -1 to 1 instead. The function shown in Equation 2.8.

$$\text{Tanh}(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.8}$$

2.4.4 Overfitting

During the training phase of neural networks, neurons establish complex relationships among themselves to capture patterns and features from the input data. However, as models grow in complexity, these relationships can become too intricate, leading to what is called co-adaptations [33]. Co-adaptations does not necessarily capture underlying patterns in the data, but instead memorize specific instances or features which are present in the training set. This way, the model becomes increasingly tailored towards the training data at the cost of its ability to generalize to unseen data [46]. This phenomena is also known as overfitting, and is a very common problem when training deep neural networks.

Overfitting is easily detected by analysing the models performance over a set of training epochs. Typically, both the validation accuracy and training accuracy increase with each epoch. A clear indication that overfitting is starting to occur is when training accuracy keeps increasing while at the same time validation accuracy is not improving or even decreasing. This divergence reveals that the model is acclimatizing to the training data and becoming worse at generalizing, as seen in Figure 2.1.

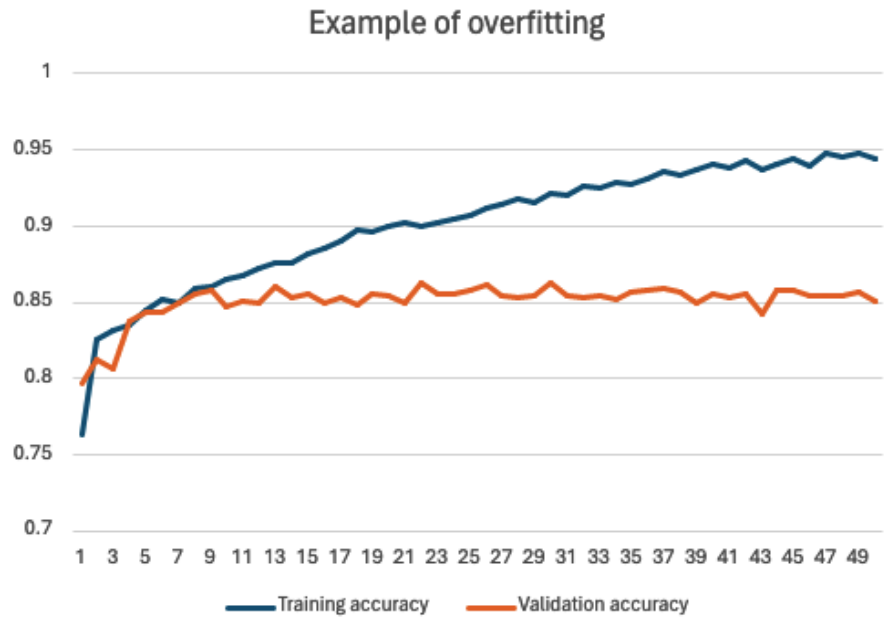


Figure 2.1: Example of overfitting occurring

One strategy to mitigate overfitting is to use early stopping [46]. Early stopping terminates the training process when the validation accuracy ceases to improve over a set amount of epochs. The model then reverts to the iteration with weights that had the best validation accuracy or lowest loss, which can be chosen based on the application. This approach aims to have the model retain its generalization capability by making sure it does not adapt too much to the training data.

More strategies include the addition of regularizers and dropout layers which are

described in 2.4.3, as well as increasing the variety and size of the training data (by means such as data-augmentation), described in 2.5.3.

2.4.5 Transfer learning

Transfer learning builds on the principle that knowledge from one field can be applied to other similar fields. This phenomenon finds most use in image recognition tasks for deep neural networks. The reason for this is that the initial stages of the learning process for a model involve acquiring low-level features. These features could include recognizing edges, colors, or light intensity in the image. All of these features are relevant for almost all applications of image recognition, whether it involves classifying humans, cars, or distinguishing between malignant and benign moles. Therefore, by importing a model that has already undergone extensive training on a diverse set of images, it can then be utilized and further trained on a more specific set of images according to the user's needs [42].

2.5 Data preprocessing

There is a common saying in computer science: "Garbage in, garbage out", meaning that if you train a model with subpar data, the model itself will be subpar. Making the data as good as possible before training a model is imperative to the models success. This in turn means that adjusting the data through preprocessing is crucial to attain a well performing model [29]. To accomplish this there are multiple techniques which can be implemented; some of which will be discussed below.

2.5.1 Data cleaning

Data cleaning involves the process of removing data points with missing entries, large outliers or other inconsistencies that could skew the model's learning. Removing poor quality images or those with missing metadata ensures the integrity of the dataset as a whole.

2.5.2 Balancing the dataset

Striving for a balance between the number of entries across all classes is imperative to draw any conclusions based on the provided accuracy from the model. If only 5% of all instances in the dataset are labeled as positive, then the model could have a 95% accuracy by just predicting negative. However, if there are an equal amount of entries for both classes the model cannot attain a high accuracy this way. To better get an idea of how the model performs there are other metrics than accuracy which can display if the model is weighted towards a specific class, these are explained in Section 2.6. Section 2.5.3 explains how to combat an unbalanced dataset.

2.5.3 Data augmentation

There are essentially two options available in the case of an unbalanced dataset. The simplest one is to use under-sampling, which entails removing data points from the dominant class until there are an equal amount of instances of both classes. There is a drawback to this method, as decreasing the size of the dataset will negatively impact the models ability to generalize, thus also decreasing its performance on unseen data [32].

The other approach is to inflate the dataset with more instances of the less prominent class. This is done through data augmentation. There are multiple different techniques used to augment data. For images, some common techniques include mirroring, rotating, zooming and contrast adjustments [32]. By applying these different alterations to an image, the dataset can be inflated with new entries with the class of the entry known. The non dominant class can be infused with more instances via augmentation to match the size of the dominant class. This option is favored when dealing with smaller datasets, due to the fact that a dataset that is too small might not be effective when training a model, since it will have a hard time generalizing to new data.

2.5.4 Train-test split

Splitting the original data into training-, validation-, and testing-sets is crucial for effectively training a model, as each set serves a distinct purpose. Initially, the dataset is divided into training and testing sets to ensure the testing data remains completely unseen during the training process. This precaution helps prevent the model from simply memorizing the test data. Subsequently, the training set is further divided to create a validation set, which is used during the training phase to evaluate the model's performance and adjust weights accordingly.

The training set is used to train the model, allowing it to learn patterns and relationships within the data. The validation set, on the other hand, plays a crucial role in tuning the model's weights and provides an unbiased evaluation of a model's performance during the training phase. Unlike the training set, it is not directly used to learn the data, but to gauge the model's generalization capabilities to perform on unseen data.

Lastly, when the model is trained, it is tested on the unseen test data. This provides an impartial evaluation on how well the model performs in a generalized situation [55].

2.6 Evaluation matrics

When measuring the performance of a model, accuracy is often used. Other common metrics of performance include recall, precision and F1 score. To calculate these, one needs to know the amount of instances of TP, TN, FP, FN for the models predictions.

A TP indicates a *correct* prediction, such as the model accurately idenfitying a malignant mole. Conversely, a FP denotes an incorrect *positive* prediction, where the model predicts a mole to be malignant, but it is actually benign. The same principles apply to TN and FN; a TN represents a correct *negative* prediction, and an FN indicates an incorrect *negative* prediction, where the model fails to identify a malignant mole.

The accuracy only measures the ratio of correct predictions to incorrect predictions. The accuracy is calculated using Equation 2.9.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{True Negatives} + \text{False Negatives}} \tag{2.9}$$

To get a more nuanced view of the performance of the model, one could use recall and precision. These metrics can be important when a dataset is imbalanced. If there was a ratio of 1:20 positive to negative labeled data in the dataset being used, the model could achieve a 95% accuracy by just predicting negative for every single point of data, simply by many of them being correct by default due to the imbalanced dataset. Recall and precision combat this by taking into the account the individual instances of FN and FP respectively.

The recall and precision are calculated using Equations 2.10 & 2.11.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.10}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.11}$$

Another scoring metric that can be used is the F1 score, which provides a balanced metric for the combined score of accuracy, recall, and precision. Equation 2.12 shows how to calculate the F1 score.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.12}$$

The F1 score values both recall and precision equally. It could however be helpful in some applications to weigh one over the other. For classifying moles as either malignant or benign, recall needs to be weighed more than the precision. This is because false negatives (i.e. classifying a cancerous mole as benign) can lead to worse consequences than false positives (i.e. creating false alarms for cancer presence in healthy moles)

To weigh recall over the precision, the F-beta score could be used. This is calculated using Equation 2.13.

$$\text{F-beta Score} = (1 + \beta^2) \times \frac{\text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}} \tag{2.13}$$

When β is higher, the recall is weighed higher than the precision. For example, setting $\beta = 2$ weighs recall twice as much as precision when calculating the F-beta score.

2.7 Hyperparameter Optimization

To achieve the highest possible performance out of a model or neural network, a lot of time must be spent on tuning the hyperparameters. The optimal values of these parameters varies depending on the task at hand. Therefore, finding the best combination may require rigorous testing. Since there are a multitude of parameters, trying all different combinations usually takes a vast amount of time. Additionally, some parameters may not impact the outcome in a meaningful way, making it a waste of time to investigate the tuning of these.

To tackle this problem different forms of hyperparameter optimization are used. The most basic form is called grid search. Grid search works by going through every possible comibination of hyperparameters within a defined search space [18]. The drawback of this method is the time complexity. If five different parameters were to be tuned all with six different values, there would be $6^5 = 7776$ different combinations of parameters to try. Training a single model can be time and resource heavy, therefore utilizing grid search within a larger model with many different parameters is not suitable.

Another approach to tuning hyperparameters is random search. It starts off by declaring search spaces for the parameters, and then randomly picking parameters to try. The advantages of this is that it is more efficient in the sense that more hyperparameters can be tried in a shorter amount of time. In grid search, they are tested iteratively, whilst in random search they are tested at random. Random search tends to be more efficient as soon as at least two or three hyperparameters are included [18].

2.7.1 Optuna

An alternative to hyperparameter tuning is to use Optuna, a software that aims to efficiently find the best hyperparameters within a given search space [35]. A search space is given for the parameters to be tuned, which Optuna then investigates by trying different combinations of these. This is done by training models with the parameters specified, and their given ranges. It is not as simple as doing a grid search on the specified parameters. Instead, it uses a more sophisticated approach, consisting of Bayesian optimization and SMBO [11, 12]. Another benefit to using Optuna is that it can use what is known as *pruning*. When trying a set of hyperparameters that do not seem to be performing well, Optuna can prune (terminate) the training

instance and begin a new one with other parameters. This greatly effectivize the search for good hyperparameters, as very little time is wasted on bad configurations. The configurations that are tested are referred to as trials. One trial is a training iteration with a set of hyperparameters. Based on the amount of trials Optuna is allowed, it will test different sets of hyperparameters in the given search space for a set amount of epochs, to determine which parameters are the best.

2.8 Models

When using transfer learning, a base model is needed. A variety of different base models can be used and may offer different advantages and performance due to differing architecture and complexity. The base models chosen for this project are all trained on the ImageNet dataset [15]. Having them all pretrained on the same dataset enables comparing of the models in a relevant manner. If they were not pretrained on the same data, their performances could vary due to the training data, which is not interesting to investigate in this project.

2.8.1 EfficientNetV2

EfficientNetV2 is a collection of small and fast models [17]. Depending on the version chosen, the parameters, performance, and resource consumption varies.

Table 2.2 describes the amount of parameters and performance on ImageNet that each of the versions have. Models marked with (21k) are trained on a different subset of ImageNet containing 13M images, compared to the ones not marked that are trained on the ILSVRC2012 subset, containing about 1.28M images. After training, the models were fine tuned using ILSVRC2012. The training was performed with a batch size of 4096 for 350 epochs [17].

Model	Top-1 Acc.	Params
EfficientNetV2-S	83.9%	22M
EfficientNetV2-S (21k)	84.9%	22M
EfficientNetV2-M	85.1%	54M
EfficientNetV2-M (21k)	86.2%	54M
EfficientNetV2-L	85.7%	120M
EfficientNetV2-L (21k)	86.8%	120M
EfficientNetV2-XL (21k)	87.3%	208M

Table 2.2: EfficientNetV2 models comparison.

Smaller, scaled down versions of EfficientNetV2 were also tested on ImageNet. These were trained with a batch size of 128, and are illustrated in Table 2.3.

Model	Top-1 Acc.	Params
EfficientNetV2-B0	78.7%	7.4M
EfficientNetV2-B1	79.8%	8.1M
EfficientNetV2-B3	82.1%	14M
EfficientNetV2-S	83.6%	24M

Table 2.3: EfficientNetV2 scaled down models comparison.

2.8.2 ResNet

ResNet was introduced in 2015 and featured residual connections as a solution to deeper networks being harder to train. This is achieved with the use of residual connections [21]. ResNet exists in different versions with differing amount of parameters. Table 2.4 below is a table describing their number of parameters as well as their performance on ImageNet [20].

Model	Top-1 Accuracy	Parameters
ResNet50	74.9%	25.6M
ResNet50V2	76.0%	25.6M
ResNet101	76.4%	44.7M
ResNet101V2	77.2%	44.7M
ResNet152	76.6%	60.4M
ResNet152V2	78.0%	60.4M

Table 2.4: Comparison of ResNet models.

2.8.3 DenseNet

DenseNet (introduced in 2016) is another group of CNNs that have been pre-trained on Imagenet. There are several advantages of using DenseNet or models with similar architecture, namely that they dampen the problem of vanishing gradients as well as contain less parameters than regular models [19]. In Table 2.5 the different iterations of DenseNet can be visualised [20].

