Department of Computer Science

BSc (Hons) Computer Science

Academic Year 2019 - 2020

Melanoma Detection with AI - Mobile App

Ben Lamb

1612127

A report submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

Brunel University
Department of Computer Science
Uxbridge
Middlesex
UB8 3PH
United Kingdom
T: +44 1895 203397
F: +44 (0) 1895 251686

Abstract

The incidence rate of melanoma skin cancer is increasing faster than any other form of cancer and there is no easy way, without going to a dermatologist, of knowing if you have it. An individual has many dozens of moles on their body, with 1/3rd of those on your back which may be difficult to see. Melanoma has very few symptoms other than a change in the skin lesion's appearance. Examples include being asymmetrical or changing over time. Without expert knowledge it is still very difficult to catch. And as with all cancers, the earlier it is diagnosed the better. Dermatologists are expensive for the NHS and have long waiting times (including other issues).

To address these problems, the approach this project takes is a convolutional neural network, trained on more than 1000 images of melanoma and benign moles each, which is used to predict how likely a new image is to be melanoma or benign. An Android application has also been developed which will embed this model, allowing users to pass their own images through it. As the model is only trained on still shots of skin lesions, it does not take into account it's evolution (change over time) – one of the most important factors. Therefore, the app also allows users to store images of moles, organise them into body parts, and view them at a later date.

The final AI model produced an accuracy of 67.6%, which is far better than a 50/50 guess. The main problem that this project faced was the lack of clinical images (taken without a microscope) of moles. Dermoscopic images (taken with a microscope) thus had to be used which, although it produced a reasonable accuracy, made it possibly ineffective when used with the app and the phone's camera.

Acknowledgements

I'd like to give thanks to Daniel Shiffman who runs The Coding Train YouTube channel that helped me learn a significant amount about TensorFlow.js and CNNs. I'd also like to thank Thomas Coleman for being a great project supervisor and for pushing me out of my comfort zone.

I certify that the work presented in the dissertation is my own unless referenced.

Signature Ben Lamb

Date 25/03/2020

Total Words: 10467

Table of Contents

List of Tables

List of Figures

1 Introduction

What are the main problems our society faces with the current system for identifying skin cancer? Firstly, the waiting times for an appointment with a dermatologist at your local surgery is commonly more than 4-12 weeks (RD & E NHS Foundation Trust, 2020) and diagnosing skin cancer quickly is very important when it comes to survival rate (IMPACT Melanoma, 2020). A faster method is needed.

It can be difficult for people, such as the elderly, disabled, or people that live in remote locations, that struggle to travel to attend regular check-ups. A more accessible method is needed.

As the incidence rate of melanoma increases, more people that have it will need to be diagnosed. This can become taxing on the NHS' resources. Either more money will need to be spent on professionals, or less people will be diagnosed in the same space of time. A cheaper method is needed.

Finally, for an untrained individual, it is very difficult to accurately determine if a mole is benign or cancerous, even using techniques such as the "ABCDE" method (American Cancer Society, 2020). Not only that, but the average person has dozens of moles on their body (SkinVision, 2015) that they may wish to track, that change by mere millimetres over weeks. This is very hard to do efficiently or effectively without any tools. An easier method is needed.

This project aims to attempt to solve most, or all of these issues to some extent by using artificial intelligence within a mobile application, so that the user has the capabilities of an expert in the palms of their hands.

1.1 Aims and Objectives

Aim: to create an Android application that allows a user to take a picture of a mole and receive a classification of benign or melanoma with an accuracy greater than 50%, as well as track multiple moles over a period of time.

Objectives:

- Study academic papers that have previously tried to develop a machine learning algorithm to detect skin cancer in order to discover how to produce the best accuracy.

- Review existing Android applications on the Google Play Store to see the features and performance that they provide, as well as any common downfalls these solutions make so that they may be avoided.
- Use the Scrum agile software development methodology to build the software in working increments.
- Design, develop, and train a machine learning model that classifies images of moles as benign or melanoma with an accuracy of >50% (better than a complete guess).
- Design and develop an Android application that allows users to pass their own images of moles through the machine learning model, as well as store, organise, and track their moles.
- Evaluate the results of the project by using appropriate metrics for determining the accuracy and performance of the machine learning model.

1.1 Project Approach

Research was done to find how this problem has been previously attempted by reading academic papers on skin cancer detection using AI, as well as reviewing the most relevant applications on the Google Play Store. As a web developer with some experience developing mobile applications, less research was required with this side of the project. Reading academic papers showed which machine learning architectures are effective, along with the structure of the algorithms, and any techniques they used to achieve a higher accuracy that previously might not have even been considered for this project (e.g. image pre-processing, dataset manipulation).

By reviewing existing applications on the Play Store, what applications did well and what they did poorly based on the user reviews came to light. The negative parts were then avoided when building this solution, including only the positive parts for a more usable and delightful user experience. As it would be infeasible to work with cancer patients in data collection and requirements analysis, only existing systems were reviewed.

Once the research phase was complete, the Scrum agile software development methodology was then used to iteratively plan and create the software. Sprints of 1 week were done, back to back until the development was complete due to the short amount of time available. This is different to continuous development as it had more structure. After each week, the sprint backlog was re-planned, and a potentially entirely different set of features was implemented than had it all been planned in advance. It also helped manage the time available and the information learned in each sprint was used to guide and prioritise the next sprint plan. Trello was used for both product and sprint backlogs.

Depending on the items being developed, each sprint began with a short design phase. For example, if building a user interface element, the design was drawn out in Adobe XD. Adobe XD is simple yet effective. It's akin to paper prototyping. It's easy to draw a design and quickly redraw the next iteration. A more complex program such as InVision would've be more appropriate if working as a designer within a team, however, the level of detail (more than simply the layout, text, size of the items) is overkill for this project and would've wasted more time than it would've be worth.

On the other hand, when working on the backend of the program, this design phase involved writing pseudocode to help structure ideas for implementing the actual code. Again, pseudocode is a simple yet effective tool for individual development. Flowcharts and UML class

diagrams would be more helpful to convey information to a team member/manager, or for a larger project than this. The pseudocode, comments within the code, and the explanations within this document should be enough to understand the processes.

This was followed by an implementation phase. Test-driven development was not used as it is unnecessary – this app will not be published and/or used by anyone therefore it does not need to be bugless. The focus of this project is the performance of the melanoma-detecting model. The utility of the app comes second. The model was evaluated on the accuracy, alongside other metrics to determine the reliability of the model.

1.2 Dissertation Outline

Chapter 2 discusses how melanoma is identified and existing solutions that have attempted to solve this problem in both machine learning and in Android applications.

Chapter 3 covers in more detail the approach to the solution used by this project given the information discovered by the previous chapter.

Chapter 4 includes the designs of the user interface created with Adobe XD as well as pseudocode written for the algorithms used to train the neural network.

Chapter 5 contains details on the implementation of the Android application created with React Native and the convolution neural network built with TensorFlow.

Chapter 6 covers the evaluation of the neural network.

Chapter 7 concludes the project with a summary of the objectives achieved and to what degree the aim was reached. It also includes suggestions on future work that could be done on this project.

2 Background

2.1 Spotting Skin Cancer

One person dies every hour of melanoma (IMPACT Melanoma, 2020). More people are diagnosed with skin cancer every year in the US than all other cancers combined, and the incidence of melanoma is rising faster than any other form of cancer (IMPACT Melanoma, 2020). If melanoma spreads to the lymph nodes, the 5-year survival rate drops to 62%, and then to just 18% if it spreads to the internal organs. However, if diagnosed early, the survival rate is 99% (IMPACT Melanoma, 2020). This demonstrates the significant importance of an early diagnoses, as well as a tool that would allow a person to do so.

Some people are more at risk than others. In terms of age, melanoma is the 2nd most common type of cancer in young people ages 15-29 (IMPACT Melanoma, 2020). After the age of 50, men are significantly more likely to develop melanoma than women (The Skin Cancer Foundation, 2020). The majority of people that develop melanoma are white men over the age of 55, with the overall average age of diagnoses at 65 (The Skin Cancer Foundation, 2020). Men and women aged 49 and under are both more likely to develop melanoma than any other form of cancer (except for breast and thyroid cancer in women) (The Skin Cancer Foundation, 2020).

Aside from correlations with age, there are many other risk factors. Having just one blistering sunburn or being sunburned more than 5 times in childhood or adolescence both double the chance of developing melanoma later in life (The Skin Cancer Foundation, 2020). Having skin that burns easily, a history of skin cancer, having blond or red hair, and having immune system-suppressing diseases or treatments all increase a person's risk of melanoma (American Cancer Society, 2020).

There are a few other useful facts to know when it comes to detecting melanoma. Only 20-30% of melanomas are found in existing moles, meaning 70-80% appear on normal skin (The Skin Cancer Foundation, 2020). Furthermore, 30% of all melanomas in men are found on their back (IMPACT Melanoma, 2020). This may be useful to inform users to assist in their choice of which moles to analyse/track first, as well as ones they might be concerned about. Lastly, a common practice to distinguish a benign mole from melanoma is to use the ABCDE rule (American Cancer Society, 2020):

A. Asymmetrical
B. Border – are the edges irregular or blurred?
C. Colour – does it have more than one shade of colour, or patches of different colours?

D. Diameter – is it larger than 6mm (roughly the size of a pencil rubber, though some melanomas can be smaller than this)?
E. Evolution – does it change over time?

2.2 Existing Machine Learning Solutions

Google Scholar was used to find academic papers on skin cancer detection using AI. Keywords, such as "AI", "machine learning", "visual recognition", and "computer vision" were combined with "skin cancer" to find relevant papers with an oldest publication year of 2010. Older papers than this may be using outdated techniques and models which may have been surpassed in more recent years. All relevant papers were picked out, and the ones, from which there seemed to be the most information gained or were highly referenced by other papers, were read.

A technique for efficient paper reading was used, whereby a paper is read in 2 passes. In the first pass, the titles, subtitles, abstract, introduction, and conclusion are read. If after this pass the paper seems to be of a reasonable quality and there is information to be gained here that hasn't been already, a second pass is done. In this pass, the rest of the paper is read to understand the specifics of how a model was implemented. Again, notes were made on the key pieces of information to be later referenced.

There are two main aspects to consider when dealing with the existing machine learning solutions for predicting skin cancer: the model itself and the dataset. This section will begin by discussing the dataset side. There are two key types of images that are taken in the diagnoses of skin diseases. The first is clinical images which are taken with a regular camera that anyone might possess. These images may contain parts of the room in the background, and they may vary in terms of lighting, as well as quality and blurring. They are the type of photo that may be taken by a smartphone, and thus the type that is used in this project's app. The second type is dermoscopic images – images taken with a dermatoscope. These images are much more detailed due to the microscope, are evenly lit by the light on the same equipment and contain only the skin lesion and some surrounding skin.

As the majority of the potential users for this mobile application are likely not to have a dermatoscope, it might be ineffective to train the machine learning model on dermoscopic images as it could produce poor results when a clinical image is supplied to the model. More testing needs to be done on this. Many of the existing solutions use dermoscopic images. The reason for this is the availability of data. To provide perspective: the ISIC archive (one of the largest open repositories of dermoscopic images) contains roughly 23,000 images (The

International Skin Imaging Collaboration, 2020). On the other hand, the number of clinical images in most datasets are in the hundreds and there are not many of them. For example, the MED-NODE dataset contains 170 clinical images, and only 70 of those are melanoma (I. Giotis, 2020). For this reason and this reason only, dermoscopic images had to be used to produce any kind of positive result.

