Development of a Web App for Skin Alteration Classification Using Machine Learning

SkinAI: A Machine Learning-Powered Web App for Dermatological Classification

Universitat Oberta de Catalunya

Student

Maria Lucas Gascón

MU Bioinf. i Bioest.
Statistical Bioinformatics and Machine Learning

TF tutor

Romina Astrid Rebrij

Responsible teacher

Carles Ventura Royo

UNIVERSITAT DE BARCELONA

January 16, 2024

FINAL PROJECT'S SHEET

Project's title:	*Development of a Web App for Skin Alteration Classification Using Machine Learning*
Author's name:	*Maria Lucas Gascón*
Consultant's name:	*Romina Astrid Rebrij*
PRA's name:	*Carles Ventura Royo*
Delivery date:	*01/2024*
Program:	*Master's degree in Bioinformatics and Biostatistics UOC-UB*
Final project's area:	*Statistical Bioinformatics and Machine Learning*
Language:	*English*
Keywords:	*Machine Learning, Dermatology, Diagnosis*

Abstract

The aim of this master's degree final project is to create a user-friendly web application supported by a robust, non-supervised machine learning algorithm. This algorithm accurately classifies various skin alterations based on user-submitted images, addressing the critical need for accessible tools in dermatology with the goal of globalizing skin condition diagnosis.

Our approach involves implementing Python, exploring a custom convolutional neural network built from scratch, and utilizing transfer learning with established models like DenseNet121 and ResNet50. This comprehensive exploration enables us to identify the most effective method for precise skin alteration classification, with the final implemented algorithm achieving an accuracy rate of 87% on the test set. The web application is built using the Django framework to ensure a seamless user experience.

The primary focus is on demonstrating the successful implementation of an operational and user-friendly method for classifying skin alterations through image analysis. The web application's intuitive interface simplifies image uploading and provides real-time classification results.

In summary, this project significantly contributes to healthcare technology by promoting early detection and empowering individuals to seek timely medical guidance. The open-access nature of the web app enhances global awareness of skin health, making dermatological expertise more accessible.

Index

1. Introduction

1.1 General description

This project focuses on developing a practical and user-friendly web application equipped with a powerful machine learning algorithm, specifically a Convolutional Neural Network (CNN). The goal is to classify a diverse range of skin lesions based on user-submitted images, aiming to revolutionize dermatology by providing an efficient and accessible means for individuals to receive preliminary assessments of their skin conditions, irrespective of their geographical location or socioeconomic status.

Key components

1. **Machine learning algorithm (CNN):** At the project's core is a robust machine learning algorithm developed in Python. The algorithm is meticulously designed to analyze and classify skin lesions, drawing from a diverse dataset and employing various machine learning techniques to ensure accuracy and reliability.
2. **Web application:** The project involves creating an intuitive web interface using the Django framework, facilitating seamless user interaction. Users can effortlessly upload images of their skin conditions through the web app and receive real-time classification results.
3. **Accessibility:** A primary objective of the project is to eliminate entry barriers. The web app is accessible to anyone with an internet connection, making it available to a global audience, including regions with limited access to dermatological expertise.
4. **Education and awareness:** Beyond classification, the web app also serves as an educational platform. Users can access information about various skin conditions, fostering awareness and encouraging proactive skin health practices.

1.2 Context and justification

Dermatology

Dermatology, the branch of medicine specializing in the study, diagnosis, and treatment of skin disorders and conditions, plays a pivotal role in healthcare worldwide. The skin, as the body's largest organ, serves as a protective barrier between the body's internal systems and the external environment. It encompasses a vast spectrum of issues, ranging from common and benign ailments to severe and life-threatening conditions, necessitating thorough examination and diagnosis for effective treatment [1].

Skin cancer has emerged as one of the most prevalent and concerning forms of cancer in recent times [2]. Given that the skin is the body's largest organ, it comes as no surprise that skin cancer ranks as one of the most commonly diagnosed types of cancer among humans [3]. Broadly categorized into melanoma and non-melanoma skin cancer, these conditions demand our utmost attention [4]. In

the realm of skin cancer management, the crux lies in the early detection of these conditions. Traditional diagnostic methods often involve the painful, time-consuming, and invasive procedure of skin lesion biopsy [5].

This project investigates various skin conditions, namely basal cell carcinoma (BCC), melanoma, and nevus. To underscore the significance of this endeavor, let's delve into specific statistics and insights related to these skin conditions.

Melanoma, in particular, represents a perilous, relatively rare, and potentially lethal variant of skin cancer. Despite accounting for only 1% of all reported cases, melanoma's fatality rate is notably higher [6]. This malignant cancer originates in melanocytes, the cells responsible for pigment production, and occurs when these melanocytes lose control over their growth, leading to the formation of cancerous tumors. Melanoma can manifest in various parts of the body, with a predilection for areas exposed to sunlight, such as the hands, face, neck, and lips. Early diagnosis is paramount for effective treatment, as untreated melanomas can metastasize to other parts of the body, resulting in devastating consequences for the individual [7]. Notably, melanoma encompasses diverse subtypes, including nodular melanoma, superficial spreading melanoma, acral lentiginous melanoma, and lentigo maligna [8].

According to the National Institute of Health (NIH), melanoma of the skin is the 5th most common type of cancer, representing approximately 5% of all cancer cases. Furthermore, melanoma cases have been rising on average 1.2% each year over 2010-2019 [9]. Recognizing and categorizing melanoma based on visual cues is a critical step in saving lives. The “ABCDE” is the standard to watch for [10]:

- **Asymmetry** - When one side doesn't mirror the other.
- **Border** - The edges appear jagged, blurred, or uneven.
- **Color** - Uneven coloring, possibly with mixtures of black, brown, and tan.
- **Diameter** - When there is a noticeable change in size, often an increase.
- **Evolving** - The mole has changed in recent weeks or months.

Conversely, the majority of skin cancer cases fall under the non-melanoma category, which includes BCC, squamous cell carcinoma (SCC), and sebaceous gland carcinoma (SGC). BCC, SCC, and SGC originate in the middle and upper layers of the epidermis. Fortunately, these non-melanoma cancers typically exhibit a lower propensity to spread to other parts of the body, making them more amenable to treatment [11].

Nevus, commonly referred to as moles, can vary in size, shape, and color. Some moles are atypical and may warrant closer examination due to their potential link to skin cancer.

Machine learning

Machine learning (ML) is a pivotal subfield within the realm of artificial intelligence (AI) that revolves around the development of algorithms and models with the capacity to learn from data and make predictions or decisions without explicit programming. It's a systematic approach to solving intricate problems by leveraging statistical and computational techniques. At its core, machine learning revolves around data, which comprises observations or examples. This data is usually represented as a collection of input features (attributes or characteristics) and output labels (the target or desired outcome) [12].

ML encompasses two primary categories: supervised and unsupervised learning. Supervised learning involves training a model using labeled data, where the correct answers are provided, enabling it to learn to map inputs to corresponding outputs. In unsupervised learning, models explore patterns and structures in unlabeled data, often through clustering or dimensionality reduction [13].

In the realm of ML, the fundamental concept revolves around models or algorithms, ranging from simple linear equations to complex neural networks. These models undergo training using datasets, adjusting internal parameters to minimize disparities between predictions and actual outcomes in the training data [14]. Once successfully trained, the models apply learned patterns and relationships to make predictions on new, unseen data, guided by algorithms like decision trees, support vector machines, and deep neural networks.

Critical to the ML process is the evaluation of model performance post-training, ensuring effective generalization to new, unseen data. Metrics such as accuracy, precision, recall, and F1 score play a pivotal role in assessing model performance [15]. Addressing challenges, overfitting, where the model learns noise and fails to generalize, and underfitting, where the model is overly simplistic, is a critical consideration [16-17]. Additionally, machine learning algorithms often involve tuning hyperparameters to optimize performance, controlling factors like learning rate, model complexity, or regularization strength [18]. This systematic approach to machine learning encompasses model training, evaluation, addressing challenges, and hyperparameter tuning, ensuring robust and effective predictive capabilities.

Deep learning is a subfield of machine learning that focuses on the use of artificial neural networks to model and solve complex problems. These neural networks are composed of multiple layers, allowing them to automatically learn and represent hierarchical patterns and features from data [19]. Deep learning has gained prominence due to its remarkable ability to excel in tasks such as image recognition, natural language processing, and speech recognition [20]. It's particularly well-suited for tasks involving large, high-dimensional datasets and has been a driving force behind recent advancements in AI, including technologies like self-driving cars and advanced recommendation systems [21].

The successful training and evaluation of machine learning models enable their deployment for real-world predictions and automated decision-making, making

machine learning an indispensable component of modern technology and data-driven decision processes. In the field of medical diagnostics, particularly dermatology, the integration of machine learning is transformative. By harnessing its ability to discern patterns from vast datasets of dermatological images, the present project aims to automate skin condition classification with high accuracy, contributing to accurate, timely, and accessible dermatological diagnostics. This underscores the pivotal role of machine learning in advancing the intersection of technology and healthcare.

1.3 State-of-the-art

Machine learning in dermatopathology

Dermatopathologists have traditionally played a central role in diagnosing skin conditions through the examination of histopathological images and clinical data. However, this process can be time-consuming, resource-intensive, and subject to variations in diagnostic accuracy among experts [22].

The integration of AI and ML techniques in dermatology has brought about transformative changes in the field [23]. AI and ML algorithms have the capacity to process vast amounts of medical data and recognize intricate patterns, significantly enhancing the accuracy and efficiency of dermatological practices [24].

AI plays a pivotal role in early skin cancer detection. By analyzing images of skin lesions, AI systems can recognize potential indicators of skin cancer in its initial stages, facilitating timely intervention and enhancing patient outcomes [25]. This capability holds particular significance within the realm of skin cancer, where timely detection is a decisive factor in successful treatment [7].

Dermoscopy, a technique that uses specialized magnifying tools for skin lesion examination, has also benefited from AI advancements [26]. AI systems can automate the analysis of dermoscopy images, assisting dermatologists and reducing the subjectivity inherent in manual analysis [27].

Teledermatology, enabled by AI, allows patients to submit images of skin issues for remote consultations. AI-driven platforms assist dermatologists in triaging cases and providing initial assessments, extending healthcare access to remote or underserved areas [28].

In addition to diagnosis and treatment, AI systems also contribute to population health management by analyzing large patient datasets to identify disease trends and evaluate the effectiveness of treatments [29]. Such insights aid in public health planning and interventions. Furthermore, AI-driven drug discovery in dermatology is a promising area. These algorithms can predict the efficacy of compounds and analyze potential side effects, expediting the development of dermatological drugs [30].

However, as AI continues to advance, addressing ethical and privacy considerations is essential. Protecting sensitive patient data and mitigating bias in AI algorithms are critical aspects of ensuring the responsible use of AI in dermatology [31].

In summary, the integration of AI and ML in dermatology is revolutionizing the field by enhancing diagnostic accuracy, treatment planning, and patient care. It holds promise for continued advancements and improved patient outcomes as technology evolves and matures.

Convolutional neural networks

CNNs are a type of machine learning model particularly well-suited for image analysis, and they play a significant role in the medical field, especially in the classification of skin conditions. These networks are inspired by the organization of neurons in the human brain and are designed to automatically extract features and patterns from images [32]. In the context of medical imaging, CNNs are employed to analyze skin images and make distinctions between different skin conditions, such as rashes, moles, or other abnormalities.

In CNNs, the convolution operation involves sliding small filters (kernels) over input images to extract local features like edges or corners, facilitating the recognition of increasingly complex patterns related to different skin conditions [33]. Subsequent pooling layers downsample feature maps, reducing spatial dimensions while retaining vital information to enhance computational efficiency and focus on pertinent details [34]. Towards the network's end, fully connected layers connect every neuron, enabling the learning of high-level features and predictions based on extracted patterns from input images [35]. Activation functions like ReLU introduce non-linearity, crucial for the network to comprehend complex relationships in the data. This structured process in CNNs, involving convolution, pooling, and fully connected layers, enhances the model's ability to recognize intricate patterns related to skin conditions.

In a CNN, layers are stacked one after the other, creating a hierarchical architecture (**Figure 1**). As you move deeper into the network, the convolution layers learn to recognize increasingly complex and abstract features. The training process of a CNN involves learning from a labeled dataset. During training, the network attempts to minimize the difference between its predictions and the actual labels. This optimization process, known as backpropagation, helps the network learn and adjust its internal parameters to make accurate predictions [37].

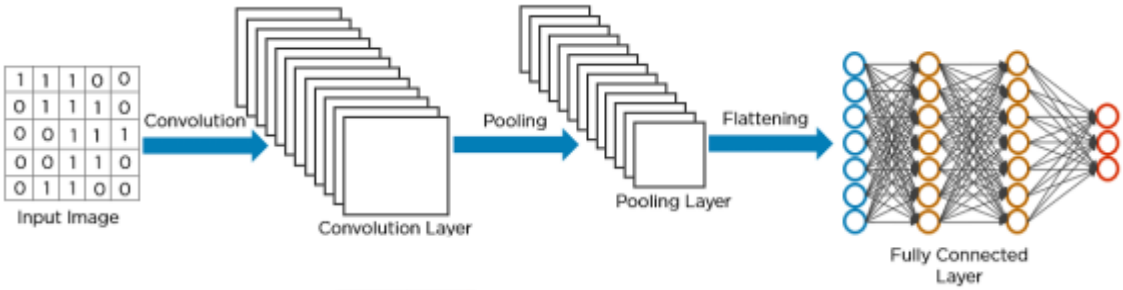


Figure 1: Architecture of a CNN. The CNN architecture comprises distinct layers. Initially, an image, presented as a pixel array, undergoes convolution layers, which extract key features. Subsequently, pooling layers reduce data dimensions while preserving crucial information for interpretation by fully connected layers. Image extracted from Biswal, 2023 [36].

The utilization of CNN-based deep learning models in dermatopathology, as evidenced by studies [38], has demonstrated significant promise in elevating diagnostic accuracy for skin conditions. This approach, trained on diverse datasets, surpasses human experts in specific diagnostic tasks, reducing the risk of misdiagnoses and facilitating more precise treatment decisions. CNNs offer a standardized and consistent diagnostic approach, enhancing efficiency by expediting the processing of whole-slide images, thus benefiting both patients and healthcare providers. The resource optimization capabilities of AI algorithms allow experts to focus on complex cases, and the potential for extending expert knowledge to underserved regions is noteworthy [39].

In summary, the development of this web-based platform with a ML algorithm for skin alteration classification addresses a pressing need in healthcare. It combines the power of technology and accessibility to bridge gaps in dermatological care, promotes early detection and appropriate medical intervention, and empowers individuals to take charge of their skin health. This project has the potential to make a significant positive impact on global healthcare and public awareness of skin conditions.

Advancements in recent years

Skin condition classification using deep learning techniques has undergone significant advancements, with CNNs playing a pivotal role in achieving remarkable accuracy and reliability in diagnosing dermatological conditions [40].

In recent years, researchers have emphasized the importance of data augmentation and preprocessing techniques to improve model performance. Augmentation methods, such as rotation, scaling, and flipping, have been employed to expand training datasets, mitigating data scarcity issues and enhancing the network's ability to generalize to unseen cases [41].

State-of-the-art CNN architectures have evolved to leverage deep learning capabilities. Models such as DenseNet, Inception, and ResNet have been adapted for dermatological image classification, demonstrating exceptional feature extraction capabilities [42]. Customized CNN architectures, often incorporating

novel techniques such as attention mechanisms and skip connections, have been designed to capture fine-grained details specific to skin condition images [43].

Transfer learning has emerged as a highly effective approach, with researchers leveraging pretrained models on large-scale datasets like ImageNet [44]. Fine-tuning these models for skin condition classification tasks has accelerated convergence and improved performance. In the pursuit of more accurate classification, researchers have also explored the integration of diverse data sources. Combining visual image data with clinical text, patient history, and dermoscopic images has shown promise in enhancing model accuracy [45]. Multimodal CNNs have emerged as a key research focus in this regard [46].

As the reliability and safety of CNN-based diagnostic systems are paramount in clinical applications, research has increasingly focused on the interpretability and explainability of these models. Techniques such as Grad-CAM, LIME, and SHAP values have been applied to visualize and interpret regions of interest within images that influence the model's decision-making process [47].

Community-driven initiatives have led to the development of benchmark datasets and challenges that serve as testing grounds for skin condition classification models. Notable examples include the ISIC (International Skin Imaging Collaboration) challenge and the HAM10000 dataset as well as the BCN20000 dataset used in this project, which have spurred healthy competition and fostered innovation in the field.

Transitioning from research to practical clinical applications, validation and integration of CNN-based skin condition classifiers are essential. Studies demonstrating real-world applicability, including large-scale clinical trials, are increasingly prevalent in the literature [48].

In the realm of dermatological image classification, several state-of-the-art applications have emerged, catering to the increasing demand for accurate and rapid skin condition assessments. One noteworthy example is “DermEngine”, a comprehensive dermatology software that utilizes artificial intelligence to facilitate the analysis of skin images. With its expansive database of dermatological conditions, “DermEngine” enables healthcare professionals to swiftly identify skin disorders and track patient progress [49]. Another prominent application is “SkinVision”, which empowers users to conduct self-assessments of skin lesions, providing risk assessments for skin cancer and other conditions [50]. Leveraging deep learning and machine vision, “SkinVision” serves as a valuable tool for early detection. Additionally, “First Derm” offers users the convenience of obtaining dermatological advice and opinions from board-certified dermatologists, making quality healthcare accessible through telemedicine [51]. These existing applications represent a leap forward in the field of dermatological image classification, enhancing diagnostic accuracy, patient engagement, and accessibility to expert medical opinions.

Our project distinguishes itself from existing dermatological applications, such as “DermEngine”, “SkinVision”, and “First Derm”, by offering a web-based application that is freely accessible and user-friendly, eliminating the need for paid standalone mobile applications. This approach aligns with the principles of open and inclusive healthcare, broadening access to efficient dermatological image classification without the barriers of app installations or financial commitments. However, it's important to note a limitation: unlike some paid applications that enable real-time consultations with certified dermatologists, our web-based application primarily focuses on providing accurate information about dermatology and skin conditions, lacking direct interaction with medical professionals. Users are advised to seek professional medical advice when needed to complement the insights offered by our application.

In conclusion, the field of skin condition classification with CNNs is evolving rapidly, driven by data augmentation, novel architectures, transfer learning, multimodal approaches, and an increasing emphasis on interpretability and clinical validation. Researchers are working towards more accurate, reliable, and clinically relevant diagnostic tools, which have the potential to revolutionize dermatological practice.

2. Objectives

2.1 Main objective

Develop a web-based platform equipped with a machine learning algorithm capable of accurately classifying various skin alterations based on user-submitted images.

2.2 Specific objectives

- Implement and evaluate a CNN machine learning model in Python to classify a diverse range of skin alterations.
- Optimize the model to achieve a minimum precision rate of 85%.
- Develop an interactive and user-friendly web application, integrating the model for skin alteration classification.

3. Sustainable development goals

In the context of Sustainable Development Goals (SDGs) established by the United Nations, our project aligns with several critical SDGs (**Figure 2**), contributing to global sustainability and development. The primary objectives of our project focus on developing a web-based platform with a machine learning algorithm capable of classifying various skin alterations based on user submitted images. This innovative solution holds a significant potential to support the following SDGs:

Figure 2: Sustainable Development Goals. An image depicting all the SDGs incorporated within this project. Extracted from United Nations, (2023) [52].

SDG 3: Good health and well-being: Our project directly advances SDG 3 by addressing the goal of good health and well being. The web based platform offers an accessible tool for early detection and assessment of a wide range of skin conditions. By enabling individuals to submit images of their skin alterations and receive rapid and accurate assessments, our project empowers users to seek timely medical intervention when needed. This proactive approach to skin health enhances overall well-being and supports the broader goal of good health.

SDG 4: Quality education: The project integrates an educational component that aligns with SDG 4, emphasizing quality education. In addition to skin condition classification, the web application provides information about various skin conditions. This educational feature fosters awareness, improves understanding of skin health, and encourages proactive practices. By offering knowledge and resources, our project contributes to the promotion of quality education, aligning with the United Nations sustainable development agenda.

SDG 9: Industry, innovation, and infrastructure: Our project exemplifies the principles of innovation and technological advancement outlined in SDG 9. The development of a web based platform equipped with a machine learning algorithm represents a significant leap in the application of technology to address healthcare challenges. By leveraging innovative approaches and technology-driven solutions, our project advances industry, innovation, and infrastructure, particularly within the healthcare sector.

SDG 10: Reduced inequalities: The project is rooted in the principle of reducing inequalities, as outlined in SDG 10. By providing open and free access to skin condition assessments, our project eliminates geographic and financial barriers to

healthcare. This accessible platform ensures that individuals worldwide, regardless of their geographical location or socioeconomic status, can benefit from early detection and assessment services. Through its inclusive design, the project contributes to the reduction of inequalities in healthcare access.

In conclusion, our project aligns with these SDGs by utilizing technology, education, and innovative approaches to improve health outcomes, reduce inequalities in access to healthcare, and promote awareness. The development of a web application and the application of ML technology exemplifies the potential for technological solutions to address global challenges, ultimately contributing to sustainable development and the broader United Nations agenda.

While our project primarily focuses on improving healthcare accessibility and awareness, it's essential to acknowledge potential negative impacts and ethical considerations. First, the operation of machine learning algorithms and web servers consumes energy, contributing to environmental impact, which we recognize and aim to mitigate through responsible server hosting and energy-efficient practices. Second, data privacy and security are paramount, as user-submitted images may contain sensitive information, and we are committed to safeguarding this data. Third, we acknowledge the potential for algorithm bias and emphasize our efforts to mitigate bias and ensure fairness in the classification results. Furthermore, we stress that our web application should complement, not replace, professional medical advice, promoting informed decision making. By addressing these concerns transparently and ethically, our project aims to ensure responsible, impactful, and equitable contributions to healthcare accessibility and awareness.

4. Approach and methodology

4.1 Methodology

In the context of enhancing the field of dermatology and improving the diagnostic accuracy of various skin conditions, the choice of utilizing CNNs as the primary methodology for skin condition classification stems from their inherent advantages. Dermatology heavily relies on visual data and clinical images for diagnosis, making CNNs particularly suitable due to their effectiveness in analyzing medical images. These networks excel at extracting complex features from images, including patterns, textures, and color variations, which are essential for the accurate classification of skin disorders. Furthermore, CNNs have demonstrated significant success in various medical imaging tasks, including skin condition detection, owing to their strong generalization capabilities and adaptability to diverse datasets. Their flexibility in model architecture, layer configuration, and hyperparameter tuning enables tailoring to the specific challenges of dermatological images, such as variations in lesion sizes, colors, and textures.