Model	Top-1 Acc.	Params
DenseNet121	75%	8.1M
DenseNet169	76.2%	14.3M
DenseNet201	77.3%	20.2M

Table 2.5: DenseNet models comparison.

2.9 Ensemble learning

Ensemble learning is the art of combining several models to make a single prediction. This could be beneficial if a model is particularly bad at something, because then it can be compensated for by other models used for the same task and prediction.

There are multiple ways of generating a single prediction from multiple models. If the models are not combined and used as standalone models, then a voting strategy is suitable. Majority voting is taking the prediction that most models predicted. Weighted majority voting is similar, just that some models can be weighted heavier, thereby allowing them more votes. This could be used if a model performs better than its counterparts in some aspect[16].

Yet another technique is called stacking. Stacking works by combining the output from multiple models and feeding it to a fully connected layer before the prediction[45].

2.10 Software & development environments

The following softwares and frameworks were used to develop the application and model.

2.10.1 Tensorflow

Tensorflow is an open source machine learning library used for training neural networks. It allows seamless importing of models which can be used with transfer learning. By transforming the inputs to the model into tensors (a multidimensional numerical matrix) the models can work efficiently. Tensorflow enables the use of GPU's, as well as TPU's to speed up training [23].

2.10.2 Android Studio

Android Studio is the official IDE for Android app development. It enables the use of virtual devices to simulate the app running without an actual phone. Pairing a physical phone to test the application is also doable in Android Studio [13]. There exists two main programming languages for app development within Android Studio, Java and Kotlin. The code responsible for the applications backend is written with either of these two languages, but the visual and frontend part is done with XML. This can be done by either writing regular XML or using Android Studios visual features. Instead of defining an object and its position XML, it can be dragged and dropped allowing for seamless developing.

2.10.3 AWS

AWS is a cloud platform and offers many possibilities within cloud computing, including hosting a server. AWS Lambda has functionality to work as a trigger, meaning it will react when an event happens. This enables the server to run responses to code upon receiving an instruction, without having to sit idle. Other options with AWS include S3 buckets for storage potential, making the storing of large models possible. Furthermore, an elastic file system mounted on an EC2, a virtual server instance, can be used to integrate large libraries which the code in the lambda function may rely upon [56].

2.10.4 Grad-cam

Gradient-weighted Class Activation Mapping, grad-cam, works by utilizing gradient information from the last convolutional layer to compute the weights importance for each neuron related to a specific class. The last convolution layer is used, since it retains the most high level features. By combining the weights with the feature maps and applying ReLU, Grad-CAM generates a class-discriminative localization map, highlighting regions in the image crucial for the model's decision [50]. This process provides insight into the model's decision-making process, aiding in interpretation and understanding of its predictions.

3

Methodology

3.1 Data handling

The ISIC archive [9] is an open source archive containing over 75000 images of moles. These images are suitable for training AI models due to them containing relevant labels that can be used for predicting whether a mole is malignant or not. These labels range from the age of the subject to the type of lesion they exhibit, and if it is benign or malignant. Only the label "benign or malignant" was used for this project.

3.1.1 Downloading the data

The dataset was downloaded using ISIC's own API. The entire dataset was downloaded at once, instead of filtering images to download. This allowed for possible adjustments along the line.

3.1.2 Data preprocessing

Using the entire dataset would result in unbalanced data. With 88% of the images classified as benign the model could achieve an 88% accuracy by only predicting benign. A balanced dataset is more desirable when training models. To achieve this, there were two possible options; augment the data by infusing the dataset with more variations of the images classified as malignant via data augmentation, or simply remove a number of the images classified as benign, also known as under-sampling. The first approach would yield a dataset containing over 140k images if it was to be balanced. Training a model with a dataset this big would not be feasible with the limited time and resourced available for the project. Under-sampling was instead opted for, removing benign images until there was a balanced split. This resulted in a dataset containing around 20k images with a more balanced split of benign and malignant which is shown in Figure 3.1.

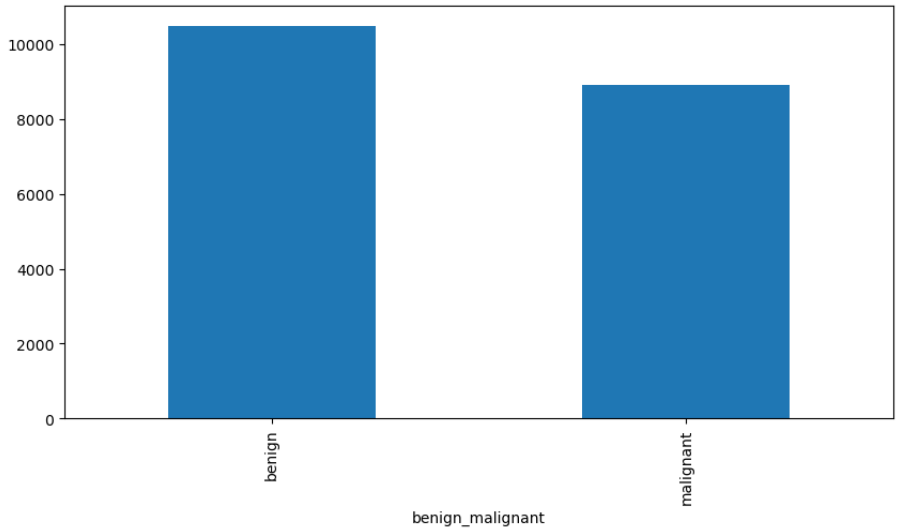


Figure 3.1: Benign (54.1%) & Malignant (45.9%) split

Another important aspect of the dataset is the age split of subjects. One potential reason why is that the model might develop a bias if it is trained only on younger people with few or no wrinkles. This could prove problematic once the model is exposed to older people with wrinkles, as it will not know if that is a sign of cancer or not. The age split of the split dataset was examined and deemed fitting, as it covered a wide variety of ages in a somewhat balanced manner. Figure 3.2 shows the arrangement of ages across the dataset.

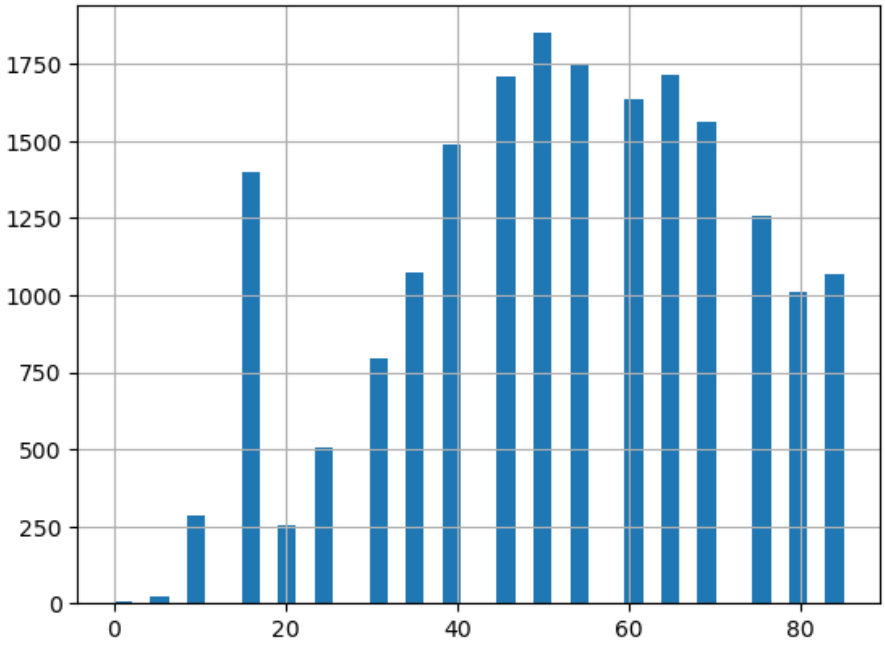


Figure 3.2: Age split among subjects in the dataset

The next step of preprocessing was to split the data into training, validation and

testing sets. The final dataset included a split of 20% testing data, 16% validation data, and 64% training data. A dataloader was employed to manage the large volumes of images efficiently. When images were passed through the dataloader, they were rescaled to a fitting size as well as divided into batches. Efficientnetv2 models accept image various sizes within the interval of 128x128 to 380x380 [17]. In this project an image size of 224x224 was settled for to strike a balance between performance and memory usage.

3.2 Training

The preprocessing of the images was verified prior to training a heavy and complex model. This step ensured that everything was in the right format and location. This was done by running the first transfer learning model for a limited number of epochs, to avoid wasting time on a lengthy training session in case something was not working.

After confirming that everything was working in order, longer training runs could start to be explored without risking having wasted time if the training failed. EfficientnetV2B3 was the first model to be tested. At first this was done on a local machine which took considerable time. Each epoch would take upwards of 12 minutes which would not be feasible in the long run.

3.2.1 Remote access

Running the training process on a local machine proved time-consuming for the EfficientnetV2B3 model. This was a problem since the number of epochs to converge was yet unknown, therefore, at least one prolonged test had to be carried out to find the appropriate amounts of epochs to run in the future. Doing this when each epoch takes 12 minutes could take a considerable amount of time, which was preferably avoided. This problem was solved by accessing one of Chalmers' remote desktops. The desktop in question is equipped with an RTX-4090 GPU, which was substantially faster than the local machines used.

Initially, remote accessing was only possible when connected to Chalmers' internet. To work around this, a connection to Chalmers VPN was established, which enabled remote accessing from outside Chalmers vicinity. After cloning the git repository and transferring all the required images to the remote machine, remote training utilizing the GPU could begin. This proved to be an effective approach, as training time for each epoch was reduced more than five-fold, going from around 12 minutes per epoch to aproximately 2 minutes per epoch.

3.2.2 Hyperparameter tuning

In an ideal scenario with unlimited time and resources, a grid search would be performed for all models trained. While it would perform an exhaustive search of all different configurations of the hyperparameters it would be inefficient due to the

nature of grid search. Another approach could be random search, a technique with potential to be faster while yielding similar results to grid search [18]. However, it boasts the same problems as grid search, where it might train very similar models without differing performances. There was not enough time available for the project to risk running obsolete combinations of parameters. Another potential issue with random search is that it is just as the name implies - random. One might never find the optimal values of parameters due to it being random. Instead, the sofware Optuna was utilized, which helped identify the most impactful parameters, and what value of each hyperparameter resulted in the best performance.

To further tune the model, an important aspect was to identify when the starts to converge. By doing this, an idea of how long upcoming experiments needs to be trained can be attained. To do this, a run which totaled 50 epochs was executed. As suspected, the model's performance started to decline and fluctuate already after around 15 epochs. Later in the project, this method was substituted for a callback method called early stopping, which automatically stopped the training process when there are no further performance gains. Doing this allowed for more unique runs, and testing of more fine-tuning techniques.

One change which proved to beneficial was adding an extra dense layer to the model's head. By doing this, the model were able to form more complex relations. However, adding even more dense layer seemed to decrease the performance. There seemed to be a fine line in keeping the model complex enough to capture different relationships, and making the model too complex.

The final technique was alternating how much of the pretrained model's weights were to be remained frozen during training. Originally, the pretrained part of the model remained entirely frozen, resulting in only the model's head being changed. This was later changed to dividing training into two parts. In the first part of training, only the weights of the model's head was changeable. Then in the second part of training, the entire model was trainable. The idea was to first make the model adapt to classifying moles, and then let the model do larger changes to hopefully increase its performance. The division of training in two parts was achieved by fitting the model to the training data twice, without re-compiling the model in between.

Multiple different runs were tested to find how many layers to unfreeze. In addition, it was found that decreasing the learning rate when unfreezing was the optimal strategy. This is mainly due to the pretrained parts of the model already have been adjusted to finding patterns, blobs or features, and by changing its weights too much, this ability can be changed towards the worse.

3.3 Using the model cloud based

An important decision along the way was whether to evaluate images with the trained model locally on the phone with the app, or instead have it on a cloud based server. Tensorflow as a library takes up a lot of space, and is required to run the

trained model. In addition to that, larger models could potentially take a considerable amount of time to run locally on a phone. This made it more suitable to run the model cloud based, as then performance and storage were not as restrained.

An alternative could have been to use Tensorflow Lite, a smaller condensed version of Tensorflow which enables it to run better on mobile devices. This was however not opted for, to guarantee that the trained model was not condensed and kept its performance.

3.4 AWS

AWS Lambda is a service for executing code without the need of provisioning or managing a dedicated server [38]. When Lambda is triggered, it will execute the code and then return chosen data to the device which invoked it. This fit the projects needs perfectly, as the general idea was to send an image from a mobile device for evaluation and receive a result back. Another coveted feature is the payment system. AWS Lambda charges based off the running time of each invocation, instead of a continuous price which is usually the case when hosting a server.

Code can be written directly in Lambda layers, or be uploaded using a zip file. Although syntax corrections are not included when coding directly into Lambda layers, it was the preferred alternative over the two. There are multiple ways of triggering the function and one way is to send a request to the function URL. In this case, it was a HTTP POST request, as the request can then also contain the image which the model should evaluate. By recommendation from our supervisors, this method was altered to instead make use of an API Gateway. The API gateway serves as a control mechanism, allowing only specific traffic to trigger the function, thereby increasing the security of the service. With the Lambda function set up together with the API Gateway, progress proceeded fluently. An image could be sent from a mobile phone, the phone would then get a confirmation that the transaction was complete and the image was also saved into a web-storage-service called S3 bucket.

The S3 buckets intended use is not to store images, but rather to store the trained models. Saving images to the S3 bucket was more of a proof of concept, as it is a similar process to load to the bucket and load from the bucket.