Neural networks need large amounts of data to perform with high accuracy, sensitivity, and specificity. Multiple partial solutions to this low data problem exist. The most applicable is image augmentation: translating, rotating, scaling, and cropping an image to generate a new image which can be used in training. It has been shown the same performance as a model trained on ~126,000 images can be achieved using less than 5000 images with an augmentation factor of 24 times and a higher image resolution to allow for that (Fujimoto, 2018).

Another option to assist with the low dataset size is utilising patient demographic metadata such as age and sex to provide a performance boost. One study showed a ~7% improvement in accuracy when including patient metadata in classification (Andre G. C. Pacheco, 2019). An issue with this is the datasets usually do not provide this information. Additionally, transfer learning can be used to improve initial performance, training speed, and the accuracy plateau of a neural network (Fabrizio Nunnari, 2019). A CNN trained on the ImageNet database with 1000 object classes can be retrained on a skin lesion dataset and show a better performance compared to being solely trained on the skin lesions.

Lastly, it is worth noting that most datasets are biased against people with skin types V and VI on the Fitzpatrick scale (people who burn minimally and always tan well). This is due to a lack of data on these skin types (Liu, 2019). Given the scope of this project, not much can be done regarding this problem – it is merely a data issue.

Next, existing solutions for the machine learning model will be discussed. Nearly all the papers read utilise a convolutional neural network (CNN). The most used architectures for the CNNs were Inception (v3, v4, ResNet), ResNet (50, 152), DenseNet (169, 210), and VGG-16 as shown in the table below.

Table 1. Various skin cancer detection models and their performance

Machine Learning Model	Performance	Architecture	Dataset	Reference
CNN	AUC: 94.4% Sensitivity: 85% Specificity: 95% Threshold: 0.5 Test size: 100	DenseNet-169	17,302 images of melanoma and benign	(Cong Tri Pham, 2020)
CNN	AUC: 84.6% Sensitivity: 85% Specificity: 76.2%	VGG-16	ISIC	(Fabrizio Nunnari, 2019)
CNN	AUC: 85%	Various	HAM10000	(Kyle Young, 2019)
CNN	Accuracy: 84.1%	Various	ISIC	(Redha Ali, 2019)
CNN	AUC: 96% Test size: 956	ResNet-152	3,797 images	(Danilo Barros Mendes, 2018)
CNN	AUC: 93.8% AUC: 94.4% AUC: 93.4% AUC: 93.2%	DenseNet-210 ResNet-152 Inception-v3 InceptionResNet-v2	HAM10000	(Amirreza Rezvantalab, 2018)
CNN	Accuracy: 82% Sensitivity: 47% Specificity: 87%	Custom	ISIC	(Sara Nasiri, 2018)
CNN	Jaccard: 77.5%	U-Net	ISIC	(Fred Guth, 2018)
CNN	Accuracy: 86.4%	VGG-16	HAM10000	(Loris Nanni, 2018)
Deep Generative Model	AUC: 86.4%	VAE	ISIC	(Yuchen Lu, 2018)

From this table we can see which datasets are commonly used, namely HAM10000 and ISIC (which contains HAM10000). Each paper evaluated the model in their own way; however,

accuracy, sensitivity, specificity, and AUC (area under the curve – how good the model is at distinguishing between melanoma and benign) seem to be the most used. These metrics are used in the evaluation of this project so that they may be compared.

Aside from the largest problem, which is the amount of training data required for any machine learning model to perform well on diagnosing skin diseases, there is the issue of neural networks being a black box. Dermatologists, and other users that would like a more detailed explanation, are obviously more sceptical of the results and less likely to trust an algorithm that has no explanation at all. In these early stages of skin cancer detection AI, where the models are less than 99% accurate, this may actually a good thing. Users right now should not trust an application on their phone to tell them they don't have cancer, and this should be conveyed to users. The accuracy is simply not good enough to rely on. But in the future, when the performance is better, users will want to know why the AI has determined one way or the other.

One of the ways around this, is to use manual feature extraction. You take a rule, such as the ABCDE rule (asymmetric, border, colour, diameter, evolution), and you use a separate model to extract each element. You then take the output of each model and pass it through a final classifier that categories the skin lesion. The benefit of this approach is that you can display more detail to the user by presenting them with this information (e.g. the mole is asymmetrical, its diameter is larger than 1.5cm, etc.). The downside is the large number of extra models that need to be trained.

An added step for potentially improving the results of the model, is to pre-process the input images (E. Nasr-Esfahani, 2016). Additional machine learning models or traditional algorithms can be used to reduce the number of variables for the classifier to learn. For example, eliminating lighting effects, normalising the colour, removing hairs, removing any vignette effect, and centring or segmenting the lesion. This might be an important step, especially for clinical images.

Of the few papers that discussed the APIs used for the machine learning, nearly all of them used TensorFlow.

2.3 Existing Mobile Applications

The Google Play Store was searched for applications related to skin cancer and 19 were found that attempt to detect it (see references for full list). Of those, 63% of them included an AI feature that predicted if a mole was cancerous. Half of those with the AI feature solely focused

on predicting melanoma, and the other half was trained for more than one type (either unspecified or multiple types).

With almost every application, aside from those with no reviews, users stated that the AI feature did not work, implying a low level of accuracy of the model or poor photography of the mole. In either case, too few training images are likely to blame. The other main shortcomings of these applications were too long of a setup process with inputting user details and logging in, too many bugs and errors, and ugly or difficult to use interfaces.

There were several common features that these applications included that were not previously considered for this project, as well as a couple unique features that may be worth considering when it comes to training the machine learning model. The features that had already been thought of were organising images into folders, organising images on a body-map, and sending notifications periodically to remind the user to take new images. The other features were the ability to load images from the phone's gallery, allowing the user to compare images side by side, and prompting the user to manually crop a new image once they had taken it.

Finally, there were four features that were unique to their applications. The first prompted the user to confirm if the automatic feature extraction steps had correct results (TeleSkin ApS, 2019). This would be less applicable to a neural network as there would be no intermediate steps in the algorithm to display to the user (a neural network is a black box). The second used a more general object recognition model to determine the size of a mole given another object in the same frame (SkinMonitor.org, 2014). Given the complexity of this, using an entire second model for a single feature, and the time available, it has not been used.

The third unique feature was compatibility with a macro lens for taking much more detailed images (DeepVision ML, 2017). A huge portion of the user base would likely not have a macro lens, meaning this would need to be an optional feature and therefore out of the scope of this project. One of the aims of this project is for it to be accessible. The last feature allowed multiple images of the same mole to be passed to the model and the result averaged (CJ63, 2019). This is one that will be considered in this application, but only after one image prediction shows reasonable accuracy. It is a simple step but one that may improve the accuracy of the prediction, even if by a small amount.

3 Methodology

A Scrum board was created on Trello with 6 columns: app product backlog, AI product backlog, sprint backlog, doing, and done. Cards were created based on the initial idea for the solution and were split into the corresponding AI or app backlogs. For example, some cards were “define CNN architecture”, “find melanoma dataset”, and “train model”. At the beginning of each week sprint, cards from each were moved into the sprint backlog. Any progress made was added as notes to the cards as they were moved into the doing and finally the done columns. As time went on, cards in each backlog could be further refined into more detailed features. E.g. “define CNN architecture” would become “define LeNet-5 architecture” and so on. Four one-week long sprints were completed.

The app was designed in Adobe XD. A data driven approach was loosely used when deciding how many screens to use and what to put on each screen. I.e. first a screen describing a singular mole was created being as simple as it could be done. Next, a screen describing multiple moles was built. Each screen was creating using Material design components where something more complex than plain text was needed. Material design is developed by Google who also create Android. Therefore, to have the most native look and feel for an app, it makes sense to use a Material design. After building some screens and seeing them on an actual phone, it was realised that a second iteration was needed to alter the layout or labelling of certain items.

Designs were not essential when it came to the code behind the screens of the app. Most React Native components look roughly the same behind the scenes, with a similar structure and patterns. The most important design for this side was the structure of the data being passed around the app. A small JSON document was created for this. For the machine learning side though, this is not the same. As the JavaScript TensorFlow library is not overly mature, it does not include extensive functionality for things like loading a directory of images. Pseudocode was written for functions that made training the neural network easier, including a data augmentation function and an image resizing function.

The app and model were then written in JavaScript using React Native and TensorFlow respectively. Although larger, more complex and modern CNN architectures performed better than older ones, the LeNet-5 architecture was chosen. It was one of the first CNNs to have the repeating pattern of convolutional layer and pooling layer, with a fully connected layer at the end. Most current CNNs build off this template, and so it is effectively a scaled down version of a

modern CNN but with a size small enough that it could be quickly trained on the machine used in this project (CPU only). Other models also exceeded the memory capacity.

To overcome the largest problem, a lack of data and therefore overtraining of the model, a technique uncovered in the background research called data augmentation was used to generate far more data than is available. Even with this, as there are hardly any clinical images, dermoscopic images had to be used to produce any reasonable accuracy. The largest repository of dermoscopic images, ISIC Archive, was utilised as a result.

Finally, a script was written to automatically evaluate the trained model using several metrics. Accuracy, loss, and the number of each true positive, true negative, false positive, and false negative cases were counted as well. These were then used to calculate the sensitivity, specificity, precision, and recall. It is important to take these into account when it comes to categorising a mole as cancerous – they need to be tuned for the properties you want when building a model for cancer. From this, we can see if the model is predicting more accurately than a complete guess (50% accuracy) and if our aim has been achieved. It will also be evaluated on the AUC (explained within the evaluation chapter) to learn the degree of separability and for comparison to other models.

4 Design

Figure 1. App screen designs

These are the final designs created for the react native application. The first four have been implemented, the "Capture New Image" was later removed, and the last design has yet to be developed. Adobe XD was used to build the designs (Adobe, 2020), making sure the padding and margins were accurate by manually adding them to the x and y coordinates as that is not a built-in feature.

A brief walkthrough of the app: a user begins on the *My Moles* home screen where they can view their saved moles filtered by body part. Selecting a mole takes them to the *My Mole* screen that contains more detail about that mole. From here the user can delete the mole, add a new image, or compare images of the mole. Adding a new image launches the phones default camera app (previously the *Capture New Image* screen) and sends the resultant image to the *Analyse Mole* screen. From this screen they can receive the AIs prediction and can select details for saving the mole. Going back to the *My Mole* screen, pressing the *Compare* button will take the user to the *Compare Mole Images* screen. The home screen also allows a user to take an image of an entirely new mole with the *New Mole* button.

A mobile form of the master-detail interface was chosen to allow the user to get a quick overview of all saved items, and then individually access moles to view them in more detail if they wish to. This would be opposed to something like a card list that would display all information in one list, however, due to the number of moles a person may have (~40 on average), this list would be difficult to search through quickly. A filter option was added at the top of the *My Moles* home screen to further enhance the search speed. The rest of the list design came from Material Design's three-line list item (Material Design, 2020).