In the pursuit of developing an effective skin condition classification model, two strategies were employed. The first strategy involved building a model from scratch, designed specifically to address the unique challenges presented by dermatological images. This approach was chosen to provide a comprehensive understanding of the deep learning process, from data preprocessing to model evaluation, in the context of healthcare applications. The second strategy involved fine-tuning existing pre-trained CNN architectures, such as DenseNet and ResNet, to adapt them to our specific dataset. This approach leverages the advantages of established models that have undergone extensive development and optimization, ensuring efficiency and reliability in skin condition classification. By fine-tuning pre-trained models with our data, we enhance their ability to generalize to the characteristics of dermatological images, expedite model development, and reduce computational resource requirements. Additionally, these strategies collectively mitigate potential biases and ensure the robustness and fairness of our skin condition classification system by using models trained on diverse data.

In the forthcoming scheme (**Figure 3**), we present a meticulously structured workflow that elucidates the comprehensive methodology for our skin condition classification project, marked by the integration of a user-centric web application.

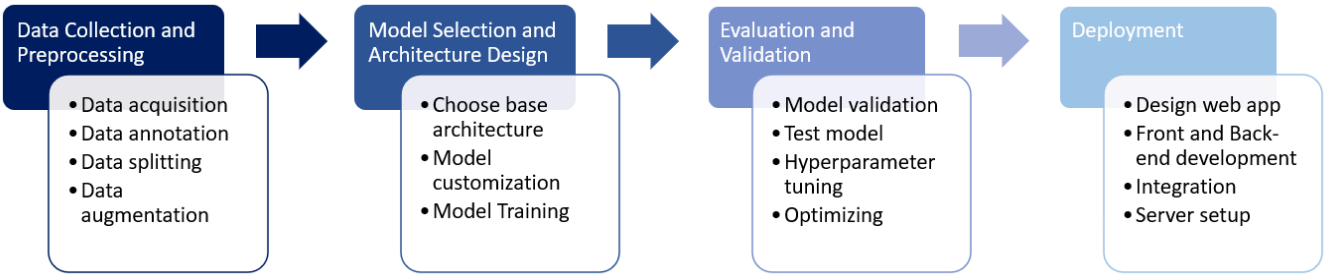


Figure 3: Workflow for skin condition classification project with web app integration. This figure illustrates the step-by-step workflow for a comprehensive skin condition classification project with the integration of a web application. The project encompasses data collection and preprocessing, model development, validation, web app development, testing and deployment.

Data collection and preprocessing: The process begins with the collection of a diverse dataset of skin condition images, which includes a metadata file with relevant labels (nevus, BCC and melanoma). The dataset is divided into training, validation and test subsets.

Model selection and architecture design: The choice of a CNN architecture, such as DenseNet and ResNet, forms the core of the model design. Additionally, a custom CNN architecture is builded from scratch. Fine-tuning strategies are implemented to adapt the base architecture to the unique characteristics of dermatological images, followed by model customization and training using the training dataset.

Evaluation and validation: The model's performance undergoes rigorous evaluation through validation, employing a distinct dataset reserved exclusively for fine-tuning and optimizing hyperparameters. This process ensures that the model adapts and generalizes well to new data, contributing to its overall robustness. Subsequently, the model's effectiveness is further scrutinized using a separate testing dataset, allowing a comprehensive assessment of its accuracy and real-world performance.

Web App development: The process of developing the web application encompasses three main components: design, front-end implementation utilizing HTML and CSS, and back-end development involving Django along with the integration of the machine learning model.

Web App testing and debugging: Comprehensive testing is conducted to identify and rectify issues, ensuring that the web app functions smoothly.

Web App deployment: The web application is hosted on a server, and domain and SSL configurations are implemented for secure data transmission.

This workflow provides a structured approach to tackle the complexities of skin condition classification and deliver a user-friendly web application for medical professionals and patients.

4.2 Planning and calendar

Main tasks

1. **Definition of the work plan:** Define the project's scope, objectives, methodology and expected outcomes. Create a project charter outlining the project's purpose and goals as well as a calendar with milestones and dates.

2. **Data collection and preparation:** Curate a comprehensive and diverse dataset of skin alteration images, ensuring that it represents various skin conditions and ages to train and validate the machine learning model. Split the data into training, validation, and test sets. Preprocess the images, normalizing pixel values to a specified range, and augmenting the data if needed.

3. **Algorithm development:** Train a ML model in Python capable of accurately classifying a diverse range of skin alterations.

4. **Web application development:** Design and develop a user-friendly web application using the Django framework to enable users to upload skin images for classification.

Extra tasks

1. **Algorithm evaluation:** Evaluate the performance of the ML model by conducting rigorous testing, including metrics such as accuracy, sensitivity, specificity, and precision, to ensure its effectiveness in skin condition classification.

2. **Algorithm optimization:** Investigate the potential of utilizing a pre-trained model such as DenseNet121, ResNet50, or MobileNet. Identify the most effective pre-processing technique to enhance the model's performance.

3. **Accessibility and scalability:** Ensure that the web application is accessible to anyone with an internet connection, and design it to be scalable for potential future enhancements and improvements.

4. **Educational content integration:** Incorporate educational content within the web application to provide users with information about various skin conditions, fostering awareness and encouraging proactive skin health practices.

5. **Data security and privacy:** Implement robust data security and privacy measures to protect user-submitted images and information, complying with relevant regulations and standards.

6. **Deployment and maintenance:** Deploy the web application to a reliable hosting environment and establish a plan for ongoing maintenance and updates to ensure continued functionality and accuracy.

Calendar

This Gantt chart (**Figure 4**) provides a visual representation of the project's timeline, helping to track progress and ensure that the project's objectives are met within the specified timeframes. A chronological schedule, presented in list format, is additionally available in the annex for reference (**Supplementary figure 1**).

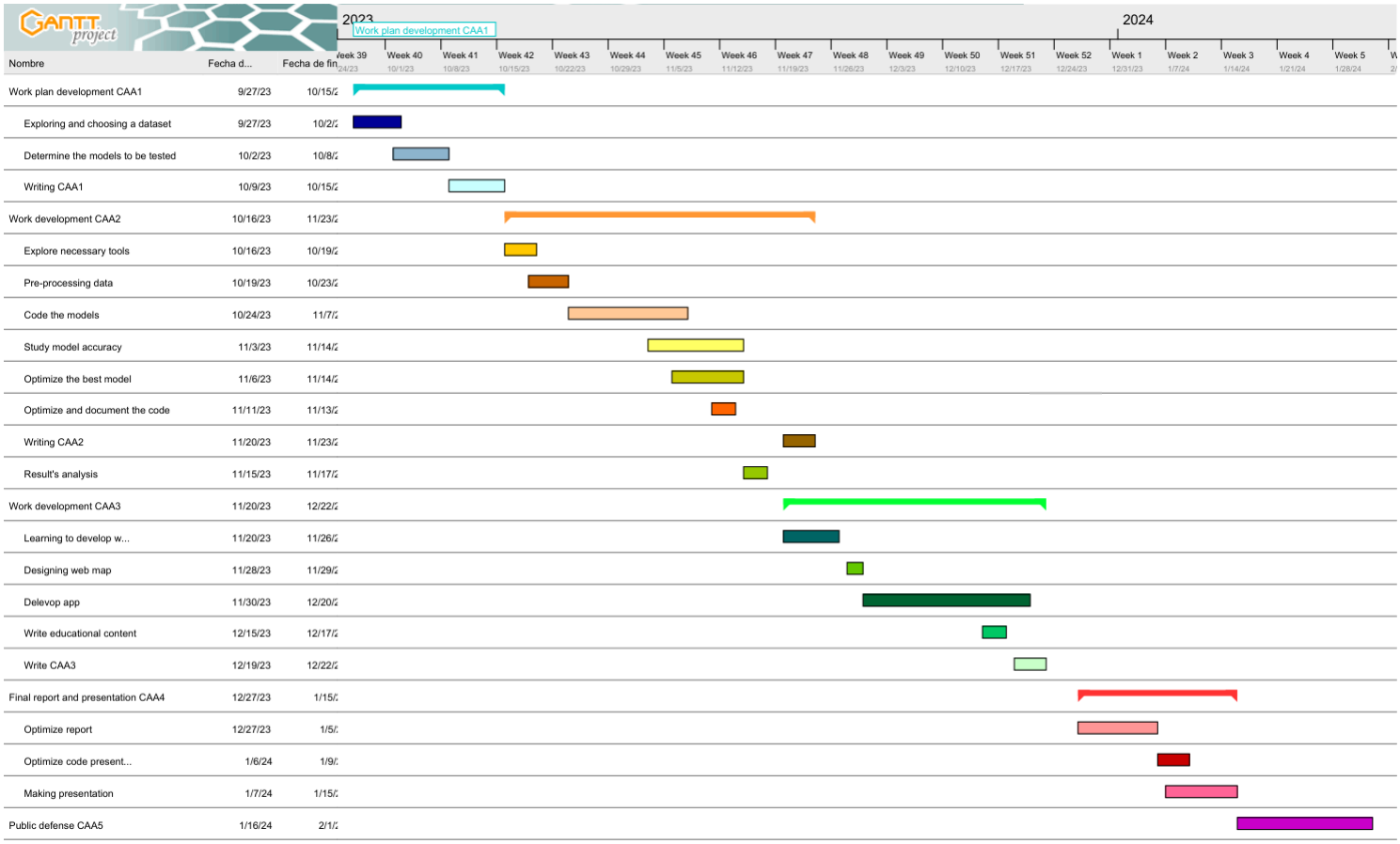


Figure 4: Project timeline. Chronological schedule presented as an image, showcasing the project's timeline through a Gantt chart. The chart visually delineates the start and end dates for each task. It was crafted using the GanttProject tool [53].

Milestones

CAA1: October 16, 2023 - Achievement of the development milestone encompassing the formulation of the comprehensive work plan.

CAA2: November 19, 2023 - Successful culmination of Phase 1 of the development process, characterized by the establishment of a model with the ability to accurately classify images, achieving an 85% accuracy rate.

CAA3: December 22, 2023 - Successful completion of Phase 2 of the development project, marked by the realization of a fully operational web application incorporating the highest performing model.

CAA4: January 15, 2024 - Accomplishment of the project's conclusion, including the delivery of a comprehensive project report, a presentation, and an explanatory video presentation.

CAA5: February 1, 2024 - Public presentation and defense of the project.

4.3 Risk analysis

The following table **(Table 1)** offers a comprehensive examination of potential project execution-related risks. This analysis encompasses an assessment of risk severity, the probability of occurrence, and corresponding mitigation strategies.

Risk	Severity	Likelihood	Mitigation
Data availability and quality	Moderate	Low	Perform an extensive data exploration and explore alternative sources and datasets.
Algorithm performance and usage	High	Moderate	Conduct rigorous algorithm evaluation and testing using cross-validation techniques and investigate optimization methods.
User adoption and awareness	High	Moderate	Include warnings about proper use of the app as well as health information.
Technical challenges	Moderate	High	Perform proper exploration of the libraries and seek guidance and mentorship from professors or experts in relevant fields.
Legal and ethical compliance	High	Low	Adhere to academic integrity guidelines and ethical standards when conducting research. Properly cite and reference sources, code, and data used in the project.
Resource constraints	Moderate	Moderate	Create a well-defined project schedule with milestones and allocate sufficient time for each phase. Plan for contingencies.

Table 1: Risk analysis. This table presents various risks associated with the project, along with their severity, likelihood, and potential mitigation measures.

4.4 Final product

CAA1: PDF document containing the comprehensive work plan and calendar, meticulously outlining all requisite tasks for project completion. It also defines the project's objectives and includes a robust risk management strategy. Available in the dedicated GitHub repository [54] through this link.

Report: PDF document providing an exhaustive account of the project's processes, encompassing investigation, development, results, conclusions, and discussions. Available at the dedicated GitHub repository [54] through this link.

Web application: Link to the web application developed in Django, which hosts the machine learning algorithm capable of classifying various skin abnormalities. Users are able to upload their own images and get a classification result. Available at the following link: https://skinai.bioedu.one/ [55]. A video demonstration of the working web site is also available in the dedicated GitHub repository [54] through this link.

GitHub repository: GitHub repository housing all the requisite code essential for the project's replication [54], encompassing both the machine learning model and the web application. Available at the following link.

Virtual presentation: Presentation in PPT format offering support and a platform for the comprehensive presentation of the undertaken work. A video recording, accompanied by explanatory narration, is also available to enhance the understanding of the project. The presentation PPT file and the explanatory video are both accessible at the dedicated GitHub repository [54] through this link.

5. Materials and methods

5.5 Dataset

The dataset employed for the purpose of this study is the BCN20000 dataset [56], which encompasses a total of 12,413 dermoscopic images depicting various skin abnormalities. These images were captured between the years 2010 and 2016 at the Clinic's Hospital of Barcelona. The dataset comprises a spectrum of potential diagnoses, including actinic keratosis, BCC, dermatofibroma, melanoma, nevus, SCC, seborrheic keratosis, solar lentigo, and vascular lesions. Additionally, the metadata associated with each image provides valuable information such as age, gender, and the anatomical location where the image was shot.

This dataset is publicly accessible through the ISIC Archive, a resource established with the primary objective of enhancing the diagnosis of skin cancer. The ISIC Archive serves as a repository for standardizing skin imaging practices, collecting and disseminating dermatological images, and fostering collaboration among medical practitioners and experts in the field of computer vision.

Initial data exploration

To comprehend the dataset thoroughly, it is crucial to analyze the distribution of images among various skin condition classes. This examination aids in identifying potential imbalances among classes and guides decisions on data preprocessing. The class distribution can be visualized in the following table (**Table 2**):

Diagnosis	Sample Size	Percentage
Nevus	4206	33,88
Melanoma	2857	23,01
Basal cell carcinoma	2809	22,62
Seborrheic keratosis	929	7,48
Actinic keratosis	737	5,93
Squamous cell carcinoma	431	3,47
Solar lentigo	209	1,68
Dermatofibroma	124	0,99
Vascular lesion	111	0,89

Table 2: Class distribution. Table illustrating the distribution of classes, showcasing the diagnosis distribution within the BCN20000 dataset. This table was generated through the execution of "**Code 1**", detailed in the annex and accessible at the dedicated GitHub [54] ("exploredata.py").

A severe class imbalance can be observed, with 4 classes getting less than 5% of the total cases. Class imbalance refers to a situation in ML where the number of instances in different classes of a dataset significantly varies, leading to an unequal distribution of classes. Unbalanced classes can substantially impact the study by causing the model to be biased towards the majority class, resulting in poor recognition of minority classes [57]. This imbalance may lead to a high accuracy rate, but it fails to address the primary objective of identifying rare or critical instances, such as dermatofibroma in this scenario.

To mitigate this issue, techniques like data augmentation, resampling, weighted loss functions, transfer learning or Ada-Boosting can be employed [58]. Reducing the scope of the project to include only the most common types of diagnosis is also an option. These approaches seek to create a more equitable representation of all classes, thus enhancing the model's ability to accurately classify rare classes and improving the overall robustness and reliability of the image classification system in a technical and formal manner.

While exploring our dataset, we initially considered various metadata variables such as sex, anatomical site, and melanocytic characteristics to gain a comprehensive understanding of the data landscape. Upon careful consideration during the model development process, we found that these variables did not contribute significantly to the algorithm's performance. The model primarily relied on the visual features of dermoscopic images for accurate classification, and the additional metadata variables were ultimately excluded from the final implementation. While these insights provided valuable context during the exploratory phase, their exclusion highlights the model's emphasis on image based features for robust skin condition classification.

Image characteristics

Understanding the inherent characteristics of the images within our dataset is of paramount importance in preparing for the development of our CNN model. These characteristics significantly influence the design and optimization of our model architecture and data preprocessing. The dataset exhibits the following key attributes:

- **Height:** The height of all the images in the dataset is 1024 pixels
- **Width:** Similarly, the width of all images in the dataset is 1024 pixels.
- **Channels:** All images in the dataset are configured with three color channels (red, green, and blue), consistent with the typical RGB format.

These results are expected, as the dataset was carefully prepared by ISICS with the objective of machine learning development in mind. Comprehending these image characteristics plays a crucial role in the development of our CNN model, guiding decisions related to input shape determination, resource allocation, data augmentation, and preprocessing procedures. This knowledge enables us to tailor

our CNN model and data processing to effectively accommodate these specific attributes.

However, meticulously designing the model to accept images of specific attributes (1024x1024 pixels and RGB color channels), poses the question of whether user-contributed images must adhere to these specifications. This issue may be necessary to address in the future with the implementation of a resizing and preprocessing of user-submitted images within the web application.

This information was derived from the execution of “**Code 1**”, which is available in the annex and on the dedicated GitHub repository [54] under the filename "exploredata.py". A detailed explanation of the code can also be found in the annex.

Data pre-processing

Resizing images

One of the primary constraints and limitations encountered during the project pertained to the extensive volume of images within the dataset. The initial challenge involved the need to downscale these images from their native resolution of 1024x1024 pixels to a reduced size of 256x256 pixels. This image resizing operation was conducted within a Linux environment through the execution of a specific bash command in the terminal. The command was applied once navigation to the directory where the images were stored was completed.

```
sudo apt install imagemagick
mogrify -resize 256x256 *.JPG
```

The decision to resize the images from their original 1024x1024 pixel dimensions to a reduced 256x256 pixel format was driven by a strategic need to optimize the utilization of available computational resources, particularly the available Random-Access Memory (RAM) of 32GB. In this context, the resizing operation was a proactive measure to mitigate the computational overhead associated with processing a large dataset.

Normalizing pixel values

Prior to the commencement of the training process, a crucial preprocessing step was undertaken for all images, involving the normalization of pixel values by dividing each value by 255 (referenced as **Code 2,** or “CNN.py”, “DenseNet.py” and “ResNet.py” in the GitHub repository [54]). This operation, indicative of a min-max scaling approach, rescaled pixel values to a range between 0 and 1. This standardized normalization facilitates optimal model convergence and ensures uniformity in the representation of image features during the training phase.

It is important to note that our selected normalization technique, based on straightforward scaling, allows for flexibility in the order of preprocessing steps. Given the simplicity of this scaling operation, normalization was consistently applied before splitting the data, avoiding the introduction of bias. This method ensures uniformity across training, validation, test, and prediction sets, contributing to a pragmatic and streamlined preprocessing workflow.

Addressing class imbalance

After conducting some testing, it became evident that the development of a high-performing CNN model was unfeasible within the prevailing context of severely imbalanced classes. Compounding this challenge was the constraint of limited computational resources, which further exacerbated the predicament.

In response to this predicament, a decision was made to address the class imbalance issue by reducing the dataset. Specifically, classes that constituted fewer than 10% of the total images were selectively removed, thereby mitigating the impact of underrepresented classes. The objective was to maintain a more equitable distribution among the remaining classes, fostering a balanced and representative dataset.

Furthermore, it was recognized that even within the reduced dataset, the nevus class, which was previously overrepresented, needed to be downsized to maintain equilibrium between the remaining classes. This adjustment aimed to ensure that the CNN model could be trained and evaluated with an equitable representation of each class, fostering a fair and robust predictive performance that could be achieved within the available computational resources.

Subsequent to the execution of **Code 3** ("reducing_dataset.py" in the GitHub repository [54]), a subset of images was meticulously generated, featuring a distinct distribution of classes as illustrated in **Table 3**. Simultaneously, the metadata CSV file underwent automatic modification to retain exclusively the pertinent information associated with the selected images.

Diagnosis	Sample Size	Percentage
Nevus	2850	33,55
Melanoma	2857	33,47
Basal cell carcinoma	2809	32,98

Table 3: Class distribution. Table outlining the distribution of classes, displaying the diagnostic breakdown of the BCN20000 dataset subset. This subset was obtained by reducing the original dataset using "**Code 3**", as detailed in the annex or "reducing_dataset.py" in the GitHub repository [54]. The table was created using the same methodology used to create Table 2.

One-hot encoding and splitting dataset

The dataset preprocessing phase included a critical procedure known as one-hot encoding for the categorical labels, facilitating their transformation into a binary format conducive to machine learning training. Each of the three distinctive classes was allocated a unique binary representation. One-hot encoding ensures the algorithm's ability to effectively interpret and assimilate information from the categorical labels during the training phase. For clarity, consider the example where the original label is “Melanoma”, the resulting one-hot encoding might be represented as [0, 1, 0], denoting the absence of “BCC” (0), the presence of “Melanoma” (1), and the absence of “Nevus” (0).

Subsequently, the dataset underwent a stratified splitting process into three subsets: training (80%), validation (10%), and test (10%). This practice adheres to a common and effective methodology in machine learning. The training subset was exclusively utilized for training the algorithm, exposing it to a diverse range of examples from each class. The validation subset played a pivotal role in monitoring the algorithm's performance during training, enabling fine-tuning to optimize accuracy and generalization. Finally, the test subset served as an independent set to assess the algorithm's true performance, providing robust evaluation metrics and insights into its accuracy across diverse skin conditions. This systematic partitioning ensures a comprehensive evaluation of the algorithm's capabilities and generalization to unseen data.

The code for this operation is available in the annex as **Code 4**, accompanied by a detailed description. Alternatively, it can be found in the dedicated GitHub repository [54] within the specific files for the complete algorithms (“CNN.py”, “DenseNet.py” and “ResNet.py”).

5.6 Machine learning algorithms

As stated previously, CNNs are a fitting choice for image classification tasks, including the categorization of skin conditions. Its suitability stems from its prowess in recognizing intricate patterns, scalability to handle high-dimensional image data, automated feature extraction, ability to generalize from training data to new examples, adaptability to improve with more data and fine-tuning, and effective handling of image complexities [59].

CNNs have emerged as the de facto standard for image classification, owing to their remarkable efficacy and unique design principles. CNNs excel in this realm due to their innate ability to automatically unearth complex hierarchical features from raw image data [60]. A detailed list of advantages and disadvantages can be found in the following table (**Table 4**).

While traditional machine learning algorithms like support vector machines (SVM), k-nearest neighbors (kNN), or naive bayes are versatile tools across various domains, their application to image classification poses inherent limitations. Unlike

CNNs, these traditional algorithms require handcrafted feature engineering from high-dimensional image data, a cumbersome and potentially inadequate process for capturing intricate patterns in images [14]. That said, you can use traditional ML algorithms for image classification, but you'll typically need to hand-craft feature vectors from the images, which can be a complex and time-consuming process. Additionally, the performance of these algorithms may not match that of CNNs on image data, especially when dealing with large and diverse datasets.

So, while you can technically apply kNN, naive bayes, decision trees, or SVM to image data by flattening the images into feature vectors, it's generally more practical and effective to use CNNs for image classification tasks due to their ability to automatically learn relevant features and hierarchical representations from raw image data. For this reason, in this project we will focus on the implementation of a CNN algorithm that can precisely classify images of skin conditions.

Pros and cons of CNN algorithms	
Pros	**Cons**
Automatic feature learning	Computational resources
Hierarchical representation	Need for large datasets
Translation invariance	Overfitting
Scalability	Interpretability
Generalization	Hyperparameter tuning
Adaptability	
Flexibility	
State of the art performance	

Table 4: Pros and cons of CNN algorithms. Table showing pros and cons of using CNN algorithms [61].

5.7 Programming language and libraries

In the construction of such an algorithm, the choice of the programming language is pivotal, and both R and Python emerge as viable options. Although there are several robust deep learning frameworks available in both languages, including TensorFlow, Keras and MXNet, the deliberate choice for the programming language in this project is Python. Python's selection is underpinned by its robust attributes, notably its extensive library ecosystem, and its status as one of the most pervasive and widely-supported programming languages. The magnitude of its community ensures ready access to valuable resources, which is particularly advantageous in the context of this project. Furthermore, Python's versatility

renders it apt for diverse tasks encompassing data preprocessing, data visualization, statistical analysis, and the availability of well-established deployment frameworks such as Django and Flask, which are conducive to the realization of the intended web application.

Concurrently, TensorFlow, a preeminent deep learning framework, asserts its prominence in the landscape of machine learning tools. Within TensorFlow, the high-level Keras API stands as an accessible and user intuitive vehicle for conceiving, training, and assessing CNN models. The endorsement of TensorFlow and Keras is substantiated by their comprehensive documentation and an expansive network of support, rendering them a superlative choice for developers. Familiarity with this library, owing to prior usage in the development of other ML algorithms, serves as a compelling rationale for its adoption in this context. While a multitude of alternative libraries such as PyTorch, Caffe, MXNet, and Fastai could be considered, the aforementioned factors collectively contribute to the preference for TensorFlow and Keras in this project.

5.8 Model architecture

CNN built from scratch

The CNN architecture presented is designed for the classification of skin images. The choice of this architecture is informed by the efficacy of CNNs in image classification tasks, owing to their inherent ability to automatically learn and extract relevant features from the input data.

The key architectural components of this CNN are as follows:

1. **Convolutional layers**:
    - Five convolutional layers are employed, each utilizing a 3x3 kernel for feature extraction.
    - The number of filters, or output channels, progressively increases, namely 32, 64, 64, 64, and 128. This hierarchical arrangement allows the network to learn increasingly intricate features.
    - The Rectified Linear Unit (ReLU) activation function is applied to introduce non-linearity to the model.
    - Following each convolutional layer, a max-pooling layer (2x2) is utilized to spatially downsample the feature maps, thereby reducing the dimensionality of the data.
    - Batch normalization is integrated after each max-pooling layer, serving to normalize activations and enhance the training process.
2. **Dropout layers**:
    - Two dropout layers are strategically positioned after the first and last convolutional layers. Multiple dropout rates were tested during hyperparameter fine-tuning.
3. **Fully connected layers**:

- Subsequent to flattening the output from the final convolutional layer, two fully connected layers are introduced.
- The initial fully connected layer encompasses 256 neurons and employs the ReLU activation function. This layer captures high-level features extracted by the preceding convolutional layers.
- A dropout layer is incorporated after the first fully connected layer to provide regularization.
- The second fully connected layer consists of 128 neurons with a ReLU activation function, offering flexibility for customizing the number of neurons and activation functions according to the specific problem.

4. **Output layer**:
   - The output layer is represented by a dense layer encompassing "num_classes" neurons, which corresponds to the number of classes in the skin image classification task, 3 in this particular project.
   - The activation function employed in this layer is "softmax", which serves to assign probabilities to each class, reflecting the likelihood of an input image belonging to a particular class.
5. **Compilation**:
   - The model is compiled using the Adam optimizer. Multiple learning rates were tested during hyperparameter fine-tuning.
   - Categorical cross-entropy loss, designated as "categorical_crossentropy", is employed as the loss function, which is particularly suited for multi-class classification tasks.

In summary, the proposed CNN architecture exhibits a structured approach to the classification of skin images, drawing upon the strengths of convolutional layers for feature extraction and pooling, as well as employing techniques such as dropout and batch normalization for model regularization. The choice of hyperparameters and network architecture is pivotal for the successful training of the model and merits meticulous adjustment and evaluation throughout the training process.

The code for this operation is available in the annex as **Code 5**, accompanied by a detailed description. Alternatively, it can be found in the dedicated GitHub repository [54] within the specific file for the complete algorithm "CNN.py".

DenseNet121

The model architecture described is based on a transfer learning approach that utilizes the DenseNet121 architecture, pre-trained on the ImageNet dataset, as the foundation. This pre-trained network is augmented with custom layers for the classification of skin alterations. The rationale behind this approach is to leverage the knowledge gained by the pre-trained model on a vast array of image data and fine-tune it for a specific classification task.

1. **Transfer learning with DenseNet121**:
   - The base model, DenseNet121, is initialized with weights pre-trained on the ImageNet dataset. This pre-training imparts valuable features and

representations, which can significantly expedite training on the target classification task.
    - The input shape is defined as (256, 256, 3), corresponding to the dimensions of the skin images.
2. **Layer freezing**:
    - To balance the use of pre-trained features and allow for the adaptation of certain layers to the new task, specific layers within the base model are frozen.
    - Freezing these layers prevents their weights from being updated during training, preserving the knowledge acquired from ImageNet. Multiple freezing points were tested with the highest performance achieved freezing layers until the 4th convolutional block.
3. **Custom layers**:
    - On top of the pre-trained model, a sequence of custom layers is introduced.
    - Global average pooling 2D is applied to the output of the base model. This layer reduces the spatial dimensions, aggregating information from the feature maps.
    - Batch normalization is incorporated to stabilize and normalize activations, contributing to faster convergence.
    - A fully connected layer with 256 neurons and ReLU activation is added to capture high-level features specific to the skin alteration classification task.
    - A dropout layer with a rate of 0.15 is included for regularization, reducing the risk of overfitting.
    - The final layer consists of a dense layer with “softmax” activation, which generates probabilities for the skin alteration classes.
4. **Compilation and training**:
    - The model is compiled using the Adam optimizer with a learning rate of 0.001 and categorical cross-entropy as the loss function.
    - The training process is characterized by 10 epochs and a batch size of 128. Multiple values were tested to ensure peak performance.
    - A learning rate scheduler is employed to dynamically adjust the learning rate during training. It monitors validation loss and reduces the learning rate by a factor of 0.5 after a patience of 3 epochs, with a minimum learning rate of 1e-6.

In summary, this model structure combines the strengths of transfer learning with DenseNet121, fine-tuning the model for a specific skin alteration classification task. The layer freezing strategy allows for a balance between leveraging pre-trained features and adapting the model to the new task. Custom layers are introduced to capture task-specific features, and the training process is managed with appropriate hyperparameters, including the learning rate scheduler, to enhance training efficiency and model performance.

The code for this operation is available in the annex as **Code 6**, accompanied by a detailed description. Alternatively, it can be found in the dedicated GitHub repository [54] within the specific file for the complete algorithm “DenseNet.py”.

ResNet50

The outlined model architecture leverages the ResNet50 architecture for the classification of skin alterations. ResNet50 is initialized with ImageNet weights, signifying pre-training on a diverse dataset, similar to the process followed for the DenseNet121.

1. **Transfer learning with ResNet50**:
   - The base model, ResNet50, is initialized with pre-trained weights from ImageNet and excludes the top classification layer.
   - The input shape is defined as (256, 256, 3), corresponding to the dimensions of the skin images.
2. **Layer freezing**:
   - To exploit the pre-trained knowledge while enabling task-specific adaptation, layers up to the 4th convolution block are frozen. This selective freezing strategy ensures that lower-level features are retained while allowing the model to learn higher-level representations.
   - Múltiple freezing points were tested with the highest performance achieved freezing layers until the 4th convolutional block.
3. **Custom layers:**
   - The same top layers as in the DenseNet121 were used for this model.
   - A dropout rate of 0.25 was used instead of 0.15 after some testing to ensure peak performance.
4. **Compilation and training**
   - The model is compiled using the Adam optimizer with a learning rate of 0.001 and categorical cross-entropy as the loss function, suited for multi-class classification tasks.
   - The training process is characterized by 30 epochs and a batch size of 128. Multiple values were tested during the fine-tuning process.
   - A learning rate scheduler dynamically adjusts the learning rate during training. It observes the validation loss and reduces the learning rate by a factor of 0.5 after a patience of 3 epochs, ensuring adaptive learning.

In summary, this model configuration integrates the strengths of ResNet50 through transfer learning, selectively freezing layers for efficient adaptation, and introducing custom layers for task-specific features. The defined hyperparameters and learning rate scheduler contribute to a well-optimized training process, providing a robust framework for skin alteration classification.

The code for this operation is available in the annex as **Code 7**, accompanied by a detailed description. Alternatively, it can be found in the dedicated GitHub repository [54] within the specific file for the complete algorithm "ResNet.py".

5.9 Optimizing hyperparameters

To facilitate the model construction process, an exhaustive GridSearch procedure was conducted, exploring various parameter combinations to identify the

configuration yielding the highest accuracy. This comprehensive optimization task was performed within the constraints of a reduced and balanced dataset comprising 1002 images, generated by the execution of **Code 8** (or "create_test_images.py" in the dedicated GitHub [54]). The computational demands were met through the utilization of Google Colab, harnessing GPU acceleration, thereby mitigating the resource limitations.

A comprehensive depiction of the image sets employed has been crafted to enhance clarity (**Figure 5**). The initial step involves modifying the original ISIC downloadable dataset, wherein minority classes are removed and remaining classes are balanced, resulting in an equilibrium of classes within the dataset. This balanced dataset serves as the foundation for constructing the definitive models, encompassing the corresponding training, validation, and test subsets.

Simultaneously, a smaller subset of 1002 images is extracted from the balanced dataset for the purpose of conducting GridSearch. Given the resource-intensive nature of this process, the reduction in dataset size enhances computational efficiency. This subset, which is also balanced, undergoes further division into training, validation, and testing sets to facilitate the GridSearch process and refine hyperparameter tuning.

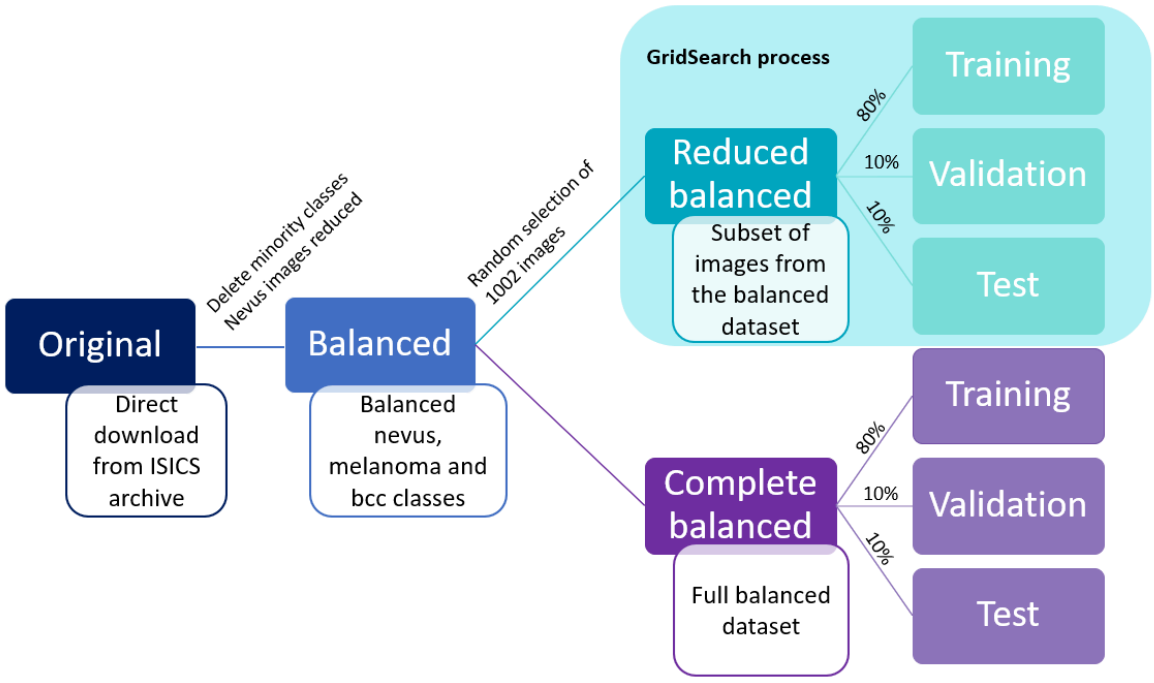


Figure 5: Data processing and subset division scheme. This scheme illustrates the meticulous process of crafting a balanced dataset from the original ISIC downloadable dataset, ensuring equitable representation of classes. The balanced dataset forms the basis for training, validation, and test subsets used in constructing the definitive models. Additionally, a resource-efficient subset of 1002 images is extracted for GridSearch, further divided into training, validation, and testing sets to streamline hyperparameter tuning.

The methodology commences by initializing a distinct CNN model for each hyperparameter combination under consideration. Each model is then compiled with the specified learning rate, dropout rate and number of epochs, setting up its optimization algorithm. Subsequently, it undergoes training on the provided reduced balanced dataset (preprocessed as previously explained), where it learns to recognize image patterns through parameter adjustments.

After training, the model's performance is assessed on a validation subset, measuring its ability to generalize to new data. The validation accuracy is recorded for each hyperparameter combination, allowing systematic exploration of different model configurations. The optimal hyperparameters, chosen based on validation accuracy, lead to the highest-performing model for image classification.

The parameters explored during this methodology, along with the corresponding results, are meticulously documented in **Table 5**. The selection of the optimal parameter combination, characterized by the highest accuracy, was instrumental in constructing the definitive models. For the custom CNN built from scratch, the chosen parameters were a learning rate of 0.001, a dropout rate of 0.15, and 30 epochs. Similarly, for DenseNet121, the optimal parameters comprised a learning rate of 0.001, a dropout rate of 0.15, and 10 epochs. In the case of ResNet50, the selected parameters were a learning rate of 0.001, a dropout rate of 0.25, and 30 epochs.

For a comprehensive view of the entire parameter exploration process and its outcomes, refer to the supplementary figures presented in the annex (**Supplementary figures 2-4**).

	Tested parameters		
Model	Learning Rate	Dropout Rate	Number of epochs
CNN built from scratch	0.1, 0.01, **0.001**, 0.0001	0.10, **0.15**, 0.25, 0.35	15, 20, 25, **30**
DenseNet121	0.1, 0.01, **0.001**, 0.0001	0.10, **0.15**, 0.25, 0.35	**10**, 15, 20, 25
ResNet50	0.1, 0.01, **0.001**, 0.0001	0.10, 0.15, **0.25**, 0.35	10, 20, **30**, 40

Table 5: Optimal parameter configurations for maximizing accuracy. This table encapsulates the various parameter configurations tested to identify those leading to the highest accuracy on the reduced subset. The parameters in focus encompass the learning rate, dropout rate, and the number of epochs. The combinations of parameters that demonstrated the highest accuracy for each model are **highlighted in yellow.**

For a detailed understanding of the entire GridSearch process, the complete code is available in the annex as **Code 9** and can be accessed directly on the dedicated GitHub repository [54] in the algorithm-specific files: "GridSearch_CNN.py", "GridSearch_DenseNet.py", "GridSearch_ResNet.py".

It is imperative to acknowledge that, given the random selection process employed for the subset of 1002 images, there is a possibility that the very images used to assess optimal hyperparameters could have also been utilized in the testing of the final model. This scenario introduces the potential for a marginal overfitting of the model. This consideration was overlooked due to the relatively small size of the reduced validation subset, comprising only 100 images out of a total dataset exceeding 9000 images employed in the comprehensive training of the final model. Despite this oversight, it is crucial to recognize the potential influence of overlapping data in both the hyperparameter optimization and model training phases, warranting cautious interpretation of the model's generalization capabilities.

5.10 Training process monitoring

In order to evaluate the training process 2 metrics are studied. The loss metric, represented by the loss function, quantifies how well the model is performing. Lower loss values indicate better alignment between predicted and actual values. The accuracy metric, on the other hand, measures the proportion of correctly classified samples. Higher accuracy values indicate better alignment between predicted and actual values.

- **Training metrics:** Computed based on the model's performance on the training dataset.
  - *Training loss:* Represents the error on the training dataset, ideally decreasing over epochs as the model learns.
  - *Training accuracy:* Indicates the proportion of correctly classified samples on the training dataset, demonstrating the model's proficiency on familiar data.
- **Validation metrics:** Calculated on a separate dataset that the model has not seen during training (validation set).
  - *Validation loss:* Reflects the model's performance on a separate dataset, helping to gauge its generalization capability. A decreasing trend is desirable.
  - *Validation accuracy:* Measures the model's accuracy on unseen data, demonstrating its ability to generalize beyond the training set.

The loss and accuracy graphs play a pivotal role in assessing the performance and behavior of the CNN models throughout the training process. They provide a visual representation of how well the models are learning and generalizing, offering insights into potential issues such as overfitting. Regular monitoring of the training and validation metrics is imperative to strike a balance between model complexity and generalization, ultimately leading to a robust and accurate skin condition classification model.

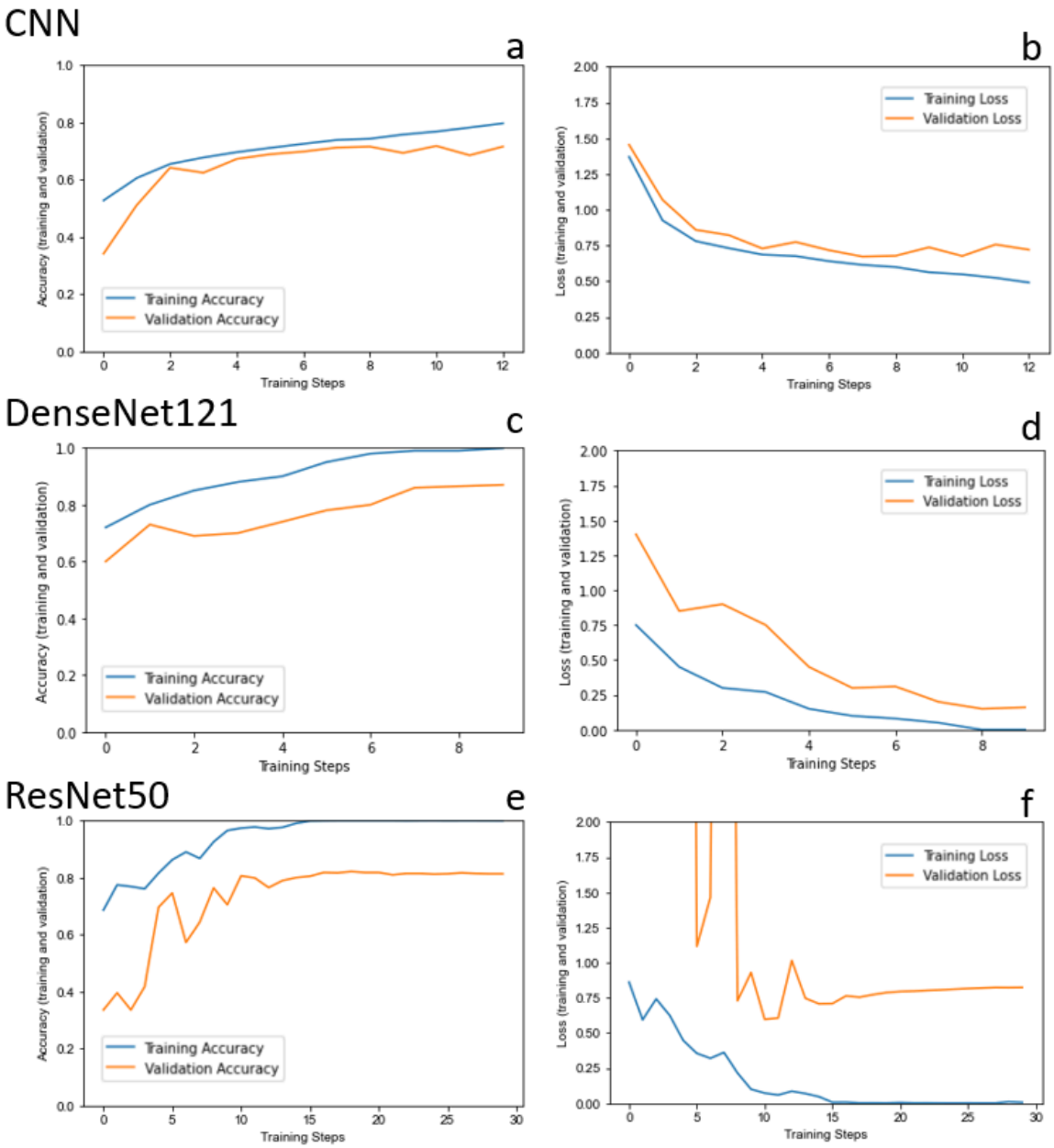


Figure 6: Accuracy and loss curves for each of the three models under study. The graphs provide a comprehensive view of accuracy and loss values across the training process. Training metrics are depicted in blue, while validation metrics are represented in orange. Sub-figures (a) and (b) present the accuracy and loss curves, respectively, for the custom CNN. Similarly, sub-figures (c) and (d) display the accuracy and loss curves for DenseNet121, and sub-figures (e) and (f) illustrate the accuracy and loss curves for ResNet50.

The analysis of the training curves (**Figure 6**) reveals nuanced insights into the performance of each model. Notably, the custom CNN, trained for an initially planned 30 epochs, activated early stopping, indicating that the validation loss reached a plateau and further training would risk overfitting (**Figure 6, b**). In contrast, the DenseNet121 and ResNet50 models employed a different strategy, involving a reduction in the learning rate instead of early stopping.

A closer examination of the ResNet50 model exposes signs of overfitting (**Figure 6, e-f**). The validation loss initially decreases but eventually begins to rise, while the training loss continues to decline (**Figure 6, f**). The notable gap between the validation and training curves suggests a potential lack of generalization to unseen data, emphasizing disparities between the training and validation sets. The use of transfer learning accentuates this point, as certain layers were not originally trained on the same dataset, highlighting a degree of unrepresentativeness in parts of the training dataset.

The DenseNet121 model presents distinct characteristics in its loss and accuracy curves (**Figure 6, c-d**). In the loss plot, a noteworthy observation is the diminishing gap between the training and validation curves, signifying an ongoing learning process as the model adapts to features from the new dataset (**Figure 6, d**). This reduction in the gap suggests an improved ability to generalize to unseen data, indicating the model's capacity to learn relevant patterns effectively.

In contrast, the custom CNN exhibits a more fitting curve, with both the training and validation losses decreasing to stable points, maintaining a smaller gap between them (**Figure 6, b**). This suggests a more stable learning process with improved generalization.

Turning to accuracy, all models demonstrate effective training without significant signs of overfitting, as the validation accuracies do not decline after reaching peak values (**Figure 6, a, c, e**). However, the ResNet50 model's accuracy curve displays a less smooth trajectory compared to the custom CNN and DenseNet121. A smoother learning curve typically indicates a stable and consistent learning process.

Notably, DenseNet121 consistently demonstrates higher accuracy compared to the other models, underscoring its efficacy in capturing intricate patterns and making accurate predictions (**Figure 6, c**). The stability and upward trend in the accuracy curve further affirm the model's robust learning and superior performance.