The first challenge appeared when it came to uploading a model and use the model to evaluate image. To load a model, the Tensorflow library is required. However, Tensorflow cannot be installed directly onto the Lambda function due its size. The solution was to download Tensorflow on an EFS which was mounted on an EC2 instance. The EFS can then be accessed from the Lambda server, solving the library problem. The drawback of this method is that EC2 is a server that runs continuously and therefore Lambdas benefit of being cheap was essentially nullified. Luckily, this wasn't a problem for Knightec that provided the budget.

Accessing the EFS from the Lambda function was problematic at first, but was eventually solved by correctly adjusting the filepaths. Another problem had to do with permissions, which was bypassed after some trial and error.

One final challenge revolved around the execution time. From invoking the function to receiving a result more than 90 seconds passed. This long wait time was not desirable, thus ways of optimizing the code was investigated. The major time consumers were importing the libraries and downloading the model. A quicker execution time was noticed after two sequential runs, and it was because the Lambda function cold starts once and is then kept warmed up for a certain amount of time. A warmed up function only executes code inside the main method, meaning that the models did not have to be loaded in once the function was warmed up. All imports were located outside of this method, but in a warmed up state the imports content were still accessible. Thus, for warmed up runs, the only noticeable delay was caused by downloading the model from the S3 bucket. By moving this section of code outside of the main function, the execution time for a warmed up function went down to only a matter of seconds. Although a cold start still takes considerable amount of time, this solution was deemed adequate for the scope of this project.

3.5 App development

The first choice in app development was whether to make the app for Android, IOS, or both. The decision was made to only make an app for Android using Android Studio. Android studio makes syncing a physical phone easy, enabling app development where progress can be seen in real time by testing the app on physical devices which further solidified the choice of IDE.

To start off, the shell of the app was developed. This shell was just a basic layout with some functioning buttons to navigate different pages in the app. The next step was to implement camera functionality. Permissions were required o use the phone camera. This means that every time a new user wants to use the app, they have to give the app access to the camera.

This early shell proved good enough for testing. The next step was to connect a trained model to a cloud based server, and enabling the app to send images to the server, and then have it receive a response back.

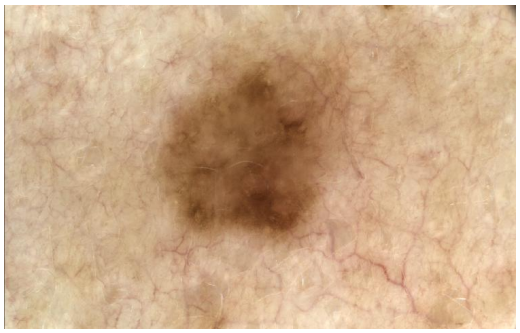
Once sending messages back and forth to the server worked, focus shifted to refining the app. Refinements included; an indication of how long it could take to get a response, processing the result based on what came back from the server, displaying the result in a meaningful way, and adding further recommendations on how to proceed based on the result.

3.6 Testing the app

Once a prototype model was up and running on the server, testing became possible. By using the app to evaluate moles known to be benign, the model's capabilities using a mobile phone camera could be tested.

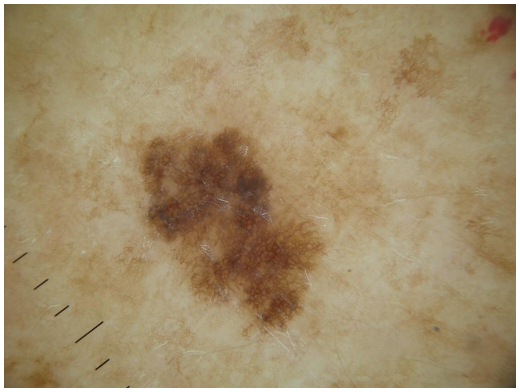
Effort was made to have the images taken with the app replicate the images from the original dataset that was used to train the model. Images depicting both benign and malignant moles taken from both the dataset and mobile phone cameras, are found in Figure 3.3.

Special care was taken to ensure that images captured with mobile phone cameras closely matched those in the dataset. There was, however, some difficulty in achieving the same image quality that was found in the dataset. The main challenge was getting as much zoom as possible without losing focus of the mole. With some effort, the images resembled the training data at least to some extent. Images similar to this were used to evaluate the app.

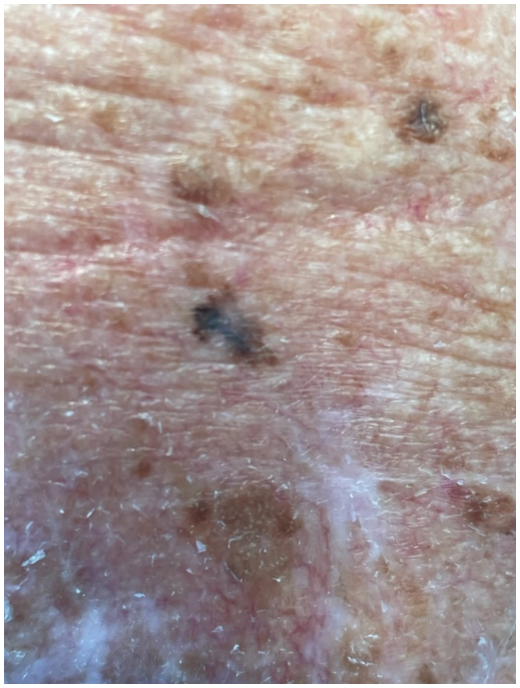


(a) Image of a benign mole from the dataset

(b) Image of a benign mole taken with a mobile phone camera



(c) Image of a malignant mole from the dataset

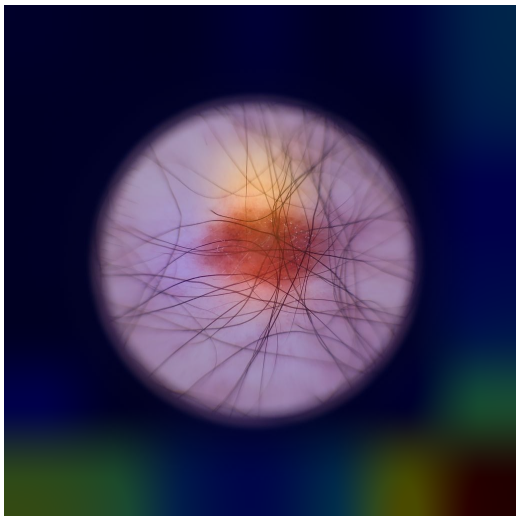


(d) Image of a malignant mole taken with a mobile phone camera

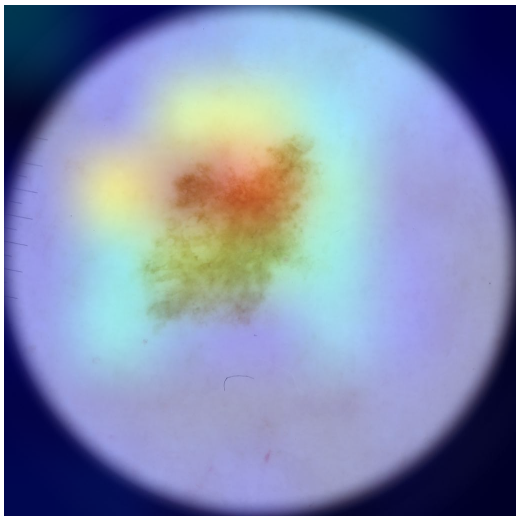
Figure 3.3: Examples of benign and malignant moles from the dataset and taken with a mobile phone camera.

3.6.1 Model evaluation through heat map analysis

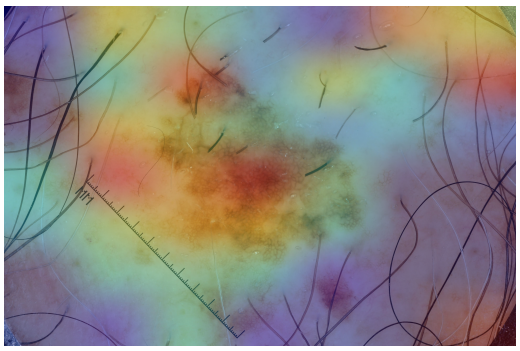
To check that the model was working as intended grad-cam was utilized. Grad-cam works by highlighting regions of an image which the model finds particularly interesting during classification. The concern was that the model did not actually look at the moles, and had learned to cheat by looking at other things in the image, such as hairs or markers next to moles. The dataset contained multiple images of moles with some sort of marker right next to the mole. These markers were rather distinct and it is possible that the model learned to pick up on these markers to make its predictions. Below in Figure 3.4 these markers and scales can be seen.



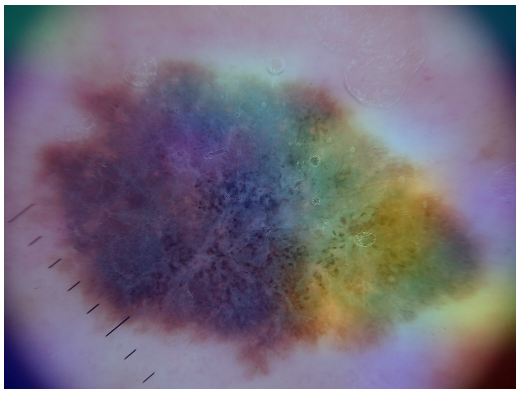
(a) Benign mole with marker present



(b) Benign mole with marker present



(c) Malignant mole with marker present



(d) Malignant mole with marker present

Figure 3.4: Images of moles from the dataset containing markers

4

Results

The following chapter will present the results of the project.

4.1 Baseline model development

The results of trained models does not say much without having something to compare it to. Therefore a dummy classifier that only predicts the most dominant class was used as a baseline. Due to the used dataset being quite balanced, the accuracy of the dummy classifier was close to 50%. The split of benign and malignant labels in the dataset can be seen in Figure 3.1. Anything above this in a trained model means it is better than the dummy classifier, and it is getting *something* right.

4.2 Models

After establishing a clear baseline and ensuring the feasibility of training models, the next step was to test transfer-learned based models. Three different models were tested, not including different variants.

4.2.1 EfficientnetV2

This chapter covers the iterative adjustments and improvements made to the EfficientnetV2B3 model. Lastly the tuned B3-model will be compared to other tuned versions of EfficientnetV2, namely the S and L versions.

Baseline
Table 4.2 contains the parameters for the first recorded run. The base model has weights which have been pre-trained on ImageNet which remained frozen throughout this training process. The trainable parameters were limited to the model's head, described in Table 4.1.

Parameter	EfficientnetV2B3
Base model	Pretrained weights
Status	Frozen
Model head	GlobalAveragePooling2D()
	DenseLayer(1, sigmoid)

Table 4.1: Model architecture

Parameter	EfficientnetV2B3
Optimizer	Adam
Learning Rate	0.001
Batch size	64
Activation function	ReLU
Epochs	50

Table 4.2: Parameters and results from first run of EfficentNetV2B3

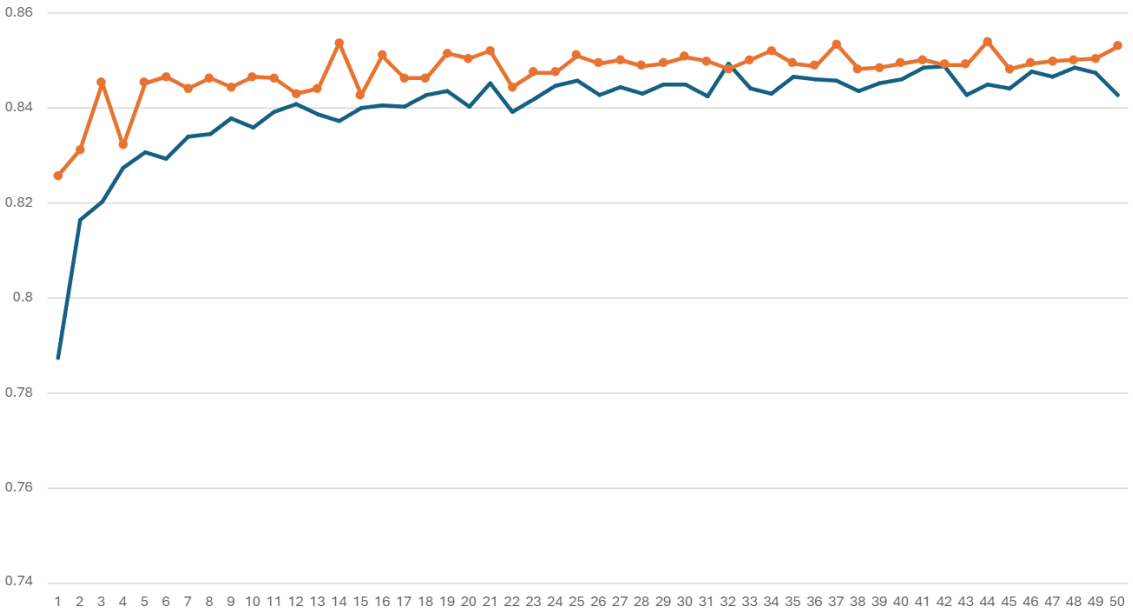


Figure 4.1: Performance over epochs of first run with EfficientnetV2B3. Orange is validation accuracy, blue is training accuracy.

Class	Precision	Recall	F1-score	F-Beta score	Accuracy
Benign	0.85	0.85	0.85	0.85	0.84
Malignant	0.82	0.83	0.83	0.83	0.84

Table 4.3: Accuracy report first run of EfficientnetV2B3

First run of Optuna
After establishing a baseline on how a model with default parameters performs, the next step was to adjust the model's hyperparameters with the goal of increasing it's performance. To accomplish this, the Optuna framework was utilized. The first run of Optuna consisted of 50 trials, where each trial ran for 5 epochs. Table 4.4 displays the optimal parameters suggested by Optuna.