The *My Mole* and *Analyse Mole* screen have basic layouts as they are displaying basic information – the simplest route was taken. Again, on the *Analyse Mole* screen, the bottom input fields follow the Material Design to give them that native Android look and feel. The *Capture New Image* screen was later replaced when a module was discovered that would replace the entire screen with a single button, as well as providing more functionality due to it being able to access the phone's default camera app and connect with the phone's gallery all in one.

Lastly, the *Compare Mole Images* screen has two large images top and bottom. They were not positioned side by side as this does not take advantage of the aspect ratio of a phone, and even though it would be more space efficient, it would not allow the user to see the images in the most detail. And then, rather than having multiple screens letting the user pick from a list of images, two horizontally scrolling lists of images were added on each side of the main images, simplifying the overall structure of the app as well as letting the user continuously see which mole image they've picked on the timeline.

Figure 2. JSON data structure

This JSON object shows the structure of the data being passed through the app. It describes a single mole. Each mole has an identifier (e.g. a name), a location on the body so it may be filtered on the main list, and a series of images. Each image has a resource identifier so it can be displayed, a date of when it was taken, and the AI's prediction on this image.

4.1 Android Studio vs Other Frameworks

This decision does not impact the functionality of the project idea, as a result it was chosen to build on the Android platform. For Android, the standard is Android Studio which uses the Java programming language and XML.

Another option is to create the application using an abstracted framework that can be compiled to be used on multiple different mobile platforms. This is useful in the event that you would like to extend a project by publishing it to more than one operating system. The most mature framework that uses web development languages is React Native. One of the main web development languages, JavaScript, is also one of the four languages supported by TensorFlow, the machine learning library used. React Native is trusted and used by very large organisations including Tesla, Uber, and Facebook (React Native, 2020).

There are other existing frameworks such as Ionic and Xamarin. Xamarin applications are written in C#, a language not supported by TensorFlow. Choosing Xamarin would mean having to find a potentially less developed machine learning library or develop in two completely different languages. Ionic is very similar to React Native in that it uses web development languages, however, the output is not compiled to native components on the target platforms. Instead, a web browser is effectively simulated when the mobile application is opened (Kremer, 2019). This can lead to a less native feel for the user. For these reasons, React Native was used.

4.2 Machine Learning Training Scripts

As the app side is built in React Native, which is mostly a form of HTML and CSS, it does not require as much planning as the machine learning side. For the machine learning, pseudocode was written for the most significant functions related to training. A slightly more relaxed version of the Pascal style pseudocode (Wikipedia, 2020) was used as it provides a useful balance between being close to the programming language for conversion but also being easy to read in almost plain English.

```
procedure data_augment (images)
    set augment_rate to 10
    set augmented_images to an empty array

    For each image in images do

        append image to augmented_images

        For i = 0 to augment_rate minus 1 do

            set clone to clone of image

            If random is less than chance then
                crop clone by random_crop

            If random is less than chance then
                flip clone by random_flip

            If random is less than chance then
                rotate clone by random_rotate

            If random is less than chance then
                brighten clone by random_brighten

            If random is less than chance then
                blur clone by random_blur

            append clone to augmented_images

    return augmented_images
end
```

Figure 3. Data augmentation pseudocode

This is the pseudocode for the data augmentation function. It takes an array of images in, and outputs an array of 10x more images with small adjustments applied to each. These adjustments are cropping, flipping, rotating, brightening, and blurring. Perhaps when the machine learning libraries for JavaScript become more mature, this method and the following one will likely be abstracted out and taken care of for us. But for now, this seems to be necessary.

```
procedure resize (images, image_size)
    For each image in images

        set cropped_size to nearest 100 less than or equal to the smaller image dimension

        set x1 to half width minus half cropped_size
        set x2 to half width plus half cropped_size

        set y1 to half height minus half cropped_size
        set y2 to half height plus half cropped_size

        crop_and_resize image to x1, y1, x2, y2 and image_size

    return images
end
```

Figure 4. Image resizing pseudocode

The resize function takes in an array of images and a desired size and returns the resized square images. It does this by punching out the largest square it can, in multiples of 100, out of each image, and then resizing. The cropping is necessary to avoid stretching of any images and the 100 was chosen for simplicity.

```
procedure load_images (dir, start_num, end_num, image_size)
    set image_paths to find_paths in dir from start_num to end_num

    set images to get_images from image_paths

    data_augment images

    set images to tensors

    resize images to image_size

    batch images

    normalise images

    return images
end
```

Figure 5. Image loading pseudocode

The data augmentation function is then used by the *load_images* function which takes a directory, that contains images, and a start and end number used to limit the number of images loaded. Loading too many images at once into memory can cause the program to fail. First an array of image paths is found within the directory, they are then loaded into memory, augmented, converted into tensors for use in TensorFlow, resized for batching, batched into one, and normalised. Functions referenced such as *find_paths* and *get_images* are lower level and built into the core of Node.js as well as the image manipulation library used in this project. Others, such as batching and normalising are part of TensorFlow.

```
procedure get_batch (start_num, end_num)
    set melanoma_images to load_images in melanoma_dir from start_num to end_num
    set benign_images to load_images in benign_dir from start_num to end_num

    For each image in melanoma_images do
        append 1, 0 to labels

    For each image in benign_images do
        append 0, 1 to labels

    set images to melanoma_images plus benign_images

    return images and labels
end
```

Figure 6. Batching pseudocode

Now that the foundation for loading and preparing images is complete, a function for preparing the training batch can be made. This loads images from both categories of data and outputs them along with their one hot encoded label (Sarah L. Harris, 2016).

```
procedure train_model
    load model

    For i = last_trained_image to last_image do

        set images and labels to get_batch from i to i plus batch_size

        train model on images and labels

        save model
end
```

Figure 7. Training pseudocode

Lastly, we have the training function which loops over sets of images and trains the model, saving it after training on a batch rather than right at the end because if something goes wrong during training (e.g. memory becomes depleted), hours of training won't be lost, only minutes.

5 Implementation

5.1 React Native Android Application

Each development sprint was broken up into two parts: the first half of the week was spent developing the React Native Android application, and the second half was spent architecting and training the convolutional neural network using TensorFlow.js.

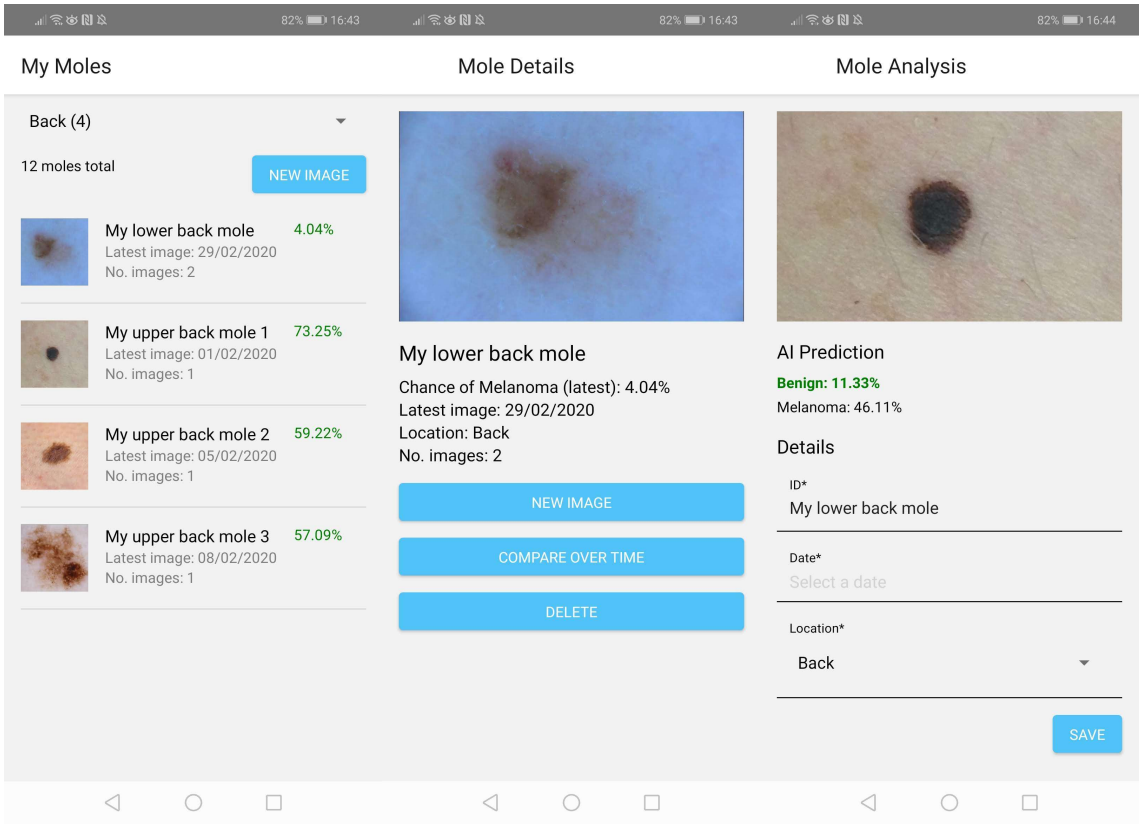


Figure 8. App screenshots. Mole list screen (left), mole detail screen (middle), and analysis screen (right).

The first sprint setup the foundation of the application by installing and running the React Native CLI (React Native , 2020). The screen layouts of the main mole list page and the mole detail page were also implemented; however, these were not functional aside from being able to navigate to each other using the React Navigation module (React Navigation, 2020).

The second sprint added access to the camera as well as the phones gallery, and the passing of data through the screens to make the mole list, and the mole detail screen, functional. Initially a photo capture screen was created during this sprint, however, after developing it a module was found that accessed both the gallery and the phones default camera app which is far more capable than this projects photo screen would have been (e.g. you can easily control the flash and other settings that this projects app would not have had (React Native Community, 2020)).

The last part of the second sprint was a prediction result page, which is navigated to after a new image has been uploaded. The third sprint was a refactoring sprint for the Android app. Bugs were fixed, such as the analysis screen inputs not being pre-filled when adding a new image of an already existing mole. Small changes were made, for example removing the location of the mole from the summary text in the list as the moles are already filtered by location. And common functions/properties from the code were abstracted to reduce duplication.

This is the process that was used to build each screen: the design for the screen was broken down into boxes, containers that when styled with flexbox would produce the look of the design. The JSX was then written (equivalent of HTML) inside the render function of the screen using elements of React Native. Styling would then be added to each element using a React Native stylesheet object (equivalent of CSS) to position and format the elements to match the design. Lastly, placeholder text was replaced by references to the properties of the component, so that data could flow down through this screen and it would be displayed. Any section of the JSX that was too large, or was being used multiple times, was abstracted into its own component. At a later time, methods would be added to the screens component to provide full functionality.

Wrapper screen components were written for some components that weren't styled with the Material design (Material Design, 2020), to provide practice writing apps with React Native instead of using a component library, and to give the app a more professional, native feel. This was done with the buttons, date pickers, and three-line list items, as well as several others.

5.2 React Native Code

In the following screenshots, you may read the comments within the code for more detail into how each section works. In React Native, everything is made of components. This subchapter will explain the structure of a component using examples from many different components so that everything important is covered.