Despite variations in the training dynamics, all three models exhibit convergence in their loss and accuracy curves, indicating that they have reached their peak performance and are no longer learning significantly from the training data.

In conclusion, the analysis of loss and accuracy curves for the custom CNN, DenseNet121, and ResNet50 models provides valuable insights into their respective training processes. While the custom CNN displays a stable and fitting curve, DenseNet121 stands out with its higher accuracy rates. Despite a slightly less smooth curve, ResNet50 attains convergence, and its characteristics highlight the intricate dynamics of transfer learning. The results collectively contribute to a nuanced understanding of model performance, reinforcing the need for tailored strategies to address specific challenges in dermatological image classification.

The code to obtain the curves is available in the annex as **Code 10**, accompanied by a detailed description. Alternatively, it can be found in the dedicated GitHub repository [54] within the specific files for each algorithm "CNN.py", "DenseNet.py" and "ResNet.py".

5.11 Evaluation metrics

The evaluation of a skin image classification model involves the application of various metrics and visualizations to assess its performance. The metrics and visualizations presented in this evaluation are derived from a distinct test subset of skin images that were intentionally excluded from the model's training phase and fine-tuning. This separation of data into training, validation and testing sets is a fundamental practice in machine learning, serving to ensure a rigorous assessment of the model's performance. The importance of this separation lies in its ability to provide an unbiased and objective evaluation of the model's generalization capabilities. Below are the metrics and visualizations used, along with their respective explanations:

1. **Accuracy:**
   - Accuracy is a fundamental metric for evaluating classification models. It measures the proportion of correctly predicted class labels out of the total predictions.
   - The test accuracy is computed using the "accuracy_score" function from the sklearn.metrics module.
   - The result is expressed as a decimal, indicating the fraction of correct predictions.
2. **Precision:**
   - Precision quantifies the model's ability to correctly predict positive class instances among all instances it predicted as positive.
   - The precision score is calculated using the "precision_score" function with a weighted average.
   - A weighted average is used to account for class imbalances, providing an overall measure of precision.
3. **Sensitivity (recall):**
   - Sensitivity, also known as recall, gauges the model's ability to correctly identify positive instances among all actual positive instances.
   - Similar to precision, the "recall_score" function computes sensitivity with a weighted average.
4. **F1-score:**
   - The F1-score is a harmonic mean of precision and recall. It balances both metrics and is useful in scenarios where precision and recall need to be considered together.
   - The "f1_score" function is applied with a weighted average to calculate the F1-score.
5. **Cohen's kappa:**

- Cohen's kappa is a statistical measure that assesses the agreement between the predicted and actual labels while accounting for chance agreement.
- The “cohen_kappa_score” function quantifies the agreement as a decimal value.

6. **Classification report:**
   - A classification report provides a comprehensive summary of classification performance, including precision, recall, F1-Score, and support for each class.
   - The “classification_report” function generates this report, offering insights into the model's performance for individual classes.
7. **Confusion matrix:**
   - A confusion matrix is a visual representation of a model's performance. It tabulates the true positives, true negatives, false positives, and false negatives.
   - The confusion matrix is obtained using the “confusion_matrix” function.
   - The seaborn library is employed to generate a heatmap representing confusion matrix data, with annotated cell values and a color map to enhance interpretability.

In conclusion, the combination of these metrics and visualizations facilitates a comprehensive evaluation of the skin image classification model. Each metric and visualization contributes to the understanding of different aspects of the model's performance, including accuracy, precision, recall, F1-score, and the distribution of classifications across classes. This comprehensive assessment aids in fine-tuning the model and making informed decisions regarding its deployment.

For a detailed understanding of the entire evaluation process, the complete code is available in the annex as **Code 11** and can be accessed directly on the dedicated GitHub repository [54] in the algorithm-specific files: “CNN.py”, “DenseNet.py” and “ResNet.py”.

5.12 Prediction process

The prediction process for user submitted images is an integral component of our web application's functionality. Upon receiving an image from the user, we implement a standardized preprocessing procedure, mirroring the steps applied to the dataset used for training and evaluation. This process begins with resizing the user submitted image to a consistent dimension of 256x256 pixels, ensuring uniformity in the analysis. Subsequently, pixel values undergo normalization by dividing each pixel value by 255. This normalization step maintains a standardized pixel range, contributing to the model's consistent interpretation of image features.

The prediction is executed using the processed image, and the algorithm determines the skin alteration classification based on the learned patterns and features. The prediction variables are then utilized to construct the results page, providing users with detailed information about the identified skin condition.

To enhance user engagement and understanding, we generate a matplotlib plot that visualizes the probabilities associated with each classification. This plot, calculated during the prediction process, adds a visual layer to the results, illustrating the confidence levels assigned to each potential skin alteration.

For those interested in exploring the technical implementation, the code for this operation is available as **Code 12** in the annex and can be accessed directly in the dedicated GitHub Repository [54] in the "predictor.py" and "views.py" files within the index function.

5.13 Web Development

In order to make the classification algorithm open to the public, a free web-based app was developed. All web-related code can be downloaded and consulted at the dedicated GitHub repository [54] under the "web" folder.

Technology stack

Our web application leverages a streamlined and effective technology stack to deliver a seamless user experience:

1. **Django for backend:**
   - Django, a high-level Python web framework, was selected for its ease of use and rapid development capabilities. With Django, we benefit from a robust backend infrastructure, simplifying the implementation of essential functionalities and ensuring efficient data handling.
2. **HTML and CSS for frontend:**
   - The frontend is crafted using HTML and CSS to provide a clean and intuitive user interface. HTML forms the structural backbone of our pages, while CSS stylizes and enhances the visual appeal, ensuring a user-friendly and aesthetically pleasing design.
3. **JavaScript for client-side validation:**
   - A concise JavaScript implementation is incorporated to perform client-side validation. This ensures an immediate response to users during the image submission process, signaling that their request is being processed. This lightweight client-side validation enhances user engagement and responsiveness.

In summary, our technology stack integrates Django for a robust backend, HTML and CSS for an intuitive frontend design, and a touch of JavaScript for client-side validation. This combination not only facilitates rapid development but also ensures a user friendly and efficient web application for skin condition classification.

Back-end development

In the realm of web development, creating a robust and efficient framework for our application is paramount. This report delves into the structural design of a web application developed using Django, a high-level Python web framework.

Leveraging the power of Django, the project adopts a meticulous organization, adhering to best practices for maintainability and performance.

General folder structure

- **“assets”:** Storing static files like images, CSS, and JavaScript in the assets folder is a common practice to keep the project organized and make it easier to manage and serve these files.
- **“core”:** Contains files dedicated to define how the web application handles requests, manages URLs, and configures various settings.
- **“templates”:** The templates folder is dedicated to holding HTML files, providing a clean separation between the presentation layer and the backend logic.
- **“DenseNet121.h5”:** Trained machine learning model stored as a “.h5” file.
- **“requirements.txt”:** The requirements.txt file is crucial for reproducing the environment. It lists all the dependencies needed for the project, making it easy for others to set up and run our Django application.
- **“manage.py”:** Command-line utility for managing various aspects of the Django project.

Core folder

- **“__init__.py”:** This file is included to treat the directory as a python package. It contains code that will be executed when the package is imported, in this case the model importation.
- **“asgi.py”:** This file configures the asynchronous server gateway interface application for deployment.
- **“views.py”:** Defines the views (functions and classes) that handle HTTP requests and return HTTP responses. These views interact with the templates.
- **“urls.py”:** Contains URL patterns for routing incoming requests to the appropriate views.
- **“settings.py”:** Includes project settings such as database configuration, static files, and middleware.
- **“predictor.py”:** Separate file for the code needed to perform predictions using the machine learning model. This organization helps achieve faster loading times by loading the model only once.
- **“wsgi.py”:** This file is part of the web server gateway configuration, it defines how web servers communicate with Python web applications.

The choice to segregate the predictor code into a standalone file underscores the commitment to efficiency. By loading the machine learning model only once, the application achieves faster loading times, enhancing the overall user experience. This organizational strategy, combined with Django's inherent strengths, positions the web application for seamless expansion and maintenance.

In conclusion, the design philosophy behind this Django web application aims for a harmonious integration of components, fostering a scalable and modular codebase.

Front-end development

Our approach to frontend development prioritizes simplicity, clarity, and an intuitive user experience. Key aspects of our design include:

1. **Layout and navigation:**
   - The web pages are structured to provide users with an easily navigable and cohesive layout. A left-side navigation menu (**Supplementary figure 5**) enhances accessibility, allowing users to seamlessly explore different sections of the application.
2. **Interactive elements:**
   - Interactive elements, strategically incorporated throughout the interface, enhance user engagement. These may include buttons, hover effects, and dynamic loading features, contributing to a more dynamic and enjoyable user experience.
3. **Client-side validation:**
   - A client-side validation using JavaScript is implemented to provide immediate feedback during the image submission process. This feature assures users that their requests are actively being processed, contributing to a responsive and reassuring user experience.
4. **Aesthetics and user experience:**
   - The combination of HTML, CSS, and JavaScript elements is meticulously crafted to ensure a visually appealing and user-friendly interface. This synergy contributes to an aesthetically pleasing design that enhances the overall user experience.

This frontend development strategy focuses on creating an interface that is not only visually appealing but also user centric, emphasizing ease of navigation and interactive elements for an engaging user journey.

Core functionalities

1. **Image submission:**
   - The Home Page serves as the starting point for users (**Supplementary figure 6**), featuring a prominent button that allows them to effortlessly upload an image of a skin condition. This straightforward process ensures a user-friendly entry into the classification system.
2. **Condition prediction:**
   - Upon image submission, users encounter a “Predict Condition” button. Clicking this initiates the algorithm, triggering the processing of the image to determine the skin condition. The “predict condition” functionality streamlines the classification process, providing users with rapid insights into the potential condition of the skin image.

3. **Results display:**
    - The results of the classification, including the predicted skin condition and the associated probability, dynamically appear on the same page (**Supplementary figure 7**). This immediate presentation of results enhances user engagement, allowing users to promptly access critical information about the submitted image.
4. **Dynamic navigation to condition info:**
    - Upon the result display, a dynamic “See more info” button appears, offering users a seamless transition to the “Condition Info” page corresponding to the predicted skin condition. This feature provides users with in depth educational content and insights related to the specific skin condition, enhancing their understanding of the results.

By combining the simplicity of image submission, real time condition prediction, and dynamic navigation to detailed condition information, our web application ensures a cohesive and informative user experience. This streamlined process empowers users to quickly obtain results and delve deeper into educational content, contributing to a comprehensive skin condition classification system.

Additional pages

1. **BCC Info, Melanoma Info, Nevus Info:**
    - These pages offer comprehensive educational content about specific skin conditions – BCC (**Supplementary figure 8**), Melanoma (**Supplementary figure 9**), and Nevus (**Supplementary figure 10**). Users can access detailed information, including causes, symptoms, and potential treatments, providing a valuable resource for enhancing their knowledge about dermatological conditions.
2. **References:**
    - The “References” page (**Supplementary figure 11**) serves as a repository of cited sources used in developing and validating the skin condition classification model. This transparency ensures users have access to the background literature supporting the accuracy and reliability of the application.
3. **Specifications:**
    - The “Specifications” page (**Supplementary figure 12**) provides technical details about the skin condition classification model. Users interested in the underlying architecture, parameters, and development methodologies can find in-depth information, adding a layer of transparency to the functionality of the application.
4. **Privacy policy:**
    - The “Privacy Policy” page (**Supplementary figure 13**) outlines how user data is handled, ensuring transparency and adherence to privacy regulations. Users can review the privacy policy to understand how their information is collected, used, and protected, fostering trust and confidence in the application.
5. **About me:**
    - The “About Me” page (**Supplementary figure 14**) introduces users to the creator of the web application, offering a personal touch and background

information. This section provides users with insights into the developer's motivations, expertise, and the journey behind creating the skin condition classification system.

Each additional page is designed to fulfill a specific purpose, ranging from educational content to technical specifications and privacy information. This multifaceted approach ensures users have access to both informative content and the transparency necessary for a trustworthy and reliable user experience.

5.14 Deployment

The deployment of our skin condition classification system followed a practical step by step process. Beginning with the project's placement on a dedicated GitHub repository, we then utilized an existing virtual machine running Linux Ubuntu within a Proxmox host OS. After setting up essential components like Git, Docker, and Docker Compose, the project was cloned onto the virtual machine. The Docker image, defined by the provided Dockerfile (available at dedicated GitHub Repository [54] root folder as “Dockerfile”), was built and used to launch a Docker container encapsulating the application.

To make the application accessible to the public, we added a DNS A record linking the subdomain "skinai.bioedu.one" to the virtual machine's IP address. For secure and efficient communication, we configured a reverse proxy in the Caddyfile, directing traffic from the specified hostname to the Docker container. A simple restart of Caddy then automatically generated SSL certificates, ensuring secure interactions with the application.

This deployment strategy, emphasizing practicality and efficiency, successfully made our skin condition classification system publicly available while maintaining security and scalability.

6. Results

6.1 Algorithm performance evaluation

In the ensuing tabulation (**Table 6**), discernible trends in the performance metrics of various models become apparent, with DenseNet121 emerging as the frontrunner in terms of superior outcomes. Specifically, DenseNet121 exhibited a noteworthy 87% accuracy and precision, surpassing its counterparts in the comparative analysis. This achievement underscores the efficacy of DenseNet121 in accurately classifying dermatological images, signifying its robust discriminative capacity. In contrast, alternative models within the experimental framework demonstrated commendable yet comparatively lower performance, with accuracies ranging from 73% to 81%. This variance in performance emphasizes the discerning nature of model selection, as the inherent architecture and characteristics of DenseNet121 evidently contribute to its heightened predictive accuracy and precision in the context of skin condition classification. The detailed performance metrics encapsulated in the presented table elucidate the nuanced distinctions in model efficacy, thereby providing valuable insights for subsequent analyses and considerations in the realm of dermatological image classification.

Metric	CNN	DenseNet121	ResNet50
Accuracy	0.73	0.87	0.81
Precision	0.73	0.87	0.81
Sensitivity	0.73	0.87	0.81
F1-Score	0.72	0.87	0.81
Cohen’s Kappa	0.59	0.80	0.71

Table 6: Model performance metrics. Comprehensive array of performance metrics for the three studied skin condition classification models. Accuracy, precision, sensitivity, F-1 score, and Cohen's Kappa are detailed for each model, providing a thorough evaluation based on a designated test subset.

Upon meticulous examination of the confusion matrices depicted in the ensuing figure (**Figure 7**), a resounding confirmation of DenseNet's superior accuracy in skin condition classification emerges. Notably, DenseNet exhibits remarkable proficiency, particularly evident in its discernment of BCC, achieving an impressive 90% correct classification rate. However, a nuanced pattern surfaces in the discrimination between melanoma and nevus, wherein the model encounters challenges, occasionally misclassifying melanoma as nevus and vice versa. This observation is substantiated by the classification reports, revealing a commendable accuracy rate of 87% for melanoma and 84% for nevus, further reaffirming the nuanced complexities inherent in distinguishing between these two classes. The model's adeptness in accurately identifying BCC juxtaposed with the intricacies

encountered in the melanoma and nevus classification underscores the nuanced nature of dermatological image analysis.

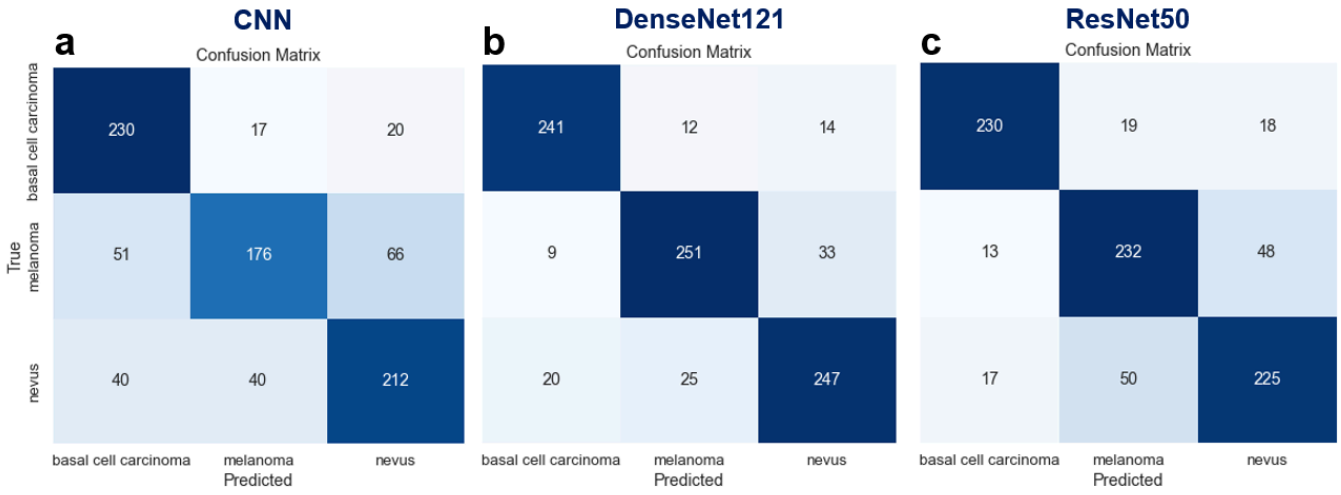


Figure 7: Confusion matrices for skin condition classification models. Confusion matrices for three models: (**a**) CNN built from scratch, (**b**) DenseNet121, and (**c**) ResNet50. True labels are represented on the y-axis, predicted labels on the x-axis. The diagonal indicates correct classifications, providing a concise visual summary of model performance in dermatological condition classification.

In light of its consistent superior performance across all evaluated instances, DenseNet121 stands as the unequivocal choice for deployment within the web application. The decision to select DenseNet121 for deployment is grounded in its demonstrable efficacy in accurately classifying diverse skin conditions, as evidenced by its notable achievements in accuracy, precision, and successful discrimination among various dermatological categories. This judicious selection aligns with the overarching objective of ensuring optimal diagnostic outcomes within the web application, fostering an enhanced user experience and bolstering the reliability of dermatological assessments. The adoption of DenseNet121 for deployment reflects a strategic decision informed by empirical evidence, aiming to deliver a high quality and reliable skin condition classification service to users engaging with the web application.

Upon scrutiny of the comprehensive classification reports (**Table 7**), a discernible pattern emerges, shedding light on the inherent complexities associated with skin condition classification. It becomes evident that nevus poses the greatest challenge for all models, followed by melanoma, and lastly, BCC, in terms of classification accuracy. This ranking of difficulty underscores the nuanced nature of distinguishing between certain skin conditions, with nevus exhibiting a particularly intricate discriminative landscape for machine learning models. This observation is of paramount importance as it not only informs the strengths and limitations of the deployed models but also holds implications for clinical practice. Understanding the relative difficulty in accurately classifying specific skin conditions is crucial for clinicians who may rely on automated systems for preliminary assessments.

Analyzing the performance of a classification model on a class wise basis is crucial for gaining nuanced insights into its discriminatory capabilities. While overall metrics like accuracy provide a general overview of a model's effectiveness, class-wise metrics, including precision, recall, and F1-score, offer a more detailed perspective. In scenarios where different classes hold varying degrees of significance, this granular evaluation becomes especially valuable. For instance, in a medical context, the importance of correctly classifying certain conditions may outweigh others, warranting a focused examination of the model's performance for each class.

In some applications, where the model is intended for general purposes and all three classes are considered equally important, a balanced performance across classes is desirable. However, in medical diagnoses reducing false positives for specific conditions may be prioritized. This could be crucial, especially in situations where false positives may lead to unnecessary concern or anxiety among users. By scrutinizing the model's precision, recall, and F1-score for each class, tailored adjustments can be made to prioritize certain classes based on the specific requirements of the application. In the context of a free to use model without doctor consultation, mitigating false positives is particularly pertinent to enhance user trust and minimize unwarranted distress.

Additionally, this insight contributes to ongoing research efforts aimed at refining model architectures and training strategies to further enhance their proficiency in tackling the intricate challenges of dermatological image classification, ultimately advancing the state-of-the-art in medical diagnostics.

	Precision	Recall	F1-score	Support
CNN				
BCC	0.72	0.86	0.78	267
Melanoma	0.76	0.60	0.67	293
Nevus	0.71	0.73	0.72	292
DenseNet121				
BCC	0.89	0.90	0.90	267
Melanoma	0.87	0.86	0.86	293
Nevus	0.84	0.85	0.84	292
ResNet50				
BCC	0.88	0.86	0.87	267
Melanoma	0.77	0.79	0.78	293
Nevus	0.77	0.77	0.77	292

Table 7: Class-wise performance metrics for the three studied models. Detailed breakdown of precision, recall, and F1-score for each class within the three studied skin condition classification models.

6.2 Web application implementation

The web application employs the best performing algorithm we just analyzed, this being the DenseNet121. The web can be visited at the following link: https://skinai.bioedu.one/ [55].

Navigation

Users can navigate through the website using the left-side menu, providing easy access to different sections (**Supplementary figure 5**). A button with a direct link to the main page is provided at the end of each page for ease of navigation.

Home page

Features a user friendly interface with image submission and prediction buttons, displaying results on the same page (**Supplementary figure 6,** link). It hosts the prediction process:

1. **Image submission:**
   - Users submit skin images through the dedicated button on the home page.
2. **Prediction calculation:**
   - Upon clicking the "Predict Condition" button, the algorithm processes the input image using DenseNet121.
3. **Result presentation:**
   - The algorithm outputs the predicted class (melanoma, nevus, or BCC) and the associated probability (**Supplementary figure 7**).
   - A dynamic button provides a direct link to the information page of the predicted condition.

The homepage also includes a link to the specifications page in case the user needs to better understand how the website works and how the algorithm works. To facilitate testing without submitting personal images, users can download nine example images from the home page using the dedicated link.

Information pages

Three dedicated pages offer health information on BCC (**Supplementary figure 8,** link), melanoma (**Supplementary figure 9,** link), and nevus (**Supplementary figure 10,** link). The general layout of the pages includes:

- **Overview:** A comprehensive introduction to the respective skin condition, covering its nature, prevalence, and general characteristics.
- **Causes and risks factors:** In depth exploration of the potential causes and risk factors associated with the development of the specific skin condition.
- **Symptoms:** Detailed information on the observable symptoms and manifestations linked to the particular skin disorder.
- **Treatment:** An overview of available treatments, therapeutic approaches, and medical interventions for managing and addressing the skin condition.

- **Prevention:** Strategies and recommendations for preventing the onset or recurrence of the skin condition, focusing on proactive measures and lifestyle choices.
- **How to tell melanomas and moles apart** (only present in the melanoma and nevus pages): Specific guidance on distinguishing between melanomas and moles, providing users with valuable insights into differentiating benign and potentially malignant skin features.

Additional Sections

- **References:** This section provides essential citations and sources of information used in the project (**Supplementary figure 11,** link).
- **Specifications:** Detailed project specifications are available to enhance user understanding of the application's functionality (**Supplementary figure 12,** link).
- **Privacy:** The privacy section outlines important statements regarding user privacy and project's licenses (**Supplementary figure 13,** link).
- **About Me:** Users can find information about the developer and the project in this section (**Supplementary figure 14,** link).

In conclusion, the application successfully leverages DenseNet121 for accurate skin condition classification, providing users with informative predictions and valuable health related content.

7. Conclusion and future vision

7.1 Conclusions

In the pursuit of developing a robust skin condition classification system, our endeavors have yielded noteworthy results. The implemented algorithm, leveraging a combination of a custom CNN model and transfer learning with DenseNet121, exhibited a commendable accuracy rate of 87%. This outcome surpasses our initial goal of achieving an 85% accuracy, affirming the efficacy of our approach in effectively classifying skin conditions.

The web application, designed for user-friendly interaction, has fulfilled its objectives. Users can seamlessly upload images, predict skin conditions with a high level of accuracy, and access comprehensive educational content related to predicted conditions. The integration of client-side validation and dynamic navigation enhances the overall user experience, ensuring a smooth journey from image submission to obtaining insightful results.

In conclusion, the set objectives have been not only met but exceeded. The algorithm's performance at 87% accuracy surpasses our predefined goal of 85%, demonstrating the robustness and reliability of our skin condition classification model. The web application operates as anticipated, providing users with a straightforward yet comprehensive tool for understanding and predicting skin conditions.

7.2 Methodology challenges and adjustments

The execution of the skin condition classification project encountered significant challenges, primarily stemming from computational limitations and class imbalances within the dataset. In response to these constraints, a strategic decision was made to streamline the dataset by excluding classes with fewer images, focusing exclusively on nevus, melanoma, and BCC. This approach aimed to address the pronounced class imbalance and improve the feasibility of achieving optimal model performance.

Compounding the computational hurdles was the inability to harness GPU acceleration, given TensorFlow's exclusive compatibility with NVIDIA GPUs. Consequently, model training became a resource intensive process, requiring 12 to 24 hours at 100% CPU usage. To address this, a Google Colab Pro subscription was acquired, offering an upgraded computational environment with 52 GB of RAM and an A1000 NVIDIA Graphics card. This subscription expedited the models fine-tuning process.

The limited VRAM led to the creation of a 1002 image subset, facilitating a GridSearch to optimize hyperparameters. Following this optimization, the model with the most favorable hyperparameters was trained on the local machine's CPU.

It is crucial to note that while these hyperparameters were deemed optimal within the constraints of the 1002 image subset, they may not necessarily represent globally optimal parameters due to the subset's inherent randomness, introducing uncertainty into the hyperparameter optimization process. These adjustments were imperative to overcome the computational challenges and achieve meaningful results in the context of the project's scope.

7.3 Limitations

While our skin condition classification project demonstrates promising results, several limitations should be acknowledged. The lack of optimal responsiveness on various devices may hinder the user experience, prompting ongoing efforts to enhance accessibility. Additionally, the absence of comprehensive user testing and feedback collection represents a limitation in evaluating the application's usability and addressing user preferences.

One notable limitation of the web application lies in its language accessibility, as the content is currently available exclusively in english. While some web browsers offer automatic translation features, providing a level of accessibility to users worldwide, the lack of native multilingual support may pose a barrier for individuals more comfortable interacting with the application in their preferred language. Future iterations could explore the implementation of multilingual support to enhance accessibility and accommodate a more diverse user base, ensuring that individuals from various linguistic backgrounds can benefit from the dermatological insights provided by the application.

It is crucial to emphasize that the web application is not a substitute for professional healthcare counseling. As the developer lacks medical expertise, the information provided is purely algorithmically generated, and users are advised to seek professional medical advice for accurate diagnosis and treatment.

Furthermore, the inherent limitations of AI and CNN algorithms must be acknowledged. The model's accuracy is contingent upon the quality and diversity of the training data, and it may not generalize perfectly to all skin conditions or demographic groups. Additionally, the model's performance is constrained by the computational resources available, impacting the scale of the dataset and the optimization of hyperparameters. It's important to note that the training and testing datasets consisted of dermoscopic images, whereas users will primarily submit non-dermoscopic images, potentially influencing the accuracy of the classification for user submitted images.

While the training dataset was carefully balanced to ensure equal representation of skin condition classes, it is essential to acknowledge the potential limitation associated with imbalanced real world scenarios. In actual user submissions, the prevalence of specific skin conditions may vary, and imbalances in class distribution could impact the model's performance. While efforts were made to train the algorithm on diverse data, the unpredictable nature of user generated

submissions introduces a level of uncertainty. This consideration is crucial for understanding the model's behavior in practical, dynamic settings where imbalances in skin condition occurrences may differ from the training dataset.

These limitations collectively underscore the need for continual refinement, user engagement, and collaboration with healthcare professionals to ensure the responsible and effective use of our skin condition classification system.

Exclusivity of training data to lighter skin tones

A fundamental limitation of our skin condition classification system arises from its exclusive reliance on a training dataset composed solely of images from individuals with lighter skin tones. Regrettably, this exclusive composition inherently restricts the tool's applicability to individuals with darker skin tones. The absence of any representation of colored skin images in the training data is not a result of a deliberate choice but stems from a lack of proper data availability.

This inherent limitation emphasizes the challenges posed by the unavailability of diverse dermatological images and the consequent inability to include individuals with darker skin tones in the training data [62]. While recognizing this constraint, it is crucial to transparently communicate the tool's current limitations, emphasizing the necessity of ongoing efforts to overcome these challenges.

Addressing this limitation necessitates a critical examination of the current state of dermatological databases and calls for a collective commitment within the scientific community [63]. There is a pressing need for substantial efforts not only to diversify the training dataset but also to initiate extensive research collaborations that deliberately encompass the full spectrum of skin tones [64]. This call extends to researchers, clinicians, and institutions involved in dermatological studies, urging a heightened awareness of the importance of inclusivity in data collection for advancing AI tools in healthcare. By actively acknowledging and rectifying the disparities in available data, we can collectively contribute to the development of more equitable and universally effective dermatological AI solutions.

7.4 SDGs revision

The surpassing of the expected accuracy by our skin condition classification algorithm aligns seamlessly with SDG 3, which focuses on health. By providing users with accurate information about their skin lesions, our application actively contributes to global health. Users gain valuable insights into dermatological conditions, empowering them to make informed decisions about their skin health.

However, when considering SDG 4, which emphasizes quality education, challenges arise due to the absence of professional healthcare participation. While rigorous research has been conducted to ensure the inclusion of quality information, the lack of direct involvement from healthcare professionals poses limitations. This prompts ongoing efforts to explore collaborations with medical

experts, aiming to enhance the educational aspects of the application and provide users with even more reliable resources.

In the realm of SDG 10, which centers on reducing inequalities, our web application stands out. It is freely accessible to anyone with an internet connection, ensuring inclusivity and distinguishing itself from other apps that often require monthly payments and the installation of dedicated, potentially incompatible applications. This approach actively contributes to reducing disparities in access to dermatological information. However, it's crucial to acknowledge a limitation, the current version of the tool primarily caters to images of lighter skin tones, highlighting the necessity for future developments to address inclusivity across a broader spectrum of skin types.

Furthermore, in line with SDG 9's focus on industry, innovation, and infrastructure, our hosting methods were carefully considered for sustainability. The chosen state-of-the-art technology involves the use of an energy efficient shared machine that runs regardless of the web hosting status, ensuring responsible energy usage. This decision not only contributes to environmental sustainability but also aligns with SDG 7, which emphasizes affordable and clean energy.

User data management adheres to the principles of privacy and security. No user data is stored, reflecting our commitment to responsible data practices and aligning with SDG 16, which emphasizes peace, justice, and strong institutions. This comprehensive approach to sustainability ensures that our skin condition classification system not only contributes to individual well being but also actively supports broader societal and environmental goals outlined in the SDGs.

7.5 Discussion

In contrast to established dermatological applications such as DermEngine [49], SkinVision [50], or FirstDerm [51], this web application stands out for its unwavering commitment to transparency and accessibility. The meticulous detailing of our methodology, extensive data exploration, and transparent presentation of results are all accessible to the public. This transparency not only empowers users to scrutinize the inner workings of the implemented machine learning algorithm but also fosters informed decision making regarding the reliability of the classification results.

Upon surveying available applications on platforms like the Android PlayStore, offerings such as AI Dermatologist [65], ModelDerm [66], SkinCheck [67], and Skinive [68] emerged. While some of these apps provide information on usage and functionality, none rival the depth of explanation found in our documentation. AI Dermatologist, for instance, discloses metrics like accuracy and employs professional revision for result validation, but it is a paid service, limiting open public access. Apps like SkinCheck and Skinive necessitate user registration, concealing their algorithmic and methodological details until after the registration process if present at all. Moreover, some apps mandate user agreement to terms

of service allowing image usage for algorithm improvement, raising privacy concerns.

Among these applications, ModelDerm may have a steeper learning curve but introduces a valuable functionality absent in our web application. It automatically detects unsuitable or blurry images, providing users with warnings and instructions to capture suitable images for accurate diagnosis, a critical step for result accuracy aligned with training data images.

While prioritizing accessibility through cost free services, our web application acknowledges a trade-off, the absence of expert dermatological consultation. In contrast to apps offering direct interaction with certified dermatologists at a fee, our application relies solely on algorithmic analysis. This limitation is transparently disclosed, emphasizing the importance of users seeking professional medical advice for crucial dermatological concerns.

In the realm of dermatological applications, the emphasis on transparency, accessibility, and cost free access positions this web application as a valuable resource for individuals seeking preliminary insights into their skin conditions. As future iterations unfold, potential collaborative efforts with healthcare professionals could further enhance the web app's utility, bridging the gap between algorithmic analysis and expert consultation.

7.6 Future vision

Looking ahead, our vision encompasses continuous improvement and expansion of the skin condition classification system. Key areas for future development include:

1. **Enhanced algorithm optimization:**
    - Continued refinement and optimization of the skin condition classification algorithm to improve accuracy and accommodate a broader spectrum of dermatological images.
    - This will mitigate limitations related to the model's generalizability and performance.
2. **User interface refinement for improved accessibility:**
    - Iterative enhancements to the web application's user interface, with a focus on improved mobile responsiveness and additional features to enhance user engagement.
    - Efforts to optimize the application for various devices aim to overcome limitations in accessibility, ensuring a seamless user experience across platforms.
3. **Incorporation of user feedback:**
    - Active solicitation and incorporation of user feedback to address evolving user needs, enhance usability, and ensure the application remains user centric.

- This strategy directly mitigates limitations associated with the lack of thorough user testing.

4. **Integration of additional skin conditions:**
    - Expansion of the skin condition classification system to include a broader array of dermatological conditions, enhancing the utility and scope of the application.
    - This expansion aims to enhance the utility and relevance of the application, overcoming limitations related to the coverage of skin conditions.
5. **Collaborative research and development:**
    - Collaboration with healthcare professionals and researchers to continually refine the algorithm's accuracy and ensure alignment with evolving dermatological knowledge.
    - This collaborative approach ensures continual refinement of the algorithm's accuracy and alignment with evolving dermatological knowledge, mitigating limitations associated with the static nature of the current model.

In conclusion, our current achievements mark a significant milestone in skin condition classification, laying the foundation for a future where our web application becomes an increasingly valuable resource for users seeking accurate and accessible dermatological insights. Through a commitment to innovation and user centric development, we envision an ever improving platform that contributes to both individual well being and broader dermatological research.

8. Glossary

8.1 Definitions

Artificial intelligence: The simulation of human intelligence in machines, allowing them to perform tasks that typically require human intelligence, such as problem-solving, understanding natural language, and visual perception.

Machine learning: A subfield of artificial intelligence that focuses on the development of algorithms and models that enable computers to learn and make predictions or decisions based on data.

Deep learning: A subset of machine learning that involves the use of artificial neural networks with multiple layers (deep neural networks) to model and process complex patterns and representations in data.

Fine tuning: The process of adjusting a pre-trained neural network on a specific dataset to adapt it to a new task. Fine tuning involves modifying the existing model to improve its performance on a targeted set of data.

Hyperparameter optimization: The process of fine-tuning the hyperparameters (configurable settings external to the model) to enhance the performance and efficiency of a machine learning model.

Transfer learning: A machine learning technique where a model trained on one task is repurposed for a different but related task. Transfer learning can expedite model development and enhance performance.

Convolutional neural network: A type of deep neural network designed for processing and analyzing visual data. CNNs are particularly effective in tasks such as image recognition and classification.

DenseNet121: Specific architecture CNN designed for image classification tasks. It is part of the DenseNet family, characterized by dense connectivity patterns, where each layer receives direct input from all preceding layers. DenseNet121 specifically refers to a DenseNet model with 121 layers, providing a balance between model complexity and computational efficiency.

Accuracy: Metric used to evaluate the performance of a machine learning model. It represents the ratio of correctly predicted instances to the total instances in the dataset. The accuracy score provides an overall measure of how well the model can correctly classify different classes, with higher values indicating better performance.

Dermoscopic images: Visual representations, typically photographs or scans, captured using dermoscopy, a non-invasive imaging technique used to examine

skin lesions. These images provide detailed views of pigmented skin lesions, aiding in the diagnosis and analysis of dermatological conditions.

Melanoma: Is a type of skin cancer that originates in the pigment producing cells of the skin. It often appears as a new mole or a change in an existing mole and can spread to other parts of the body if not detected and treated early.

Nevus: Commonly referred to as a mole, is a benign growth on the skin composed of pigment producing cells. Moles can vary in color, shape, and size, and most are harmless. However, changes in the appearance of a nevus may warrant medical attention for evaluation.

Basal cell carcinoma: One of the most common types of skin cancer. It originates in the basal cells, which are found in the lower epidermis. BCC typically appears as a shiny, pink, or red bump on the skin and tends to grow slowly. While it rarely spreads to other parts of the body, early detection and treatment are essential.

Web App: An interactive software application accessed and operated through a web browser.

User centric development: A development approach that prioritizes the needs, preferences, and experiences of end users throughout the design and implementation of a product or system.

Sustainable development goals: A set of global goals established by the United Nations to address various societal and environmental challenges, with aims such as good health and well-being, quality education, industry innovation, infrastructure, and reduced inequalities.

8.2 Abbreviations list

AI	Artificial Intelligence
BCC	Basal Cell Carcinoma
CNN	Convolutional Neural Network
ISIC	International Skin Imaging Collaboration
kNN	k-Nearest Neighbors
ML	Machine Learning
NIH	National Institute of Health
RAM	Random-Access Memory
ReLU	Rectified Linear Unit
SCC	Squamous Cell Carcinoma
SDG	Sustainable Development Goals
SGC	Sebaceous Gland Carcinoma
SVM	Support Vector Machines

9. Annex

9.1 Acknowledgments

I would like to express my sincere gratitude to my tutor, Romina Astrid Rebrij, whose invaluable guidance, recommendations, and unwavering support played a crucial role in the successful completion of this project. Her expertise and insights greatly contributed to the development and refinement of the web application and machine learning algorithm.

I am also grateful to all the educators and mentors who have been instrumental in shaping my educational journey. Their dedication to fostering learning and intellectual growth has been crucial in my development.

A heartfelt thank you goes out to my friends and family for their unwavering support and encouragement throughout this academic endeavor. Their belief in my abilities and constant encouragement provided the motivation needed to overcome challenges and reach the culmination of this project.

9.2 Bibliography

[1] Braun-Falco, O., Plewig, G., Wolff, H. H., & Winkelmann, R. K. (2013). Dermatology. *Springer Science & Business Media*.

[2] Craythorne, E., & Al-Niami, F. (2017). Skin cancer. *Medicine*, 45(7), 431-434.

[3] Leiter, U., & Garbe, C. (2008). Epidemiology of melanoma and nonmelanoma skin cancer—the role of sunlight. *Sunlight, vitamin D and skin cancer,* 89-103.

[4] Netscher, D. T., Leong, M., Orengo, I., Yang, D., Berg, C., & Krishnan, B. (2011). Cutaneous malignancies: melanoma and nonmelanoma types. *Plastic and reconstructive surgery, 127(3)*, 37e-56e.

[5] Abhishek, K., & Khunger, N. (2015). Complications of skin biopsy. *Journal of cutaneous and aesthetic surgery, 8(4),* 239.

[6] Holme, S. A., Malinovszky, K., & Roberts, D. L. (2000). Changing trends in non‐melanoma skin cancer in South Wales, 1988–98. *British Journal of Dermatology, 143(6),* 1224-1229.

[7] Khan, M. Q., Hussain, A., Rehman, S. U., Khan, U., Maqsood, M., Mehmood, K., & Khan, M. A. (2019). Classification of melanoma and nevus in digital images for diagnosis of skin cancer. *IEEE Access, 7,* 90132-90144.

[8] Katalinic, A., Kunze, U., & Schäfer, T. (2003). Epidemiology of cutaneous melanoma and non-melanoma skin cancer in Schleswig-Holstein, Germany: incidence, clinical subtypes, tumour stages and localization (epidemiology of skin cancer). *British Journal of Dermatology, 149(6),* 1200-1206.

[9] NIH: National Cancer Institute. (2022). *Cancer Stat Facts: Melanoma of the Skin.* Retrieved October 15, 2023, from https://seer.cancer.gov/statfacts/html/melan.html

[10] NIH: National Cancer Institute. (2019, December 16). *Melanoma*. Retrieved October 15, 2023, from https://medlineplus.gov/melanoma.html

[11] Madan, V., Lear, J. T., & Szeimies, R. M. (2010). Non-melanoma skin cancer. *The lancet, 375(9715),* 673-685.

[12] Lantz, B. (2021). Overview of Machine Learning Tools. In The Machine Age of Customer Insight (pp. 79-90). *Emerald Publishing Limited.*

[13] Donalek, C. (2011). Supervised and unsupervised learning. *In Astronomy Colloquia*. USA (Vol. 27, p. 8).

[14] Lantz, B. (2019). Machine learning with R: expert techniques for predictive modeling. *Packt publishing ltd.*

[15] Handelman, G. S., Kok, H. K., Chandra, R. V., Razavi, A. H., Huang, S., Brooks, M., ... & Asadi, H. (2019). Peering into the black box of artificial intelligence: evaluation metrics of machine learning methods. *American Journal of Roentgenology, 212(1),* 38-43.

[16] Dietterich, T. (1995). Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR), 27(3)*, 326-327.

[17] Roelofs, R., Shankar, V., Recht, B., Fridovich-Keil, S., Hardt, M., Miller, J., & Schmidt, L. (2019). A meta-analysis of overfitting in machine learning. *Advances in Neural Information Processing Systems, 32*.

[18] Probst, P., Boulesteix, A. L., & Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research, 20(1)*, 1934-1965.

[19] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521(7553)*, 436-444.

[20] Erickson, B. J., Korfiatis, P., Akkus, Z., Kline, T., & Philbrick, K. (2017). Toolkits and libraries for deep learning. Journal of digital imaging, 30, 400-405.

[21] Shinde, P. P., & Shah, S. (2018). A review of machine learning and deep learning applications. In 2018 *Fourth international conference on computing communication control and automation (ICCUBEA)* (pp. 1-6). IEEE.

[22] Norman, G. R., Rosenthal, D., Brooks, L. R., Allen, S. W., & Muzzin, L. J. (1989). The development of expertise in dermatology. *Archives of dermatology, 125(8),* 1063-1068.

[23] Jartarkar, S. R., Cockerell, C. J., Patil, A., Kassir, M., Babaei, M., Weidenthaler‐Barth, B., ... & Goldust, M. (2023). Artificial intelligence in Dermatopathology. *Journal of Cosmetic Dermatology, 22(4),* 1163-1167.

[24] Efimenko, M., Ignatev, A., & Koshechkin, K. (2020). Review of medical image recognition technologies to detect melanomas using neural networks. *BMC bioinformatics, 21,* 1-7.

[25] Wells, A., Patel, S., Lee, J. B., & Motaparthi, K. (2021). Artificial intelligence in dermatopathology: Diagnosis, education, and research. *Journal of Cutaneous Pathology, 48(8),* 1061-1068.

[26] Young, A. T., Xiong, M., Pfau, J., Keiser, M. J., & Wei, M. L. (2020). Artificial intelligence in dermatology: a primer. *Journal of Investigative Dermatology, 140(8)*, 1504-1512.

[27] Saravanan, S., Heshma, B., Shanofer, A. A., & Vanithamani, R. (2020). Skin cancer detection using dermoscope images. Materials Today: *Proceedings, 33,* 4823-4827.

[28] Du-Harpur, X., Watt, F. M., Luscombe, N. M., & Lynch, M. D. (2020). What is AI? Applications of artificial intelligence to dermatology. *British Journal of Dermatology, 183(3),* 423-430.

[29] Reisinho, J., Coimbra, M., & Renna, F. (2020). Deep convolutional neural network ensembles for multi-classification of skin lesions from dermoscopic and clinical images. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)* (pp. 1940-1943). IEEE.

[30] Yuan, Y., Han, Y., Yap, C. W., Kochhar, J. S., Li, H., Xiang, X., & Kang, L. (2023). Prediction of drug permeation through microneedled skin by machine learning. *Bioengineering & Translational Medicine, e10512*.

[31] Kroll, J. A. (2018). Data science data governance [AI ethics]. *IEEE Security & Privacy, 16(6),* 61-70.

[32] O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

[33] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)* (pp. 1-6). IEEE.

[34] Nigat, T. D., Sitote, T. M., & Gedefaw, B. M. (2023). Fungal Skin Disease Classification Using the Convolutional Neural Network. *Journal of Healthcare Engineering*, 2023.

[35] Zhang, N., Cai, Y. X., Wang, Y. Y., Tian, Y. T., Wang, X. L., & Badami, B. (2020). Skin cancer diagnosis based on optimized convolutional neural network. *Artificial intelligence in medicine, 102,* 101756.

[36] Biswal, A. (2023). Convolutional Neural Network Tutorial. *Simplilearn*. Retrieved October 15, 2023, from https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

[37] Cullell-Dalmau, M., Noé, S., Otero-Viñas, M., Meić, I., & Manzo, C. (2021). Convolutional neural network for skin lesion classification: understanding the fundamentals through hands-on learning. *Frontiers in Medicine, 8,* 644327.

[38] Brinker, T. J., Hekler, A., Enk, A. H., Klode, J., Hauschild, A., Berking, C., ... & Schrüfer, P. (2019). A convolutional neural network trained with dermoscopic images performed on par with 145 dermatologists in a clinical melanoma image classification task. *European Journal of Cancer, 111,* 148-154.

[39] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). *Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639),* 115-118.

[40] Gore, J. C. (2020). Artificial intelligence in medical imaging. *Magnetic resonance imaging, 68,* A1-A4.

[41] Yanagisawa, Y., Shido, K., Kojima, K., & Yamasaki, K. (2023). Convolutional neural network-based skin image segmentation model to improve classification of skin diseases in conventional and non-standardized picture images. *Journal of dermatological science, 109(1)*, 30-36.

[42] Wei, M., Wu, Q., Ji, H., Wang, J., Lyu, T., Liu, J., & Zhao, L. (2023). A Skin Disease Classification Model Based on DenseNet and ConvNeXt Fusion. *Electronics, 12(2),* 438.

[43] Xie, F., Yang, J., Liu, J., Jiang, Z., Zheng, Y., & Wang, Y. (2020). Skin lesion segmentation using high-resolution convolutional neural network. *Computer methods and programs in biomedicine, 186,* 105241.

[44] Morid, M. A., Borjali, A., & Del Fiol, G. (2021). A scoping review of transfer learning research on medical image analysis using ImageNet. *Computers in biology and medicine, 128*, 104115.

[45] Höhn, J., Hekler, A., Krieghoff-Henning, E., Kather, J. N., Utikal, J. S., Meier, F., ... & Brinker, T. J. (2021). Integrating patient data into skin cancer classification using convolutional neural networks: systematic review. *Journal of medical Internet research, 23(7),* e20708.

[46] Li, Z., Wang, H., Han, Q., Liu, J., Hou, M., Chen, G., ... & Weng, T. (2022). Convolutional Neural Network with Multiscale Fusion and Attention Mechanism for Skin Diseases Assisted Diagnosis. *Computational Intelligence and Neuroscience, 2022*.

[47] Nunnari, F., Kadir, M. A., & Sonntag, D. (2021). On the overlap between grad-cam saliency maps and explainable visual features in skin cancer images. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction (pp. 241-253)*. Cham: Springer International Publishing.

[48] Brinker, T. J., Hekler, A., Utikal, J. S., Grabe, N., Schadendorf, D., Klode, J., ... & Von Kalle, C. (2018). Skin cancer classification using convolutional neural networks: systematic review. *Journal of medical Internet research, 20(10),* e11936.

[49] MetaOptima. (2023). *DermEngine*. Retrieved November 05, 2023, from https://www.dermengine.com

[50] SkinVision. (2021). *Skin cancer melanoma detection app*. Retrieved November 05, 2023 from https://www.skinvision.com/

[51] FirstDerm. (2022). *The Fastest Way to Ask an Online Dermatologist*. Retrieved November 05, 2023 from https://firstderm.com/

[52] United Nations. (2023). *The 17 Goals*. Retrieved November 16, 2023 from https://sdgs.un.org/goals

[53] GanttProject [Computer software]. (2021). *Free desktop project management software*. Retrieved September 10, 2023 from https://www.ganttproject.biz/

[54] Lucas, M. (2024). TFM. *GitHub*. Available at https://github.com/HummusDeArialux/TFM

[55] Lucas, M. (2024). *Skin Condition Prediction Tool with AI*. Available at https://skinai.bioedu.one/

[56] ISICS. (2016). BCN20000. Retrieved September 12, 2023 from https://api.isic-archive.com/collections/249/

[57] Buda, M., Maki, A., & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural networks, 106,* 249-259.

[58] Taherkhani, A., Cosma, G., & McGinnity, T. M. (2020). AdaBoost-CNN: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. *Neurocomputing, 404,* 351-366.

[59] Zafar, K., Gilani, S. O., Waris, A., Ahmed, A., Jamil, M., Khan, M. N., & Sohail Kashif, A. (2020). Skin lesion segmentation from dermoscopic images using convolutional neural network. *Sensors, 20(6)*, 1601.

[60] Cheng-Hong, Y., Jai-Hong, R., Huang, H. C., Li-Yeh, C., & Po-Yin, C. (2021). Deep Hybrid Convolutional Neural Network for Segmentation of Melanoma Skin Lesion. *Computational Intelligence and Neuroscience: CIN, 2021*.

[61] Thompson, D. (2022, July 03). The Pros And Cons Of Convolution Neural Networks. *iTechPost*. Retrieved October 16, 2023 from https://www.itechpost.com/articles/109452/20220307/the-pros-and-cons-of-convolution-neural-networks.htm

[62] Adamson, A. S., & Smith, A. (2018). Machine learning and health care disparities in dermatology. *JAMA dermatology*, *154*(11), 1247-1248.

[63] Chen, C. Y., Kahanamoku, S. S., Tripati, A., Alegado, R. A., Morris, V. R., Andrade, K., & Hosbey, J. (2022). Systemic racial disparities in funding rates at the National Science Foundation. *Elife*, *11*, e83071.

[64] Narla, S., Heath, C. R., Alexis, A., & Silverberg, J. I. (2023). Racial disparities in dermatology. *Archives of dermatological research*, *315*(5), 1215-1223.

[65] Acina. (2023). AI Dermatologist: Skin Scanner (Version 2.4) [Mobile App]. *Play Store*. Retrieved January 04, 2024 from https://play.google.com/store/apps/details?id=com.aidermatologist

[66] IDerma. (2023). ModelDerm - Enfermedades de la piel (Version 13.8.76) [Mobile App]. *Play Store*. Retrieved January 04, 2024 from https://play.google.com/store/apps/details?id=com.skincheck.cancerscreening.melanoma.mole.check

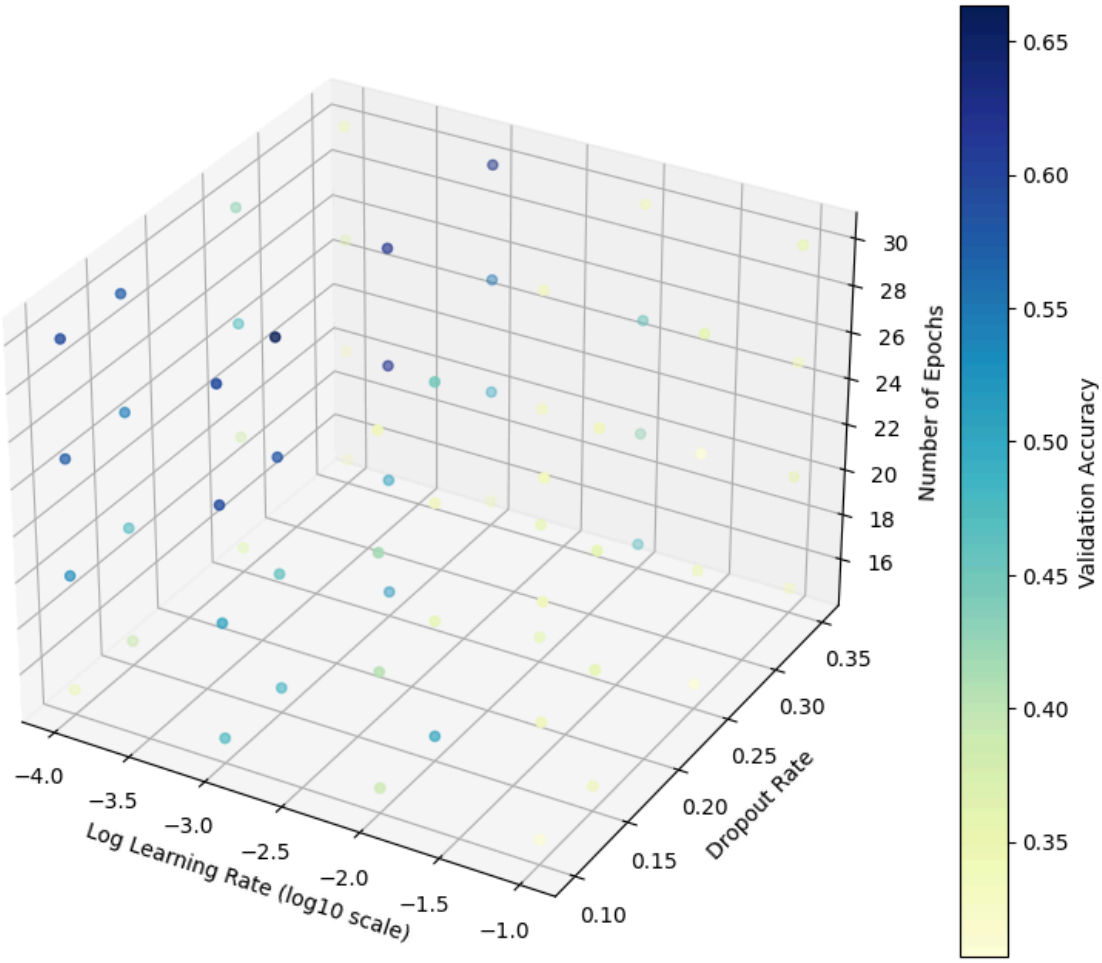
[67] Labirent Artificial Intelligence UG. (2022). SkinCheck: Dermatology App (Version 1.5.0) [Mobile App]. *Play Store*. Retrieved January 04, 2024 from https://play.google.com/store/apps/details?id=com.phonegap.whichderm

[68] Skinive B.V. (2023). Skinive - Dermatólogo de AI(Version 1.0.16) [Mobile App]. *Play Store*. Retrieved January 04, 2024 from https://play.google.com/store/apps/details?id=com.skinive.App

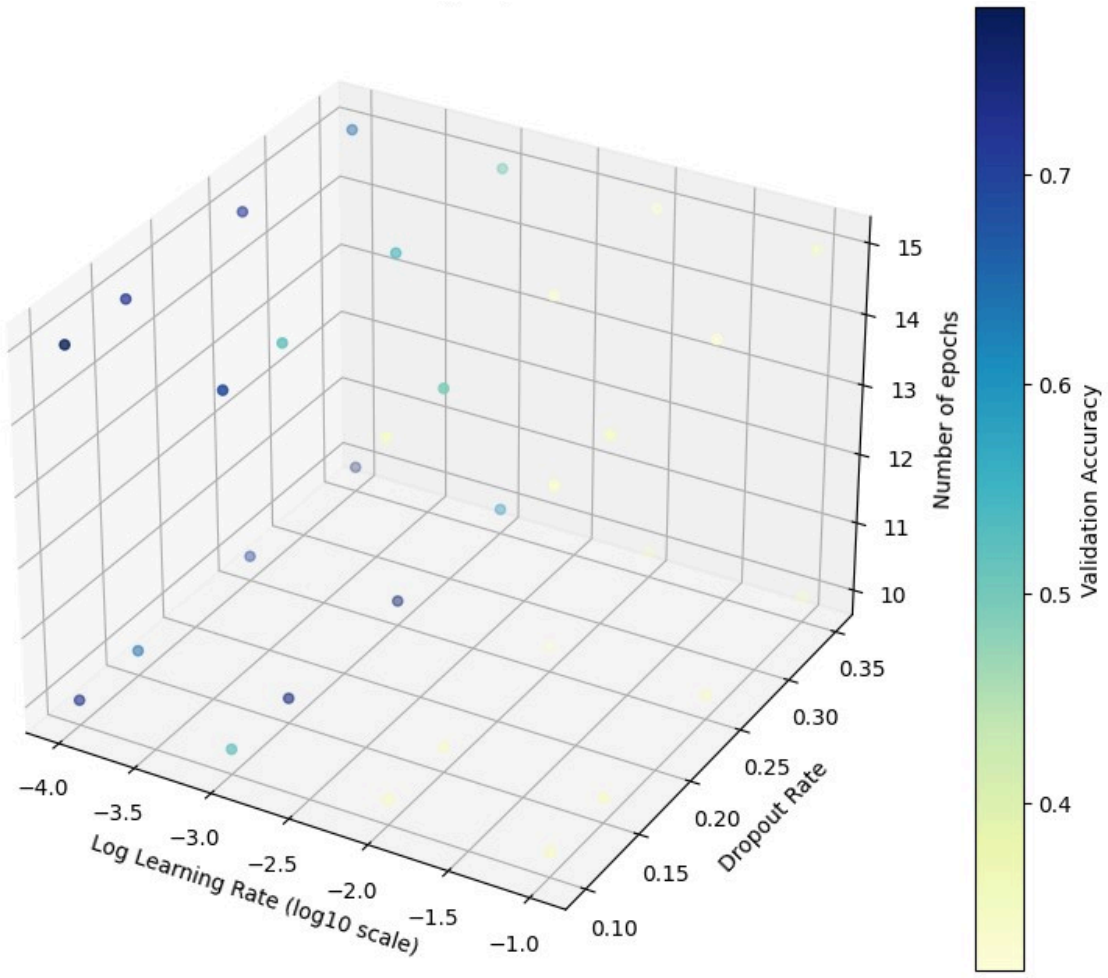
9.3 Supplementary information

Nombre	Fecha de inicio	Fecha de fin
Work plan development CAA1	9/27/23	10/15/23
Exploring and choosing a dataset	9/27/23	10/2/23
Determine the models to be tested	10/2/23	10/8/23
Writing CAA1	10/9/23	10/15/23
Work development CAA2	10/16/23	11/23/23
Explore necessary tools	10/16/23	10/19/23
Pre-processing data	10/19/23	10/23/23
Code the models	10/24/23	11/7/23
Study model accuracy	11/3/23	11/14/23
Optimize the best model	11/6/23	11/14/23
Optimize and document the code	11/11/23	11/13/23
Writing CAA2	11/20/23	11/23/23
Result's analysis	11/15/23	11/17/23
Work development CAA3	11/20/23	12/22/23
Learning to develop web-app in Django	11/20/23	11/26/23
Designing web map	11/28/23	11/29/23
Delevop app	11/30/23	12/20/23
Write educational content	12/15/23	12/17/23
Write CAA3	12/19/23	12/22/23
Final report and presentation CAA4	12/27/23	1/15/24
Optimize report	12/27/23	1/5/24
Optimize code presentation and documentation	1/6/24	1/9/24
Making presentation	1/7/24	1/15/24
Public defense CAA5	1/16/24	2/1/24

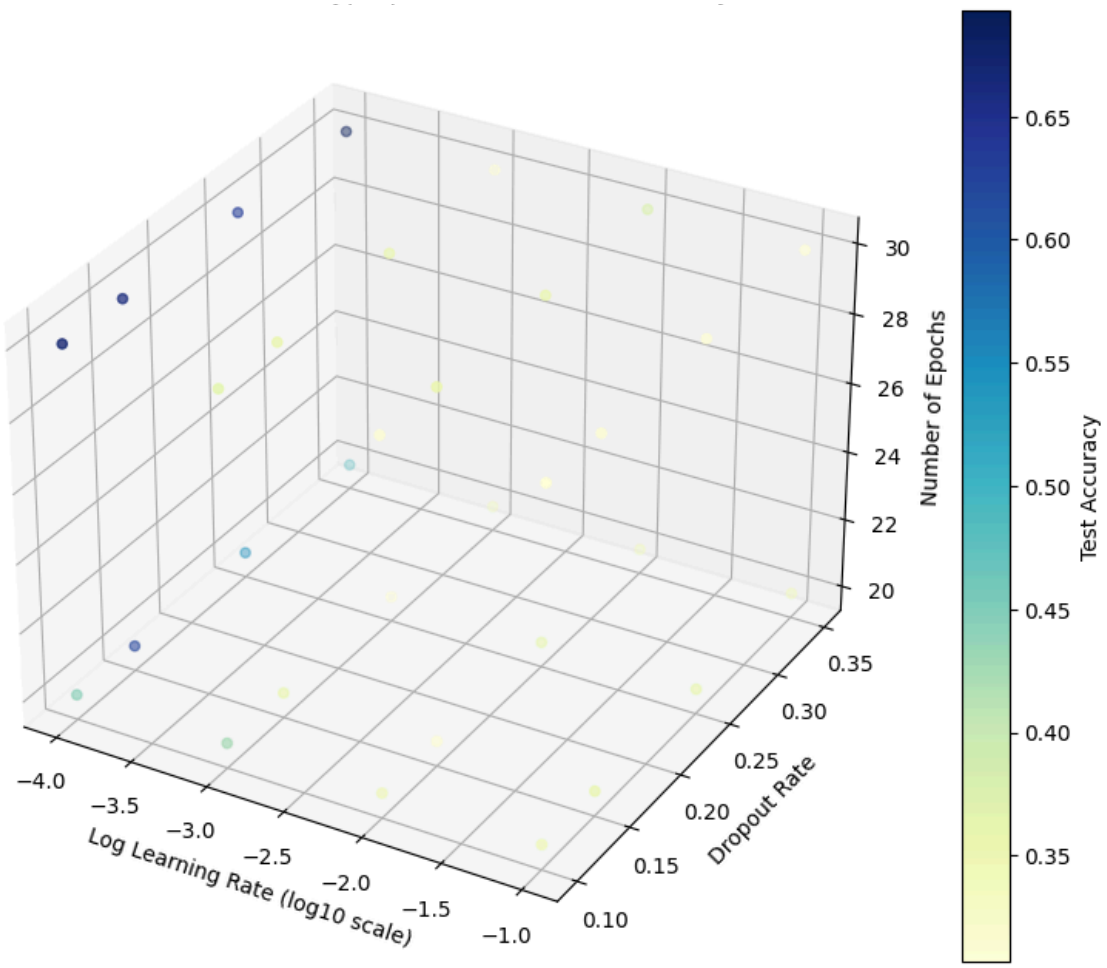
Supplementary figure 1: Chronological schedule. Chronological schedule presented in list format, detailing all tasks undertaken throughout the project. Each entry includes the task's initiation date and completion date, organized into four distinct phases for clarity.



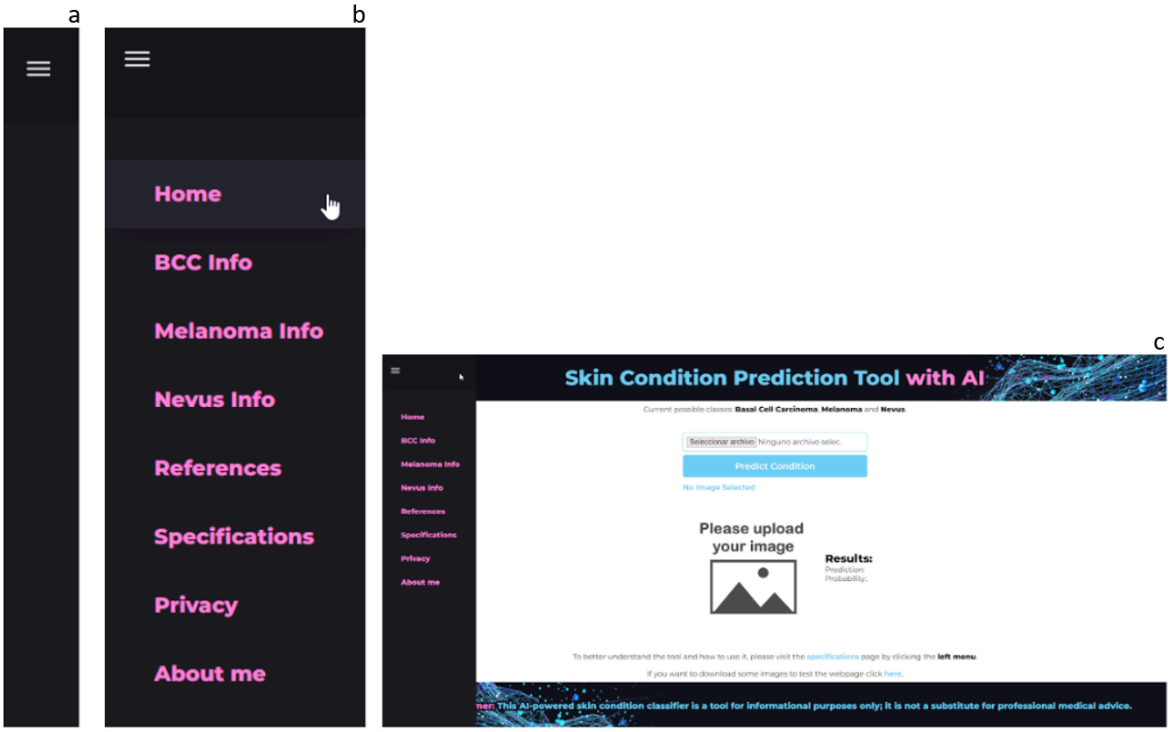
Supplementary figure 2: Hyperparameter exploration heatmap for custom CNN. 3D heatmap portrays the interplay of hyperparameters with model accuracy. The x-axis represents the learning rate (log10 scale), the y-axis denotes the dropout rate, and the z-axis signifies the number of training epochs. Each point's color indicates the model's accuracy on a test subset, providing a visual guide to discern optimal hyperparameter configurations for enhanced dermatological image classification.



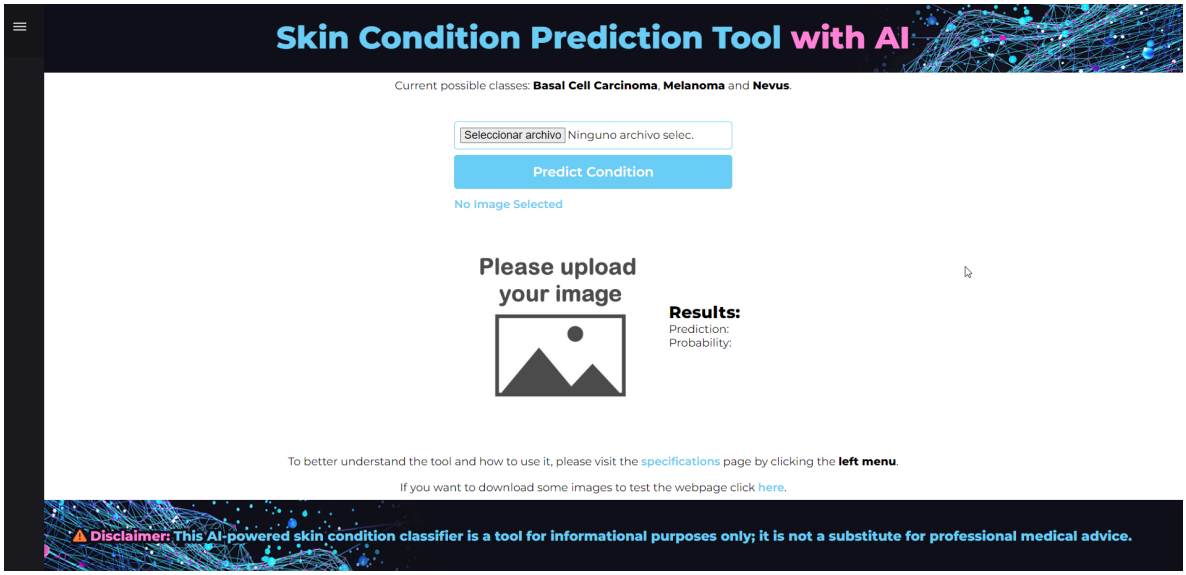
Supplementary figure 3: Hyperparameter exploration heatmap for DenseNet121. 3D heatmap portrays the interplay of hyperparameters with model accuracy. The x-axis represents the learning rate (log10 scale), the y-axis denotes the dropout rate, and the z-axis signifies the number of training epochs. Each point's color indicates the model's accuracy on a test subset, providing a visual guide to discern optimal hyperparameter configurations for enhanced dermatological image classification.



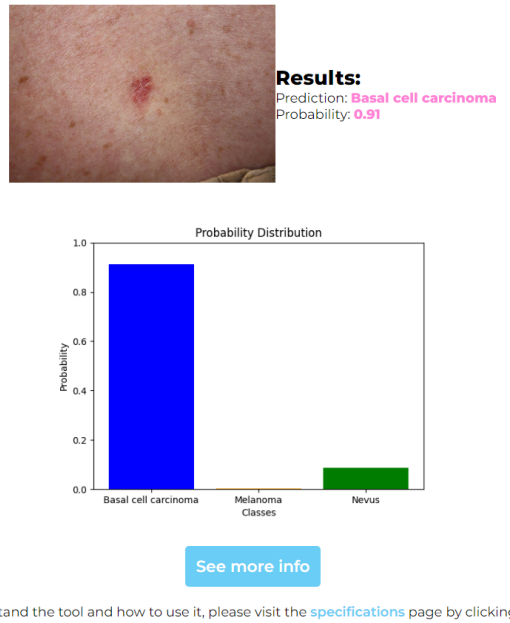
Supplementary figure 4: Hyperparameter exploration heatmap for ResNet50. 3D heatmap portrays the interplay of hyperparameters with model accuracy. The x-axis represents the learning rate (log10 scale), the y-axis denotes the dropout rate, and the z-axis signifies the number of training epochs. Each point's color indicates the model's accuracy on a test subset, providing a visual guide to discern optimal hyperparameter configurations for enhanced dermatological image classification.



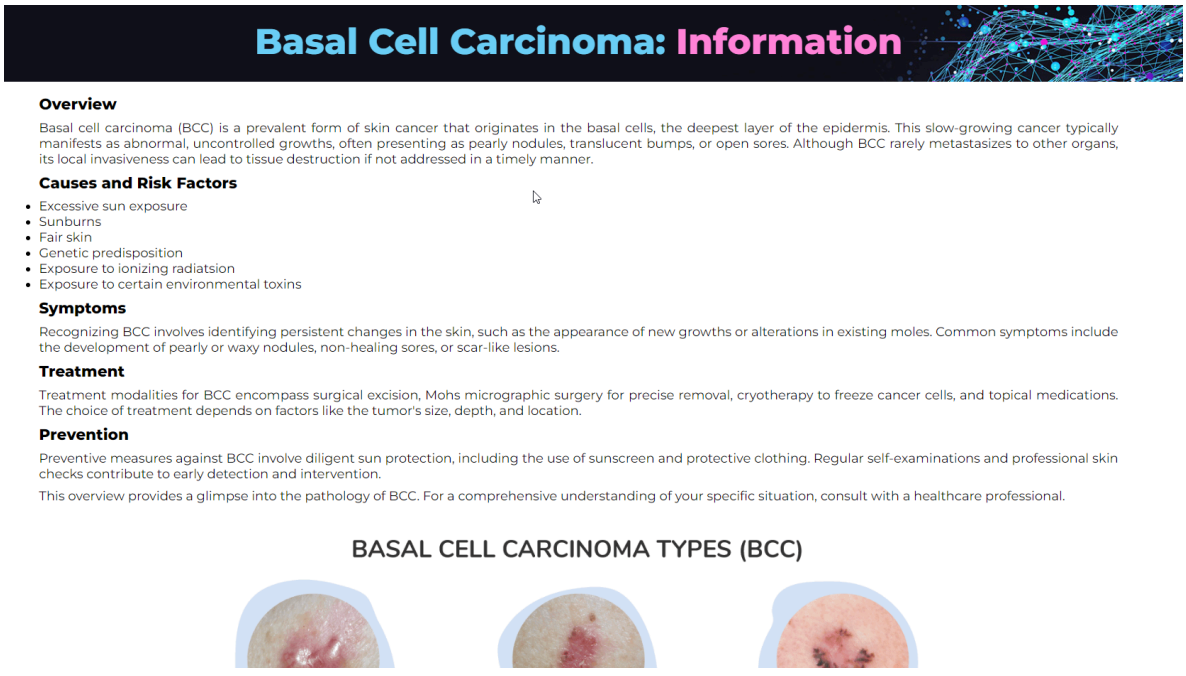
Supplementary figure 5: Navigation menu. The Navigation menu, positioned on the left side of the page, enhances user experience and accessibility. Initially folded (a), it expands (b) upon mouse hover, revealing the available sections for seamless navigation (c).



Supplementary figure 6: Home page. The Home page serves as the focal point for users, featuring image uploading and prediction functionalities.



Supplementary figure 7: Results page. The Results page dynamically loads upon completion of the prediction process, presenting users with the predicted class and a probability distribution for enhanced interpretability.



Supplementary figure 8: BCC info page. The BCC Information page offers detailed information about BCC, providing users with valuable insights and educational content related to this specific skin condition.

Melanoma: Information

Overview

Melanoma, a type of skin cancer originating in melanocytes, the pigment-producing cells, is known for its potential to metastasize, making early detection crucial. This aggressive cancer often presents as an irregularly shaped mole or dark lesion on the skin, with a tendency to evolve in size, color, or texture.

Causes and Risk Factors

- Excessive sun exposure
- Sunburns
- Fair skin
- Genetic predisposition
- Numerous moles
- Weakened immune function

Symptoms

Symptoms of melanoma include the development of new moles or changes in existing ones. Pay attention to irregular borders, variations in color, diameter exceeding 6 millimeters, and evolution over time.

Treatment

Treatment options for melanoma include surgical excision, lymph node biopsy, immunotherapy, targeted therapy, and chemotherapy, depending on the stage and extent of the disease. Early-stage melanomas are often curable with surgical removal.

Prevention

Preventing melanoma involves sun protection practices, including sunscreen use, protective clothing, and avoiding tanning beds. Regular self-examinations and professional skin checks aid in the early identification of suspicious lesions.

This overview provides a glimpse into the pathology of Melanoma. For a comprehensive understanding of your specific situation, consult with a healthcare professional.

Supplementary figure 9: Melanoma info page. The Melanoma Info page delivers comprehensive information on Melanoma, equipping users with educational resources and insights into this particular skin condition.

Nevus: Information

Overview

Nevi, commonly known as moles, are benign growths on the skin composed of pigment-producing cells called melanocytes. These clusters of cells often appear as small, round, or oval spots with a consistent color and regular borders. While most moles are harmless, changes in size, shape, or color may warrant medical attention.

Causes and Characteristics

The development of nevi is influenced by a combination of genetic factors and sun exposure. They can vary in color, ranging from tan to dark brown, and may be flat or raised. Nevi typically emerge in childhood and adolescence, with the number stabilizing in adulthood.

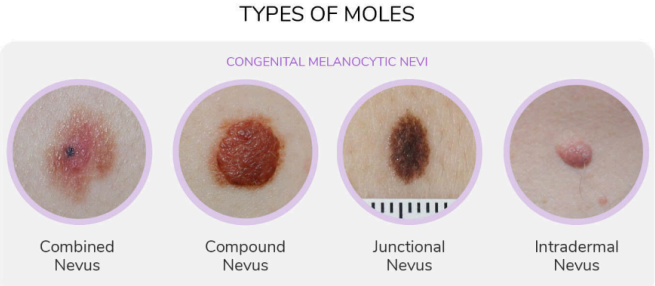
Treatment

Generally, nevi do not require treatment unless there are signs of atypical features or changes suggestive of malignancy. Removal may be considered for cosmetic reasons or if there is suspicion of skin cancer.

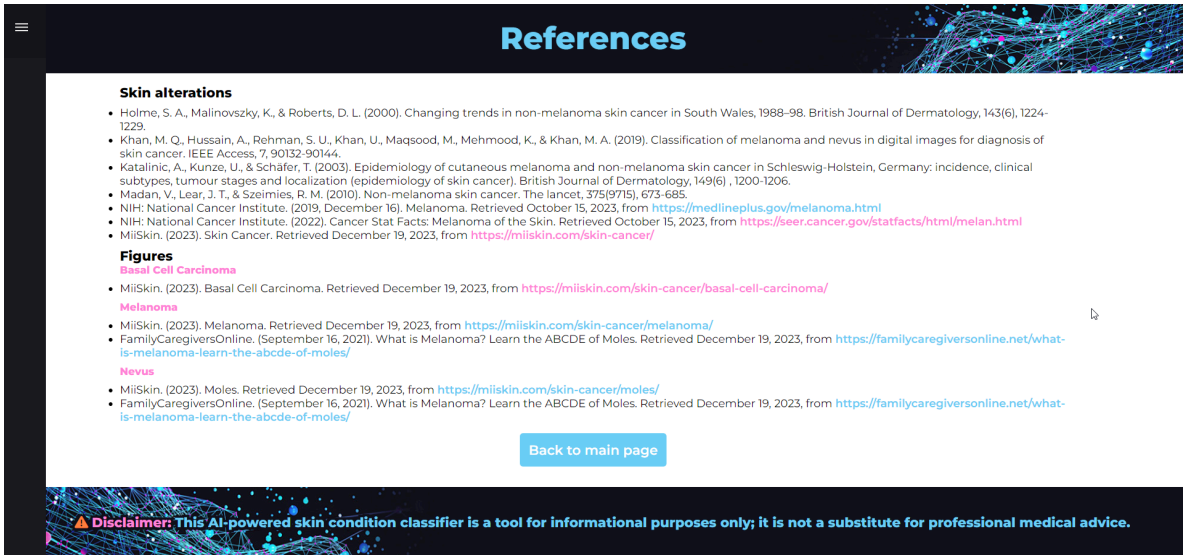
Prevention and monitoring

Preventive measures against BCC involve diligent sun protection, including the use of sunscreen and protective clothing. Regular self-examinations and professional skin checks contribute to early detection and intervention.

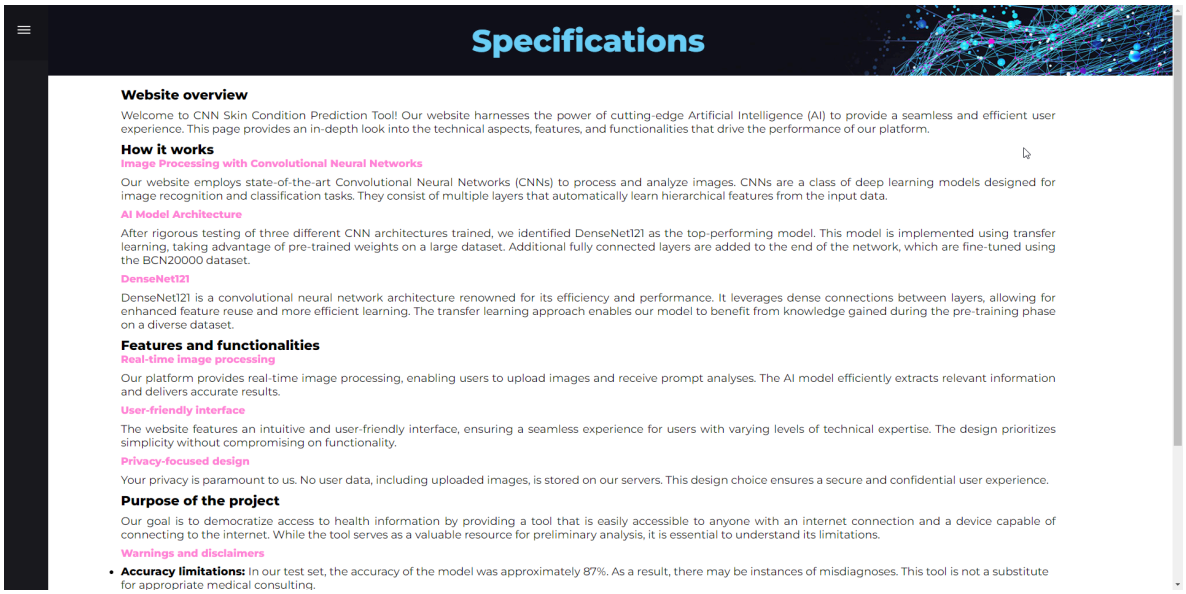
This overview provides essential information about nevi. If you observe any unusual changes in your moles or have concerns, seek guidance from a healthcare provider for a thorough evaluation.



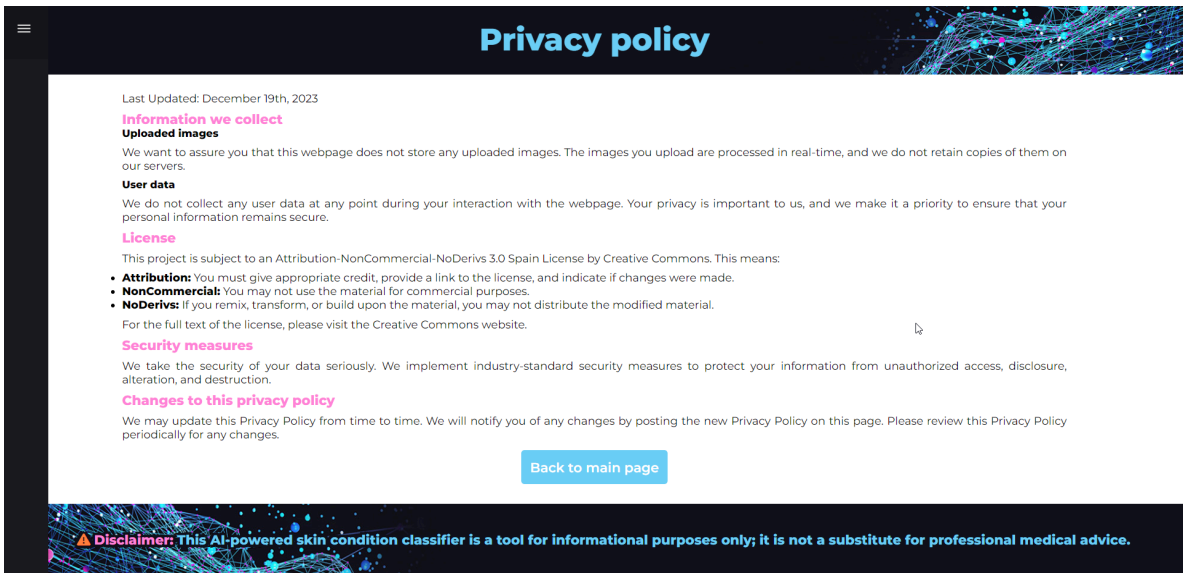
Supplementary figure 10: Nevus info page. The Nevus Info page provides users with detailed information about Nevus, fostering a deeper understanding of this skin condition through educational content.



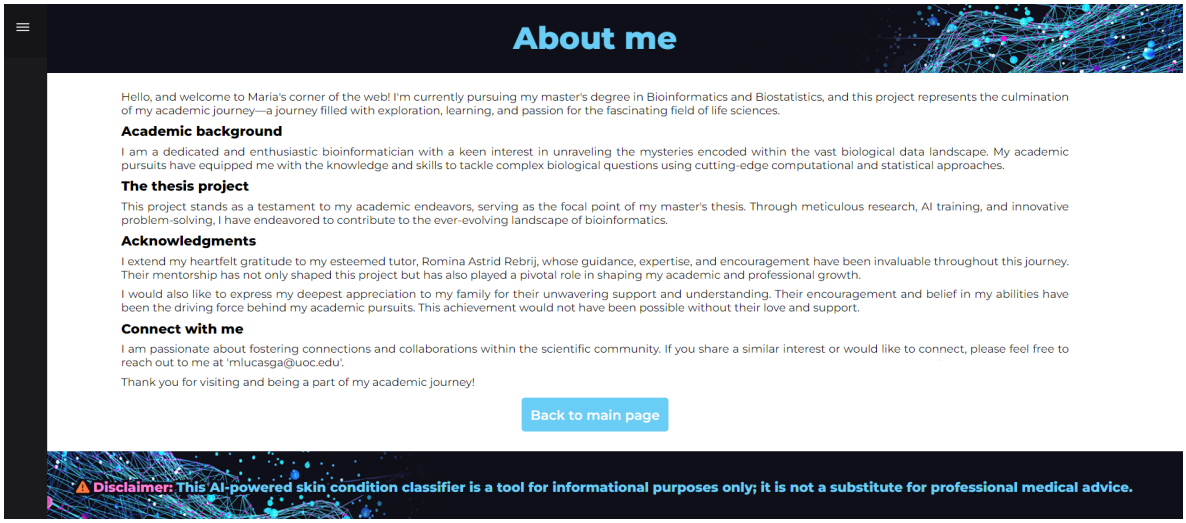
Supplementary figure 11: References page. The References page serves as a repository of cited sources, ensuring transparency and facilitating further exploration for users interested in the background and supporting literature.



Supplementary figure 12: Specifications page. The Specifications page offers an in depth look into the technical aspects and parameters of the skin condition classification model, catering to users seeking detailed insights into the system's architecture and functionality.



Supplementary figure 13: Privacy policy page. The privacy policy page outlines the app's commitment to user data protection, ensuring transparency in data handling and privacy measures.



Supplementary figure 14: About me page. The “About Me” page provides users with information about the creator of the web app, offering a personal touch and fostering a connection between the developer and users.

9.4 Code

The entire codebase is accessible for examination and replication in the following GitHub repository [54]: https://github.com/HummusDeArialux/TFM. Within this annex, a condensed iteration of the code is presented, omitting library imports and redundant segments. Additionally, the code structure has been adjusted to prioritize comprehension over direct execution.

Code 1: Exploratory data analysis script.

The provided code serves a multifaceted purpose within the context of the medical image dataset BCN20000 [56]. Initially, it loads metadata from a CSV file and conducts a preliminary examination by displaying column names and the total number of records. Non-informative columns are subsequently removed, optimizing the dataset for relevant analyses. The code then delves into data exploration, calculating the percentage distribution of diagnoses and generating a comprehensive diagnosis table. This table is both displayed and exported to a CSV file for further reference. Simultaneously, the script undertakes an examination of additional pertinent information, including gender distribution, anatomical site distribution, and the frequency of melanocytic classifications.

Moreover, the code encompasses functionality for the recursive extraction of image information from a specified directory. This includes attributes such as image height, width, and channels, which are aggregated into a dataframe for subsequent analysis. The resulting image statistics are systematically presented, shedding light on the distribution of image dimensions and channels within the dataset.

```
# Load file
filepath = 'E:\TFM\code\reduced_metadata.csv'
dataset = pd.read_csv(filepath)

# Examine file
print(f'Variables: {dataset.columns}')
print(f'Number of images: {len(dataset)}')

# Delete non-informative columns
col_drop = ['attribution', 'copyright_license', 'diagnosis_confirm_type',
'image_type', 'lesion_id']
data = dataset.drop(columns = col_drop)
print(f'Variables: {data.columns}')

# Data exploration
diagnosis_counts = data['diagnosis'].value_counts()

# Creation of diagnosis table
diagnosis_table = pd.DataFrame({'Diagnosis': diagnosis_counts.index, 'Sample
Size': diagnosis_counts.values})
diagnosis_table['Percentage'] = (diagnosis_table['Sample Size'] /
diagnosis_table['Sample Size'].sum()) * 100
diagnosis_table['Percentage'] = diagnosis_table['Percentage'].round(2)

# Display the resulting table
print(diagnosis_table)

# Export the DataFrame to a CSV file
diagnosis_table.to_csv('diagnosis_table.csv', index=False)

# Other info
print(data.sex.value_counts())
print(data.anatom_site_general.value_counts())
print(data.melanocytic.value_counts())

# Image directory
image_directory = 'E:\TFM\BCN20000\Imagenes_256_reduced'

# Function to gather image information recursively
def gather_image_info(directory):
    """
Recursively traverses the specified directory to gather information about
images with the '.JPG' extension. Retrieves the height, width, and channels
of each image using OpenCV.

Parameters:
- directory (str): The path to the directory containing images.

Returns:
- list: A list of dictionaries, where each dictionary contains image
information
        including height, width, and channels.
    """
```

```
    image_info = []
    for root, _, files in os.walk(directory):
        for filename in files:
            if filename.endswith('.JPG'):  # Images are in JPG format
                file_path = os.path.join(root, filename)
                image = cv2.imread(file_path)
                if image is not None:
                    height, width, channels = image.shape
                    image_info.append({
                        'Height': height,
                        'Width': width,
                        'Channels': channels
                    })
    return image_info

# Call the function to gather image information
image_info = gather_image_info(image_directory)

# Create a DataFrame to store the gathered information
image_df = pd.DataFrame(image_info)

# Print necessary info
print(image_df.Height.value_counts())
print(image_df.Width.value_counts())
print(image_df.Channels.value_counts())
```

Code 2: Data loading and preprocessing script. Minor modifications may have been introduced depending on the data to load.

The presented code follows a systematic procedure for handling the dataset. Initially, it loads metadata from a specified CSV file, removing non-informative columns to streamline the dataset. Subsequently, the code establishes a list, “image_info”, to store pertinent image information. The image directory is then defined, and the script iterates through the metadata to gather crucial image details. For each entry, the code forms the image file name using the “isic_id”, constructs the image path, and extracts the corresponding diagnosis label. Utilizing OpenCV, the code reads and preprocesses each image, normalizing pixel values to the [0, 1] range by dividing each pixel value by 255. The processed image and its associated label are appended to the “image_info” list. This systematic approach ensures the extraction and organization of normalized image data along with their corresponding diagnostic labels.

```
# Load file
filepath = 'E:/TFM/code/reduced_metadata.csv'
metadata = pd.read_csv(filepath)  # Use the specified file path

# Delete non-informative columns
col_drop = ['attribution', 'copyright_license', 'diagnosis_confirm_type',
'image_type', 'lesion_id']
data = metadata.drop(columns=col_drop)  # Elimination of the specified columns

# Define a list to store image information
image_info = []

# Image directory
image_directory = 'E:/TFM/BCN20000/Imagenes_256_reduced'

# Iterate through the metadata and gather image information
for index, row in metadata.iterrows():
    # Form the image file name using the isic_id
    image_filename = row['isic_id'] + '.jpg'
    image_path = os.path.join(image_directory, image_filename)
    label = row['diagnosis']

    # Read and preprocess the image
    image = cv2.imread(image_path)
    image = image / 255.0  # Normalize the pixel values to the range [0, 1]
```

```
    # Store the image and label information
    image_info.append((image, label))
```

Code 3: Data reduction script for melanoma, basal cell carcinoma, and nevus classes.

This code strategically trims the dataset by selecting specific labels (“melanoma”, “basal cell carcinoma”, and “nevus”). It limits the number of “nevus” images to a defined maximum while saving a randomized subset. The retained images are stored in an output directory. The resulting dataset, with reduced labels and controlled “nevus” images, is stored as ”reduced_metadata.csv”.

```
# Set seed
random_seed = 2023
random.seed(random_seed)

# Load file
filepath = 'E:/TFM/code/metadata.csv'
metadata = pd.read_csv(filepath)

# Image directory
image_directory = 'E:/TFM/BCN20000/Imagenes_256'

# Define the labels you want to keep
labels_to_keep = ['melanoma', 'basal cell carcinoma', 'nevus']

# Define the output directory for all labels
output_directory = 'E:/TFM/BCN20000/Imagenes_256_reduced'

# Create the output directory if it doesn't exist
os.makedirs(output_directory, exist_ok=True)

# Define the maximum number of "nevus" images to keep
max_nevus_images = 2850

# Shuffle the indices of "nevus" images for random selection
nevus_indices = list(metadata[metadata['diagnosis'] == 'nevus'].index)
random.shuffle(nevus_indices)

# Iterate through the metadata and gather image information
nevus_count = 0  # Keep track of the number of "nevus" images saved
save_nevus = True  # Flag to control saving of nevus images

# Create a list to store the indices of rows to keep in the reduced metadata
rows_to_keep = []

for index, row in metadata.iterrows():
    label = row['diagnosis']
    if label not in labels_to_keep:
        continue  # Skip images with labels you don't want to keep

    if label == 'nevus' and save_nevus == False:
        continue # Skip nevus images if max is reached

    # Form the image file name using the isic_id
    image_filename = row['isic_id'] + '.jpg'
    image_path = os.path.join(image_directory, image_filename)

    # Read and preprocess the image
    image = cv2.imread(image_path)

    # Form the output file path
    output_file_path = os.path.join(output_directory, image_filename)

    # Save the image to the output directory
    cv2.imwrite(output_file_path, image)

    if label == 'nevus':
        nevus_count += 1
```

```
        if nevus_count >= max_nevus_images:
            save_nevus = False  # Set save_nevus to False

    # Store the index of this row to keep in the reduced metadata
    rows_to_keep.append(index)

# Create the reduced metadata with only the rows you want to keep
reduced_metadata = metadata.loc[rows_to_keep]

# Save the reduced metadata to 'reducedmetadata.csv'
reduced_metadata.to_csv('E:/TFM/code/reduced_metadata.csv', index=False)
```

Code 4: One-hot encoding script for labels and splitting into training, validation and test sets.

This code facilitates the preparation of a machine learning dataset by encoding labels into one-hot format and dividing the data into training, validation, and test sets. Initially, it extracts images and labels from a previously gathered “image_info” list. The labels, representing different medical conditions (“nevus”, “melanoma”, and “bcc”), are encoded using the LabelEncoder to convert them into numerical format. Subsequently, one-hot encoding is applied to the encoded labels, ensuring compatibility with machine learning models.

To facilitate model training and evaluation, the dataset is partitioned into training (80%), validation (10%), and test (10%) subsets using the “train_test_split” function. This division allows for robust model training on the training set, fine-tuning and hyperparameter tuning using the validation set, and a final unbiased evaluation on the test set. Overall, this code prepares the dataset for effective utilization in a machine learning pipeline, ensuring a balanced representation of labeled classes and facilitating model generalization.

```
# Extract images and labels from image_info
images = np.array([item[0] for item in image_info])  # Convert image data to
NumPy arrays
labels = [item[1] for item in image_info]

# Unique number of diagnosis
num_classes = len(Counter(labels))

# Encode labels using LabelEncoder
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(labels)

# Convert encoded labels to one-hot encoding
y_one_hot = to_categorical(y_encoded, num_classes=num_classes)

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(images, y_one_hot,
test_size=0.2, random_state=2023)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=2023)
```

Code 5: Custom CNN model implementation.

This code defines and trains a convolutional neural network (CNN) for a multi-class image classification task. It starts by setting hyperparameters, including the learning rate, dropout rate, and the number of training epochs. The CNN architecture is then constructed using sequential layers, featuring convolutional layers with max-pooling, batch normalization, and dropout for effective feature extraction and model generalization. The fully connected layers follow, incorporating dropout to prevent overfitting. The output layer uses the softmax activation function for multi-class classification.

The model is compiled using the Adam optimizer, categorical cross entropy loss (suitable for multi-class problems), and accuracy as the evaluation metric. Early stopping is implemented to prevent overfitting, and the model is trained on the training set with validation on a separate validation set. The training history is stored for later analysis, and the trained model is saved to a specified path. This code encapsulates the essential steps in designing, compiling, training, and saving a custom CNN for image classification.

```
# Define hyperparameters
learning_rate = 0.001 # Learning rate
dropout_rate = 0.15 # Dropout rate
num_epochs = 30 # Number of epochs

# Create the CNN model
model = models.Sequential()

# Convolutional Layers with MaxPooling and BatchNormalization
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256,
3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(dropout_rate))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.SpatialDropout2D(dropout_rate))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.SpatialDropout2D(dropout_rate))

model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())

# Flatten the output for the fully connected layers
model.add(layers.Flatten())

# Fully Connected Layers with Dropout
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(dropout_rate))

model.add(layers.Dense(512, activation='relu'))

# Output Layer
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=learning_rate),
              loss='categorical_crossentropy',  # Because it's a multi-class
classification problem
              metrics=['accuracy']) # Evaluate model based on accuracy

# Define early stopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Train the model
```

```
history = model.fit(
    X_train, # Training images
    y_train, # # One-hot encoded training labels
    epochs=num_epochs, # Number of training epochs
    validation_data=(X_val, y_val), # Validation data with one-hot encoded
labels
    callbacks=[early_stopping]  # Add the early stopping callback
)

# Save the model to a specific path
model.save('E:/TFM/Results with code/Trained_models/CNN.h5')
```

Code 6: DenseNet121 model implementation.

This code implements the definition and training of a transfer learning model using DenseNet121 architecture pre-trained on ImageNet. The pre-trained model is loaded, and layers up to "conv4_block" are frozen to retain pre-trained features. Hyperparameters such as dropout rate, number of epochs, batch size, and learning rate are specified. Custom layers are added on top of the pre-trained model, including global average pooling for dimensionality reduction, batch normalization for stability, and dense layers with ReLU activation and dropout for regularization. The final model is compiled with the Adam optimizer and categorical cross entropy loss, focusing on accuracy as the evaluation metric.

A learning rate scheduler is incorporated, adjusting the learning rate during training to enhance convergence. The model is then fitted on the training set, with validation performed on a separate validation set. The training history is stored for analysis, and the trained model is saved to a specified path. This code streamlines the process of utilizing transfer learning with DenseNet121 for image classification while allowing customization of the model architecture for specific tasks.

```
# Load the pre-trained DenseNet121 model with weights from ImageNet
base_model = DenseNet121(weights='imagenet', include_top=False)

# Freeze all layers up to block4
for layer in base_model.layers:
    if 'conv4_block' in layer.name:
        break
    layer.trainable = False

# Define hyperparameters
dropout_rate = 0.15 # Dropout rate
num_epochs = 10 # Number of epochs
batch_size = 128 # Batch size
learning_rate = 0.001 # Learning rate

# Add custom layers on top of the pre-trained model
x = base_model.output # Base model
x = GlobalAveragePooling2D()(x) # Global Average Pooling to reduce spatial
dimensions
x = BatchNormalization()(x) # Batch Normalization for improved training
stability
x = Dense(256, activation='relu')(x) # Dense layer with ReLU activation
x = Dropout(dropout_rate)(x) # Dropout for regularization
predictions = Dense(num_classes, activation='softmax')(x) # Final Dense layer
for classification

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model with Adam optimizer and categorical crossentropy loss
model.compile(optimizer=Adam(learning_rate=learning_rate),
              loss='categorical_crossentropy',  # Because it's a multi-class
classification problem
```

```
              metrics=['accuracy']) # Evaluate model based on accuracy

# Define a learning rate scheduler
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
min_lr=1e-6)

# Fit the model
history = model.fit(
    X_train, # Training images
    y_train, # One-hot encoded training labels
    validation_data=(X_val, y_val), # Validation data with one-hot encoded
labels
    epochs=num_epochs, # Number of training epochs
    batch_size=batch_size, # Batch size
    callbacks=[lr_scheduler] # Add the learning rate scheduler
)

# Save the model to a specific path
model.save('E:/TFM/Results with code/Trained_models/DenseNet121.h5')
```

Code 7: ResNet50 model implementation.

This code defines and trains a transfer learning model using the ResNet50 architecture pre-trained on ImageNet. It begins by loading the pre-trained model, specifying the input tensor shape to accommodate the dataset's dimensions. Layers up to 'conv4_block' are frozen to retain pre-trained features. Hyperparameters, including the number of epochs, batch size, learning rate, and dropout rate, are defined. Custom layers are added on top of the pre-trained model, incorporating global average pooling for spatial dimension reduction, batch normalization for stability, and dense layers with ReLU activation and dropout for regularization.

The final model is compiled with the Adam optimizer and categorical cross entropy loss, focusing on accuracy as the evaluation metric. A learning rate scheduler is implemented to adapt the learning rate during training for enhanced convergence. The model is fitted on the training set with validation performed on a separate set, and the training history is stored. Finally, the trained model is saved to a specified path. This code streamlines the process of utilizing transfer learning with ResNet50 for image classification while allowing customization of the model architecture for specific tasks.

```
# Load the pre-trained ResNet50 model with weights from ImageNet
base_model = ResNet50(weights='imagenet', include_top=False,
input_tensor=Input(shape=(256, 256, 3)))

# Freeze layers up to block4
for layer in base_model.layers:
    if 'conv4_block' in layer.name:
        break
    layer.trainable = False

# Define hyperparameters
num_epochs = 30 # Number of epochs
batch_size = 128 # Batch size
learning_rate = 0.001 # Learning rate
dropout_rate = 0.25 # Dropout rate

# Add custom layers on top of the pre-trained model
x = base_model.output # Base model
x = GlobalAveragePooling2D()(x) # Global Average Pooling to reduce spatial
dimensions
x = BatchNormalization()(x) # Batch Normalization for improved training
stability
x = Dense(256, activation='relu')(x) # Dense layer with ReLU activation
x = Dropout(dropout_rate)(x) # Dropout for regularization
predictions = Dense(num_classes, activation='softmax')(x) # Final Dense layer
```

```
for classification

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model with Adam optimizer and categorical crossentropy loss
model.compile(optimizer=Adam(learning_rate=learning_rate),
              loss='categorical_crossentropy',  # Because it's a multi-class
classification problem
              metrics=['accuracy']) # Evaluate model based on accuracy

# Define a learning rate scheduler
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
min_lr=1e-6)

# Fit the model
history = model.fit(
    X_train, # Training images
    y_train, # One-hot encoded training labels
    validation_data=(X_val, y_val), # Validation data with one-hot encoded
labels
    epochs=num_epochs, # Number of training epochs
    batch_size=batch_size, # Batch size
    callbacks=[lr_scheduler] # Add the learning rate scheduler
)

# Save the model to a specific path
model.save('E:/TFM/Results with code/Trained_models/ResNet50.h5')
```

Code 8: Test subset reduction script.

This code refines the already reduced dataset to a maximum of 1002 images, evenly distributed across the three specified classes (“nevus”, “melanoma”, and “bcc”). It initiates by setting a random seed for reproducibility and loading the previously reduced metadata. The dataset is then shuffled randomly to ensure an unbiased selection process. Images are stored in an output directory, and the maximum number of images per class is defined. A dictionary, “class_counts”, keeps track of the count for each class, and a list, “rows_to_keep”, is created to store the indices of selected rows.

The code iterates through the metadata, checking and skipping images if the maximum count for a class is reached. It saves each selected image to the output directory, incrementing the class count accordingly. The process is controlled to stop once the required number of images for all classes is attained. The resulting refined metadata, containing the specified number of images for each class, is saved as “test_metadata.csv”. This code facilitates the creation of a balanced and constrained dataset for testing purposes, ensuring a controlled representation of the targeted classes.

```
# Set seed
random_seed = 2023
random.seed(random_seed)

# Load file
filepath = 'E:/TFM/code/reduced_metadata.csv'
metadata = pd.read_csv(filepath)

# Shuffle the entire dataset randomly
metadata = metadata.sample(frac=1,
random_state=random_seed).reset_index(drop=True)

# Image directory
image_directory = 'E:/TFM/BCN20000/Imagenes_256_reduced'

# Define the labels you want to keep
labels_to_keep = ['melanoma', 'basal cell carcinoma', 'nevus']

# Define the output directory for all labels
```

```
output_directory = 'E:/TFM/BCN20000/Imagenes_test'

# Create the output directory if it doesn't exist
os.makedirs(output_directory, exist_ok=True)

# Define the maximum number of images per class
total_images_to_keep = 1000

# Create a dictionary to keep track of the count for each class
class_counts = {label: 0 for label in labels_to_keep}

# Create a list to store the indices of rows to keep in the reduced metadata
rows_to_keep = []

# Iterate through the metadata and gather image information
for _, row in metadata.iterrows():
    label = row['diagnosis']

    # If the maximum number of images for a class is reached, skip the image
    if class_counts[label] >= total_images_to_keep / len(labels_to_keep):
        continue

    # Form the image file name using the isic_id
    image_filename = row['isic_id'] + '.jpg'
    image_path = os.path.join(image_directory, image_filename)

    # Read and preprocess the image
    image = cv2.imread(image_path)

    # Form the output file path
    output_file_path = os.path.join(output_directory, image_filename)

    # Save the image to the output directory
    cv2.imwrite(output_file_path, image)

    # Increment the count for the current class
    class_counts[label] += 1

    # Store the index of this row to keep in the reduced metadata
    rows_to_keep.append(_)

    # Check if the required number of images for each class is reached
    if all(count >= total_images_to_keep for count in class_counts.values()):
        break  # Stop once the required number of images for all classes is
reached

# Create the reduced metadata with only the rows you want to keep
test_metadata = metadata.loc[rows_to_keep]

# Save the reduced metadata to 'reducedmetadata.csv'
test_metadata.to_csv('E:/TFM/code/test_metadata.csv', index=False)
```

Code 9: GridSearch implementation script. Minor adjustments are introduced depending on the model (CNN, DenseNet121 and ResNet50).

This code performs a grid search to identify optimal hyperparameters for a neural network model on a smaller subset of the dataset. It explores different combinations of learning rates, dropout rates, and the number of epochs to find the configuration that maximizes validation and test accuracy. The grid search results are stored in dictionaries, and the best hyperparameters for both validation and test accuracy are identified. Visualizations in 3D plots illustrate the impact of hyperparameters on accuracy, providing insights into the model's sensitivity to these parameters. This systematic approach aids in selecting hyperparameters that enhance model performance on unseen data, contributing to the overall effectiveness of the neural network for the given classification task.

```
# GRID SEARCH
# Define hyperparameters
```

```
learning_rates = [0.1, 0.01, 0.001, 0.0001]
dropout_rates = [0.10, 0.15, 0.25, 0.35]
num_epochs = [15, 20, 25, 30]

results = {}  # Dictionary to store results for different hyperparameters on
val accuracy
results2 = {}  # Dictionary to store results for different hyperparameters on
test accuracy

# Function to create and train the model
def train_model(X_train, y_train, X_val, y_val, X_test, y_test, learning_rate,
dropout_rate, epochs)

# ... model ...

    test_loss, test_accuracy = model.evaluate(X_test, y_test)

    return history.history['val_accuracy'][-1], test_accuracy

# Perform grid search
for learning_rate in learning_rates:
    for dropout_rate in dropout_rates:
        for epochs in num_epochs:
            val_accuracy, test_accuracy = train_model(X_train, y_train, X_val,
y_val, X_test, y_test,
                                                      learning_rate,
dropout_rate, epochs)
            results[(learning_rate, dropout_rate, epochs)] = val_accuracy
            results2[(learning_rate, dropout_rate, epochs)] = test_accuracy

# Find the best hyperparameters for validation accuracy
best_hyperparameters = max(results, key=results.get)
print(f"Best hyperparameters for validation accuracy: {best_hyperparameters} -
Validation Accuracy: {results[best_hyperparameters]}")

# Find the best hyperparameters for test accuracy
best_hyperparameters_test = max(results2, key=results2.get)
print(f"Best hyperparameters for test accuracy: {best_hyperparameters_test} -
Test Accuracy: {results2[best_hyperparameters_test]}")

# Plot the grid search results for validation accuracy
hyperparameters_val = list(results.keys())
accuracies_val = list(results.values())
learning_rates_val, dropout_rates_val, epochs_val = zip(*hyperparameters_val)

log_learning_rates_val = np.log10(learning_rates_val)

fig_val = plt.figure(figsize=(10, 8))
ax_val = fig_val.add_subplot(111, projection='3d')

sc_val = ax_val.scatter(log_learning_rates_val, dropout_rates_val, epochs_val,
c=accuracies_val, cmap='YlGnBu', marker='o')

ax_val.set_xlabel('Log Learning Rate (log10 scale)')
ax_val.set_ylabel('Dropout Rate')
ax_val.set_zlabel('Number of Epochs')
cbar_val = fig_val.colorbar(sc_val)
cbar_val.set_label('Validation Accuracy')

plt.title("Grid Search for Hyperparameters (Validation Accuracy)")
plt.show()

# Plot the grid search results for test accuracy
hyperparameters_test = list(results2.keys())
accuracies_test = list(results2.values())
learning_rates_test, dropout_rates_test, epochs_test =
zip(*hyperparameters_test)
```

```
log_learning_rates_test = np.log10(learning_rates_test)

fig_test = plt.figure(figsize=(10, 8))
ax_test = fig_test.add_subplot(111, projection='3d')

sc_test = ax_test.scatter(log_learning_rates_test, dropout_rates_test,
epochs_test, c=accuracies_test, cmap='YlGnBu', marker='o')

ax_test.set_xlabel('Log Learning Rate (log10 scale)')
ax_test.set_ylabel('Dropout Rate')
ax_test.set_zlabel('Number of Epochs')
cbar_test = fig_test.colorbar(sc_test)
cbar_test.set_label('Test Accuracy')

plt.title("Grid Search for Hyperparameters (Test Accuracy)")
plt.show()
```

Code 10: Monitoring training process.

This code generates two plots to monitor the training process of a neural network model. The first plot illustrates the training and validation loss over training steps, providing insights into the convergence and generalization of the model. The second plot displays the training and validation accuracy, offering a visual assessment of the model's performance on both the training and validation datasets throughout the training process. These plots serve as valuable tools for practitioners to assess the model's learning dynamics, identify potential overfitting or underfitting, and make informed decisions regarding model adjustments or hyperparameter tuning.

```
# Plot training and validation loss
plt.figure()
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.ylim([0, 2])
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])

# Plot training and validation accuracy
plt.figure()
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")
plt.ylim([0, 1])
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
```

Code 11: Model performance evaluation script.

This code assesses the performance of a trained neural network model using unseen test data. It starts by evaluating the model's test accuracy. Subsequently, predictions are generated on the test set, and one-hot encoded predictions are converted back to class labels for further analysis. The code calculates and prints various evaluation metrics, including accuracy, precision, sensitivity (recall), F1-score, and cohen's kappa. Additionally, a detailed classification report is generated to provide insights into the model's performance in each class. The confusion matrix is computed and visualized using a heatmap, offering a comprehensive view of the model's predictions. These metrics and visualizations assist practitioners in understanding the model's strengths and weaknesses, facilitating informed decisions for model refinement or further optimization.

```
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_accuracy}')

# Predictions on test set
y_pred = model.predict(X_test)
```

```
# Convert one-hot encoded predictions back to class labels
y_pred_labels = label_encoder.inverse_transform(np.argmax(y_pred, axis=1))
y_true_labels = label_encoder.inverse_transform(np.argmax(y_test, axis=1))

# Calculate and print various evaluation metrics
# Calculate accuracy
accuracy = accuracy_score(y_true_labels, y_pred_labels)
print(f'Test accuracy: {accuracy:.2f}')

# Calculate precision
precision = precision_score(y_true_labels, y_pred_labels, average='weighted')
print(f'Precision: {precision:.2f}')

# Calculate sensitivity (recall)
sensitivity = recall_score(y_true_labels, y_pred_labels, average='weighted')
print(f'Sensitivity (Recall): {sensitivity:.2f}')

# Calculate F1-Score
f1 = f1_score(y_true_labels, y_pred_labels, average='weighted')
print(f'F1-Score: {f1:.2f}')

# Calculate Cohen's Kappa
kappa = cohen_kappa_score(y_true_labels, y_pred_labels)
print(f"Cohen's Kappa: {kappa:.2f}")

# Generate a classification report
report = classification_report(y_true_labels, y_pred_labels)
print("Classification Report:")
print(report)

# Calculate the confusion matrix
confusion = confusion_matrix(y_true_labels, y_pred_labels)
print("Confusion Matrix:")
print(confusion)

# Get class labels
classes = unique_labels(y_true_labels, y_pred_labels)

# Plot a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", xticklabels=classes,
yticklabels=classes)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Code 12: Predicting script.

This combined script handles a user-submitted image in a web application. Upon receiving an image through the request, it is read and processed using OpenCV and Pillow libraries. The image is then resized to 256x256 pixels, normalized by dividing pixel values by 255, and prepared for model prediction. The model, defined and loaded from a saved file (“DenseNet121.h5”), is used to predict the class probabilities for the processed image. The prediction results, including the predicted class, alteration label, and probability values, are stored as variables.

A bar chart is generated using Matplotlib to visualize the probability distribution across different classes. The resulting plot is encoded as base64 to embed it in the web application. The script utilizes a separate file (“predictor.py”) where the model is imported and the “predict_image” function is defined. This function takes the processed image as input, predicts the class probabilities using the loaded model, and returns relevant information about the prediction, such as the predicted class label, alteration, and probability. The entire process allows for seamless integration of image

prediction capabilities within a web application, providing users with insights into the model's classification of user-submitted images.

```
# Script in index function in the views.py file
image = request.FILES["image"]

# Read the image
image_content = image.read()
imag = cv2.imdecode(np.frombuffer(image_content, np.uint8), cv2.IMREAD_COLOR)

# image details
image_url =
f"{base64.b64encode(image_content).deco
de('utf-8')}"

# Process the image in-memory
img_from_ar = Image.fromarray(imag, 'RGB')
resized_image = img_from_ar.resize((256, 256))

test_image = np.expand_dims(resized_image, axis=0)
def_image = test_image / 255.0

class_names = {0: 'Basal cell carcinoma',
               1: 'Melanoma',
               2: 'Nevus'}

# Make prediction using the predict_image function
alteration, result_probability, result, prediction = predict_image(def_image)
probability_values = result[0]

# Plot the bar chart
plt.switch_backend('agg')
fig, ax = plt.subplots()
ax.bar(class_names.values(), probability_values, color=['blue', 'orange',
'green'])
ax.set_title('Probability Distribution')
ax.set_xlabel('Classes')
ax.set_ylabel('Probability')
ax.set_ylim(0, 1)

# Save the Matplotlib plot to a BytesIO buffer
buffer = io.BytesIO()
plt.savefig(buffer, format='png')
buffer.seek(0)
plt.close()

# Encode the image as base64
plot_image = base64.b64encode(buffer.getvalue()).decode('utf-8')


# Script in predictor.py file
# Import and define model as model
model = tf.keras.models.load_model(os.getcwd() + '/DenseNet121.h5')

# Define predict function
def predict_image(def_image):
    # Make the prediction
    result = model.predict(def_image)

    # Create dictionary with pertinent label encoding
    class_names = {0: 'Basal cell carcinoma',
                   1: 'Melanoma',
                   2: 'Nevus'}

    # Save the results as variables
    prediction = int(np.argmax(result))
    alteration = class_names[prediction]
```

```
result_probability = round(result[0][prediction], 2)
return alteration, result_probability, result, prediction
```