Parameter	**Optimal value suggested**
Activation Function	Tanh
Batch Size	256
Batch Normalization	True
Learning Rate	0.000715576
Optimizer Type	Adam
Dropout Rate	0.223860214
Highest measured val acc	0.849548995

Table 4.4: First run of Optuna

The next step was to increase the number of epochs, to make sure the model converged during training.

Before and after Optuna
Figure 4.2 displays a graph with three runs. The first run used the same parameters as the baseline, the second run used the parameters suggested by Optuna. The third run was a slightly modified version of the second run, the difference being that an extra dense layer was added in the head of the model and the activation function was swapped from Tanh to ReLU. As the graph suggests, there is a slight improvement to validation accuracy with each experiment, which is also why the second dense layer and the activation function ReLU was to be kept in further experiments.

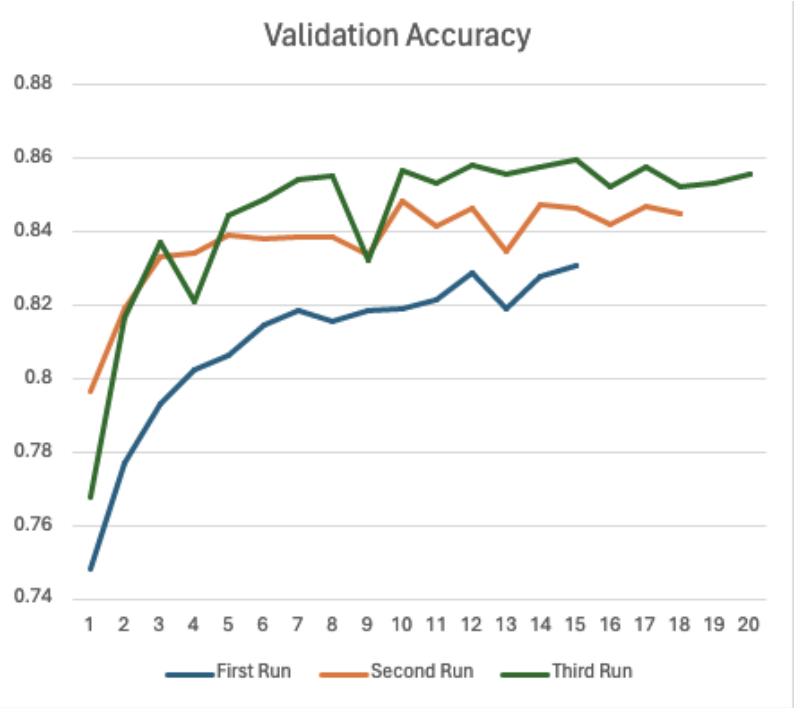


Figure 4.2: The validation accuracy across epochs for three different runs

Unfreezing base model

Previously the base model have remained frozen during the entire training process. Moving forwards, the base model is frozen for the first couple of epochs only. This technique allows the entire model to fine tune itself, possibly increasing the performance. Figure 4.3 displays two runs, in which the first run the base model was unfrozen without any reduction to the learning rate. As can be seen, this results in a fluctuating line. In the second run, the learning rate is also decreased by a factor of ten when unfreezing the other layers. This yields a more stable line, due to the lower learning rate causing smaller updates in weights. In both runs, the entire base model was unfrozen, a parameter which will be altered in upcoming experiments.

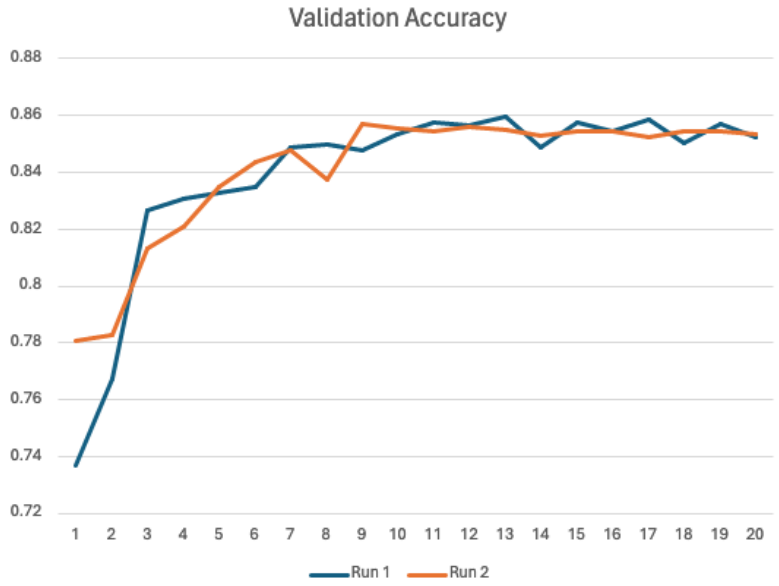


Figure 4.3: Comparing two runs where the base model is unfrozen after 10 epochs

Second run of Optuna

For the second run of Optuna the number of trials was kept at 50, however, each trial was increased to 12 epochs in addition to more parameters being tested. This study took longer than expected and was interrupted after 38 trials due to limitations in the GPU usage. Optuna's suggested optimal values were savable, and can be seen in Table 4.5.

Parameter	Optimal value suggested
Batch Size	64
Batch Normalization	True
Learning Rate	0.001424
Optimizer Type	Adam
Second Dense layer	False
Dropout Rate	0.34
Dropout Rate2	0.13
Regularizer	None
Highest Measured Val Acc	0.8608

Table 4.5: Best parameters from Optuna second run

Testing the suggested parameters

Testing the optimal parameters suggested by Optuna yields another small increase in the performance of the model. Figure 4.4 shows the differences between the previously best run found in 4.2. A new highest validation accuracy of 86% was achieved.

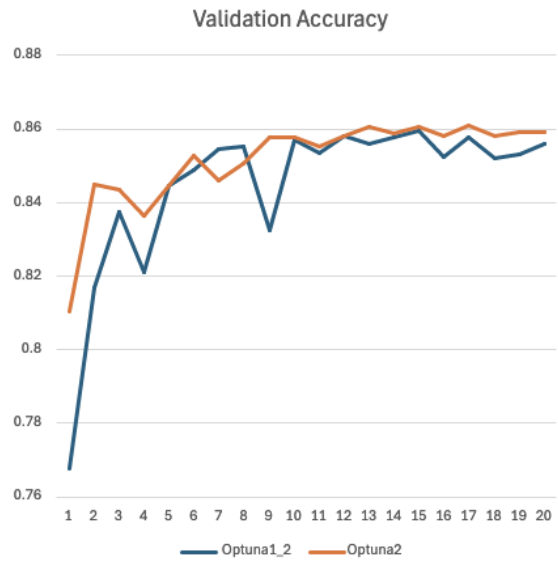


Figure 4.4: Validation Accuracy

Third run of Optuna

In this study the number of epochs for each trials was kept at 12, but the number of trials were decreased to 30. Another parameter was investigated, namely the number of units within a dense layer. Furthermore, a pruning technique was added which automatically terminated trials which did not exceed 80% validation accuracy within 5 epochs. The results are presented in table 4.9.

Parameter	Optimal value suggested
Batch Size	256
Batch Normalization	False
Learning Rate	0.001
Dense Units	1024
Use 2nd Dense	False
Dense Units2	0
Dropout Rate1	0
Dropout Rate2	0.3
Regularizer	None
Highest Measured Val Acc	0.864

Table 4.6: Best trial-values for three different models when using Optuna

Testing the suggested parameters

Using the suggested parameters from Optuna, along with fine tuning, by unfreezing 50% of the model after 15 epochs, further improvements across all metrics were yielded. The new highest recorded validation accuracy was measured to 86.73% and can be seen in Figure 4.5. Also, a comparison which displays the incremental increase in performance after each Optuna run can be seen in Figure 4.6.

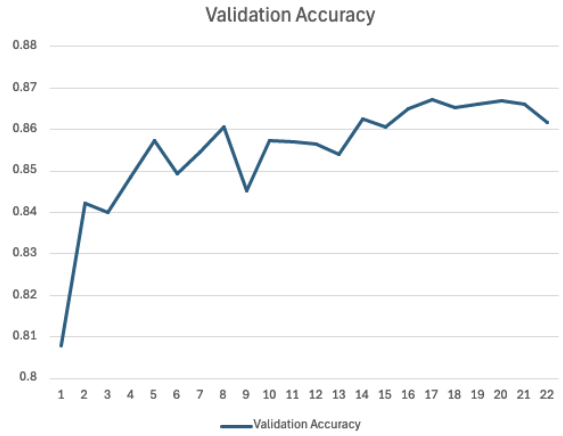


Figure 4.5: Validation Accuracy

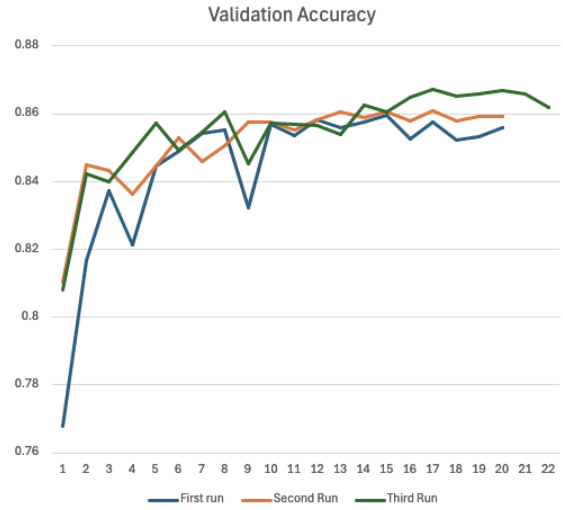


Figure 4.6: Validation Accuracy

Final fine-tuning

Lastly, a technique called reduce learning rate on plateau was applied which helped the model become more stable by decreasing the learning rate when there were no improvements to validation accuracy for three consecutive epochs. A comparison between the previous run in Figure 4.5 can be seen in Figure 4.7.

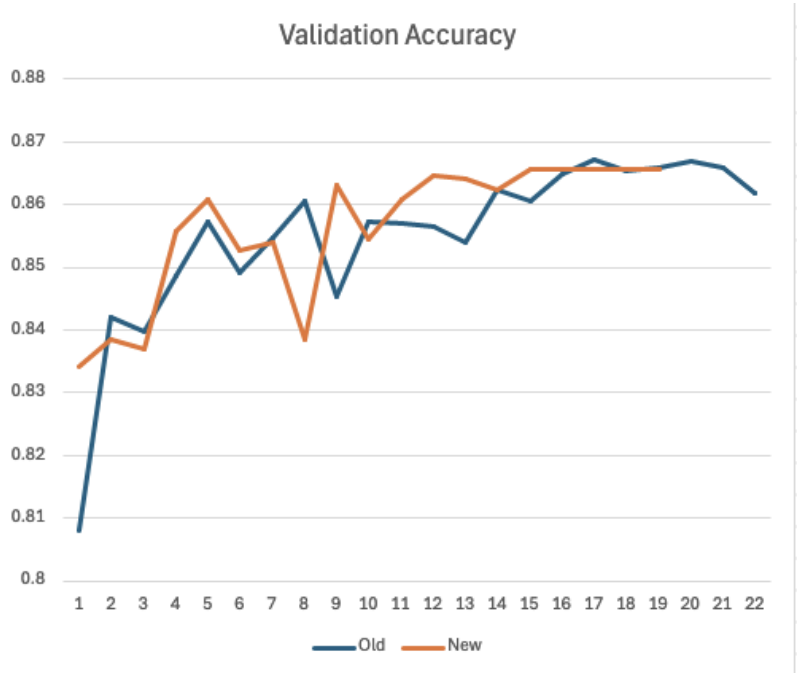


Figure 4.7: Validation Accuracy

EfficientnetV2-S & EfficientnetV2-L
Due to time constraints, both of these models used the same parameters suggested by the third Optuna run found in Table 4.9. The results yielded might not be optimal, since the parameters are optimized for a model with lower complexity. This is verified in Figure 4.8 which compares the B3 model to the large and small versions.

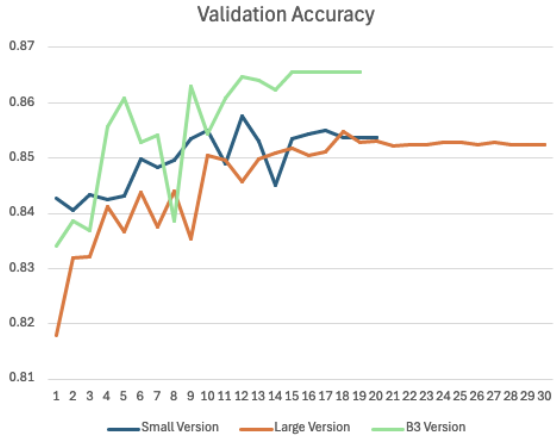


Figure 4.8: A comparison between the different EfficientnetV2 models

4.2.2 ResNet

A less comprehensive Optuna study was composed for the ResNet model. Initially, a study was conducted to determine the most effective parameters. Afterwards, a longer run was executed with the suggested parameters. The resulting graph can be seen in Figure 4.9.