At the top of each component, first React is imported, along with all the built-in components created by React Native. Then third-party components such as Image Pickers are imported. After that any of the components made for this project are imported locally.

```
// Import React Native
import React, { Component } from 'react';
import { View, StyleSheet, ScrollView, Text, Picker } from 'react-native';
import ImagePicker from 'react-native-image-picker';
import AsyncStorage from '@react-native-community/async-storage';

// Import components
import MoleListItem from '../components/MoleListItem';
```

Figure 9. React Native imports

Each component is a class that extends from the React component class.

```
export default class MoleListScreen extends Component {
```

Figure 10. Component definition

The classes usually start with a constructor which takes a *props* parameter ("properties"). This contains any information passed to this component when it is used as a tag in XML in another components render function. The constructor initialises the state – information going to be displayed or used by this component that may change over time.

Props can be directly rendered but not changed by this component. For example, on the analysis screen shown below, the information surrounding a mole is created with default values and it attempts to find actual values if we were updating an existing mole rather than analysing a completely new mole. The *navigation.getParam* function accesses an object similar to *props*, however passed from a screen change rather than when simply rendered as JSX.

```
constructor(props) {
    super(props);

    // Initialize any information that is going to be displayed on this page and might change
    this.state = {
        id: null,
        // Default location is 'back' as this is the most common
        location: 'Back',
        // Receives image from previous screen
        imageUri: this.props.navigation.getParam('imageUri'),
        date: null,
        // AI prediction is random until real AI model is exported into this app
        aiPrediction: [Math.random(), Math.random()]
    };

    // If we're given a mole, load that info as we're updating it instead of creating a new one
    const mole = this.props.navigation.getParam('mole');
    if (mole) {
        this.state = {
            ...this.state,
            id: mole.id,
            location: mole.location
        };
    }
}
```

Figure 11. The analyse screen constructor

Some components need to read data saved on the local storage rather than data that's just being passed around from screen to screen, or from component to component. We detect when the component is mounted (when it's constructed and ready to be displayed) and call to load the data inside a listener. Due to the way the React Navigation system works on mobile devices, when you change page, the previous page is not popped off the history stack and unloaded because you might be going back to that same page soon.

Therefore, you can't rely on the constructor to update the information already on the page because it won't be run again. To get around this, you listen for an event called "did focus". This means the users attention has been brought back to this page. Now that we know when to load the data, we take it all out of Async Storage, a third-party component for storing data to the local storage on the device. The example below is from the home screen which displays all of the moles stored.

```
componentDidMount() {
    // Re-loads the data if the user comes back to this screen to display any changed moles
    this.unsubscribe = this.props.navigation.addListener('didFocus', () => {
        this.loadData();
    });
}

componentWillUnmount() {
    // Have to unsubscribe for memory reasons
    this.unsubscribe.remove();
}

// Loads all moles saved in local storage
async loadData() {
    try {
        const keys = await AsyncStorage.getAllKeys();
        const keyValues = await AsyncStorage.multiGet(keys);

        const moles = keyValues.map(keyValue => JSON.parse(keyValue[1]));
        this.setState({
            loadedMoles: moles
        });
    } catch (err) {
        // If there was an error in any of this, show the user with a toast
        ToastAndroid.show('Error loading moles', ToastAndroid.SHORT);
    }
}
```

Figure 12. Data loading in a component

A component's render function is XML code, similar to HTML but using React Native elements and custom components created in this project (e.g. *MatTextField* as shown at the bottom of the screenshot below). Some notable parts of this screenshot include the ability to place JavaScript expressions inside the XML to display live data. The same can be done for passing properties to components through the XML elements attributes. Even functions can be passed, as seen on the

second from bottom line (*onChangeText* attribute) which is how upstream communication is done (sending data from a child back up to the parent that created it).

```
render() {
    return (
        <View style={style.body}>
            {/* Image of the mole */}
            <Image source={{ uri: this.state.imageUri }} style={style.moleImage} />

            {/* AI prediction results */}
            <View style={style.section}>
                <Text style={style.sectionHeading}>AI Prediction</Text>
                <Text style={style.goodAIPrediction}>Benign: {Utils.toPercentage(this.state.aiPrediction[0])}</Text>
                <Text>Melanoma: {Utils.toPercentage(this.state.aiPrediction[1])}</Text>
            </View>

            {/* Input details for saving */}
            <View style={style.section}>
                <Text style={style.sectionHeading}>Details</Text>

                {/* ID */}
                <MatTextField label="ID" text={this.state.id} onChangeText={text => this.setState({ id: text })}
                    required={true} placeholder="Upper back mole 1" style={style.input} />
```

Figure 13. The analyse screen render function

The following images show some unique code in different components, such as taking an image of and saving a mole.

```
// Saves the inputted data to the local storage and returns to the home screen
async saveData() {
    // If we're missing any information, don't save
    if (!this.state.id || !this.state.location || !this.state.date) {
        return ToastAndroid.show('The ID, date, and location fields are required.', ToastAndroid.LONG);
    }

    const mole = this.createMoleFromState();

    try {
        // Check if this mole already exists under this ID
        const existingMoleJSON = await AsyncStorage.getItem(mole.id);

        if (existingMoleJSON) {
            // Just updates the mole by adding the new image if it does
            this.saveToExistingMole(existingMoleJSON, mole);
        } else {
            // Otherwise it's saved under its ID
            await AsyncStorage.setItem(mole.id, JSON.stringify(mole));
        }

        // Go back to the home screen
        this.props.navigation.navigate('MoleList');

    } catch (err) {
        ToastAndroid.show('Error saving new mole', ToastAndroid.SHORT);
    }
}
```

Figure 14. Saving data function

```
// Loads the third party image picker and sends the result to the analysis screen
addNewImage() {
    ImagePicker.showImagePicker({
        mediaType: 'photo'
    }, res => {
        if (res.didCancel || res.error) return;

        this.props.navigation.navigate('Analysis', { imageUri: res.uri });
    });
}
```

Figure 15. Taking a new image with the camera or from the gallery

This component is a good example of how composition is used rather than inheritance for building up complex structures. Although the *MatInput* inherits from *Component* initially, it will never be further inherited from.

```
// A styled container for any form of input
export default class MatInput extends Component {

    render() {
        return (
            <View style={[style.container, this.props.style]}>
                {/* Can inherit styles given to this component as a label style */}
                <Text style={[style.label, this.props.labelStyle]}>{this.props.label}</Text>

                {/* Contains anything - from text inputs to date pickers */}
                {this.props.children}
            </View>
        );
    }
}
```

Figure 16. The MatInput component (Material Designed Input)

MatInput is then used as a wrapper when building components such as the *MatDatePicker*. The date picker does not inherit from *MatInput* but it does build with it.

```
return (
    // Wrapped in a styled input component
    <MatInput label={this.props.label} required={this.props.required} style={this.props.style} labelStyle={style.label}>
        {
            // The calendar, only shown if it's enabled
            this.state.showCalendar &&
            <DateTimePicker mode="date" display="calendar" value={new Date()} onChange={(e, date) => this.setDate(date)} />
        }
```

Figure 17. The MatDatePicker component

Finally, most components have style to them, and this is defined as a JavaScript object in a format similar to CSS.

```
// Define the style
const style = StyleSheet.create({
    body: {
        padding: 16
    },

    moleImage: {
        // Null lets the image fill the available width
        width: null,
        height: 200,
        marginBottom: 16
    },

    // Uses flex box
    footer: {
        alignItems: 'flex-end'
    },

    goodAIPrediction: {
        marginBottom: 4,
        color: 'green',
        fontWeight: 'bold'
    },
```

Figure 18. React Native styles

5.3 Convolutional Neural Network

The initial plan was to load an Inception-v4 network (Szegedy, et al., 2016) pre-trained on the ImageNet database (Stanford University, 2016) and using transfer learning to train it on a melanoma dataset. However, it seems due to the immaturity of the JavaScript version of TensorFlow, there are no existing Inception-v4s pre-made and uploaded anywhere. Instead, the decision was made to define an architecture from scratch to gain a far deeper understanding of CNNs and TensorFlow. A step down in complexity is the AlexNet model (Krizhevsky, 2012) though, due to the size, the PC it was being trained on quickly ran out of memory and caused the program to crash. Another step down led to the use of the 1998 LeNet-5 architecture (LeCun, 1998). Although it is simple to define and not very resource intensive to train, it is well-known for being one of the first CNNs to use the pattern of repeating convolutional and pooling layers with a fully connected dense layer at the end (a template for nearly all future networks), perfect for learning and working with the available hardware.

The layers of the model were defined according to the original architecture, for example, with a specific number of filters in the convolutional layers, a set kernel size, and a certain number of units in the last dense layers. Training began on a much larger cats and dogs dataset from

Microsoft (Microsoft, 2017) to ensure that the model was correctly defined and working before training on the real data. This revealed two problems: the first was that the validation set accuracy was not improving at all. The second was that the validation set loss was increasing.

```
// Wrapped in a function to allow for easy changing of the inputted image size
function getLeNet5Model(inputShape) {
    const model = tf.sequential();

    // Convolutional layer
    model.add(tf.layers.conv2d({
        filters: 6,
        kernelSize: 5,
        strides: 1,
        activation: 'relu',
        inputShape,
        dtype: 'float32',
        kernelInitializer: 'varianceScaling',
        kernelRegularizer: tf.regularizers.l2({ l2: 0.001 })
    }));

    // Pooling layer
    model.add(tf.layers.averagePooling2d({
        poolSize: 2
    }));

    // Convolutional layer
    model.add(tf.layers.conv2d({···
    }));
```

Figure 19. Part of the LeNet-5 CNN architecture defined in TensorFlow

After a dozen hours of learning about how CNNs work, several changes were made to the architecture. To help improve the initial performance with the accuracy, kernel initialisers were added, set to variance scaling, to all convolutional and dense layers. Variance scaling was chosen as it works well with another change made, the use of the ReLU activation function instead of tahn. ReLU improves the training speed of the network and reduces the vanishing gradient problem by rectifying negative outputs from nodes (Bing Xu, 2015).

The next two changes to the architecture help with over training (assumed from the high validation set loss but low training batch set loss – the model could not generalise to new data well). Two dropout layers in between the fully connected layers were added with a rate of 50%, meaning half of the time these nodes could not be relied on by the network leading to a simpler and more robust model. This has the downside of increasing the amount of training time required by a significant amount.

```
// Returns a single tensor containing all the specified pre-processed images in the directory
// Handles discovering, loading, augmentation, resizing, batching, normalising
async function loadImages(dir, imageSize, startIndex, endIndex, augment = true) {
    console.time('Load Images Duration');
    console.log('--------------------------------');
    console.log(`Loading images from ${dir}`);

    // Discovering files
    console.log('Finding images');
    const imagePaths = await getImagePaths(dir, startIndex, endIndex);
    console.log(`${imagePaths.length} images found`);

    // Loading, augmenting, and converting to tensors
    console.log('Augmenting');
    const augmentedImages = await augmentImages(imagePaths, augment);
    console.log(`${augmentedImages.length} augmented images generated`);

    console.log(`Number of tensors in memory: ${tf.memory().numTensors}`);

    // Resizing images to all one size
    console.log('Resizing');
    const resizedImages = resizeImages(augmentedImages, imageSize);
    augmentedImages.forEach(t => t.dispose());

    console.log(`Number of tensors in memory: ${tf.memory().numTensors}`);

    // Batching into one tensor
    console.log('Batching');
    const batchedTensor = batchTensors(resizedImages);
    resizedImages.forEach(t => t.dispose());

    console.log(`Number of tensors in memory: ${tf.memory().numTensors}`);

    // Normalising pixel data for input into the CNN
    console.log('Normalising');
    const tensor = normaliseImages(batchedTensor);
    batchedTensor.dispose();
```

Figure 20. The image loading and pre-processing code

The second change for overtraining was the use of L2 kernel regularisers. This directly helps with overtraining by keeping the weight values from becoming too large.

A few of the model's hyperparameters were chosen due to hardware restraints. These include the iteration batch size (200), and the batch size (32) used by the internal workings of TensorFlow when calculating weight changes. Numbers much larger than these cause the computer to run out of memory and crash. The loss and optimisation functions were chosen based on a similar task being demonstrated on a Google Codelab tutorial, categorical cross entropy and the Adam optimiser (Google Codelabs, 2020). The number of epochs were chosen after running it for a short period of time and viewing the improvement rate of the batch accuracy and loss. The batch accuracy reached around 90% in 5 epochs, and so the epochs were reduced to that, from 10, to avoid needless overtraining and save time.

```
// Model training hyperparameters
const hyperparams = {
    imageSize: 500,
    originalImagesPerBatch: 20,
    epochsPerBatch: 10,
    modelBatchSize: 32,
    validationSplit: 0.2,
    loss: 'categoricalCrossentropy',
    optimizer: tf.train.adam()
};

// The main training function
async function train() {
    console.time('Model Training Duration');
    // Load the model
    const model = await tf.loadLayersModel(`${modelConfig.saveDir}/${modelConfig.name}.json`);
    compileModel(model);

    // Get an even number of images from both positive/negative image categories
    const cat1Count = await getImageCount(modelConfig.dataAPath);
    const cat2Count = await getImageCount(modelConfig.dataBPath);
    // If we have more of one than the other, use the smaller number
    // So that the model doesn't overtrain on one category
    const maxImageCount = Math.min(cat1Count, cat2Count);
    let iteration = 1;

    // Until there are no more images left
    // Only increase by half a batch as one batch is made up of half of each category
    for (let i = modelConfig.continueFromImageNum; i < maxImageCount; i += (hyperparams.originalImagesPerBatch / 2)) {
        console.log(`\nIteration ${iteration}`);
        iteration++;
        // Get the next batch of images
        const { xs, ys } = await getTrainingBatch(modelConfig.dataAPath, modelConfig.dataBPath, i, i + (hyperparams.originalImagesPerBatch / 2));

        // Train the model
        const trainingResults = await model.fit(xs, ys, {
            shuffle: true,
            epochs: hyperparams.epochsPerBatch,
            batchSize: hyperparams.modelBatchSize,
            validationSplit: hyperparams.validationSplit
        });

        xs.dispose();
        ys.dispose();

        // Save the model
        await model.save(modelConfig.saveDir);
```

Figure 21. The main training method with hyperparameters at the top

The biggest problem that arose in this project was the amount of data for training. As the only existing datasets available for clinical images of melanoma were only in the hundreds, the decision was made to train on dermoscopic images in the hope that it would translate well to clinical images. The largest publicly available archive of dermoscopic images of melanoma was used, the ISIC Archive (ISIC, 2020). Instead of using their API to fetch each image when it was needed for training, the approach that was taken was to download the entire archive (~50GB) and write a script that would use the accompanying simple metadata to extract just the melanoma images and an equal number of benign images.

```
// Automatically runs this function when the file is executed
// Used to extract all melanoma images and an equal number of benign images
// From the ISIC Archive into a separate folder for ease of use
(async function () {
    const imageNames = await getImageNames('nevus', 0, 2168);
    const imagePaths = await getImagePaths(imageNames);
    moveFiles(imagePaths, './ISIC/Nevus');
})();
```

Figure 22. The main ISIC data preparing function

getImageNames reads an Excel spreadsheet. The images from this archive were not grouped into folders by classification but rather by dataset (the archive is made up of a series of other datasets).

```
// Returns the list of filenames from the spreadsheet the ISIC Archive came with
async function getImageNames(diagnosis = 'melanoma', startIndex, endIndex) {
    const csv = fs.readFileSync('./ISIC-images/metadata.csv', 'utf8');

    // Uses a promise to use async await with csvParse
    const requestedRecords = await new Promise((res, rej) => {
        // Converts the CSV to a js array
        csvParse(csv, (err, metadata) => {
            if (err) return rej(err);

            const records = metadata.map(record => {
                // Takes out the filename and the diagnosis
                return {
                    filename: `${record[1]}.jpg`,
                    diagnosis: record[6]
                };
                // Only keeps the requested diagnosis
            }).filter(record => record.diagnosis === diagnosis);

            res(records);
        });
    });
    // Only take the amount of records we want so it doesn't have to be every single file
    return requestedRecords.map(record => record.filename).slice(startIndex, endIndex);
}
```

Figure 23. The method used to extract the images from the ISIC Achieve download

Images needed to be analysed to find the best size to use when passing them through the CNN. A larger image means more data will be used by the CNN to make a prediction. Picking too large a base size means that a lot of the images will need to be upscaled to that resolution and will become very blurred. A method was written to provide details on the sizes of these images, include smallest, largest, average, and the number of images in each set of a series of bucket

sizes (e.g. 100x100, 300x300, 500x500).

```
// The list of sizes to be scanning for
const sizes = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000];
// Stores the counts as an object with a key for each size
const countsAbove = {};
sizes.forEach(size => {
    countsAbove[size] = 0;
});

// Go through each image (as a tensor)
tensors.forEach(tensor => {
    const width = tensor.shape[1];
    const height = tensor.shape[0];

    // Update the min
    if (width < min.width) min.width = width;
    if (height < min.height) min.height = height;

    // Update the max
    if (width > max.width) max.width = width;
    if (height > max.height) max.height = height;

    // Update the individual sizes
    sizes.forEach(size => {
        if (width >= size && height >= size) countsAbove[size]++;
    });
});

// Get rid of tensors we're no longer using for memory purposes
tensors.forEach(t => t.dispose());

// Display the results
console.log(`Min: ${min.width}x${min.height}`);
console.log(`Max: ${max.width}x${max.height}`);
// Mean based on min and max
console.log(`Mean: ${min.width + ((max.width - min.width) / 2)}x${min.height + ((max.height - min.height) / 2)}`);

console.log(`% of images above each size:`);
sizes.forEach(size => {
    console.log(`${size}: ${((countsAbove[size] / tensors.length) * 100).toFixed(1)}%`);
});
```

Figure 24. The image size summary function

```
Finding images
200 images found
Converting images into tensors
200 tensors created
Images Summary
--------------------------------
Min: 679x540
Max: 3872x2592
Mean: 2275.5x1566
% of images above each size:
100: 100.0%
200: 100.0%
300: 100.0%
400: 100.0%
500: 100.0%
600: 86.5%
700: 80.0%
800: 60.0%
900: 53.0%
1000: 52.5%
```

Figure 25. Image dataset size summary

As you can see in this screenshot, all of images in this dataset are at least 500x500 pixels in size. This means that the algorithm will be able to make use of all of the images without scaling any images up and losing detail.

```
function resizeImages(tensors, size) {
    // These two values are used by the final TensorFlow resizing function
    const cropSize = [size, size];
    const boxInd = [0];

    // Turns the 3D tensors into 4D tensors, required for resizing
    const tensors4d = tensors.map(tensor => tf.expandDims(tensor));

    // For each image
    const resizedImages = tensors4d.map(image => {
        const width = image.shape[2];
        const height = image.shape[1];

        // Get the smaller side to the nearest 100 (for simplicity) inside the bounds
        // For finding the largest box that would fit into this image
        let croppedSize = 0;
        if (width < height) {
            croppedSize = Math.floor(width / 100) * 100;
        } else {
            croppedSize = Math.floor(height / 100) * 100;
        }

        // Determine the box's coordinates
        const y1 = (height / 2) - (croppedSize / 2);
        const x1 = (width / 2) - (croppedSize / 2);

        const y2 = (height / 2) + (croppedSize / 2);
        const x2 = (width / 2) + (croppedSize / 2);

        const boxes = [
            [y1 / height, x1 / width, y2 / height, x2 / width]
        ];

        // Crop and scale up/down if needed
        return tf.image.cropAndResize(image, boxes, boxInd, cropSize);
    });
```

Figure 26. The image resizing algorithm

Even with this dataset of ~2000 images for each category, it would still not be enough to produce any form of a trained model. To counter this, data augmentation was implemented. JavaScript does not seem to be very mature in the machine learning world, and there are few image augmentation libraries. The “image-augment” library was tried first, however, its documentation was so poor that it was very difficult to use. In the end, a library called JIMP was used (JavaScript Image Manipulation Program), a standard image manipulation library (Moran, 2020). As it is a general-purpose library and not one that is specifically meant to be used with machine learning (working with thousands of images at a time), it is very slow and a large bottleneck for the training time.

```
const maxCrop = 0.2;
const maxRotation = 45;
const maxBrighten = 0.5;
const maxBlur = 3;

// Returns a list of images X times larger than the paths given
// Augment can be turned off to skip and just load into tensors
// Uses normal for loops to only do one image at a time for limited memory reasons
async function augmentImages(imagePaths, augment = true) {
    if (!imagePaths) return;

    try {
        const augmentedImages = [];

        // For each original image
        for (let i = 0; i < imagePaths.length; ++i) {
            // Load it
            const original = await Jimp.read(imagePaths[i]);

            // Keep a copy of the original
            augmentedImages.push(await jimpToTensor(original));

            if (!augment) continue;

            // For each number of augmentations to make
            for (let j = 0; j < (augmentRate - 1); ++j) {
                if (verbose) console.log(`${i + 1}: ${j + 1}`);
                // Make a copy so we don't change the original
                const image = original.clone();

                // Get random values to augment it by
                const crop = getRandomCrop(image, maxCrop, cropSideChance);
                // Doubled and subtracted by the max to centre it on 0
                // E.g. instead of 0 to 90, it'll be -45 to +45
                const rotation = (Math.random() * maxRotation * 2) - maxRotation;
                const brightness = (Math.random() * maxBrighten * 2) - maxBrighten;
                const blur = Math.ceil(Math.random() * maxBlur);

                // A random check for each augmentation
                if (Math.random() < augmentChance) image.crop(crop.x, crop.y, crop.width, crop.height);
                if (Math.random() < augmentChance) image.flip(Math.random() < flipChance, Math.random() < flipChance);
                if (Math.random() < augmentChance) image.rotate(rotation, false);
                if (Math.random() < augmentChance) image.brightness(brightness);
                if (Math.random() < augmentChance) image.blur(blur);

                // Save this new image
                augmentedImages.push(await jimpToTensor(image));
```

Figure 27. The data augmentation function

The data augmentation used does between 1 and 5 manipulations: cropping, flipping, rotating, brightening, and blurring. It keeps one of the original images and produces 9 altered images, resulting in 10 times the number of images (although this number can be changed very easily). The picture below has an example, and it's clear to see the cropping, flipping, and rotating effects.

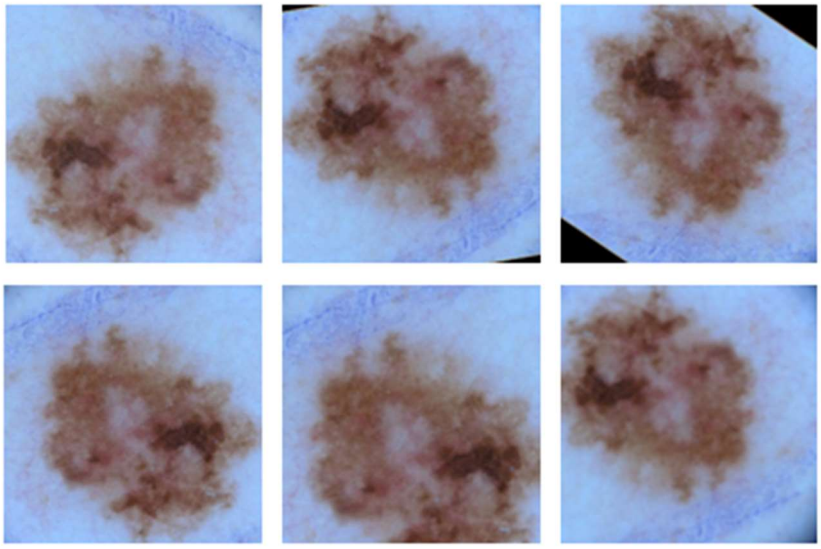


Figure 28. Data augmentation example

Node and TensorFlow only provide functions to load one image at a time and convert it into the data format used by TensorFlow (a tensor). To improve the codebase, I wrote several utility wrapper functions that allowed multiple images to be loaded from a directory and into tensors.

The evaluation function is separated into two, the first passes the test images through the model and stores the results which is quite a slow process. The second function takes the results and calculates and prints the evaluation metrics which is a very fast process. The reason for this is that the threshold for categorising a prediction as one or the other was unknown, so a lot of repeated runs of the evaluation were needed to find the exact threshold. Separating them meant only the fast function needed to be re-run with a different value.

```
// Declare the images to evaluate on
// Images past about 1100 have not been seen by the model
const startIndex = 1500;
const batchSizePerClass = 50;
const totalTestSize = 400;

let accuracies = [];
let losses = [];

// For each batch
for (let j = startIndex; j < startIndex + totalTestSize; j += batchSizePerClass) {
    console.log(`Evaluating images ${j} to ${j + batchSizePerClass}`);
    // Load the images without augmentation
    const { xs, ys } = await getTrainingBatch(modelConfig.dataAPath, modelConfig.dataBPath, j, j + batchSizePerClass, false);

    // Use TensorFlows evaluate method to get the accuracy and loss
    const metrics = model.evaluate(xs, ys, { batchSize: 32 });
    losses.push(metrics[0].dataSync()[0]);
    accuracies.push(metrics[1].dataSync()[0]);
    metrics.forEach(t => t.dispose());

    // Convert the true labels tensor into an array of words
    let trueLabels = tf.split(ys, ys.shape[0]);
    trueLabels = trueLabels.map(y => {
        const intY = y.dataSync().map(prob => Math.round(prob));
        y.dispose();
        return intY[0] === 1 ? 'Melanoma' : 'Nevus';
    });

    const images = tf.split(xs, xs.shape[0]);
    // For each image
    images.forEach((image, i) => {
        // Run it through the model and get a prediction
        const predictionTensor = model.predict(image);
        image.dispose();
        const probabilities = predictionTensor.dataSync();
        predictionTensor.dispose();

        // Save the true label and the probabilities of each category from the prediction
        // Separates the metrics calculations and the length evaluation process
        save({
            trueLabel: trueLabels[i],
            probabilities: [probabilities['0'], probabilities['1']]
        }, './TensorFlow/eval.json');
```

Figure 29. The evaluation function

```
function computeMetrics(predictions, threshold = 0.75) {
    if (!predictions) return {};

    let tp = 0, tn = 0, fp = 0, fn = 0;

    // For each prediction
    predictions.forEach(prediction => {
        let label = 'Nevus';
        const melanomaChance = prediction.probabilities[0];
        const nevusChance = prediction.probabilities[1];
        // If it was more likely melanoma than benign AND that likelihood was greater than the threshold
        // Then categorise it as melanoma
        if (melanomaChance > nevusChance && melanomaChance >= threshold) label = 'Melanoma';

        // Compare this to the true label to get our true/false positive/negatives
        if (label === 'Melanoma' && prediction.trueLabel === 'Melanoma') tp++;
        if (label === 'Nevus' && prediction.trueLabel === 'Nevus') tn++;
        if (label === 'Nevus' && prediction.trueLabel === 'Melanoma') fn++;
        if (label === 'Melanoma' && prediction.trueLabel === 'Nevus') fp++;
    });

    const classCount = predictions.length / 2;
    // Sensitivity = (True +) / (Class +)
    const sensitivity = tp / classCount;
    // Specificity = (True -) / (Class -)
    const specificity = tn / classCount;

    // Precision = (True +) / ((True +) + (False +))
    const precision = tp / (tp + fp);
    // Recall = (True +) / ((True +) + (False -))
    const recall = tp / (tp + fn);

    return {
        sensitivity,
        specificity,
        precision,
        recall,
        // Keep the counts for the confusion matrix
        counts: {
            tp,
            tn,
            fp,
            fn
        },
        threshold
    };
```

Figure 30. The evaluation metrics function

The area under the curve (AUC) of the receiver operating characteristics (ROC) was also found by writing a script that executes the above *computeMetrics* function on increasing values of the threshold. These values were then saved in a comma separated format text file (CSV) and later analysed in Excel. The ROC graph plots the true positive rate (TPR) and the false positive rate (FPR)

```
(function evalAUC() {
    const evalResJSON = fs.readFileSync('./TensorFlow/eval.json', 'utf8');
    const evalRes = JSON.parse(evalResJSON);

    let csv = "";
    // Generate a table of points with an increasing threshold value
    for (let threshold = 0; threshold <= 1; threshold += 0.01) {

        // Compute the TPR and FPR using the confusion matrix values
        const confusionMatrix = computeMetrics(evalRes.results, threshold).counts;
        const tpr = confusionMatrix.tp / (confusionMatrix.tp + confusionMatrix.fn);
        const fpr = 1 - (confusionMatrix.tn / (confusionMatrix.tn + confusionMatrix.fp));

        // Save them in a CSV
        csv += `${threshold},${tpr},${fpr}\n`;
    }

    fs.writeFileSync('./TensorFlow/auc.txt', csv);
})();
```

Figure 31. ROC table generating script

6 Testing and Evaluation

Although this project is made up of two parts, the Android app and the convolutional neural network, it was solely evaluated on the performance of the CNN. The Android app is a key part of the project as it allows a user access to the AI and it covers the "evolution" tracking aspect of a skin lesion, an important factor when diagnosing melanoma. However, due to time constraints, the evaluation will only cover the far more technically challenging aspect, the machine learning model.

The model was evaluated on several metrics using the holdout method. As the model was only trained on half of the data (2168 images), there was more than enough data to use for testing. It was decided to use an 80/20 split and use 20% of the dataset for evaluation, which is roughly 800 images or 400 per class (Melanoma/benign). This is a standard amount used by many other machine learning models, along with other common splits being 70/30 and 60/40 (racheldraelos, 2019).

After evaluating this model, it is surprising to find it has a relatively high accuracy given the problem. Its accuracy is 67.6%. This is 17.6% better than a complete guess. An important thing to note is that this model's final layer uses SoftMax activation, meaning it doesn't output a binary yes or no, it outputs a probability that it is one class or another. The reason for this is that it leaves the final decision up to the user. It doesn't explicitly tell the user they have cancer; it simply gives them a high likely hood of it. Therefore, this adds another variable when it comes to converting it into a binary classification when evaluating it – a threshold of confidence in the probabilities. If the model were just evaluated on the accuracy, it would look as though it is performing quite well. However, a confusion matrix provides more information.

To speed up the process of evaluating the model, the prediction results, along with their true labels for each tested image, were saved to a file. Instead of rerunning each image through the network every time a variable was changed to see how the results would be affected, this file could then be read and evaluated which shorted the evaluation time from around 30 minutes per trial to 2-3 seconds.

Then various values for the threshold of confidence were tried to find a confusion matrix with the most evenly spread values. The thresholds tested were 0.5, 0.6, 0.7, and 0.8. It was then narrowed down to precisely 0.74763 using a binary search method when choosing these values. E.g. if the number of false positives was too large, it was too sensitive, and the threshold was

lowered. The threshold has a direct and significant effect on every metric except from loss and accuracy which are determined by the inner workings of TensorFlow's evaluate method.

More insight can be gained by using the number of true/false positives/negatives to build a confusion matrix, aside from simply the overall accuracy. As you can see from the table below, the model is correctly identifying a significant number of the Melanoma and benign images and the table is very evenly balanced due to the precise threshold found. It's sensitivity (74.8%), specificity (75%), precision (74.9%), and recall (74.8%) are all very close. It's evenly balanced. The extra information that can be gained from this table now is that it simply needs more training to catch the other cases and perhaps a more complex architecture if it began to plateau.

Table 2. Confusion matrix with confidence threshold of 0.74763

		Actual	
		Melanoma	Benign
Predicted	Melanoma	299	100
	Benign	101	300

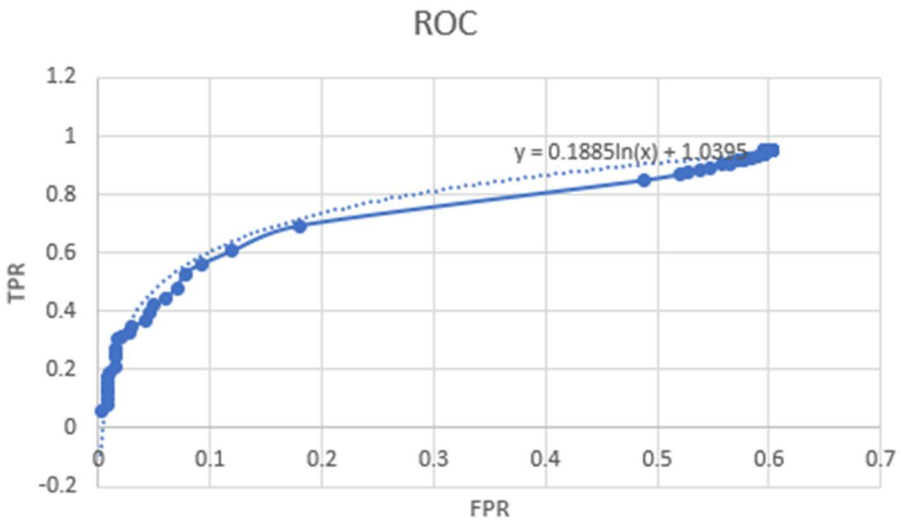


Figure 32. ROC graph

The true positive rate (FPR = Sensitivity) and false positive rate (FPR = 1 – Specificity) were then computed across a range of threshold values from 0 to 1 in increments of 0.01 and plotted in Excel. This produces the receiver operating characteristics graph (ROC) from which the area under the curve (AUC) can be calculated for comparison to the other models found in the background chapter. In Excel, a trendline, that matches the points as closely as possible, was overlaid on the graph, showing its equation. This can be used to easily give a close approximation for the AUC. The equation was put into a simple online calculator (EMathHelp,

2020) to give us an AUC value of 85.2%. This tells us how good the model is at distinguishing between melanoma and benign. A value of 85.2% shows that it is comparable in this performance to those papers found in the background research, with a high degree of separability.

Another question that could be asked is: do we want the model to be this balanced? For example, lowering the threshold to 0.5, meaning it only has to be half confident that it is cancer, increases the amount of cancerous cases it catches to 95.5% (sensitivity). This is obviously a good thing; however, it comes with a trade-off with specificity. It would also predict a large number of false positive cases as the model is so easily triggered into outputting the Melanoma label. It would have a specificity of just 39.8%. Another way to think of it would be in terms of precision and recall. With a threshold of 0.5, it would identify 95.5% of Melanoma cases but in doing so, it would also bring up a lot of false positive cases, with a precision of 61.3%. With this threshold, it would catch far more cancer cases, but it would also tell a lot of people that don't have cancer, that they do.

You could also go in the other direction and raise the threshold to 0.8. This would have the opposite effect, meaning it would be much more precise (88.1%) and specific (94%). When it decides an image is Melanoma, it is usually right. Still, it would select far fewer images with a sensitivity of just 44.5%. This would mean that it might tell people that actually have cancer, that they don't. It would always be better to scare someone, who's concerned about a mole anyway, into visiting a dermatologist rather than let them believe they're completely fine. Even if it would be a waste of time, it's better safe than sorry. In an ideal world, it would be both sensitive and specific. This is why such a precise threshold has been chosen.

The CNN has a high loss value of 59%. It is likely caused by underfitting due to the many overfitting countermeasures introduced to the network. For example, L2 regularisation was used which punishes the network for having high weight values, making it more difficult to fit too closely to a smaller dataset. However, this by default means that the loss value will be higher. This, in combination with the dropout layers used and data augmentation, have potentially resulted in underfitting. To fix this, the strength of each countermeasure could be reduced. E.g. the L2 value could be reduced from 0.001 to 0.0005 and the dropout rate from 50% to 35%. With re-training and more training, the loss could be reduced along with the underfitting.

7 Conclusions

7.1 Objectives

Study academic papers that have previously tried to develop a machine learning algorithm to detect skin cancer in order to discover how to produce the best accuracy.
More than 10 relevant and recent papers on skin cancer detection using artificial intelligence were read and it was found that the large majority of them used a convolutional neural network model. A lot of information was also uncovered regarding the available data of melanoma images, their different types, and several techniques on how to deal with this such as data augmentation and image pre-processing.

Review existing Android applications on the Google Play Store to see the features and performance that they provide, as well as any common downfalls these solutions make so that I may avoid them.
The description, images, and reviews of 19 of the most relevant applications on the Google Play store were analysed. The common features, as well as more unique ones, were taken into consideration when building the app for this project. The minimum number of features were then implemented for the minimum viable product of this app.

The frequent negative reviews were used to avoid similar mistakes. For example, a long start-up and requiring unnecessary amount of user input. This app has virtually no start-up and no user details required before the functionality can be used. Another negative comment on other apps were the ugly or difficult to use interfaces. This app uses material design, and a small number of screens, reducing the likelihood that the UI will be inherently ugly or difficult to use.

Use an agile software development methodology to build the software in working increments.
Scrum was used and tracked using Trello as a scrum board. Four week-long sprints were completed, spending the first half of the week on the Android app, and the second half of the week on the AI. Each week, items from the product backlog were moved into the sprint backlog and they were used to keep track of what was complete.

Design, develop, and train a machine learning model that classifies images of moles as benign or melanoma with an accuracy of >50% (better than a complete guess).
A famous architecture for a convolutional neural network, LeNet-5, was implemented with custom changes to enhance the performance (due to its outdated design). It was trained on 1000 unique images of melanoma and benign skin lesions, and the evaluation showed it had an accuracy of 67.6% which is greater than 50%.

Design and develop an Android application that allows users to pass their own images of moles through the machine learning model, as well as store, organise, and track their moles.
An Android application was designed with Adobe XD and developed with React Native, however, it does not currently allow users to pass their images through the machine learning model due to time limitations (having the model work was a higher priority). Right now, it simply generates a random value as dummy data in place of a real prediction. It does on the other hand, allow the user to store, organise and track their moles.

Evaluate the results of the project by using appropriate metrics for determining the accuracy and performance of the machine learning model.
The model was evaluated using multiple metrics including accuracy, loss, a confusion matrix, sensitivity, specificity, precision, and recall. This allowed further insight into the performance of the model rather than just the accuracy. For example, depending on the confidence threshold, the same model might have a very high sensitivity but a very low specificity.

Aim: to create an Android application that allows a user to take a picture of a mole and receive a classification of benign or melanoma with an accuracy greater than 50%, as well as track multiple moles over a period of time.
An Android app was created that allows users to take a picture of a mole, but it is not currently linked with the machine learning model to produce a prediction. Even if it were, as previously stated, the model is only trained on dermoscopic images of melanoma and likely would not perform well on clinical images of melanoma. However, this would need to be tested, as well as tested on clinical images that have been manually or automatically cropped to produce images more similar to the dermoscopic images the model has been trained on.

Regardless, the model can provide a prediction on how likely a given mole is to be benign or melanoma with an accuracy of 67.6% which is significantly greater than the 50% this project was aiming for. The app can also be used to track moles over a period of time as it is fundamentally an image storage app with a focus on moles. Features such as a screen to compare two images of the same mole from different dates would make tracking a mole over time easier, but it is effectively possible with the current app.

Although the aim has not technically been met, the app has come very far (~95% complete for the minimum viable product) and the model has come further than anticipated, even if it was trained on a dataset that might not be compatible with the apps original purpose.

7.2 Future Work

Had there been more time, this project would benefit from having several of these additions. Firstly, on the app side, an obvious first step would be to embed the AI model within the app so that pictures could receive a prediction. Next, a screen that allows you to view older images of a mole, as well as compare two images of the same mole side by side to allow the user to take manual comparisons of the evolution of a mole. This would likely be the most effective part of the app in terms of practical application until AI models reach a much higher accuracy on clinical images. One other add-on to the app would be a body map that displayed moles on an image of a body rather than simply a drop down (or in combination with it) to make it more intuitive for beginners to use.

The most obvious improvements that could be made to the AI model would be to use a more complex CNN by either using expert knowledge to re-create the modern Inception-v4 networks, or to wait until the libraries mature and someone uploads a pre-built one to be used for training. This would also require a more powerful computer though due to the increased size of the number of the learnable parameters (Inception-v4 has roughly 700 times more parameters than the LeNet-5 used here).

Next, pre-processing of the input images could be done that might improve the performance of the network. For example, making sure the lighting is the same for each image – lightening dark images, darkening light images, etc. Removing shadows and even hairs using other existing algorithms/AI models might also help.

Finally, models trained on dermoscopic images should be tested against clinical images. They would likely perform very poorly when unedited as they would contain things in the background like the room that dermoscopic images do not have at all. It would then be useful to test if any improvements are made by manually cropping the images. If it did, automatic cropping could be used to isolate the mole from the image so that the user doesn't have to do it themselves.

8 References

Adobe, 2020. *XD.* [Online]
Available at: https://www.adobe.com/products/xd.html
[Accessed 27 1 2020].

American Cancer Society, 2020. *Risk Factors for Melanoma Skin Cancer.* [Online]
Available at: https://www.cancer.org/cancer/melanoma-skin-cancer/causes-risks-prevention/risk-factors.html
[Accessed 25 3 2020].

American Cancer Society, 2020. *Signs and Symptoms of Melanoma Skin Cancer.* [Online]
Available at: https://www.cancer.org/cancer/melanoma-skin-cancer/detection-diagnosis-staging/signs-and-symptoms.html
[Accessed 25 3 2020].

AmiciiCompany, 2019. *Visus: Skin Cancer Detection.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.companyname.XRayAnalysis
[Accessed 25 3 2020].

Amirreza Rezvantalab, H. S. S. K., 2018. Dermatologist Level Dermoscopy Skin Cancer Classification Using Different Deep Learning Convolutional Neural Networks Algorithms. *arXiv,* p. 15.

Andre G. C. Pacheco, R. R. K., 2019. Recent Advances in Deep Learning Applied to Skin Cancer Detection. *arXiv,* p. 8.

Bing Xu, N. W. T. C. M. L., 2015. Empirical Evaluation of Rectified Activations in Convolution. *arXiv,* p. 5.

Boreal Open Systems S.L., 2018. *Molexplore "Skin Cancer App".* [Online]
Available at: https://play.google.com/store/apps/details?id=com.borealos.medical.molexplore
[Accessed 25 3 2020].

Cancer Research UK, 2020. *Melanoma skin cancer statistics.* [Online]
Available at: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/melanoma-skin-cancer
[Accessed 25 3 2020].

CJ63, 2019. *Deep Learning for Melanoma.* [Online]
Available at:
https://play.google.com/store/apps/details?id=chiranjeewee.deeplearningformelanoma
[Accessed 25 3 2020].

Cong Tri Pham, M. C. L. D. Y. H. A. D., 2020. AI outperformed every dermatologist.... *arXiv,* p. 21.

Danilo Barros Mendes, N. C. d. S., 2018. Skin Lesions Classification Using Convolutional Neural Networks in Clinical Images. *arXiv,* p. 11.

DeepVision ML, 2017. *MoleScreener.* [Online]
Available at: https://play.google.com/store/apps/details?id=ml.deepvision.molescreener
[Accessed 25 3 2020].

E. Nasr-Esfahani, S. S. N. K. S. S. M. J. K. W. K. N., 2016. Melanoma detection by analysis of clinical images using convolutional neural network. *IEEE,* p. 7.

EMathHelp, 2020. *Area Between Curves Calculator.* [Online]
Available at: https://www.emathhelp.net/calculators/calculus-2/area-between-curves-calculator
[Accessed 26 3 2020].

Fabrizio Nunnari, D. S., 2019. A CNN toolbox for skin cancer classification. *arXiv,* p. 8.

Firma Technologies, 2019. *Derma Analytics - Analyse for Skin Cancer.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.firmagroup.derma
[Accessed 25 3 2020].

Fred Guth, T. E. d., 2018. Skin lesion segmentation using U-Net and good training strategies. *arXiv,* p. 5.

Fujimoto, Y. F. Y. O. Y. O. Y. N. R. F. Y. I. R. W. N. O. K. O. M., 2018. Deep-learning-based, computer-aided classifier developed with a small dataset of clinical images surpasses board-certified dermatologists in skin tumour diagnosis. *British Journal of Dermatology,* p. 9.

GBASOFT, 2019. *Melanoma Detection.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.bakten.skincancer
[Accessed 25 3 2020].

GeniaLabs, 2020. *DermIA Pro - Analyze Skin Cancer with your camera.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.genialabs.dermia
[Accessed 25 3 2020].

Google Codelabs, 2020. *TensorFlow.js — Handwritten digit recognition with CNNs.* [Online]
Available at: https://codelabs.developers.google.com/codelabs/tfjs-training-classfication/index.html#4
[Accessed 7 2 2020].

I. Giotis, N. M. S. L. M. B. M. J. N. P., 2020. *Dermatology database used in MED-NODE.* [Online]
Available at: http://www.cs.rug.nl/~imaging/databases/melanoma_naevi/
[Accessed 11 3 2020].

IMPACT Melanoma, 2020. *Melanoma Facts and Statistics.* [Online]
Available at: https://impactmelanoma.org/learn/melanoma-facts-and-statistics/
[Accessed 25 3 2020].

ISIC, 2020. *The International Skin Imaging Collaboration.* [Online]
Available at: https://www.isic-archive.com/#!/topWithHeader/wideContentTop/main
[Accessed 14 2 2020].
Kremer, M., 2019. *Ionic vs. Everyone: Comparing Cross-Platform Frameworks.* [Online]
Available at: https://ionicframework.com/resources/articles/ionic-vs-react-native-a-comparison-guide
[Accessed 26 3 2020].
Krizhevsky, A., 2012. ImageNet Classification with Deep Convolutional. p. 9.
Kyle Young, G. B. B. S. R. D. S. S., 2019. Deep neural network or dermatologist. *arXiv,* p. 8.
LeCun, Y., 1998. Gradient-Based Learning Applied to Document Recognition. p. 46.
Liu, Y., 2019. Using Deep Learning to Inform Differential Diagnoses of Skin Diseases. *Google AI Blog,* p. 5.
Locher, M., 2019. *Smart Skin Cancer Detection.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.martinlocher.smartskin
[Accessed 25 3 2020].
Loris Nanni, A. L. S. G., 2018. Ensemble of Deep Learned Features for Melanoma Classification. *arXiv,* p. 11.
MasseranoLabs LLC, 2017. *MySkinPal - Skin Cancer App.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.masseranolabs.myskinpalpro
[Accessed 25 3 2020].
Material Design, 2020. *Components.* [Online]
Available at: https://material.io/components/
[Accessed 27 1 2020].
Material Design, 2020. *Lists.* [Online]
Available at: https://material.io/components/lists/#specs
[Accessed 27 1 2020].
MetaOptima Technology Inc, 2020. *MoleScope.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.molescope
[Accessed 25 3 2020].
Microsoft, 2017. *Kaggle Cats and Dogs Dataset.* [Online]
Available at: https://www.microsoft.com/en-us/download/details.aspx?id=54765
[Accessed 6 2 2020].
Miiskin , 2020. *Miiskin - Melanoma Skin Cancer.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.miiskin.android
[Accessed 25 3 2020].

Moran, O., 2020. *Jimp.* [Online]
Available at: https://www.npmjs.com/package/jimp
[Accessed 12 2 2020].
Munteanu, C., 2016. *SpotMole.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.spotmole
[Accessed 25 3 2020].
Nature Healthy Care, 2018. *Skin Cancer.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.Prevent.Skin.Cancer.Steps
[Accessed 25 3 2020].
Pulsar d.o.o. Croatia, 2019. *Skin Cancer Risk Prediction.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.bamblek.skin_cancer
[Accessed 25 3 2020].
racheldraelos, 2019. *Best Use of Train/Val/Test Splits, with Tips for Medical Data.* [Online]
Available at: https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/
[Accessed 26 3 2020].
RD & E NHS Foundation Trust, 2020. *Dermatology - Information for Patients.* [Online]
Available at:
https://www.rdehospital.nhs.uk/patients/services/dermatology/Info_patients.html
[Accessed 22 1 2020].
React Native , 2020. *Getting Started.* [Online]
Available at: https://reactnative.dev/docs/getting-started
[Accessed 27 1 2020].
React Native Community, 2020. *React Native Image Picker.* [Online]
Available at: https://github.com/react-native-community/react-native-image-picker
[Accessed 4 2 2020].
React Native, 2020. *Who's using React Native?.* [Online]
Available at: https://reactnative.dev/showcase
[Accessed 26 3 2020].
React Navigation, 2020. *Getting started.* [Online]
Available at: https://reactnavigation.org/docs/getting-started
[Accessed 28 1 2020].
Redha Ali, R. C. H. M. S. D. S. T. M. K., 2019. Skin Lesion Segmentation and Classification for ISIC 2018 by Combining Deep CNN and Handcrafted Features. *arXiv,* p. 4.
Sara Nasiri, M. J. J. H. M. F., 2018. Deep-CLASS at ISIC Machine Learning Challenge 2018. *arXiv,* p. 4.

Sarah L. Harris, D. M. H., 2016. *One-Hot Encoding.* [Online]
Available at: https://www.sciencedirect.com/topics/computer-science/one-hot-encoding
[Accessed 26 3 2020].
SkinMonitor.org, 2014. *Skin Monitor: check your moles.* [Online]
Available at: https://play.google.com/store/apps/details?id=org.skinmonitor
[Accessed 25 3 2020].
SkinVision B.V., 2020. *SkinVision - Detect Skin Cancer. Track your Moles.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.rubytribe.skinvision.ac
[Accessed 25 3 2020].
SkinVision, 2015. *Our guide to calculating the number of moles on your body.* [Online]
Available at: https://www.skinvision.com/articles/our-guide-to-calculating-the-number-of-moles-on-your-body/
[Accessed 26 3 2020].
Skinzy Software Solutions Pvt. Ltd, 2020. *DermaPhoto by Skinzy: Skin Disease Prediction.* [Online]
Available at: https://play.google.com/store/apps/details?id=skinzy.dermaphoto10.model1
[Accessed 25 3 2020].
Stanford University, 2016. *ImageNet.* [Online]
Available at: http://www.image-net.org/
[Accessed 4 3 2020].
Szegedy, C., Ioffe, S. & Alemi, A., 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv,* p. 12.
TeleSkin ApS, 2019. *skinScan.* [Online]
Available at: https://play.google.com/store/apps/details?id=com.teleskin.skinscan
[Accessed 25 3 2020].
The International Skin Imaging Collaboration, 2020. *The International Skin Imaging Collaboration.* [Online]
Available at: https://isic-archive.com/
[Accessed 11 3 2020].
The Skin Cancer Foundation, 2020. *Skin Cancer Facts & Statistics.* [Online]
Available at: https://www.skincancer.org/skin-cancer-information/skin-cancer-facts/
[Accessed 25 3 2020].
Wikipedia, 2020. *Pseudocode.* [Online]
Available at: https://en.wikipedia.org/wiki/Pseudocode#Syntax
[Accessed 26 3 2020].

Yuchen Lu, P. X., 2018. Anomaly Detection for Skin Disease Images Using Variational Autoencoder. *arXiv,* p. 8.

Appendix A Personal Reflection

A.1 Reflection on Project

If this project were to be repeated, there are several things that could be done differently. The main thing would be not creating a mobile app. Although this a valuable and practical skill to learn, in terms of project depth, this may have hindered it. This project could've had more depth and less breadth. The time taken creating an Android app could have been spent on improving the machine learning aspects. Another reason for this idea comes from the evaluation of the project. The two parts were too separate and distinct and the evaluation of both using two different methods felt like too much of an undertaking. This therefore sacrificed the evaluation of the application side making it hard to tell what value the app really had.

Without the need to develop an application, that time could have been spent learning the syntax of the Python programming language. Python has many more mature libraries for machine learning, improving development time as well as performance of the final model. The faster libraries would have also provided more time to train on the entire dataset (still using the holdout method for evaluation), producing a better model all around (~25% of the data was not used). To be specific, TensorFlow, for example, has more useful functionality on the Python library than the JavaScript library. It has more methods built in for image manipulation and data augmentation that otherwise had to be custom written for this project. These custom scripts built on top of non-machine learning image manipulation libraries that were therefore very slow and required further extension to work with TensorFlow.

Next, if TensorFlow was still being used when redoing this project, it would be far, far quicker to use a computer running a Linux OS. This could be done by either dual booting a PC or by using a completely separate computer. Running on Linux gives TensorFlow access to the graphics processing unit (GPU), increasing the training speed by around 10x. Many hours were spent training a model, looking at the results and adjusting the hyperparameters to improve performance. Because it was so slow to train, it sometimes took a while before any problems became apparent and training had to be restarted with the adjustments. With a faster machine, this could be greatly reduced.

This may be "scope creep", however, it would be interesting to test clinical images against a model trained on dermoscopic images. Further tests could be done, such as seeing if cropping the images improved the predictions, or if some sort of segmentation of the mole could be done. Transfer learning would also be interesting: firstly, training on dermoscopic and then using

transfer learning to train on clinical images. Image pre-processing techniques could also be added to find if they altered the accuracy at all.

Finally, it would have been much better to have first learned a standardised technique for discovering journal papers around a given subject. The method used by this project was simply to search for them on Google Scholars and arXiv and select what seemed as the most recent and relevant ones. This has the problem of not knowing how fundamental a paper is to the given field. By finding it directly on a search, there are not many ways of knowing how well critiqued a paper is or how many times it is referenced by other papers. All there is to go off is how well written the paper is and if the logic used is reasonable. A search method should have been used that was more efficient and found the best papers to have read. The papers searched for were also specifically regarding skin cancer, when perhaps other skin diseases might have had useful papers whose principles transferred to skin cancer.

It might have been helpful to do research iteratively in the sprints as the design and implementation were. This would have provided more relevant specific details to the item being worked on that day rather than research being used to get an overall idea of what should be done.

A.2 Personal Reflection

The biggest change I would make personally to the way I worked on this project would be to start much earlier. I started development in late January, giving me only two months rather than five or six months if I had started at the beginning of this year of university.

Appendix B Appendices

The appendix material submitted on WISEFlow include:

- All of the software – the React Native app and the machine learning code within the TensorFlow folder. This includes raw results from the AI model evaluation. (3.4MB)
- A video demo of the React Native app (3.4MB)
- A copy of the trained LeNet-5 ISIC TensorFlow model (106.6MB)

B.1 BREO

College of Engineering, Design and Physical Sciences Research Ethics Committee
Brunel University London
Kingston Lane
Uxbridge
UB8 3PH
United Kingdom

www.brunel.ac.uk

20 November 2019

LETTER OF CONFIRMATION

Applicant: Mr Benjamin Lamb

Project Title: Skin Cancer Mole Analyser

Reference: 19261-NER-Nov/2019- 21155-1

Dear Mr Benjamin Lamb

The Research Ethics Committee has considered the above application recently submitted by you.

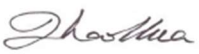
The Chair, acting under delegated authority has confirmed that on the basis of the information provided in your application, your project does not require ethical review.

Please note that:

- Approval to proceed with the study is granted providing that you do not carry out any research which concerns a human participant, their tissue and/or their data.
- The Research Ethics Committee reserves the right to sample and review documentation relevant to the study.
- If during the course of the study, you would like to carry out research activities that concern a human participant, their tissue and/or their data, you must inform the Committee by submitting an appropriate Research Ethics Application. Research activity includes the recruitment of participants, undertaking consent procedures and collection of data. Breach of this requirement constitutes research misconduct and is a disciplinary offence.

Good luck with your research!

Kind regards,



Professor Hua Zhao

Chair of the College of Engineering, Design and Physical Sciences Research Ethics Committee

Brunel University London