Parameter	ResNet152V2
Number of trials	25
Epochs	12
Batch Size	512
Batch Normalization	False
Learning Rate	0.001
Dense Units	1024
Use 2nd Dense layer	True
Dense Units2	256
Dropout Rate1	0.3
Dropout Rate2	0.1
Regularizer	None

Table 4.7: Best trial-values for ResNet when using Optuna

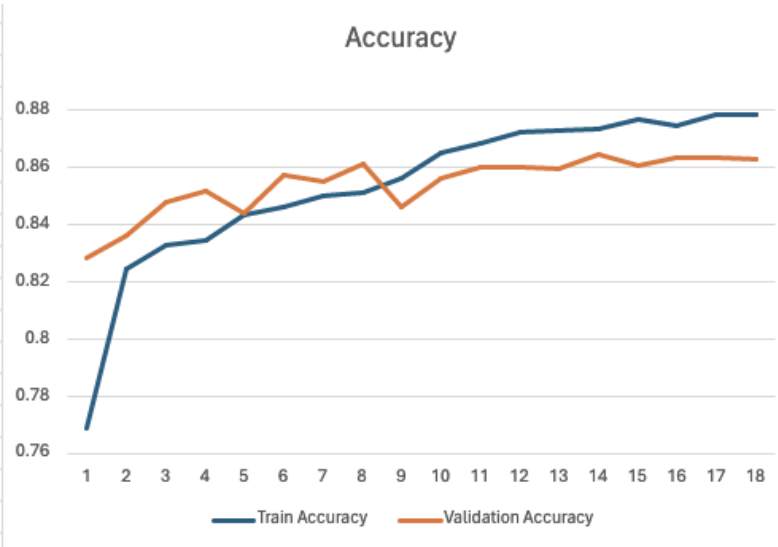


Figure 4.9: ResNet training process

4.2.3 DenseNet

DenseNet was also investigated through the utilization of Optuna. As can be seen in Figure 4.10 the training accuracy reached 1.0. This is likely the cause of Optuna neither suggesting the use of dropout layers, nor a regularizer.

Parameter	DenseNet201
Number of trials	25
Batch Size	128
Batch Normalization	True
Learning Rate	0.001
Dense Units	512
Use 2nd Dense	False
Dense Units2	0
Dropout Rate1	0.0
Dropout Rate2	0.0
Regularizer	None

Table 4.8: Best trial-values for DenseNet when using Optuna

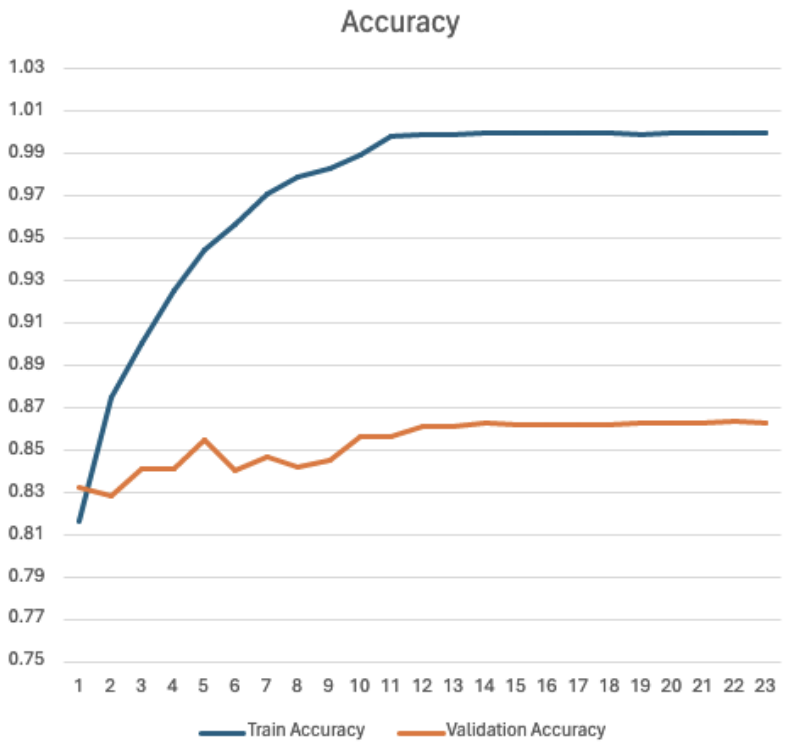


Figure 4.10: Training and Validation accuracy across epochs

4.3 Final results

This chapter summarizes the best parameters suggested by Optuna for each model, as well as the highest accuracy reached across multiple metrics when using those parameters along with fine-tuning by unfreezing the base model after 10 epochs.

Parameter	EfficientnetV2B3	ResNet152V2	DenseNet201
Batch Size	256	512	32
Batch Normalization	False	False	True
Learning Rate	0.001	0.001	0.001
Dense Units	1024	1024	512
Use 2nd Dense	False	True	False
Dense Units2	0	256	0
Dropout Rate1	0	0.3	0
Dropout Rate2	0.3	0.1	0
Regularizer	None	None	None

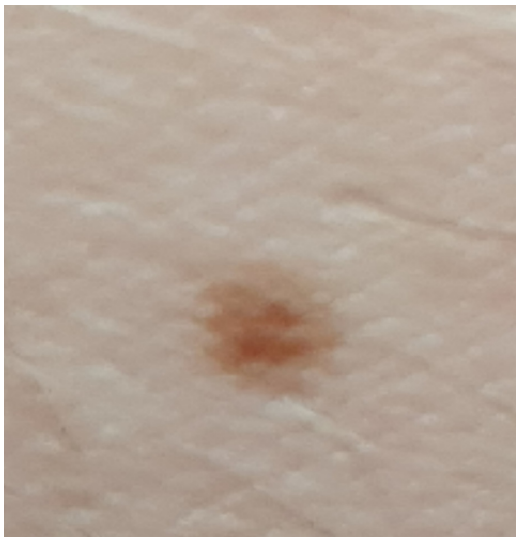
Table 4.9: Best hyper-parameter values for each model according to Optuna

Model	Class	Accuracy	Precision	Recall	F1 score	F-beta score
Baseline	Benign	0.84	0.85	0.85	0.85	0.850
	Malignant		0.82	0.83	0.83	0.828
EfficientnetV2B3	Benign	0.855	0.887	0.835	0.860	0.849
	Malignant		0.822	0.877	0.849	0.865
ResNet152V2	Benign	0.831	0.853	0.828	0.84	0.833
	Malignant		0.808	0.835	0.821	0.829
DenseNet201	Benign	0.849	0.867	0.848	0.857	0.852
	Malignant		0.829	0.849	0.839	0.845

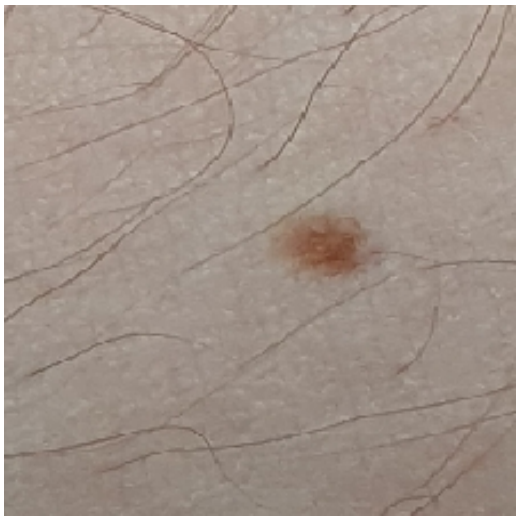
Table 4.10: Results on test-set across different models

4.4 Results via the application

The model integrated into the application was trained using skin microscopy images and therefore it might not acclimatize well to images taken with a mobile phone camera. The very same benign mole was photographed and evaluated by the model to see if the results changed based on the quality of the image, as seen in Figure 4.11. The model that was used for this test was the best performing iteration of EfficientnetV2B3, as presented in Table 4.10.

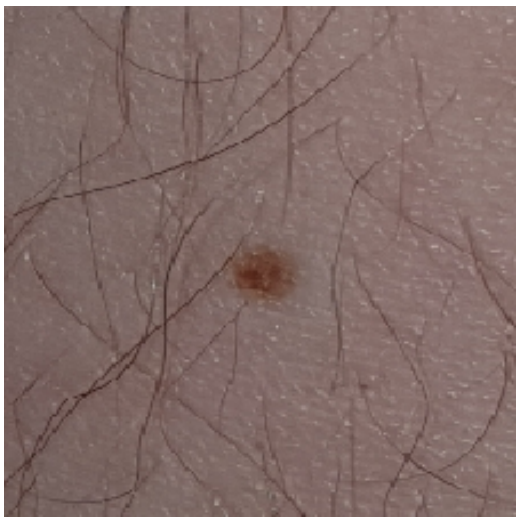


(a) 67.2% Benign



(b) 88.6% Benign

(c) 90.7% Benign (taken with flash)

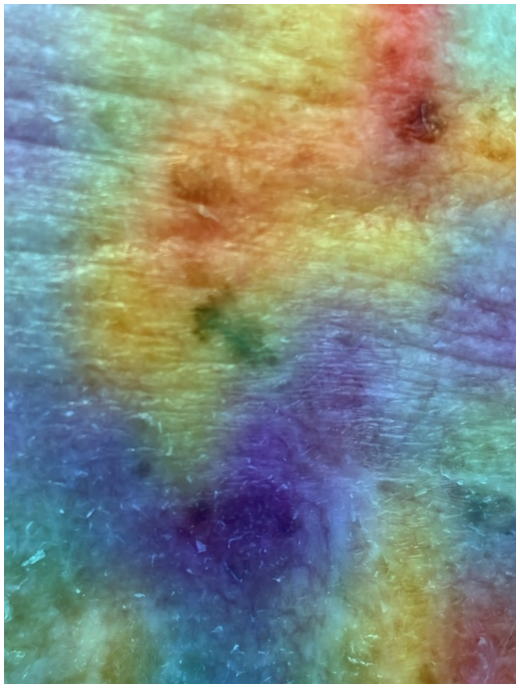


(d) 85.1% Benign (taken with flash)

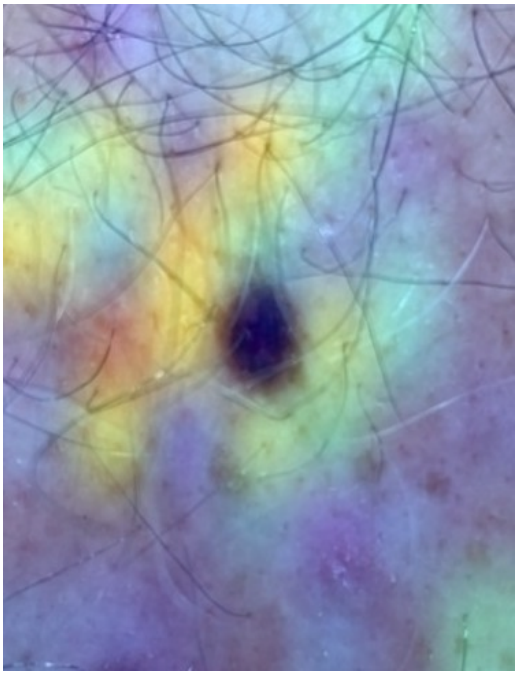
Figure 4.11: Images of the same mole with different zoom and lighting conditions

4.5 Moles with heatmaps applied

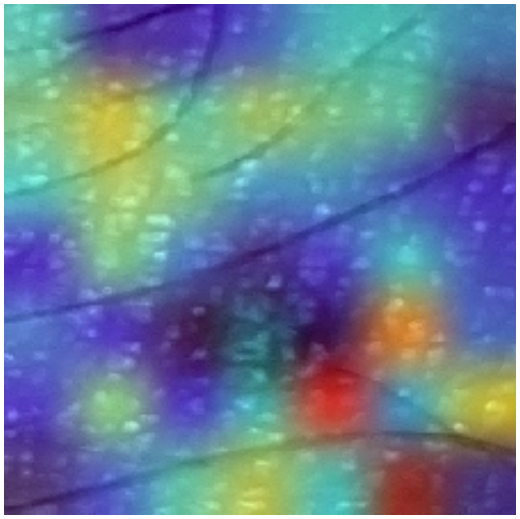
Below in Figure 4.12 is a collection of images with heatmaps applied created using Grad-CAM to visualize the areas the model focuses on for its predictions. All the images were taken with mobile phone cameras.



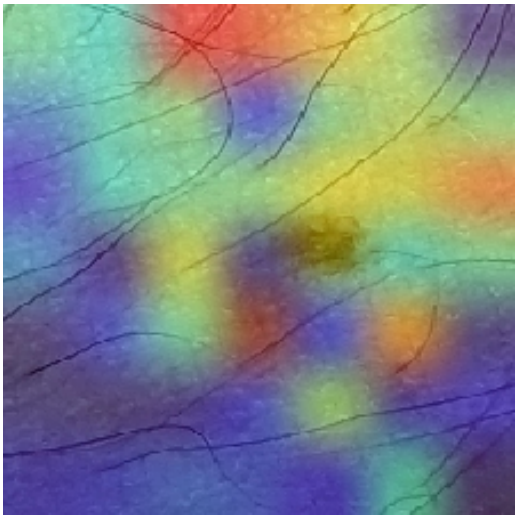
(a) Malignant mole using a phone camera. Predicted class malignant.



(b) Benign mole using a phone camera. Predicted class malignant



(c) Benign mole using a camera with flash. Predicted class benign

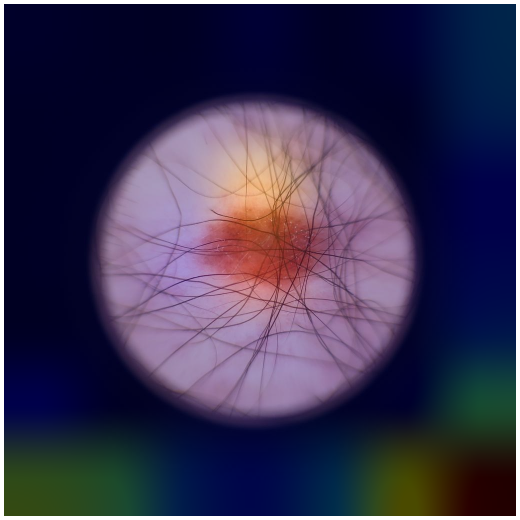


(d) Benign mole using a phone camera without flash. Predicted class benign

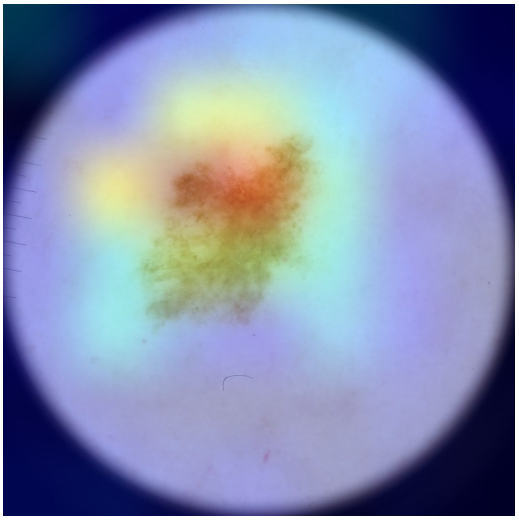
Figure 4.12: Images of moles from the dataset containing markers

It is not clear what the model looks for in the images without a malignant mole present, which will be discussed more in Section 5.5.

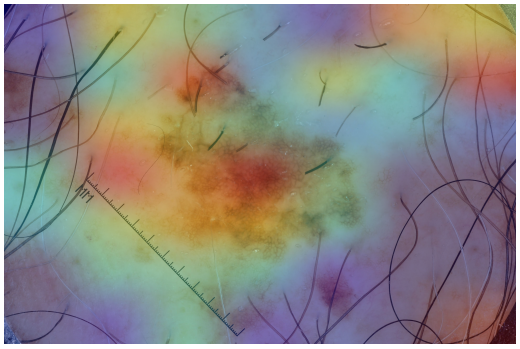
To investigate whether the model cheated or not, a heatmap was applied to images with clear markings on them, to see if the model looked at the markings or not. Figure 4.13 illustrates this.



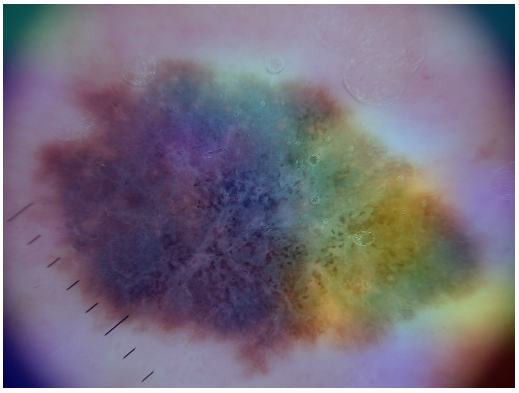
(a) Benign mole with marker present. Predicted class benign.



(b) Benign mole with marker present. Predicted class malignant



(c) Malignant mole with marker present. Predicted class malignant



(d) Malignant mole with marker present. Predicted class malignant

Figure 4.13: Images of moles from the dataset containing markers

4.6 The Application

This section showcases key features of the application and provides guidance on navigation.

Opening the application Upon launching the application the user is presented with a set of instructions designed to ensure optimal image quality. The instructions detail the steps necessary to capture the best possible image of a mole, though these instructions are not present here.

Navigating the application
Following the instructions, the user is directed to the main menu of the app, as depicted in Figure 4.14. The main menu includes several options:

- **Instructions and tutorial**: Redirects the user back to the initial start page containing the instructions.
- **Camera**: Activates the device's camera to capture an image of a mole.
- **Advanced information**: Provides detailed guidance based on the model's prediction.

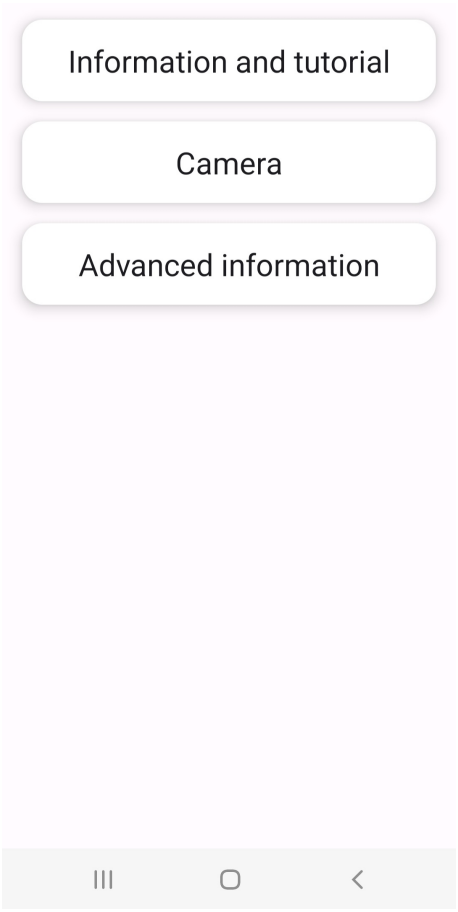


Figure 4.14: Main menu of the developed application

Waiting for evaluation
Once an image is submitted, the user must wait for the server to process the image and return a prediction. The waiting period is visualized using a progress bar, as shown in Figure 4.15.

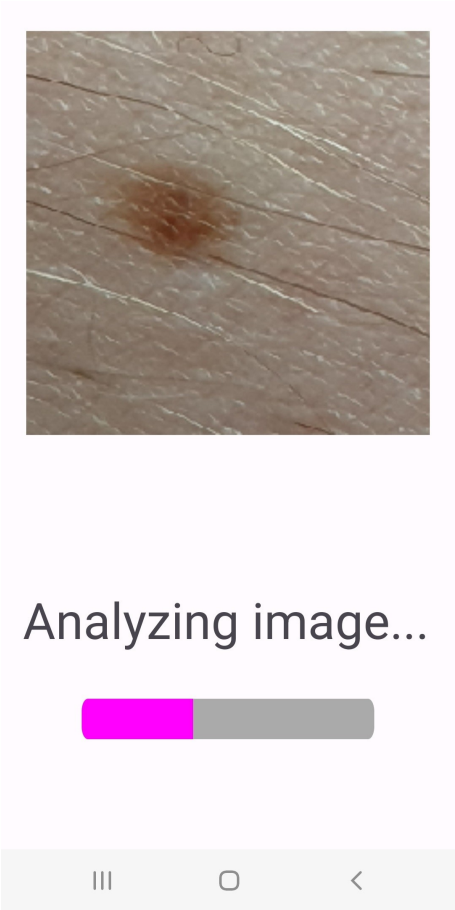


Figure 4.15: Waiting screen in the application

Getting the result
Once the server has responded, the user is directed to the next screen which displays the result. A percentage is presented that represents the model's prediction confidence. An example is illustrated in Figure 4.16.

Course of action
Clicking 'Next' on the results screen leads to the 'Advanced information' page, which presents users with advice based on the model's prediction. This screen is depicted in Figure 4.17.

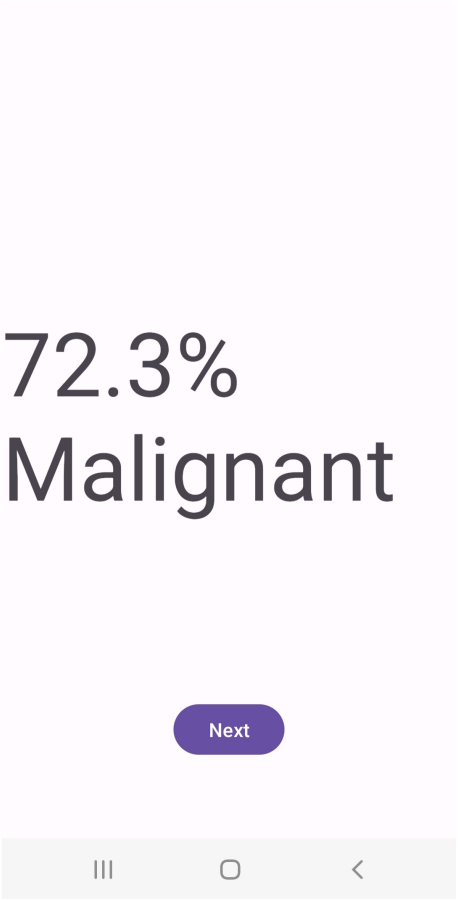


Figure 4.16: Results screen

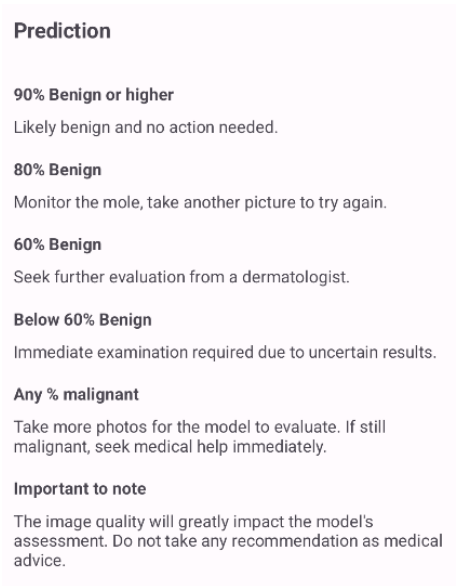


Figure 4.17: Information screen

5

Discussion

5.1 Selection of initial model

At the start of the project, limitations to the available graphics card were still unknown, and the decision on whether to operate the app cloud based or locally on mobile devices remained undecided. These factors introduced complexities in choosing an initial model to train. A balance was sought between model performance and resource intensity. EfficientNetV2, which was the first model to be evaluated offers various configurations with differing levels of performance and complexity, detailed in Tables 2.2 and 2.3. While the most complex models were appealing for their performance, their feasibility given the potential constraints were still uncertain. Consequently, EfficientNetV2-B3 was selected as the starting model due to the balance of performance and complexity.

The intermittent access to GPU resources throughout the project limited the ability to thoroughly test all potential models with Optuna, leading to potential discrepancies in the results section where model performance may vary unexpectedly. This is primarily because models outside of EfficientNetV2-B3 were not properly optimized using Optuna.

5.2 Optuna

While Optuna proved valuable for hyperparameter optimization, there was a learning curve on how to utilize it effectively. Part of this learning curve had to do with not having access to the GPU all the time, which made it necessary to not waste any potential runs. In the first study each trial lasted only for five epochs, to allow for more trials within the allotted GPU time. However, five epochs was not enough for the model to converge and consequently not an accurate representation of the potential of the specific sets of hyperparameters. Furthermore, there will also be bias towards certain hyperparameters when training for such a short amount of time. For instance, Optuna will favor high learning rate and low drop out rate simply because that is advantageous in short runs, due to the model making bigger adjustments in its limited epochs. Lower learning rates and higher dropout rates might be a better alternative if the model is run until convergence to ensure that convergence is reached. Striking a balance between the number of epochs versus the

number of trials would become a recurrent problem throughout all studies in the project. The optimal solution would be to increase both, but that was not possible due to the time frame of which the borrowed GPU could be used.

In the second study additional parameters were tested and the number of epochs for each trial was increased to 12, which is when most runs have shown to begin stagnating in their performances. Although the suggested parameters from Optuna lead to an increase in performance when tested, there were still signs of biases toward certain parameters. Parameters which increased complexity of the model (extra dense layer) or the addition of regularizers seemed to perform poorly in general. The reason could either be that those parameters were not favourable for this task, or the same problem as before occurred; the number of epochs for each trial was simply not high enough.

In the last study a pruning technique was added to remove trials which did not exceed a set accuracy after a set number of epochs. This allowed for faster training times which was crucial due to the limited access to the GPU. Optuna pruned each trial after 5 epochs, which may have eliminated trials with lower learning rates, due to them not having enough time to change their weights by a significant amount.

5.3 Interpreting model confidence in skin image analysis

Once the model makes a prediction, it returns a number corresponding to how sure it is on its prediction. If the model returns "70% Benign" that means that the model is 70% certain that the mole in question is benign. Due to the nature of this application and its objectives, ensuring the accuracy of predictions is of utmost importance to safeguard people's health. This exact number is returned and displayed in the application to give every individual a choice on how to act on the information they received. This is coupled with information regarding initial suggestions based on what the model predicted for them. The exact numbers can be seen in Table 5.1.

Prediction	**Recommendation**
90% Benign or higher	Likely benign and no action needed.
80% Benign	Monitor the mole, take another picture to try again.
60% Benign	Seek further evaluation from a dermatologist.
Below 60% Benign	Immediate examination required due to uncertain results.
Any % malignant	Take more photos for the model to evaluate. If still malignant, seek medical help immediately.

Table 5.1: Model predictions and corresponding recommendations

These numbers are all preliminary and should not be used as medical advice. They

would require thorough validation and refinement before being considered for any practical medical application if the app was ever to be published. A conservative approach should be used when setting these numbers due to the consequences of a potential misdiagnosis, which are discussed in Chapter 6.

It is important to note that the correlation between the models output and the actual class of the mole is not linear. If the model predicts a mole to be benign with a 75% certainty, this does not imply that there is a 25% chance of cancer being present. Instead, this number represents the models confidence in its prediction based on similarities from the testing data. As seen in Figure 4.11, this confidence is influenced by several factors, such as lighting, image quality, zoom etc.

5.4 Offloading back-end processing to the cloud

A decision that had to be made was to run the model either locally on the phone or use a dedicated server to run the model. Running a model on the phones hardware could pose some challenges due to Tensorflow and the trained models requiring a lot of space. A solution to this is to use TensorFlow Lite, a tool to convert Tensorflow models into smaller, more efficient versions of themselves [24]. However, making a model more compact and optimized might cause the performance to be affected. Using the trained model for the app where the performance is known was opted for, to alleviate any possible risk of losing performance.

A server based approach would allow for the entire trained model to be uploaded and run without major delays due to better performance in the hardware. Having access to an AWS account via Knightec made it even more possible to use a server sided approach.

There exist some possible disadvantages of choosing to run the model via the cloud instead of locally. One is that the app requires an internet connection to work due to its connection to a server. Another issue is the problem with privacy, and that the users privacy and integrity may not be well protected if using a cloud based solution, more on that in 6.3. Releasing the app to the public would be more suitable if the model was run locally on devices, due to monetary reasons since the server is not free.

5.5 Information gained using heatmaps

With markers present in a substantial amount of images, there was a concern that the model looked at the markers rather than the mole itself. The markers range from colourful stickers, presumably used for camera focus, to scales for measuring the size of the mole. This was investigated using Grad-CAM. As can be seen in Figure 4.13, it appears as though the model is not solely looking at the markings to base its predictions on.

Another thing that was discovered is how the model classified benign moles. It appears as in the cases where a malignant mole is not present, the model looks for clues outside of the actual mole to find evidence that there might be a malignant mole present. It has been observed to look at hairs or shadows, as can be seen in Figure 4.13a. However in cases where a malignant mole is present, the model examines the mole itself to determine its nature — whether it is benign or malignant.

5.6 The importance of good images

As was seen in Figure 4.11, the very same mole can give different readings when evaluated by the model. The models prediction varied by more than 23 percentage units in its prediction. This can be due to a multitude of factors, including lighting, zoom, focus, amount of hair strands present etc. Capturing a good picture of the mole is challenging, and requires either help or the usage of some stabilising tool to keep the camera still. The limit for if a mole is considered safe or not was decided to be 90%, and harmless moles might not reach that percentage prediction from the model just due to poor image quality. This could lead to false alarms for the users, indicating that they may have cancer while in reality it was only the image quality that was poor. The same can be said for malignant moles. There is a chance that they could reach the threshold for being considered harmless by the model based on poor images, however these cases have not been tested due to not having physical access to malignant moles.

What is considered a bad quality image was not determined during the project. There are a multitude of factors that could impact the model's prediction capabilities, such as the level of blur or noise being present in the image. The size of the mole in the image might also play a part in determining the performance of the model on it. This would be a good point to investigate if the project is ever revisited in the future.

There is a possibility that users may accidentally submit images that do not contain moles, such as a finger over the camera. By introducing a third class in the model labeled "not a mole", images which doesn't contain a mole could be filtered out and thereby increase the reliability of the model in such cases. This could be a valuable addition for future iterations of similar projects.

5.7 Different performance based on different skin colors

There is a possibility that the trained model has a strong bias connected to the subjects skin color and heritage. The concern comes from the fact that the dataset does not contain labels on the ethnicity of the person possessing the mole, which means that the skin color around the mole could vary drastically - or worse, not vary a lot. There is information about which hospital the image is taken from, and with some research one could investigate the demographic of that specific area and

come up with some educated guesses on the demographic. This would be pointless due to the fact that just because the demographic is split a certain way, does not mean that the same split of people will go to the hospital for suspected skin cancer. The outcome is that without a label on the image, there is no good way to get a picture for the demographic split in the dataset. If the model is trained on primarily white skin colored individuals, then it might have difficulties making predictions on darker skin colors because it might confuse the skin color for moles.

5.8 Doctors

An interesting aspect of this project is to compare it to real world doctors and healthcare. A performance significantly stronger than trained professionals might indicate potential for the future within the field of using machine learning for medical diagnosis.

The best model trained for this project achieved a test accuracy of 85.5%, illustrated in Table 4.10. When compared to Table 2.1 it can be clearly seen that the model's performance is several percentage units higher than the most experienced doctors and significantly better than the registrars of less experience.

Worth noting is that the sample size was rather small in the performed study on doctors diagnostic capabilities, as there were less than 200 moles investigated. There might have been a different result if the doctors were to evaluate the same dataset that the model was trained on, due to there possibly being harder or easier moles in one of these instances.

Regardless, the model's accuracy and performance appears to perform better than the medical professionals from this this study at the very least.

5.8.1 Validation of Model Predictions with Clinical Evaluations

Two photographs, taken with a mobile device, of moles suspected to be malignant were obtained from doctor at a skin clinic [52]. These images (without the heat map applied) labeled as (a) and (b) in Figure 4.12, were evaluated by the doctor who suspected malignancy in both cases. While awaiting laboratory confirmation of the moles nature, the trained model was employed to assess them. The model predicted both moles to be malignant. However, the lab results indicated that only the mole in image (a) was cancerous. This outcome suggests that the model identified similar characteristics to those the doctor considered indicative of malignancy.

5.9 Informing users about image quality

A crucial aspect of the app is to educate users on the limitations of the model in relation to the way it is used. The model itself might perform well on its test data,

but not translate well to real world use. This will impact its predictions which is important to highlight for any potential user of the app. The users need to be aware of these limitations which is why this information is present in the app, after the model has evaluated an image.

This transparency is vital for both ethical reasons and also in managing expectations about the apps capabilities. By clearly stating and communicating these limitations, it is ensured that the user does not use the app's prediction as medical advice. Any uncertainty should be brought to a doctor even if the model is certain that the mole is benign.

5.10 Future work

This section will present ideas on what to implement in future projects similar to this one.

5.10.1 Image segmentation

To avoid the problem of possible markers and other features present in images that were not a part of the mole, image segmentation could be a solution. Image segmentation is marking which part of the image is the mole, which can then be used for the prediction. The issue with this however, is that the skin around the mole might be significant to the class of the mole.

A workaround that could satisfy both of these issues is cropping the image to remove any potential markers, either manually or by using some script designed to detect markers. These should be cropped out completely, and the image should be resized accordingly. This could present yet another problem though, the model could learn that some images are shaped differently than others due to this cropping, and learn to draw conclusions from that.

The absolute best approach to this would be to use a dataset with no markers present in any images. This would ensure that the model does not take the markers into account and would provide a good environment for it to acclimatize to unseen data.

5.10.2 Training the model with relevant images

To make the model even better at classifying malignant or benign melanoma from an image captured with a mobile phone camera, an approach would be to train it on images like this. This would enable the model to work better in the use case scenario of using the app.

A glaring issue with this however is that there can be a lot of variability in images captured with a mobile phone. The quality, zoom, lighting and focus are all things that could impact how the model learns, while also being things that are hard to keep constant when taking pictures with a mobile phone. To combat this, a huge

dataset would be needed to cover a vast amount of different settings for the image in terms of the aforementioned parameters.

The collecting of such a dataset could prove difficult, due to there not being a widely available dataset like this. Contact with hospitals and clinics could be established with the intention of collecting images, but it could take years to get a proper dataset.

5.10.3 Potential of future models

Trying to tune the model to be as good as possible would require a lot of time. The many different hyperparameters available along with the resource needs for training a model does not allow for it to be trained and tuned on short notice. Along with this, new models are constantly released. Trying the latest and best performing model for this project could improve the performance.

Just in a few years since Länsberg and Manfredsson [25] did a similar project, a lot of new models have been released. The accuracy and recall achieved in this project surpass their accuracy and recall of around 15 and 20 percentage units respectively, mostly due to new and better models being released since then. If this project is ever revisited, there might be even better models available.

5.10.4 Approach to ensemble learning

No form of ensemble learning was utilized in the project due to time constraints. This could be something worth exploring in future work, as it might be beneficial for the performance of the app as a whole. In majority voting, several models are used to give a single prediction. This could increase accuracy as some models might pick up on details that other models miss, causing them to complement each other.

Possible disadvantages of using ensemble learning include increased resource consumption and extended processing times. Additionally, the implementation of ensemble learning might make local deployment on mobile devices impossible, due to the sheer size of the models. This is particularly relevant when there are multiple models required.

5.10.5 Develop app for widespread use

The app that was developed for testing the model was only used as an interface to said model. It was not developed with the intention to be released to the public, and would not be possible due to budgetary reasons since it is running on a paid server with pricing based usage. This could be circumvented if the model could run locally on the phone, and therefore open up the possibility of making a proper app to be released for the public.

Along with running the model locally, the app would have benefited from being developed for both Android and Iphone. As of now, the app only works on Android which limits a vast majority of the user market.

The app could then be polished to offer a good user experience, which was not a focus when developing the current app.

5.10.6 Potential for real world use

If there was a way to somehow extract the performance of the models and have them compatible with mobile phone cameras, then there might actually be some potential for real world use from this project. The problem lies in that the image quality can not be guaranteed when using a mobile phone camera. This is because of two reasons, the first one being that different phones have different cameras, and some may meet the minimum requirements for the application. The other big problem is that people might have difficulties taking good quality pictures which might confuse the model, as seen in Figure 4.11. The images in that figure were taken with the same camera, and so the results might differ even more if different cameras were used. This scenario was however not tested in practise.

Not everyone possess phones with cameras capable of taking good quality pictures, or even phones with operative systems that allow third part applications. This part of the population would probably struggle with taking a good quality image even if they did possess the necessary means for it. This might be the group of people that could benefit the most from the application if it worked well for them.

With the technology of today there does not seem to be a suitable solution for the application for its main target demographic. Not being able to guarantee good quality images makes this application not reliable enough for real world use.

Another approach to real world use of the trained model is if it was applied directly on images from a microscopy. The performance of the trained model exceeded that of trained medical professionals (see Section 5.8). This was only on the specific dataset used for training and testing the model though, and it is unclear how it would perform in a real world setting. A clinic could employ the use of a model if at least two minimum requirements are met:

1. **Access to microscopy imaging Eequipment:**
   The clinic has access to equipment that allows the capturing of images similar to those in the dataset the model was trained on.

2. **Verification of model efficacy:**
   The clinic can verify that the model works on real-world data by testing it in the field on moles with a known label.

If both of these criteria are met, then it could be an idea to incorporate a trained model into the clinics workflow. The model could be incorporated in a way so that if the doctor predicts a mole to be benign, but the model predicts it to be malignant, then the model should be taken into account and further testing should be performed to verify the true nature of the mole. If a doctor thinks a mole is malignant but the model thinks its benign, further testing should be explored to exhaust the possibility of missing a malignant mole due to the models wrong prediction.

6

Ethics and sustainability

As of today's day and age, the healthcare system in Sweden is heavily strained. Only 72% of people get to see a doctor on the day of contact [14]. This number stretches to 90% when examined over a three day period. This effectively means that one in ten people has to wait longer than three days to see a doctor in potentially life threatening conditions. Lessening the strain on the healthcare system is therefore a high priority.

This project ties into UN's third goal of "Ensuring healthy lives and promote well-being for all at all ages" [8]. By classifying moles as either benign or malignant at home via an app before a patient even arrives at the doctors appointment could help lessen the strain on the healthcare system by filtering out people who do not need medical assistance.

Important to note is that the project was not carried out by people in the medical field. The trained models and the developed app should therefore **not** be considered medical advice, and any predictions should carefully considered.

The most glaring issue with the model is the discrepancy between the high quality dermatological images used for training and the lower quality images captured using a mobile phone camera. This inconsistency can lead to less accurate predictions since the model is optimized for data that users cannot produce at home. This issue is vital to keep in mind at all times when using the app, because it underscores the importance of proceeding with caution after getting a recommendation from the model.

There is always a possibility of having false predictions by the model. The trained model achieved a test accuracy of 86.73% on the dataset used for training. However, because the dataset consists of clinical grade dermatological images, this accuracy may not translate well to the mobile app. This emphasizes the need for transparency due to the models limitations.

Imagine a scenario in which a model of 95% accuracy was trained. Assume it to handle images from the app just as well as the training data. This **still** means that the model will be wrong around 1 in 20 times. A situation could appear where a person with a cancerous mole has it classified as benign and delays in seeking medical assistance.

Due to the reasons stated above, it is crucial to maintain transparency of the models limitations and remind the user that it is not to be substituted for medical advice. Regardless of the models prediction, potentially cancerous mole should always be examined by a medical professional. This approach aims to align with ethical standards and avoid potential consequences of misdiagnosing due to the models prediction. This technology is meant to complement the healthcare system, not replace it.

6.1 The potential lack of ethnic diversity in the dataset

There could be a bias in the dataset, as discussed in 5.7. This could lead to the model having a hard time detecting malignant melanoma on people of color, due to it being trained primarily on people with lighter skin tones. This could lead to severe consequences in a scenario where a malignant mole is missed due to this bias in the model.

Section 67 of the EU AI act specifically emphasises this point, by stating that the datasets should also "have the appropriate statistical properties". It also states that attention should be paid to trying to mitigate biases in the datasets, especially those that might impact the health of people [28]. This project definitely could impact the health of individuals. For real world use another dataset would have to have be used to alleviate the risk of biases.

6.2 Different forms of sustainability

There are different forms of sustainability that the project might target.

Economic sustainability is clearly targeted by paving the way for a future app that minimizes unnecessary doctor visits and hastening the diagnostic process. This could be especially beneficial in developing countries or under resources areas.

Social sustainability is another perspective to be considered. The project may in the future increase accessibility to healthcare diagnosis, which may reduce disparities in healthcare access. By enabling early detection of health issues through an app, the project could contribute to better health outcomes across diverse social groups. It might increase awareness about health issues for people without the necessary knowledge to know when a doctor visit is suitable.

6.3 Privacy

The privacy of the users is to be taken into consideration when working with an app like this. Although the app is not designed to save information about the images sent to the server, there might still be metadata about images available. Things like

where the image was taken or which phone it was sent from. Such information could be used to identify the individual in the image. Potential consequences of having the image and identity of the person leaked could be higher insurance costs if an insurance company figures out who has cancer or not. This has not been considered in the development of the app, and is yet another reason why the app is not suitable for the general public.

7

Conclusion

This thesis set out to answer two questions regarding the performance of deep learning models in classifying moles and the effectiveness of using images taken with a mobile phone for evaluation using the trained models.

What performance is achievable using deep learning models in classifying malignant and benign moles, and how can its accuracy, recall, and precision be further optimized?

The trained model, EfficientNetV2B3, was able to achieve promising results for the future medical industry. This performance was improved using the Optuna framework, and the model's validation accuracy was incrementally improved across multiple runs. The highest validation accuracy achieved using this model was measured at 86.73%, with the highest test validation measured at 85.5%. The recall factor in the best performing iteration of EfficientNetV2B3 was measured to 87.7% when classifying malignant moles. This performance was achieved through careful tuning of hyperparameters, as well as fine tuning techniques such as unfreezing the base model or reducing the learning rate during training.

This performance is promising and might prove useful in the future of the medical field.

How effective can a mobile camera be in capturing and classifying images of moles, and what factors influence its usability and accuracy in a real-world setting?

The development and testing of the mobile application revealed that a mobile camera, when combined with proper instructions and optimal conditions, has the potential to capture images of moles for classification purposes. This is not the case without optimal conditions though, which might be hard to reproduce, especially for older people that might benefit the most from the application. These conditions include lighting, image clarity and user compliance in following the provided instructions. The application demonstrated potential to in the future serve as a preliminary diagnostic tool, although this is not possible yet due to the uncertainty that comes with mobile devices and image quality.

Other critical findings

Image quality significantly impacted the quality of the model's predictions. Factors such as lighting, zoom, focus can drastically alter the model's certainty prediction, sometimes more than 23 percentage points. This underscores the importance of ensuring consistent image quality when using a mobile phone camera to capture images, as it is essential for reliable predictions.

More extensive studies with Optuna for hyperparameter optimization would be beneficial for less biased suggestions of optimal hyperparameters. In this project, Optuna studies were restricted to 36 hours a week due to limited GPU access time. Consequently, the studies were shortened by both decreasing the number of trials and reducing the number of epochs per trial. Ideally, each trial should exceed the epochs required for the model to converge. This approach would allow hyperparameters that mitigate overfitting, such as regularizers, and those that benefit from extended training periods, like a lower learning rate, to potentially influence the model's performance positively. In this project, Optuna studies rarely suggested the use of regularizers since the model did not run long enough for their potential benefits to take place. Additionally, increasing the number of studies and expanding the range within which a hyperparameter is tested could lead to better model performance.

In conclusion, the combination of multiple deep learning techniques and an accompanying mobile application shows promise in enhancing early detection of skin cancer. Future work should focus on further improving the model, solutions for finding a reliable way to get good quality images using mobile phone cameras, and expanding the dataset to ensure diversity. By addressing these areas, the system has the potential to become a valuable tool in the medical field, facilitating the early detection of malignant melanoma.

Bibliography

[1] Gustaver, M. (2020) A Chalmers University of Technology Master's thesis template for LaTeX. Unpublished.

[2] Cancerfonden, "Hudcancer," 2023 [Online]. Available: https://www.cancerfonden.se/om-cancer/cancersjukdomar/hudcancer (Accessed: 2024-01-25).

[3] Skincancer, "Skin Cancer Facts and Statistics", 2024 [Online]. Available: https://www.skincancer.org/skin-cancer-information/skin-cancer-facts/ (Accessed: 2024-01-25)

[4] Cancer Research UK. "About cancer: Melanoma." [Online]. Available: https://www.cancerresearchuk.org/about-cancer/melanoma (Accessed: Jan. 25, 2024).

[5] Michael A. Nielsen, "Neural Networks and Deep Learning" Determination Press, 2015 [Online]. Available: http://neuralnetworksanddeeplearning.com/chap6.html (Accessed: 2024-01-25)

[6] International Business Machines Corporation, "What are convolutional neural networks?", [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks (Accessed: 2024-01-29)

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436-444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[8] United Nations. "Ensure healthy lives and promote well-being for all at all ages." Accessed: Jan. 25, 2024. [Online]. Available: https://www.un.org/sustainabledevelopment/health/ (Accessed: 2024-02-2)

[9] International Skin Imaging Collaboration, "ISIC Archive," [Online]. Available: https://www.isic-archive.com/ (Accessed: Jan. 25, 2024)

[10] M. Mostafizur Rahmanand D. N. Davis, "Addressing the Class Imbalance Problem in Medical Datasets," *International Journal of Machine Learning and Computing*, vol. 3, no. 2, [Online]. Available: https://www.ijmlc.org/papers/307-K0020.pdf

[11] Peter I. Frazier, "A tutorial on Bayesian Optimization", arXiv, 2018. [Online], Available: https://doi.org/10.48550/arXiv.1807.02811

[12] A. Lacoste, H. Larochelle, F. Laviolette, and M. Marchand, "Sequential Model-Based Ensemble Optimization," arXiv, 2014. [Online]. Available: https://arxiv.org/abs/1402.0796

[13] Google Developer Program, Android Studio. [Online]. Available: https://developer.android.com/studio (Accessed: 2024-04-16)

[14] Sveriges Kommuner och Regioner, Medicinsk bedömning i promärvården. [Online]. Available: https://skr.se/vantetiderivarden/vantetidsstatistik/medicinskbedomning.54389.html (Accessed: 2024-02-1)

[15] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, [Online]. Available: doi: 10.1109/CVPR.2009.5206848

[16] Polikar, R. (2012). Ensemble Learning. In: Zhang, C., Ma, Y. (eds) Ensemble Machine Learning. Springer, New York, NY. [Online]. Available: https://doi.org/10.1007/978-1-4419-9326-7_1

[17] M. Tan and Quoc V. Le, EfficientNetV2: Smaller Models and Faster Training, arXiv, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2104.00298

[18] P. Liashchynskyi and P. Lianshchynskyi, Grid Search, Random Search, Genetric Algorithm: A Big Comparison for NAS, arXiv, 2021, [Online]. Available: arXiv:1912.06059

[19] G. Huang et al., Densely Connected Convolutional Networks, ArXiv, 2018, [Online]. Available: arXiv:1608.06993

[20] Keras, Keras Applications, [Online]. Available: https://keras.io/api/applications/#usage-examples-for-image-classification-models

[21] K. He et al., Deep Residual Learning for Image Recognition, arXiv, 2015, [Online]. Available: https://arxiv.org/pdf/1512.03385.pdf

[22] Chen L, Li S, Bai Q, Yang J, Jiang S, Miao Y. Review of Image Classification Algorithms Based on Convolutional Neural Networks. Remote Sensing. 2021; 13(22):4712. [Online]. Available: https://doi.org/10.3390/rs13224712

[23] Tensorflow, "An end-to-end platform for machine learning," 2024. [Online] Available: https://www.tensorflow.org/

[24] Tensorflow, "Deploy machine learning models on mobile and edge devices," 2024. [Online]. Available: https://www.tensorflow.org/lite

[25] Simon Länsberg, Anna Manfredsson, "Detection of malignant melanomas using neural networks", BSc thesis, Department of Computer Engineering, Chalmers University of Technology, Gothenburg, Sweden, 2022. [Online]. Available: https://hdl.handle.net/20.500.12380/304921

[26] C.A. Morton, R.M. Mackie, Clinical accuracy of the diagnosis of cutaneous malignant melanoma, British Journal of Dermatology, Volume 138, Issue 2, 1 February 1998, Pages 283–287, [Online]. Available: https://doi.org/10.1046/j.1365-2133.1998.02075.x

[27] Dana-Farber Cancer Institute, How We Diagnose Melanoma, [Online]. Available: https://www.dana-farber.org/cancer-care/types/melanoma/diagnosis

[28] EU Artificial Intelligence Act, Final draft (2024), [Online]. Available: https://artificialintelligenceact.eu/the-act/

[29] R. Geiger et al., “Garbage in, garbage out” revisited: What do machine learning application papers report about human-labeled training data?, [Online]. Available: https://arxiv.org/pdf/2107.02278v1.pdf

[30] Michael Nielsen, "Neural Networks and Deep Learning," Determination Press, 2015. Available: http://neuralnetworksanddeeplearning.com/index.html

[31] Han SH, et al., Artificial Neural Network: Understanding the Basic Concepts without Mathematics. Dement Neurocogn Disord. 2018 Sep;17(3):83-89. [Online]. Available: doi: 10.12779/dnd.2018.17.3.83. Epub 2018 Dec 13. PMID: 30906397; PMCID: PMC6428006.

[32] Harriet L. Dawson, Olivier Dubrule, Cédric M. John, Impact of dataset size and convolutional neural network architecture on transfer learning for carbonate rock classification, Volume 171, 2023, 105284, ISSN 0098-3004, [Online]. Available: https://doi.org/10.1016/j.cageo.2022.105284

[33] N. Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, 2014, Vol. 15, [Online]. Available: https://jmlr.org/papers/v15/srivastava14a.html

[34] Sharma, Siddharth, Simone Sharma and Anidhya Athaiya. “ACTIVATION FUNCTIONS IN NEURAL NETWORKS.” International Journal of Engineering Applied Sciences and Technology (2020): n. pag. Available: doi: 10.33564/ijeast.2020.v04i12.054

[35] Optuna, "Optimize Your Optimization," 2024. [Online]. Available: https://optuna.org/ (Accessed: 2024-03-18)

[36] Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org/ (Accessed: 2024-03-05)

[37] S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, arXiv, 2015, [Online]. Available: https://arxiv.org/abs/1502.03167

[38] Amazon Web Services, What is AWS Lambda?, [Online]. Available https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (Accessed: 2024-04-19)

[39] Ibrahem Kandel, Mauro Castelli, The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset, ICT Express, Volume 6, Issue 4, 2020, Pages 312-315, ISSN 2405-9595, [Online]. Available: https://doi.org/10.1016/j.icte.2020.04.010

[40] Y. Wu and L. Liu, Selecting and Composing Learning Rate Policies for Deep Neural Networks, arXiv, 2022, [Online]. Available: https://arxiv.org/abs/2210.12936

[41] Wang, Q., Ma, Y., Zhao, K. et al. A Comprehensive Survey of Loss Functions in Machine Learning. Ann. Data. Sci. 9, 187–212 (2022). [Online]. Available: https://doi.org/10.1007/s40745-020-00253-5

[42] Hosna, A., Merry, E., Gyalmo, J. et al. Transfer learning: a friendly introduction. J Big Data 9, 102 (2022). [Online]. Available: https://doi.org/10.1186/s40537-022-00652-w

[43] Naqvi, M., Gilani, S. Q., Syed, T., Marques, O., & Kim, H. C. (2023). Skin Cancer Detection Using Deep Learning-A Review. Diagnostics (Basel, Switzerland), 13(11), 1911. [Online]. Available: https://doi.org/10.3390/diagnostics13111911

[44] Zafar M, Sharif MI, Sharif MI, Kadry S, Bukhari SAC, Rauf HT. Skin Lesion Analysis and Cancer Detection Based on Machine/Deep Learning Techniques: A Comprehensive Survey. Life (Basel). 2023 Jan 4;13(1):146. [Online]. Available: doi: 10.3390/life13010146 PMID: 36676093; PMCID: PMC9864434.

[45] Shorfuzzaman, M. An explainable stacked ensemble of deep learning models for improved melanoma skin cancer detection. Multimedia Systems 28, 1309–1323 (2022). [Online]. Available: https://doi.org/10.1007/s00530-021-00787-5

[46] Xue Ying 2019 J. Phys.: Conf. Ser. 1168 022022, [Online]. Available: doi:10.1088/1742-6596/1168/2/022022

[47] R. Johansson, Applied Machine Learning Introduction to Convolutional Neural Networks, unpublished. [Online]. Available: https://www.cse.chalmers.se/~richajo/dit866/lectures/l5/l5_2.pdf

[48] R. Johansson, Applied Machine Learning Introduction to Convolutional Neural Networks, unpublished. [Online]. Available: https://www.cse.chalmers.se/~richajo/dit866/lectures/l7/l7_1.pdf

[49] R. Johansson, Applied Machine Learning Introduction to Convolutional Neural Networks, unpublished. [Online]. Available: https://www.cse.chalmers.se/~richajo/dit866/lectures/l8/l8_1.pdf

[50] Selvaraju et al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization, arXiv, 2019, [Online]. Available: doi:10.1007/s11263-019-01228-7

[51] Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv, 2015, [Online]. Available: arXiv:1412.6980

[52] Doctor Dan Fagrell, The Swedish Medical Association, private communcation, March-May 2024

[53] K. O'Shea and R. Nash, An Introcution to Convolutional Neural Networks, arXiv, 2015, [Online]. Available: https://arxiv.org/abs/1511.08458

[54] Srivastava, Nitish and Hinton, Geoffrey and Krizhevsky, Alex and Sutskever, Ilya and Salakhutdinov, Ruslan, "Dropout: a simple way to prevent neural networks from overfitting," The Journal of Machine Learning Research, Volume 15, Issue 1pp 1929–1958, January 2014, [Online]. Available: https://dl.acm.org//abs/10.5555/2627435.2670313 (Accessed: 2024-03-04)

[55] Scikit Learn, "train_test_split," 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (Accessed: 2024-02-03).

[56] Amazon, "Welcome To AWS Documentation," 2024. [Online]. Available: https://docs.aws.amazon.com/ (Accessed: 2024-04-21).

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY