Politecnico di Torino

Politecnico di Torino

Master's Degree in Biomedical Engineering

Academic Year 2024/2025

Master's Degree Thesis

Development of Deep Learning Integrated Mobile Frameworks for Dermatological Image Analysis

Supervisors:
Prof. Massimo Salvi
Prof. Kristen M. Meiburger
Ing. Francesco Branciforti

Candidate:
Ibrahim Ghadre

July 2025

Table of Contents

List of Figures **5**

List of Tables **8**

Abstract **9**

Introduction **11**

1.1 Dermatology 11

1.1.1 Skin Cancer and the Importance of Early Diagnosis 12

1.1.2 Teledermatology: Opportunities and Challenges 15

1.1.3 Existing Mobile Applications for Dermatological Support 16

1.2 Artificial Intelligence in Diagnostic Support 19

1.2.1 Artificial Neural Networks (ANNs) 19

1.2.2 Convolutional Neural Networks (CNNs) 23

1.2.3 Generative Adversarial Networks (GANs) 26

1.2.3.1 GAN Fundamentals 26

1.2.3.2 Deep Convolutional GAN (DCGAN) 27

1.2.3.3 Progressive Growing GAN (ProGAN) 28

1.2.3.4 Style-Based GANs (StyleGAN) 29

1.2.3.5 Conditional GANs (cGAN) 29

1.2.3.6 Image-to-Image Translation: CycleGAN and Pix2Pix 30

1.2.4 Image Super-Resolution and Clinical Applications 31

1.2.4.1 Learning-Based Super-Resolution Methods 33

1.2.4.2 GAN-Based Super-Resolution: The SRGAN Model 33

1.2.4.3 Clinical Relevance and Applications 34

1.2.5 The Real-ESRGAN Model 34

1.2.5.1 Underlying GAN Framework 35

1.2.5.2 Degradation Modeling and Training Process 36

1.2.5.3 Model Architecture and Evolution 37
1.3 Frameworks for Mobile Deployment 37
1.3.2 NCNN Framework 38
1.3.3 ExecuTorch Framework 39
Materials and Methods 41
2.1 Dataset 41
2.2 Image Preprocessing Pipeline 42
2.3 Architecture and Configuration of the Real-ESRGAN Model 44
2.4 Model Conversion: from PyTorch to NCNN and ExecuTorch 45
2.5 Android App Development with NCNN and ExecuTorch 47
2.5.1 NCNN-Based Application 48
2.5.2 ExecuTorch-Based Application 49
2.6 Inference Benchmarking on Google Colab and Android Devices 49
Results 52
3.1 Qualitative Assessment 52
3.2 Quantitative Benchmark: Inference Time 53
3.3 Quantitative Image Quality Metrics 54
3.4 Visual Analysis of Images 59
3.4.1 Histogram Comparison 59
3.4.2 Pixel-by-pixel Difference Maps 62
3.5 Consistency and Reliability 63
Conclusions 65
4.1 Results Analysis 65
4.2 Improvements and Future Developments 66
4.3 Conclusions 68
Bibliography 69

List of Figures

Figure 1: Structure of the human skin. [2] 11

Figure 2: Incidence rates of skin cancer in the U.S. from 1999 to 2021, by gender (per 100,000 population). [3] 12

Figure 3: The five stages in melanoma evolution process. [8] 13

Figure 4: Dermatoscopic imaging setup and example. (A) Illuco IDS-1100 dermatoscope; (B) dermoscopic image of a malignant melanocytic lesion (ISIC_0000283). 14

Figure 5: Example of a teledermatology workflow. 15

Figure 6: Example of a mobile dermatology application prototype (eSkin) designed for skin lesion analysis and risk assessment. [8] 18

Figure 7: Nurugo™ Derma smartphone dermatoscope: (A) device structure, (B) attachment to a mobile phone, and (C) real-time image acquisition. [12] 18

Figure 8: Structure of an artificial neuron. 20

Figure 9: Example of a deep neural network architecture. The input layer receives image features, two hidden layers perform feature extraction, and the output layer assigns the images to one of possible diagnoses. [26] 21

Figure 10: Schematic representation of a convolutional neural network (CNN). The input image undergoes feature extraction through convolution and pooling layers, is flattened, and processed by fully connected layers to produce class probabilities. [29] 22

Figure 11: Comparison of different explainability techniques that highlight which regions of the image most influenced the prediction of the model. [34] 22

Figure 12: Sliding filter operation in convolutional layers. A 3×3 kernel moves across the input matrix, computing local features that are used to build activation maps in a CNN. [35] 24

Figure 13: Max-pooling process in convolutional neural networks. From left to right: (1) input matrix, (2) selected region proposal, (3) division into pooling sections, (4) identification of maximum values within each section, (5) resulting output matrix after max pooling. [36] 24

Figure 14: Examples of model fitting behaviors. Balanced fitting reflects an optimal model with good generalization. [37] 25

Figure 15: Architecture of a Generative Adversarial Network (GAN). The generator produces synthetic samples from a latent space, while the discriminator evaluates their authenticity by comparing them to real data. Training proceeds through adversarial fine-tuning of both netwoks. [39] 27

Figure 16: DCGAN generator architecture. [33] 28

Figure 17: ProGAN generator architecture. [33] 28

Figure 18: StyleGAN generator architecture. [33] 29

Figure 19: cGAN architecture. The label vector m is provided as extra information to both the generator and the discriminator. [40] 30

Figure 20: Training mechanism of CycleGAN. Generators G and F translate between domains X and Y, with cycle-consistency loss guiding the reconstruction of the original input from its translated form. [33] 30

Figure 21: Pix2Pix architecture. The generator learns to translate input sketches (x) into realistic images (G(x)), while the discriminator distinguishes between generated pairs (x, G(x)) and real pairs (x, y). [33] 31

Figure 22: An example of nearest neighbor interpolation. [42] 32

Figure 23: An example of bilinear interpolation. [42] 32

Figure 24: An example of bicubic interpolation. [42] 33

Figure 25: Architecture of SRGAN. The generator upsamples a low-resolution image through residual blocks and sub-pixel convolutions, while the discriminator distinguishes between real and generated high-resolution output. [45] 34

Figure 26: The Real-ESRGAN generator adopts the same architecture as ESRGAN, featuring residual-in-residual dense blocks and upsampling through pixel-shuffle layers. [47] 35

Figure 27: Structure of a Residual-in-Residual Dense Block (RRDB), composed of three dense blocks with residual and skip connections. [46] 36

Figure 28: Degradation pipeline used for training the Real-ESRGAN model. [47] ..36

Figure 29: Benchmark of deep learning frameworks on mobile CPUs. The chart reports the framework achieving the lowest inference latency for each combination of model and mobile device. [49] 39

Figure 30: High-level overview of the ExecuTorch deployment pipeline. A PyTorch model is exported and compiled into an optimized ExecuTorch program, which is executed at inference time through a lightweight runtime on edge devices. [51] 40

Figure 31: High-resolution dermoscopic images selected from the ISIC archive. Specifically, the chosen images are ISIC_0000062, ISIC_0000206, and ISIC_0000271. 41

Figure 32: Degraded version of the dermoscopic images obtained as an output of the preprocessing pipeline. 42

Figure 33: Preprocessing pipeline applied to the dermoscopic images before inference. 44

Figure 34: NCNN conversion and deployment pipeline. [54] 46

Figure 35: Executorch conversion and deployment pipeline. [52] 47

Figure 36: User interface flow of the two Android applications (NCNN above, ExecuTorch below). Each row shows the main steps performed by the user: (A) app launch, (B) image selection, and (C) inference result after image enhancement. 51

Figure 37: Qualitative comparison of the outputs obtained across different frameworks. Each row shows the degraded input and the corresponding results for the same image. 52

Figure 38: Pixel intensity histogram for image 1. 60

Figure 39: Pixel intensity histogram for image 2. 61

Figure 40: Pixel intensity histogram for image 3. 61

Figure 41: Pixel-by-pixel difference map for image 1. 62

Figure 42: Pixel-by-pixel difference map for image 2. 63

Figure 43: Pixel-by-pixel difference map for image 3. 63

Figure 44: Examples of corrupted outputs generated by the ExecuTorch-based application during inference. 64

List of Tables

Table 1: Inference time in milliseconds (ms) for each image across the three deployment frameworks. PyTorch was separated into CPU and GPU executions. Values reflect the average time required to process one image. 53

Table 2: PSNR values for each image processed by the three frameworks. 57

Table 3: MSE values for each image processed by the three frameworks..... 58

Table 4: SSIM values for each image processed by the three frameworks..... 58

Table 5: LPIPS values for each image processed by the three frameworks..... 58

Abstract

Skin cancer is one of the most potentially lethal cancers in the world, with malignant melanoma presenting aggressive clinical behavior and an increasing incidence over the last decade. Early and accurate diagnosis is essential to improve prognosis, and in this regard teledermatology has emerged as an increasingly viable solution to overcome spatial and temporal barriers in healthcare delivery. With the growing accessibility of smartphones and mobile dermatoscopes, remote dermatological screening is becoming increasingly common in both clinical and non-clinical settings. However, the quality of images acquired by patients remains a limiting factor, often compromised by poor resolution, improper focus, and variable lighting conditions, which can compromise the reliability and accuracy of diagnosis.

This study focuses on evaluating the implementation of the Real-ESRGAN model on mobile devices as a means of improving dermatological images acquired under suboptimal conditions. Real-ESRGAN is a super-resolution framework based on Generative Adversarial Networks (GANs) that generates high-resolution images from low-quality versions. Given the computational limitations typical of smartphones, the project evaluates the model performance, feasibility of integration, and responsiveness in mobile environments.

The Real-ESRGAN model was tested in three different environments. First, inference was conducted using the official PyTorch implementation in a Google Colab environment, which served as a baseline for output quality and processing speed under ideal computational conditions. Second, the model was converted to NCNN format and run on Android using a native C++ application, developed with JNI and OpenMP optimizations to maximize real-time performance on ARM processors. Third, a deployment was performed using ExecuTorch, a new framework developed by Meta that allows PyTorch models exported in .pte format to run efficiently on Android devices. In this case, a Java-based Android application was created to perform inference on the device. Prior to inference, all images underwent a pre-processing pipeline that degraded their quality to reproduce the imperfections typically found in teledermatology and to meet the input specifications of the model. Each custom application allowed users to upload dermatological images, apply super resolution, and view the results directly within the mobile interface, enabling end-to-end evaluation of model responsiveness and integration complexity in constrained environments. Each model version was evaluated through both visual inspections and objective image

quality metrics, including PSNR, SSIM, MSE, and LPIPS. Inference time was recorded on mobile devices to evaluate computational performance in terms of latency and responsiveness.

The results showed that NCNN outperformed Executorch in terms of speed, while the latter facilitated a more streamlined and native integration within PyTorch-based pipelines. Despite differences in execution frameworks, visual output quality remained high, confirming the effectiveness of Real-ESRGAN across all deployment environments. These results support the integration of super-resolution algorithms into mobile dermatology workflows, with potential applications in remote triage and diagnostic support where hardware limitations or limited bandwidth could otherwise compromise image fidelity.

Introduction

1.1 Dermatology

Dermatology is the medical discipline that focuses on the prevention, diagnosis, and treatment of conditions that affect the skin, as well as related structures like hair, nails, and mucous membranes.

Given its extensive surface area and constant exposure to external stimuli, the skin functions as both a protective barrier and a mirror of internal health status, which explains the high frequency of dermatological manifestations in clinical encounters [1].

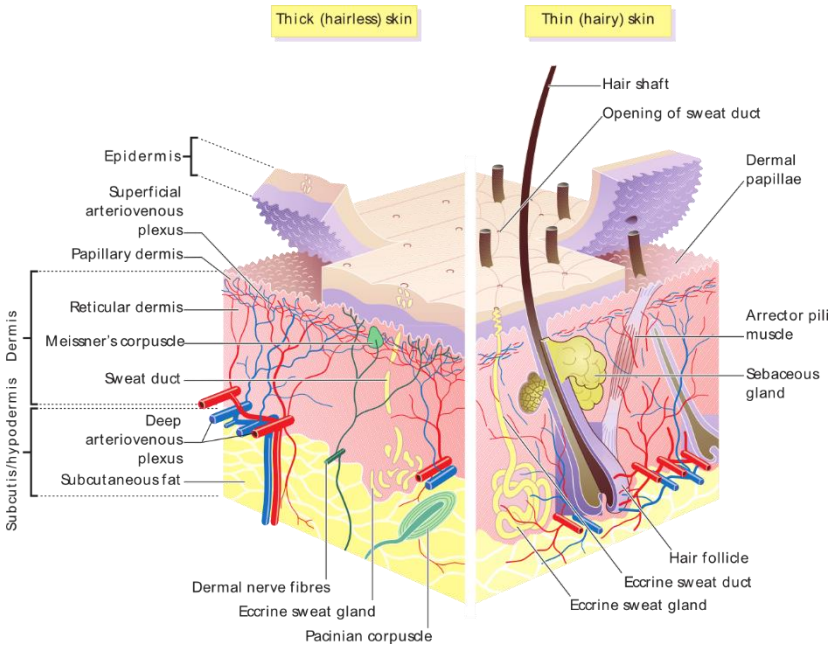


Figure 1: *Structure of the human skin. [2]*

In the past decade, dermatologists have faced a consistent rise in skin cancer cases, highlighting the significance of prompt management and early detection initiatives.

At the same time, technology has begun to play a more central role in everyday dermatological practice. Tools such as digital dermoscopy, mobile health apps, and even AI-assisted image analysis are becoming part of a broader shift toward more accessible and data-driven care [2].

1.1.1 Skin Cancer and the Importance of Early Diagnosis

Currently, skin cancer is one of the most frequently diagnosed cancers worldwide, and the incidence rates are consistently increasing across all continents. As shown in Figure 2, recent epidemiological studies show a rising incidence in skin cancer in the past decades [3].

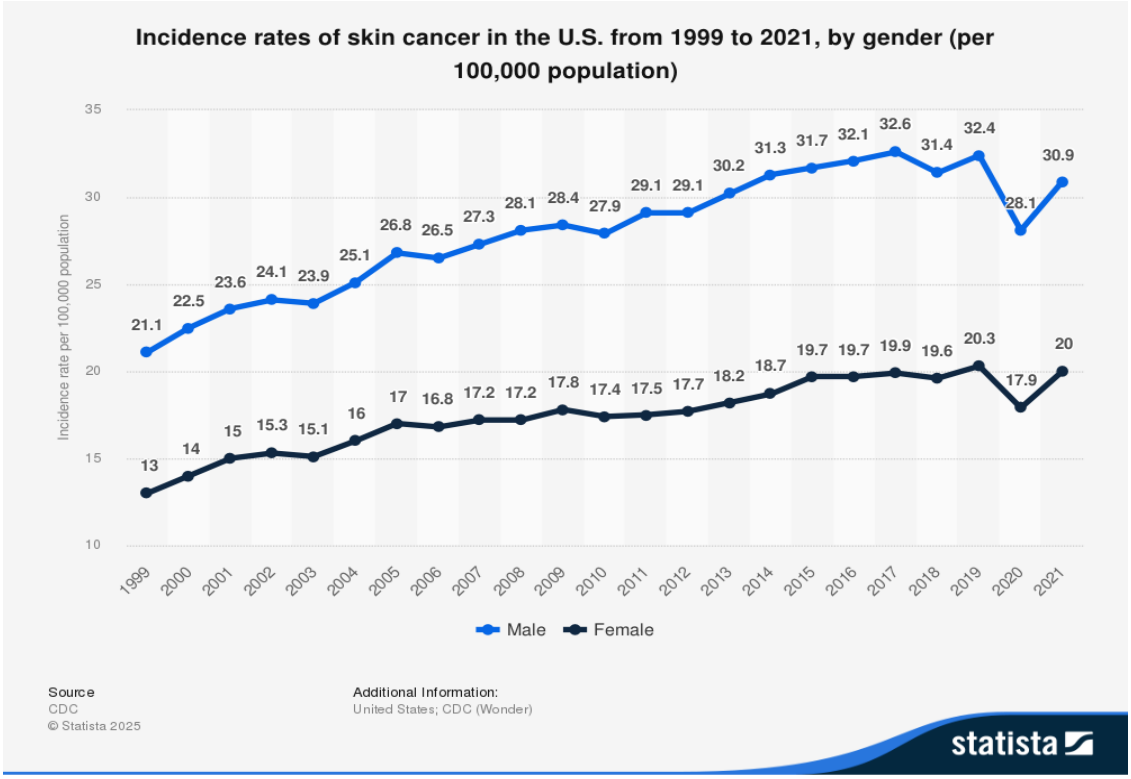


***Figure 2**: Incidence rates of skin cancer in the U.S. from 1999 to 2021, by gender (per 100,000 population). [3]*

This rise in incidence is primarily attributed to cumulative ultraviolet (UV) exposure, an aging demographic, and increased awareness among the public resulting in more frequent screenings [4], [5]. Although basal cell carcinoma (BCC) and squamous cell carcinoma (SCC) often exhibit modest metastatic potential and are considered as less aggressive, their large prevalence significantly affects healthcare system costs and resource allocation [4]. In contrast, melanoma, despite its lower incidence, presents a significantly higher risk as a result of its rapid metastasis and high mortality rate if not diagnosed early [6].

The clinical management of melanoma is significantly influenced by the stage of the disease. In the United States (US), five-year survival rates exceed 98% when the lesion is localized and confined to the epidermis, but these rates decline dramatically as the tumor penetrates deeper into the dermis [7]. When melanoma exceeds 4 mm in size, it begins to reach lymphatic sites and then spreads to other organs, as shown in Figure 3. Therefore, early diagnosis is not only helpful but essential. The treatment of early-stage melanoma is often possible through surgical excision alone, thus avoiding the need for systemic therapies such as immunotherapy or chemotherapy, which are more frequently required in advanced cases and are associated with increased cost and morbidity [8], [9].

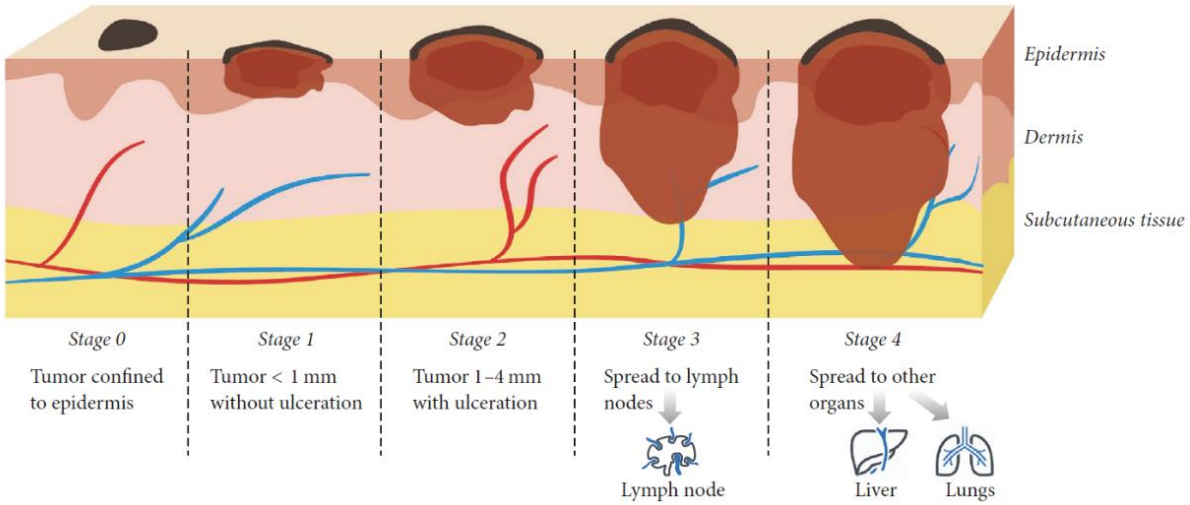
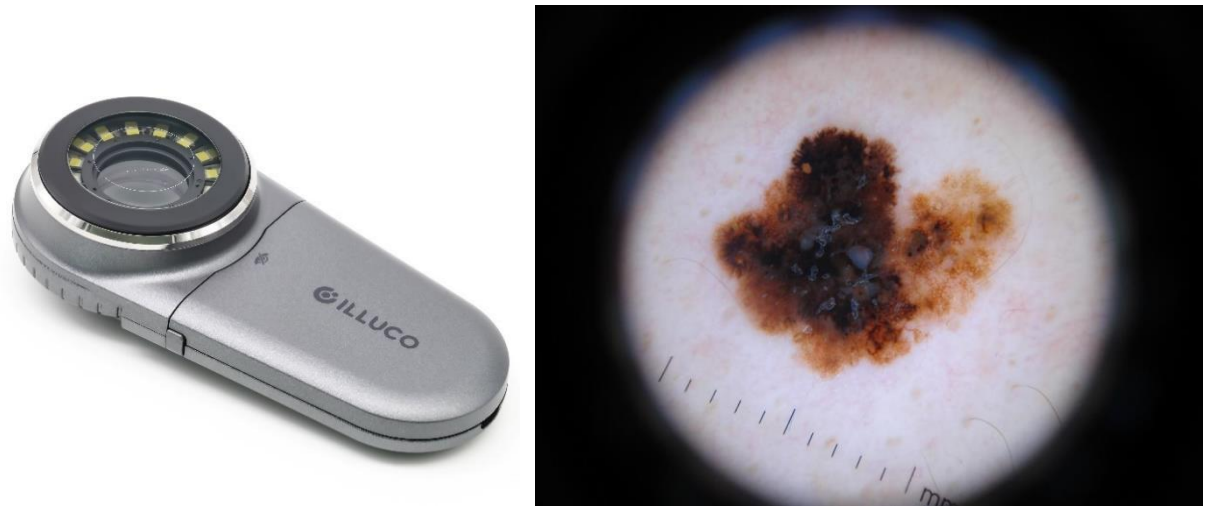


***Figure 3**: The five stages in melanoma evolution process. [8]*

The primary method for early detection is the visual examination of the skin, which is frequently enhanced through dermoscopy, a non-invasive diagnostic technique that enables clinicians to observe sub-surface skin structures that are not visible to the naked eye. Dermoscopy has been shown to significantly improve diagnostic accuracy, particularly in the hands of trained dermatologists. However, diagnostic performance is heavily influenced by the level of experience of the clinician. In rural areas or primary care settings, where dermatologists are scarce, non-specialist physicians might lack the necessary training or instruments to confidently identify early-stage melanomas [10].



(A) (B)

Figure 4: *Dermatoscopic imaging setup and example. (A) Illuco IDS-1100 dermatoscope; (B) dermoscopic image of a malignant melanocytic lesion (ISIC_0000283).*

In response to these challenges, public health efforts in recent years have promoted awareness campaigns, sun protection guidelines, and screening programs aimed at encouraging individuals to monitor changes in their skin and seek medical consultation when observing suspicious lesions [4].

In addition, the increasing availability of mobile technology has allowed the development of new approaches for early detection. Smartphone cameras, when combined with dermoscopy adapters or AI-based diagnostic tools, are increasingly being evaluated as potential solutions for remote screening and self-monitoring, especially in settings with limited access to dermatological care [11], [12], [13].

However, numerous obstacles persist in spite of these developments. The reliability of remote assessments is affected by variability in image quality, lighting conditions, focus, and resolution. Furthermore, the lack of standardization in how skin lesions are photographed by patients contributes to diagnostic uncertainty [14]. For these reasons, improving the quality of skin images obtained via smartphones has become a critical area of research. Techniques such as image super-resolution, which are designed to reconstruct fine visual details that are lost in low-quality images, gained attention for their potential to improve diagnostic accuracy, particularly when combined with machine learning-based classification systems [15], [16].

1.1.2 Teledermatology: Opportunities and Challenges

Across many healthcare systems, dermatology faces the dual challenge of growing patient volumes and an insufficient supply of specialists. This imbalance has fueled the adoption of teledermatology, a branch of telemedicine which allows for the remote diagnosis and monitoring of cutaneous conditions using digital platforms *(Fig. 5)*. The model is particularly well-suited for extending care to populations in remote or underserved areas, where traditional face-to-face visits may be unfeasible. Thanks to both asynchronous and real-time modalities, clinicians can manage caseloads more effectively by reducing bottlenecks, accelerating referrals, and directing urgent cases toward timely treatment [17].

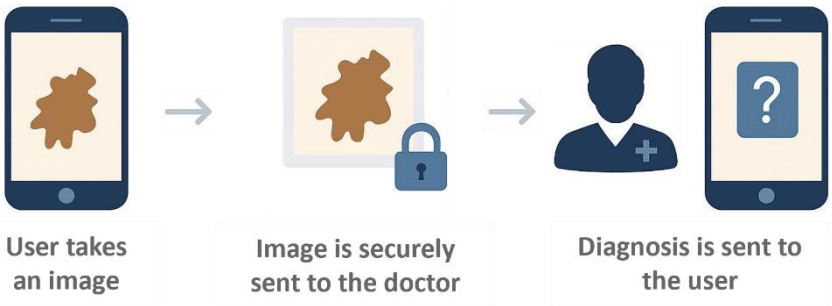


***Figure 5**: Example of a teledermatology workflow.*

Effectively structured teledermatology programs have shown diagnostic accuracy comparable to that of in-person consultations, especially in the evaluation of common dermatologic disorders like acne, eczema, and psoriasis [18]. Its role in oncologic screening is also expanding, with remote assessments demonstrating value in identifying early-stage malignancies. However, the inability to conduct tactile evaluations remains a significant limitation, particularly for subtle clinical signs that rely on direct palpation or subtle visual indicators.

The outbreak of the COVID-19 pandemic acted as a major catalyst for the widespread implementation of teledermatology. Healthcare systems across the globe were compelled to integrate digital platforms into standard dermatologic workflows to ensure continuity of care. While the transition enabled sustained care delivery and adherence to treatment for chronic skin diseases, it also exposed underlying weaknesses, including disparities in digital access, variations in technological literacy, and gaps in institutional readiness [19].

Despite its promise, the effectiveness of teledermatology is highly dependent on the quality of the images submitted by patients. Inadequate lighting, motion blur, low resolution, and improper framing are among the most common issues that hinder accurate remote evaluation [14]. Furthermore, the wide range of mobile devices and the lack of standardized acquisition methods contribute to inconsistencies that pose a challenge to both human reviewers and AI models. Therefore, researchers and doctors

have started developing strategies aimed at improving the visual accuracy of photographs sent via smartphones and refining image acquisition protocols.

Efforts to improve the reliability of patient-submitted dermatological images have included both the education by healthcare providers regarding the use of the camera and the deployment of advanced image enhancement models [16], [20]. Such tools are able to compensate for common visual flaws and reconstruct diagnostically relevant features. In low-resource healthcare settings, where high-end equipment is unavailable, these methods may prove essential to ensuring the effectiveness and scalability of remote dermatologic care.

While regulatory frameworks for teledermatology continue to evolve, especially regarding data privacy and cross-border consultations, ongoing collaboration between clinicians, engineers, and public health authorities is essential to ensure that the expansion of remote dermatologic services maintains both diagnostic quality and equitable access.

1.1.3 Existing Mobile Applications for Dermatological Support

The increasing availability of mobile devices with high resolution cameras has opened the door to a new generation of dermatology-related applications. Designed to assist with lesion diagnosis and monitoring, these tools range from simple trackers to complex systems that incorporate machine learning for classification purposes. Several platforms, such as SkinVision and eSkin, have been subjected to varying degrees of clinical assessment, revealing both potential and inconsistency in their diagnostic contributions.

One of the most well-studied examples is SkinVision, a CE-certified application that employs a support vector machine (SVM) classifier to estimate the malignancy risk of photographed skin lesions. In a recent review, the app demonstrated a sensitivity of 95% and a specificity of 78% for skin cancer detection. However, the study also noted a major limitation: the absence of clinical confirmation via physical examination, raising concerns about false reassurance and misclassification in self-assessment contexts [21].

A more recent and rigorous study evaluated the diagnostic capabilities of SkinVision by comparing its risk assessment performance with histopathological examination, the clinical gold standard in skin cancer diagnosis. The analysis conducted by the authors involved 1,204 pigmented skin lesions from 114 patients, revealing that the application tends to identify a disproportionately high number of lesions as high risk, with an overdiagnosis rate of 45.4%. This trend led to numerous false positive classifications, particularly in benign nevi, raising concerns about possible over-referrals, increased anxiety among users, and unnecessary pressure on clinical resources. The authors attributed part of this problem to the underlying architecture of the app, which relied on a support vector machine (SVM) rather than more adaptive deep learning methods.

While the tool is not intended to replace professional assessments, it is designed as a triage mechanism to encourage early dermatological consultation. However, its limitations highlight the need for more explicit instructions for use and further algorithmic validation before its integration into clinical protocols [22].

In addition to purely software-based solutions, several mobile teledermatology systems have incorporated external optical devices, such as clip-on dermatoscopic lenses or microscopes adapted for smartphones, to improve image quality and diagnostic reliability. These accessories aim to overcome the optical limitations of native smartphone cameras by enabling a more thorough inspection of subcutaneous skin structures.

A study conducted in the Swedish public healthcare system has shown that mobile teledermoscopy has drastically reduced waiting times for diagnosis and treatment. A first consultation via teledermoscopy not only allowed patients to receive initial feedback within 24 hours but also enabled patients with possible melanomas to be given higher priority. In fact, the waiting time for diagnosis and treatment for patients who used the teledermoscopy service was 36 days, compared to 85 days for those who relied on traditional paper-based referrals [13].

A prototype application developed for the early diagnosis of melanoma is eSkin. This application, whose interface is shown in Figure 6, was designed to capture dermatoscopic images by pairing a smartphone camera with a dermatoscopic lens and then transmitting them to a remote server for lesion evaluation. The system adopts a client-server architecture and implements two different algorithms to analyze possible melanomas and micro-melanomas (lesions with a diameter $\leq$ 5 mm). The study reported promising results, with a sensitivity of 90% for micro-melanomas and 86% for melanomas. However, there have been no further developments since the article was published in 2018 [8].

Figure 6: *Example of a mobile dermatology application prototype (eSkin) designed for skin lesion analysis and risk assessment. [8]*

Another study evaluated a low-cost portable dermatoscope for smartphones as part of a teledermatology platform. The system proved capable of capturing diagnostically relevant features and showed promising results for early diagnosis workflows. Nevertheless, the study also highlighted some practical limitations such as inconsistent image quality affected by lighting conditions and device stability [23].

A more recent and technically advanced study examined the use of the Nurugo™ Derma lens, a consumer-grade smartphone dermatoscope, in combination with a convolutional neural network (CNN) trained on ISIC images *(Fig. 7)*. The CNN ensemble was able to classify melanoma, nevi, and seborrheic keratoses from Nurugo™-acquired images with a diagnostic accuracy of 83.9% and an F1 score of 80.8%, results comparable to those of experienced dermatologists. The system showed the potential of integrating deep learning with low-cost imaging for real-world dermatologic applications, despite some limitations such as glare and a restricted field of view [12].

Figure 7: *Nurugo™ Derma smartphone dermatoscope: (A) device structure, (B) attachment to a mobile phone, and (C) real-time image acquisition. [12]*

These results highlight the potential of integrating hardware augmentation with AI-based analysis in mobile dermatology. At the same time, they highlight the need to

standardize acquisition protocols, validate heterogeneous devices, and carefully calibrate both software and hardware to ensure reliable and equitable implementation in different clinical settings.

1.2 Artificial Intelligence in Diagnostic Support

Artificial intelligence (AI) encompasses a wide range of computational methods that allow machines to replicate cognitive functions that are typically associated with human intelligence, such as learning, reasoning, and decision-making. These systems are engineered to autonomously analyze information, adjust to incoming data, and improve their performance over time without the need of explicit reprogramming for each task [24]. While AI is extensively utilized in consumer applications like search engines, recommendation systems, voice recognition, and autonomous vehicles, its growing impact in scientific and medical domains is particularly significant.

Over the past decade, the growing availability of clinical data, along with improved computational power and refined learning algorithms, has facilitated the integration of AI into diagnostic medicine. In diagnostics, machine learning and deep neural networks have demonstrated the capacity to process large volumes of complex data, detect subtle anomalies, and assist clinicians in making accurate and timely decisions. Techniques such as support vector machines and convolutional neural networks have become central to applications involving image classification, segmentation, and biosignal analysis, helping to minimize diagnostic discrepancies and improve early detection rates [25], [26], [27].

1.2.1 Artificial Neural Networks (ANNs)

Machine learning (ML), a prominent subset of artificial intelligence, focuses on developing systems that can identify patterns in data and refine their behavior through experience. These algorithms are capable of constructing predictive models by analyzing prior examples, allowing them to generalize effectively to new and unseen data without explicit rule-based instructions. A wide range of ML techniques exists, from simple linear regressors to complex ensemble methods, each tailored to specific data types and structural properties [28], [24].

Among the most versatile models in this field are artificial neural networks (ANNs), which are particularly effective in domains involving high-dimensional and unstructured data, such as biomedical images or physiological signals. Inspired by the architecture of biological neural systems, ANNs consist of layers of interconnected processing units known as neurons. Each neuron receives signals as input, multiplies them by specific weights, aggregates the "weighted" signals, and finally applies a nonlinear activation function that produces an output *(Fig. 8)*. The result is then propagated forward to the next layers of neurons, and through iterative training, the network progressively adjusts its weights to minimize errors [28].

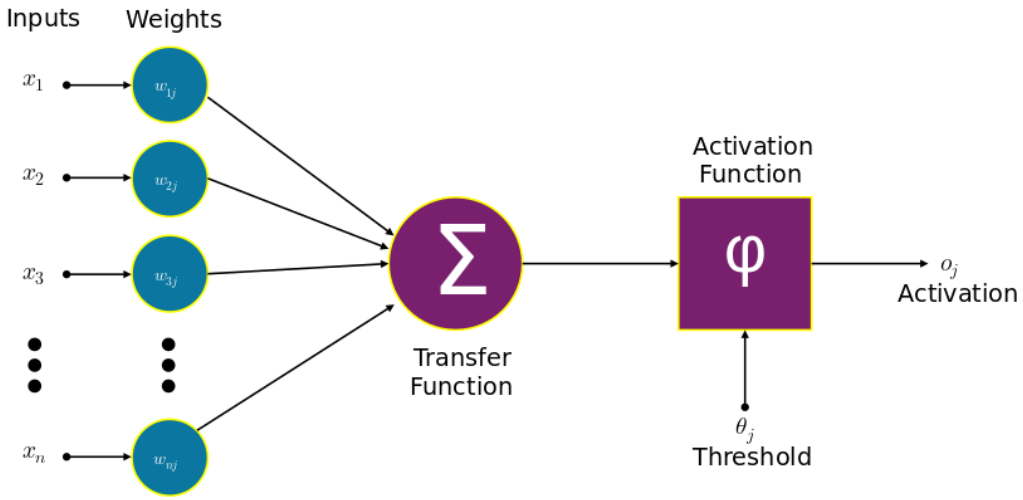


Figure 8: *Structure of an artificial neuron.*

The expansion of computational capabilities and the availability of large-scale datasets have enabled ANNs to evolve into more elaborate configurations, giving rise to the field of deep learning (DL). In deep neural networks (DNNs), the presence of multiple hidden layers facilitates the learning of hierarchical feature representations directly from raw input, eliminating the need for manually engineered features, a task that is often time-consuming and domain specific [28].

A typical feedforward neural network is composed of an input layer, several hidden layers, and an output layer. During training, optimization algorithms, such as stochastic gradient descent, adjust the internal weights of the network to minimize a chosen loss function that quantifies the gap between predicted and actual outcomes. The process relies on backpropagation, a mechanism that propagates error signals backward through the network to iteratively update each weight, thereby improving performance over successive epochs [28]. An example of a deep neural network architecture for image classification is shown in Figure 9 [26].

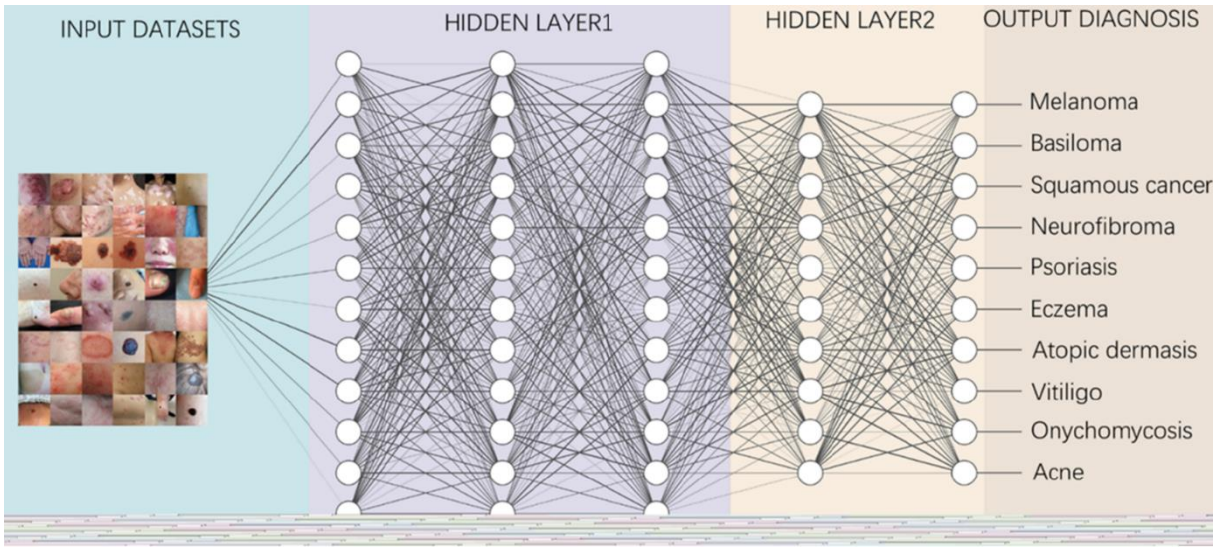


Figure 9: *Example of a deep neural network architecture. The input layer receives image features, two hidden layers perform feature extraction, and the output layer assigns the images to one of possible diagnoses. [26]*

These architectural advances have significantly enhanced the capacity of the model to interpret complex inputs. Deep networks are capable of learning abstract features at increasing levels of complexity, making them especially well-suited to fields such as medical imaging and temporal signal analysis, where data are both structured and high in dimensionality [25], [27]. Unlike traditional machine learning models that depend on feature engineering, deep networks automatically extract relevant representations from the data itself, an important advantage in clinical contexts, where domain expertise may be limited or inconsistently applied.

Within this domain, convolutional neural networks (CNNs) have become the architecture of choice for tasks involving image analysis. By applying trainable filters across localized regions of the input, CNNs can detect features such as edges, textures, and geometric shapes. Pooling layers then reduce the spatial resolution, allowing the model to retain key information while simplifying computational complexity. These layered operations enable CNNs to capture both local detail and global structure. In clinical diagnostics, including the analysis of skin lesions, diabetic retinopathy screening, and the interpretation of radiographic images, CNN-based models have

achieved results that, in many cases, approach or match the accuracy of expert practitioners [26], [30], [31].

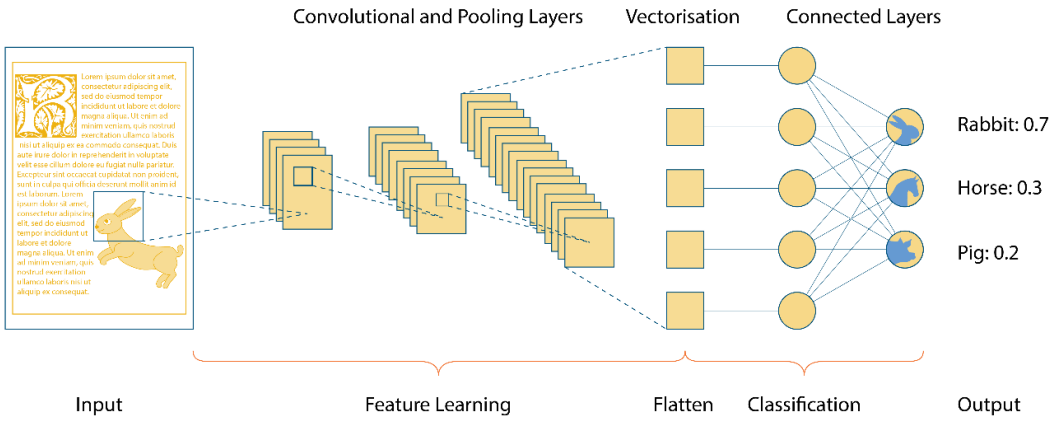


Figure 10: *Schematic representation of a convolutional neural network (CNN). The input image undergoes feature extraction through convolution and pooling layers, is flattened, and processed by fully connected layers to produce class probabilities. [29]*

Despite their effectiveness, deep learning models are heavily dependent on access to extensive, high-quality labeled datasets. In the medical field, such datasets are difficult to assemble due to privacy considerations, annotation variability, and the low prevalence of certain conditions. To mitigate these limitations, researchers have adopted strategies such as data augmentation, which synthetically expands training sets through transformations like rotation or scaling; transfer learning, which adapts pretrained models to new medical tasks; and generative approaches, such as the use of generative adversarial networks (GANs) to simulate realistic synthetic data [24], [25], [32], [33].

However, one of the principal challenges facing the adoption of deep models in healthcare remains their lack of transparency. While performance metrics are often impressive, the inner workings of these systems are not inherently interpretable. This has prompted the development of explainable AI (XAI) techniques, including saliency maps, Grad-CAM, and attention-based models, that offer visual insight into which features most influence a given prediction, providing a bridge between black-box systems and clinical accountability *(Fig.11)* [34].

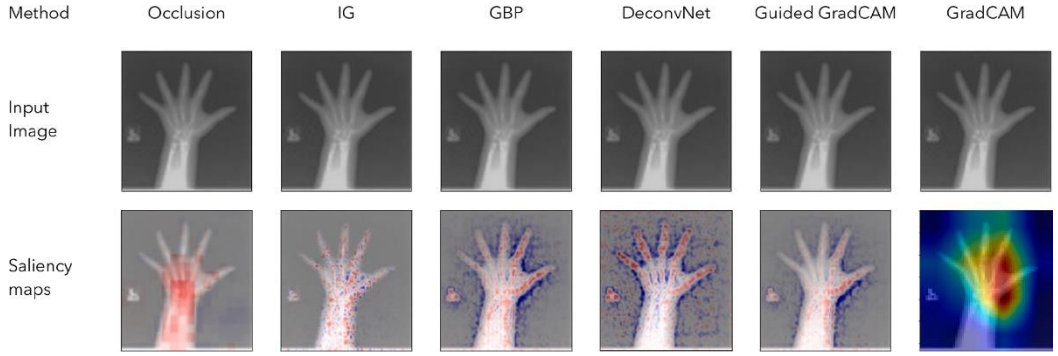


Figure 11: *Comparison of different explainability techniques that highlight which regions of the image most influenced the prediction of the model. [34]*

That said, more conventional machine learning models, such as logistic regression, decision trees, and support vector machines, retain strong relevance in medical applications, particularly when interpretability and robustness in low-data regimes are priorities. In some cases, these models may outperform deep architectures when dealing with structured, low-dimensional datasets or when transparency is critical for clinical decision-making.

Finally, depending on the nature of the activity and the structure of the available data, the learning paradigms used for both ML and DL models can be divided into supervised, unsupervised, and reinforcement approaches [24].

- **Supervised learning:** the model is trained on a dataset of labeled data. During training, the model makes predictions based on the inputs received and then compares the predicted values with the actual values corresponding to the labels provided. The error made is then measured and the network parameters are adjusted based on this. This learning method is the dominant technique for regression or classification tasks.
- **Unsupervised learning:** the model receives raw data as input, without any indication or label. During the training phase, the network tries to discover hidden and intrinsic patterns in the data. This method is widely used for clustering or dimensionality reduction tasks.
- **Reinforcement learning:** This is a trial-and-error approach where the model makes decisions within an environment from which it receives feedback. Based on this feedback, the model then adjusts its future actions. This approach is widely used in fields such as robotics and autonomous driving.

1.2.2 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are deep neural network architectures widely employed in domains where spatial relationships within data are crucial such as image classification and medical diagnostics. Their design, which is inspired by the way the human visual cortex processes stimuli, allows them to recognize and generalize patterns in localized regions of an image. These initial detections are gradually integrated into more abstract features as data is propagated through successive layers. What makes CNNs particularly interesting in clinical applications is their ability to learn directly from raw pixel values, minimizing dependence on handcrafted features or expert-curated descriptors [25].

Each CNN typically begins with convolutional operations, where small, trainable filters slide over the input to generate feature maps that emphasize specific visual cues [35]. During the learning process, these filters adapt to improve the recognition of informative elements within the image *(Fig. 12)*. This is followed by the application of

a non-linear transformation, frequently ReLU, which improves the ability of the network to model complex and non-linear relationships [28].

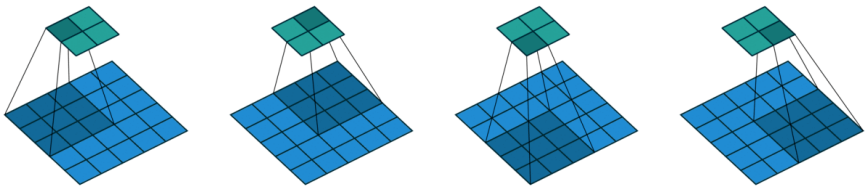


***Figure 12**: Sliding filter operation in convolutional layers. A 3×3 kernel moves across the input matrix, computing local features that are used to build activation maps in a CNN. [35]*

Pooling layers are used to reduce the spatial dimensions of the feature maps, typically via max-pooling or average-pooling, thereby decreasing computational load while preserving essential information [36]. The data is downsampled by these layers, which select representative values within local regions, thereby preserving critical visual information and removing redundancy *(Fig. 13)*. Pooling also provides a degree of spatial invariance, which allows the network to maintain performance even when the input is slightly translated or distorted [28].

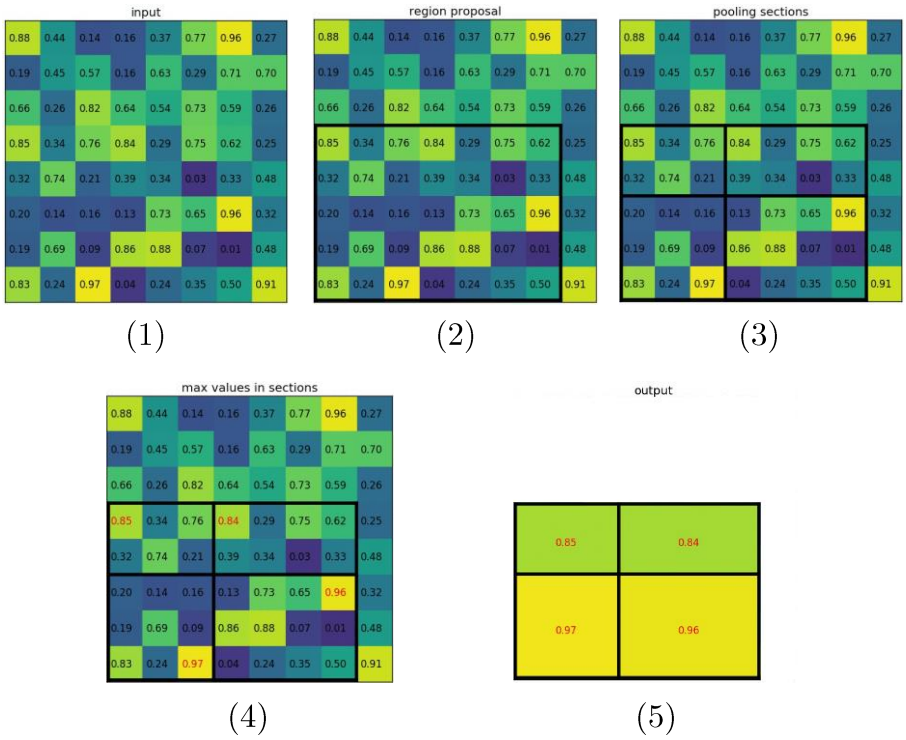


***Figure 13:** Max-pooling process in convolutional neural networks. From left to right: (1) input matrix, (2) selected region proposal, (3) division into pooling sections, (4) identification of maximum values within each section, (5) resulting output matrix after max pooling. [36]*

As the network advances toward its final layers, spatial features are flattened and passed into one or more dense layers, which are also referred to as fully connected (FC) layers. These layers generate a prediction based on the acquired features received as input. Each neuron in a FC layer is connected to all neurons in the previous layer. This structure, also known as Multi-Layer Perceptron (MLP), allows for the input management of very complex data and proves to be particularly effective in tasks requiring classification or regression outputs [28].

Typically, the training process is supervised and focuses on the optimization of a loss function, which quantifies the mismatch between the true labels and the predictions of the model. In order to improve the accuracy of predictions, backpropagation is used to propagate the error backward through the network, adjusting parameters layer by layer. This iterative process is guided by optimization algorithms, such as Stochastic Gradient Descent (SGD), Adam, and Root Mean Square Propagation (RMSProp), which evaluate the computed gradients to determine the updates [28].

The performance of a CNN is significantly influenced by the selection and tuning of its hyperparameters, including the learning rate, the number of hidden layers, kernel dimensions, and batch size, which affect how the model learns from data and adapts during training. The learning rate determines the step size of the updates to the weights of the model during optimization, the kernel size defines the spatial dimensions of the filters used in convolutional layers to extract localized features, and the batch size determines how many samples are processed simultaneously before the model updates its parameters. An inappropriate configuration of these elements can hinder convergence, slow down learning, or cause the model to settle on suboptimal solutions [28].

Two common issues that can arise during model development are overfitting and underfitting. Overfitting occurs when the network becomes overly tailored to the training data and does not perform well on new inputs, while underfitting occurs when it fails to capture the underlying pattern of the data and performs poorly even on known samples *(Fig. 14)*. To address these challenges, several types of regularization methods can be used. Dropout randomly disables a fraction of neurons during training, encouraging the model to learn redundant and more generalizable patterns. Batch normalization helps stabilize activations across layers, accelerating convergence and reducing sensitivity to initialization. Additionally, weight decay imposes a penalty on large parameter values to discourage overcomplex solutions [28].

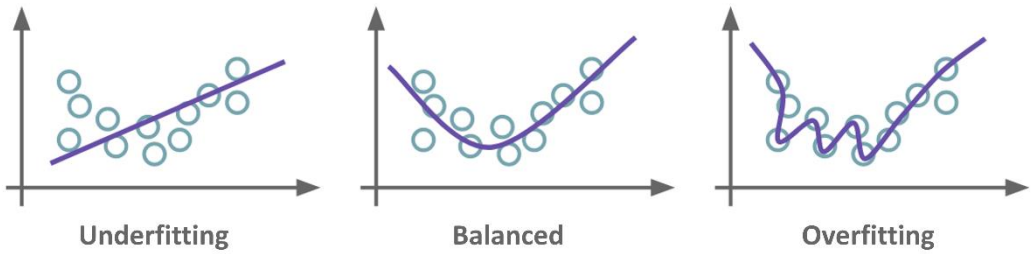


Figure 14: *Examples of model fitting behaviors. Balanced fitting reflects an optimal model with good generalization. [37]*

In clinical settings, acquiring large annotated datasets can sometimes be challenging. To address this, data augmentation is utilized to enhance training datasets by applying transformations such as rotation, scaling, and flipping. Furthermore, transfer learning has become a standard technique: CNNs pretrained on extensive datasets like ImageNet can be adapted to smaller, domain-specific medical datasets to achieve competitive outcomes with limited resources [25].

In recent years, several convolutional neural network architectures have been introduced to address limitations in training depth, computational efficiency, and deployment feasibility. Models such as ResNet, DenseNet, and MobileNet exemplify these advancements. ResNet incorporates residual connections that allow gradients to flow more effectively through deep networks, mitigating vanishing gradient issues. DenseNet connects each layer to all subsequent layers within a block, promoting feature reuse and improving gradient propagation. MobileNet, designed with depthwise separable convolutions, achieves substantial reductions in model size and computation, making it particularly suitable for applications on mobile and embedded devices [37], [38].

As CNNs have gained traction in clinical workflows, interpretability has emerged as a core requirement. Clinicians must be able to trust and validate algorithmic outputs, particularly when they support high-stakes decisions. Visualization tools, including Grad-CAM and saliency maps, assist in this process by highlighting image regions that most influenced the decision of the network. This visual feedback allows medical professionals to judge whether the attention of the model aligns with clinical expectations and to identify potential failure points [34].

CNNs remain a cornerstone of image-based diagnostics, especially in specialties that rely on detailed spatial and morphological features. Fields such as dermatology, radiology, and ophthalmology continue to benefit from deep learning models that are capable of extracting multi-scale features and maintaining spatial coherence, two aspects essential for high diagnostic precision.

1.2.3 Generative Adversarial Networks (GANs)

1.2.3.1 GAN Fundamentals

Generative Adversarial Networks (GANs) constitute a class of generative models developed to approximate complex data distributions by learning how to generate new samples that closely resemble those found in a target dataset. Originally introduced in 2014, the architecture comprises two neural networks with opposing objectives. The first, called the generator G, receives as input a random noise vector $\mathbf{z} \sim \mathbf{p_z(z)}$ and maps it to the data space in an attempt to create realistic outputs G(z). The second, the discriminator D, takes as input a sample and outputs a probability value estimating whether the sample is drawn from the true data distribution $\mathbf{p_{data}(x)}$ or was generated synthetically [32], [39].

These two networks are trained simultaneously in a competitive scenario where the generator is optimized to deceive the discriminator, while the discriminator is trained to correctly distinguish real from synthetic data *(Fig. 15)*. This process is formalized as a two-player min-max game with the following objective function:

$$\min_G \max_D E_{x\sim p_{\mathrm{dt}}(x)} [\log D\,(x)] + E_{z\sim p_z(z)} \left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

This equation contains two terms. The first term, $E_{x\sim p_{\mathrm{dt}}(x)}[\log D\,(x)]$, represents the expectation over real data samples and encourages the discriminator to assign a high probability to authentic data. The second term, $E_{z\sim p_z(z)} \left[\log\left(1 - D\big(G(z)\big)\right)\right]$, penalizes the discriminator when it incorrectly classifies generated data as real. Simultaneously, the generator seeks to minimize this same term by producing samples G(z) that the discriminator is less able to identify as synthetic.

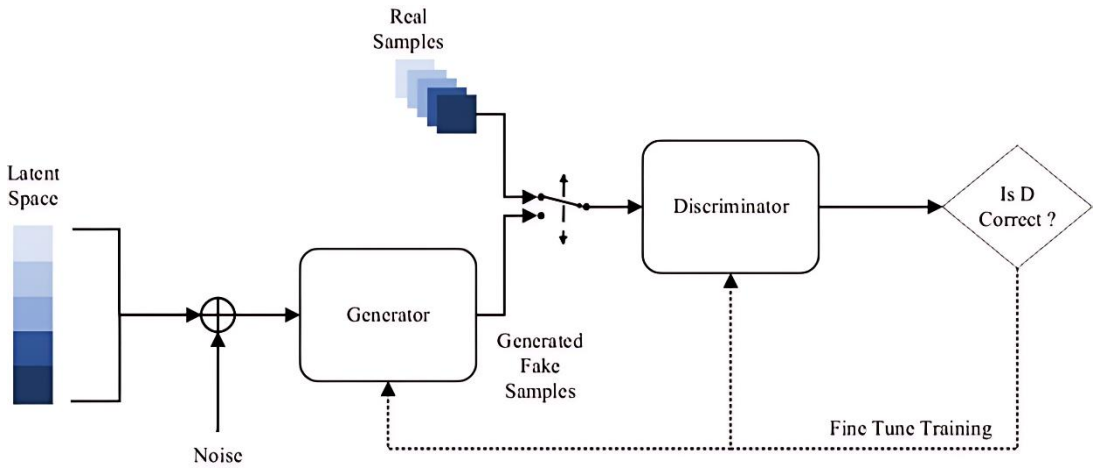


Figure 15: *Architecture of a Generative Adversarial Network (GAN). The generator produces synthetic samples from a latent space, while the discriminator evaluates their authenticity by comparing them to real data. Training proceeds through adversarial fine-tuning of both netwoks. [39]*

As training progresses, the generator improves its ability to mimic the data distribution, while the discriminator becomes more refined in detecting subtle inconsistencies. The ideal equilibrium is reached when D(x) = 0.5 for all x, meaning the discriminator is no longer able to distinguish between real and generated samples, implying that G has learned to produce data indistinguishable from the true distribution [32].

1.2.3.2 Deep Convolutional GAN (DCGAN)

One of the earliest and most influential adaptations is the Deep Convolutional GAN (DCGAN), which replaced fully connected layers with convolutional and transposed convolutional operations to better capture spatial hierarchies in image data. The

generator in DCGAN utilizes transposed convolutions to iteratively upsample from a low-dimensional latent vector, therefore progressively constructing image features at progressively higher resolutions *(Fig. 16)*. Batch normalization is applied to stabilize gradient flows and facilitate convergence, while ReLU and Tanh activations are utilized to introduce non-linearity and bound the output values. These design choices improve the coherence and structural consistency of the generated samples, particularly in visual domains requiring fine-grained texture synthesis [33].

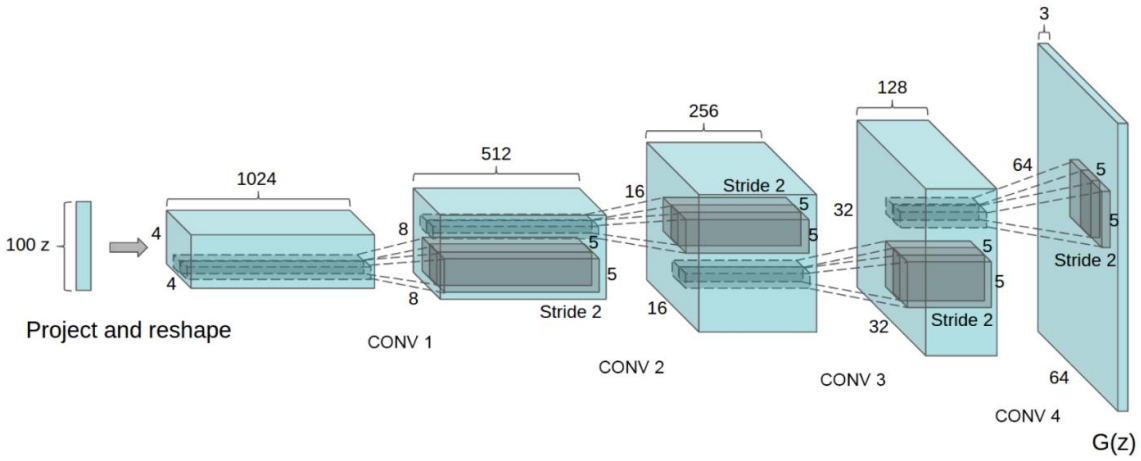


Figure 16: *DCGAN generator architecture. [33]*

1.2.3.3 Progressive Growing GAN (ProGAN)

To address the instability often encountered in training GANs at high resolutions, the Progressive Growing GAN (ProGAN) architecture was introduced in 2018. Training starts at a low resolution, and the dimensionality of the generator and discriminator is incrementally increased through subsequent layer additions in this model *(Fig. 17)*. At each stage, the network is allowed to stabilize before being exposed to more detailed

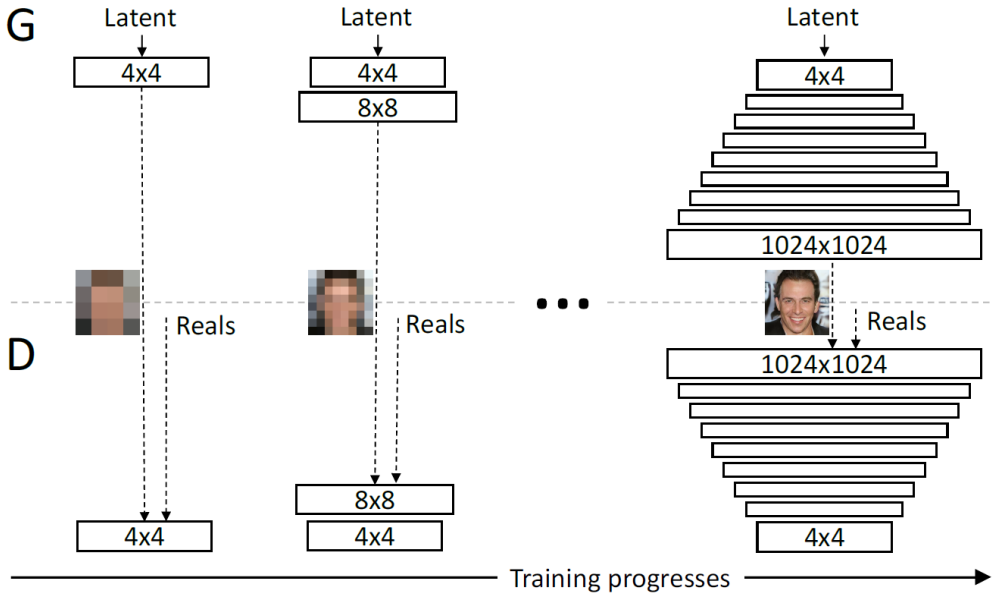


Figure 17: *ProGAN generator architecture. [33]*

representations. This approach allows the generator to initially capture global structural patterns before learning finer textures, thereby minimizing the risk of mode collapse and gradient vanishing [33].

1.2.3.4 Style-Based GANs (StyleGAN)

The StyleGAN family introduced a novel method for controlling image synthesis via modulation of the latent space. Instead of relying solely on a fixed latent vector input at the beginning of the network, StyleGAN injects learned intermediate representations at various layers through adaptive instance normalization (AdaIN), allowing disentangled control over individual attributes of the generated image *(Fig. 18)*. This approach enables the fine manipulation of features such as color gradients, spatial frequency, and morphology. The network architecture also includes a mapping network that transforms input latent codes into an intermediate space, improving semantic coherence and enabling more intuitive editing of output characteristics. Subsequent versions, such as StyleGAN2 and StyleGAN3, refined this architecture by addressing artifacts like droplet distortions and improving spatial consistency across layers [33].

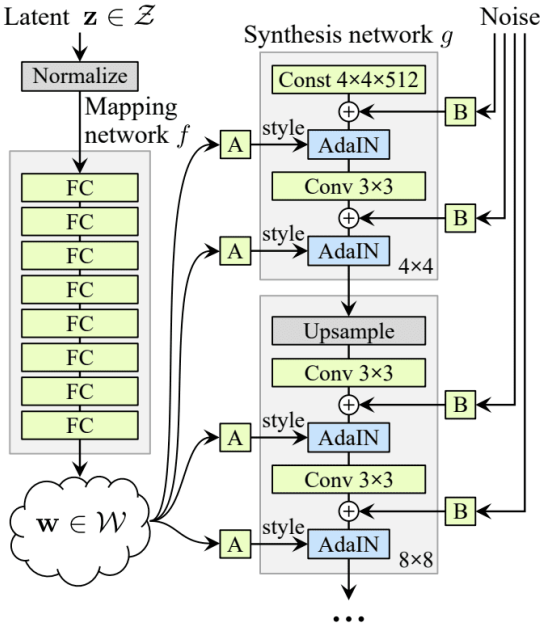


Figure 18: *StyleGAN generator architecture. [33]*

1.2.3.5 Conditional GANs (cGAN)

Conditional GANs (cGANs) introduce additional input variables, such as class labels or segmentation masks, into both the generator and discriminator *(Fig. 19)*. This conditioning steers the generation process, enabling the production of content aligned with specific categories. Such control is particularly valuable in medical imaging, where outputs can be tailored to depict pathologies or anatomical variations with diagnostic relevance [33], [40].

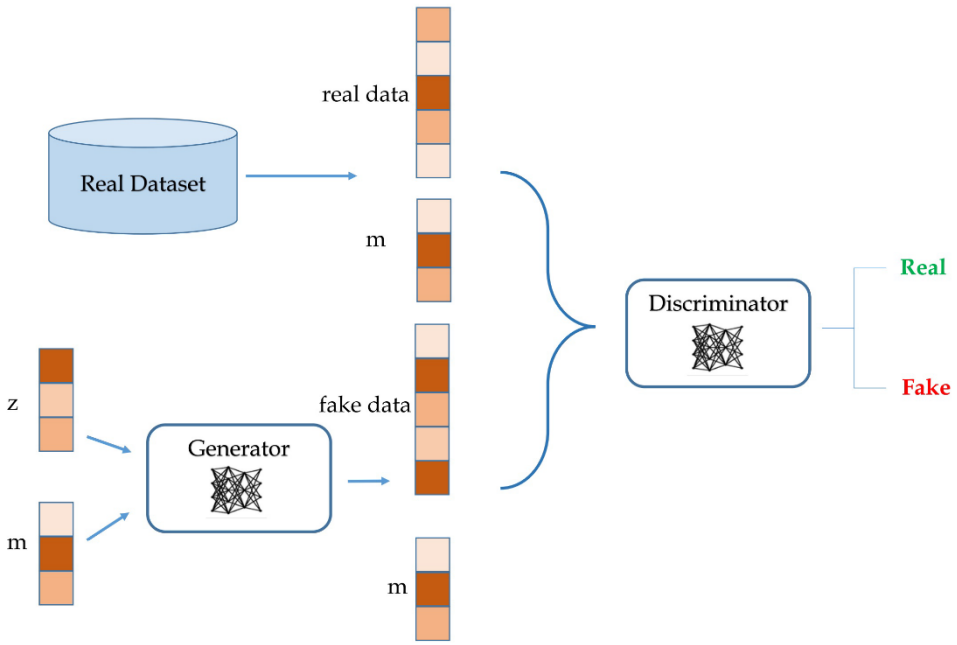


Figure 19: *cGAN architecture. The label vector m is provided as extra information to both the generator and the discriminator. [40]*

1.2.3.6 Image-to-Image Translation: CycleGAN and Pix2Pix

CycleGAN introduced an architecture capable of learning mappings between two unpaired domains, using cycle-consistency loss to enforce that translating an image to the target domain and back to the original should yield a result similar to the input *(Fig. 20)*. This has been applied in cross-modality image translation, such as converting MRI to CT, or in improving image quality through denoising and contrast enhancement without paired datasets [33].

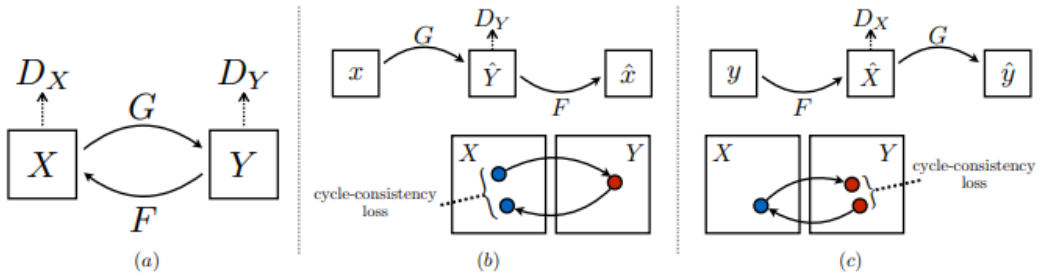


Figure 20: *Training mechanism of CycleGAN. Generators G and F translate between domains X and Y, with cycle-consistency loss guiding the reconstruction of the original input from its translated form. [33]*

Pix2Pix, in contrast, relies on paired datasets to learn direct mappings between structured inputs and their photorealistic counterparts *(Fig. 21)*. [33].

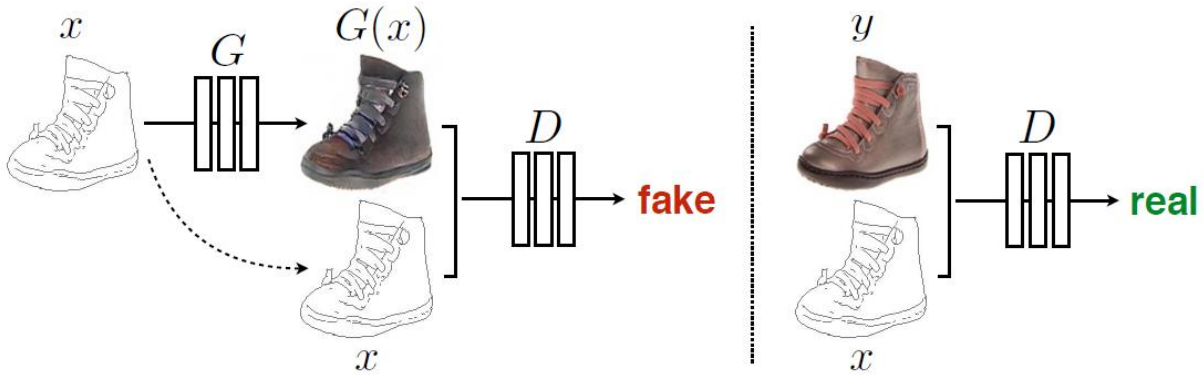


***Figure 21**: Pix2Pix architecture. The generator learns to translate input sketches (x) into realistic images (G(x)), while the discriminator distinguishes between generated pairs (x, G(x)) and real pairs (x, y). [33]*

1.2.4 Image Super-Resolution and Clinical Applications

Super-resolution (SR) encompasses a set of techniques designed to derive a high-resolution (HR) image from one or more low-resolution (LR) inputs. Low-resolution images are often the result of intrinsic limitations of the acquisition devices, motion-induced blur, optical distortions, or aggressive image compression protocols. The primary objective of SR methods is to enhance the overall perceptual quality of degraded images, reconstruct textures, and restore lost spatial details [41].

Traditional SR methods use interpolation techniques to approximate new pixel values using information already present in the image.

The most commonly used techniques are:

- **Nearest neighbor interpolation**: For each target pixel location in the high-resolution grid, the algorithm assigns the value of the closest pixel from the original low-resolution image. The method requires minimal computational resources and is frequently applied in real-time applications or hardware-constrained systems. However, its naivety often results in block-like artifacts and abrupt transitions, making it unsuitable for tasks that require visual fidelity or anatomical precision [41], [42].

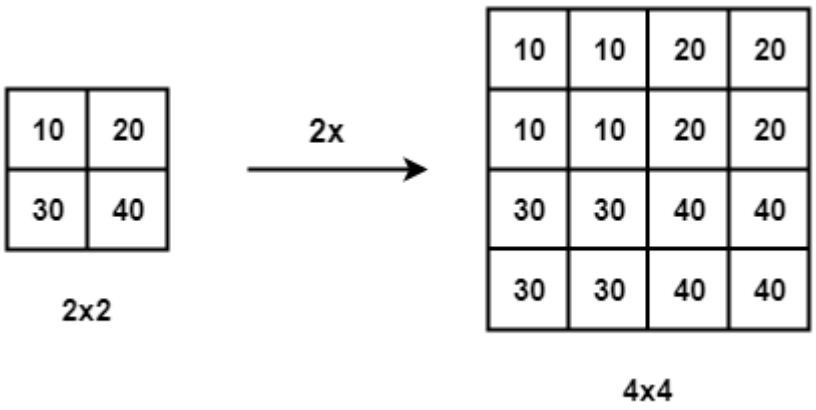


Figure 22: *An example of nearest neighbor interpolation. [42]*

- **Bilinear interpolation**: It is a two step process that involves a linear combination of the four nearest surrounding pixels to estimate each new pixel value. It performs interpolation first along one axis and then along the other, producing smoother transitions and reducing aliasing effects. It offers smoother transitions than nearest-neighbor but struggles with preserving sharp edges and fine structures [41], [43].

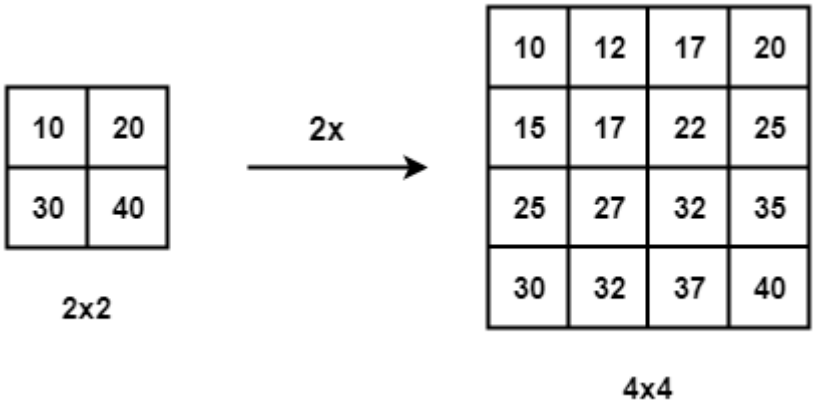


Figure 23: *An example of bilinear interpolation. [42]*

- **Bicubic interpolation**: More sophisticated than the previous two, bicubic interpolation calculates new pixel values using the 16 nearest neighbors arranged in a 4×4 grid. It applies cubic convolution to both horizontal and vertical axes, allowing for a smoother and more continuous reconstruction. Bicubic methods tend to preserve edge gradients better and generate fewer visual artifacts. Although computationally more demanding, they are often preferred in applications where reconstruction quality is prioritized over speed. Notably, bicubic interpolation with anti-aliasing remains a standard preprocessing step in the construction of synthetic super-resolution datasets, where high-resolution

images are intentionally degraded to low-resolution versions for supervised training [43].

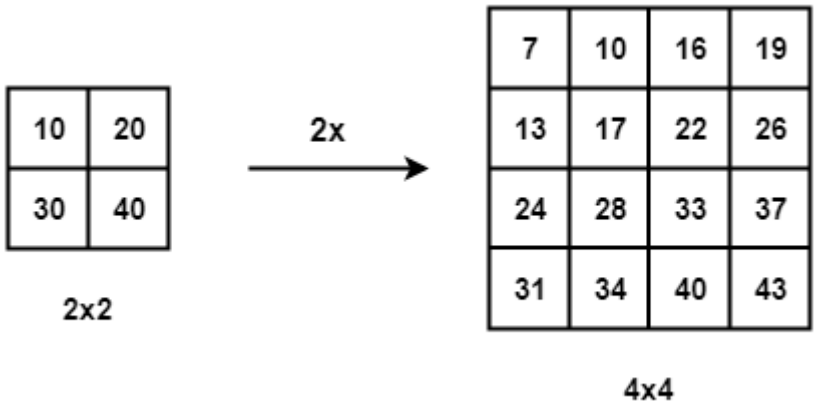


***Figure 24**: An example of bicubic interpolation. [42]*

However, these interpolation methods are inherently limited by their inability to restore lost high-frequency information. To overcome these constraints, learning-based techniques have emerged as a more effective alternative.

1.2.4.1 Learning-Based Super-Resolution Methods

Deep learning has significantly improved the ability to reconstruct high-quality images from degraded inputs through its integration into super-resolution frameworks. The Super-Resolution Convolutional Neural Network (SRCNN) was one of the initial contributions, showing that learning-based models might exceed conventional interpolation by improving pixel reconstruction using convolutional layers [44].

As a result of subsequent advancements, architecture became more complex, incorporating mechanisms such as residual blocks, attention modules, and perceptual loss functions. These improvements allow models to preserve texture continuity and structural integrity, particularly in fields like medical imaging.

1.2.4.2 GAN-Based Super-Resolution: The SRGAN Model

The Super-Resolution GAN (SRGAN) model introduced by Ledig et al. (2017) marked a turning point by leveraging adversarial training to generate high-quality, photo-realistic outputs. Unlike conventional CNN-based models, SRGAN uses a discriminator to assess the realism of the generated HR image and a generator trained to fool the discriminator by producing increasingly realistic reconstructions [45].

In addition to adversarial loss, SRGAN incorporates a perceptual loss computed in a feature space defined by a pretrained network, which enables the model to prioritize perceptual fidelity over pixel-wise accuracy. This approach helps the model reproduce textures and boundaries in a way that better matches how clinicians interpret visual information, which is crucial when medical decisions depend on small anatomical irregularities [45].

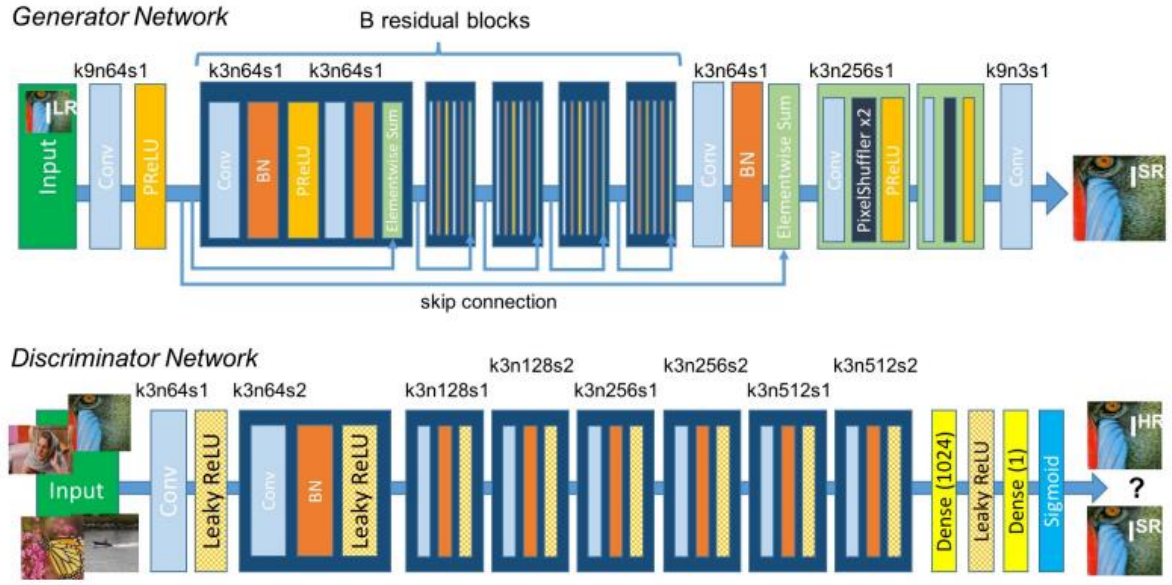


Figure 25: *Architecture of SRGAN. The generator upsamples a low-resolution image through residual blocks and sub-pixel convolutions, while the discriminator distinguishes between real and generated high-resolution output. [45]*

Subsequent extensions of this approach, including ESRGAN (Enhanced SRGAN) and Real-ESRGAN, further refined this architecture by introducing residual-in-residual blocks, dense connectivity, and more stable loss functions. These models improve the sharpness and robustness of reconstructions, even when trained on heterogeneous or noisy datasets [46], [47].

1.2.4.3 Clinical Relevance and Applications

High-resolution imaging is essential for the detection and characterization of pathological features in clinical workflows. However, imaging technologies frequently operate under limitations imposed by technical specifications or regulatory constraints on radiation exposure. Super-resolution techniques represent a methodological advancement that enhances visual quality, supporting more informed and precise diagnostic assessments.

1.2.5 The Real-ESRGAN Model

The Real-Enhanced Super-Resolution Generative Adversarial Network (Real-ESRGAN) is a state-of-the-art deep learning framework developed to perform image

super-resolution on real-world, low-quality images [47]. While classical SR approaches often assume that input images are degraded through known processes (e.g., bicubic downsampling), such assumptions rarely hold in real-world situations where image corruption is unpredictable and multifactorial.

Real-ESRGAN addresses this limitation by introducing a high-order degradation model that better simulates practical degradations that can occur in the real world, as well as a GAN-based architecture that allows the system to recover realistic textures by suppressing noise and artifacts [47].

The model is particularly relevant in contexts where user-acquired images often suffer from diverse forms of degradation due to hardware limitations, environmental lighting, compression, or camera movement. Unlike earlier methods constrained by synthetic degradation assumptions, Real-ESRGAN is trained on images that undergo a more complex degradation pipeline, making it robust to a variety of input conditions encountered in non-controlled environments [47].

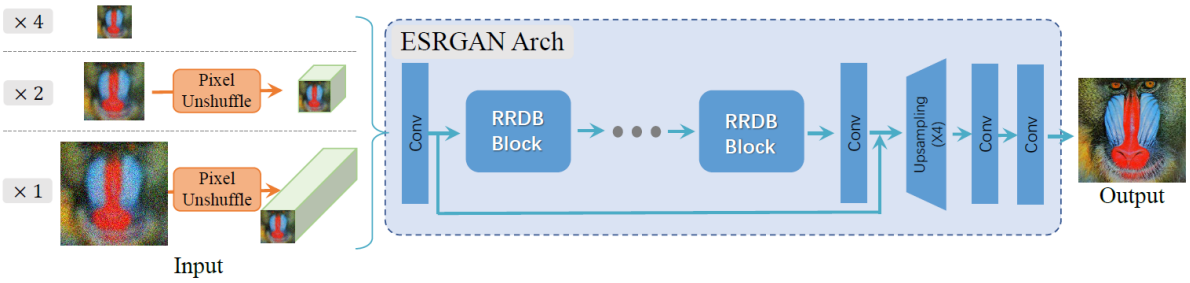


Figure 26: *The Real-ESRGAN generator adopts the same architecture as ESRGAN, featuring residual-in-residual dense blocks and upsampling through pixel-shuffle layers. [47]*

1.2.5.1 Underlying GAN Framework

Real-ESRGAN extends the foundation established by ESRGAN by refining both the architecture and the training methodology to improve robustness and applicability in real-world scenarios. Similar to ESRGAN, it uses a deep generator architecture based on residual-in-residual dense blocks (RRDB). This architecture enables efficient feature extraction and stable training without the need for batch normalization layers [46]. Unlike its predecessor, which was trained on images degraded exclusively via bicubic

downsampling, Real-ESRGAN incorporates a more comprehensive degradation model. As a result, the model is better equipped to generalize beyond idealized scenarios [47].

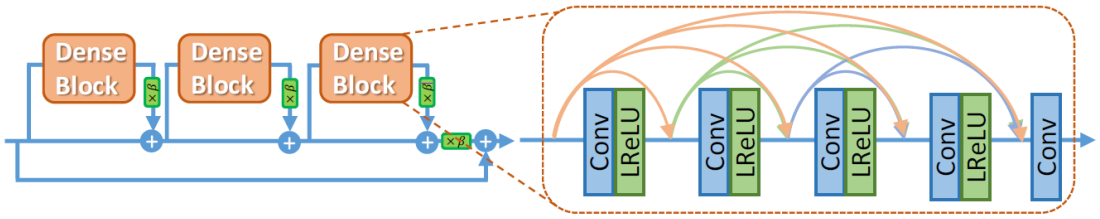


Figure 27: *Structure of a Residual-in-Residual Dense Block (RRDB), composed of three dense blocks with residual and skip connections. [46]*

1.2.5.2 Degradation Modeling and Training Process

A core innovation of Real-ESRGAN lies in its realistic degradation pipeline, which replicates complex and compound forms of image degradation *(Fig. 28)*. Instead of relying on simple downscaling kernels, the authors introduced a two-stage degradation model involving random blur kernels, additive Gaussian noise, JPEG compression, and downsampling by non-ideal filters [47]. This process ensures that the generator learns to reverse degradations that are more representative of those encountered in unconstrained real-world images.

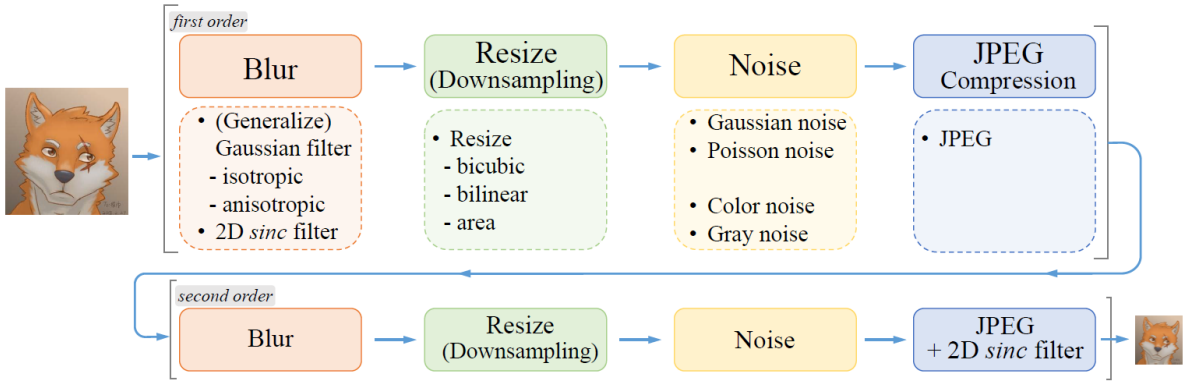


Figure 28: *Degradation pipeline used for training the Real-ESRGAN model. [47]*

The training of Real-ESRGAN is structured in two phases. Initially, a PSNR-oriented generator is trained using only the L1 loss to ensure pixel-wise reconstruction fidelity. This pre-trained model is then used to initialize the generator of Real-ESRGAN, which is subsequently fine-tuned using a combination of L1 loss, perceptual loss and GAN loss. This strategy balances numerical accuracy with perceptual realism, ensuring that fine details and textural information are preserved.

1.2.5.3 Model Architecture and Evolution

The first versions of Real-ESRGAN were primarily built on the RRDB architecture, which proved highly effective for capturing texture details and spatial patterns in high-resolution image synthesis [46].

However, the more recent versions of Real-ESRGAN, particularly those intended for real-time deployment, adopt a lightweight architecture known as SRVGGNetCompact. This architecture, introduced in the official Real-ESRGAN GitHub repository, simplifies the network by performing upsampling only in the final layer, and avoiding convolutions in the HR feature space. This reduces memory usage and computational overhead [48].

While RRDB-based models offer superior performance in terms of fidelity on high-end machines, SRVGGNetCompact allows real-time inference without substantial sacrifice in image quality, making it a practical choice for applications on smartphones and embedded devices.

1.3 Frameworks for Mobile Deployment

The integration of deep learning models into mobile and embedded systems poses a number of challenges that differ significantly from those faced in cloud-based or desktop environments. Unlike traditional computing platforms, mobile devices are subject to significant limitations in terms of memory availability, computational power, energy consumption, and thermal dissipation [49].

Furthermore, the diversity of hardware accelerators across smartphone architectures introduces considerable variability in runtime performance, often requiring platform-specific optimization solutions.

In addition to hardware heterogeneity, mobile deployment must account for real-time responsiveness, which is essential for applications such as medical diagnostics or augmented reality. Models must therefore be sufficiently compact to ensure low-latency inference, while still preserving the accuracy necessary for the task at hand. Techniques such as model pruning, quantization, and architectural simplification have been widely adopted to achieve this balance. These methods reduce the number of operations and memory footprint without significantly impairing the ability of the network to generalize.

Bandwidth availability and data privacy also play a critical role. Offloading computation to remote servers may be impractical in situations with poor connectivity or when sensitive data, such as medical images, should remain on the device. For this reason, on-device inference has gained traction as a preferred strategy, even though it entails additional design constraints [49].

The implementation of deep learning models on mobile hardware demands a multidisciplinary approach that encompasses not only algorithmic innovation but also low-level system optimization and consideration of user experience requirements.

1.3.2 NCNN Framework

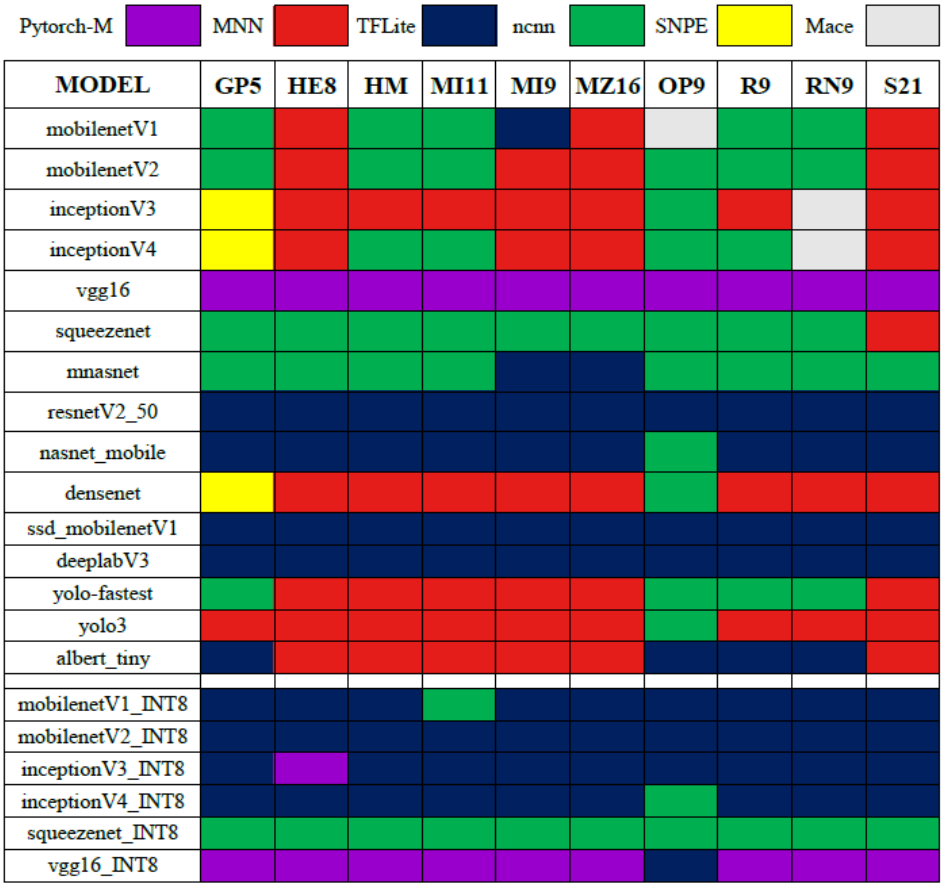
The NCNN framework is an open-source neural network inference engine developed by Tencent specifically for mobile platforms. Unlike many frameworks that rely on external dependencies or cloud offloading, NCNN is optimized with a zero-dependency design to prioritize efficiency, portability, and on-device execution [50].

A significant technical advantage of NCNN is its architectural optimization for ARM-based processors, which are the primary computing units in Android mobile platforms. The framework is specifically engineered to exploit NEON SIMD (Single Instruction, Multiple Data) instructions, an ARM extension that enables parallel processing of multiple data points within a single operation, thereby accelerating core computations such as convolutions and activation functions. In parallel, NCNN employs OpenMP, a widely adopted API for shared-memory multiprocessing, to implement dynamic multithreading strategies. This allows the inference engine to distribute workloads efficiently across multiple cores, improving throughput without sacrificing latency. The threading configuration is adaptive by default, scaling with the number of available logical processors, but it also permits manual tuning to fit heterogeneous system-on-chip designs with asymmetric core performance.

In addition to CPU optimization, NCNN includes a Vulkan backend that enables inference acceleration on supported mobile GPUs. Vulkan is a cross-platform API for high-performance graphics and compute operations, offering more explicit control over hardware resources compared to older APIs like OpenGL ES. By leveraging Vulkan, NCNN can offload computational workloads, such as convolutions and matrix multiplications, from the CPU to the GPU of the mobile device, thereby reducing latency and improving energy efficiency. Although Vulkan-based inference in NCNN is still under active development, it presents a scalable and portable solution for real-time deployment across a wide spectrum of Android devices [50].

Compared to other mobile inference frameworks such as TensorFlow Lite, MNN, or PyTorch Mobile, NCNN frequently exhibits superior performance, particularly in CPU-bound scenarios. Independent benchmarking studies have shown that NCNN achieves faster inference times on models, such as MobileNetV2 and SqueezeNet, across multiple hardware configurations [49].

In addition, NCNN simplifies the model deployment workflow by offering a specific toolchain that simplifies the conversion of models from popular formats into a streamlined representation that is suitable for mobile inference. Overall, NCNN offers a reliable and effective solution for the deployment of deep learning models on mobile hardware, while simultaneously ensuring portability and performance.



MODEL	GP5	HE8	HM	MI11	MI9	MZ16	OP9	R9	RN9	S21
mobilenetV1										
mobilenetV2										
inceptionV3										
inceptionV4										
vgg16										
squeezenet										
mnasnet										
resnetV2_50										
nasnet_mobile										
densenet										
ssd_mobilenetV1										
deeplabV3										
yolo-fastest										
yolo3										
albert_tiny										
mobilenetV1_INT8										
mobilenetV2_INT8										
inceptionV3_INT8										
inceptionV4_INT8										
squeezenet_INT8										
vgg16_INT8										

(a) CPU

Figure 29: *Benchmark of deep learning frameworks on mobile CPUs. The chart reports the framework achieving the lowest inference latency for each combination of model and mobile device. [49]*

1.3.3 ExecuTorch Framework

ExecuTorch is an embedded inference framework introduced by Meta to address the challenges associated with deploying PyTorch models on resource-constrained devices [51].

Built atop the export graph infrastructure of PyTorch 2.0, ExecuTorch marks a substantial evolution from its predecessor, PyTorch Mobile. While PyTorch Mobile requires bundling much of the full Python runtime and results in relatively large

binaries and longer initialization times, ExecuTorch introduces a static format called .pte (Portable Torch Executable). This format encapsulates the model in a highly portable, serialized structure that can be interpreted directly by the runtime with minimal overhead. As a result, developers benefit from faster application start-up times, reduced package sizes, and a more deterministic execution environment [52].

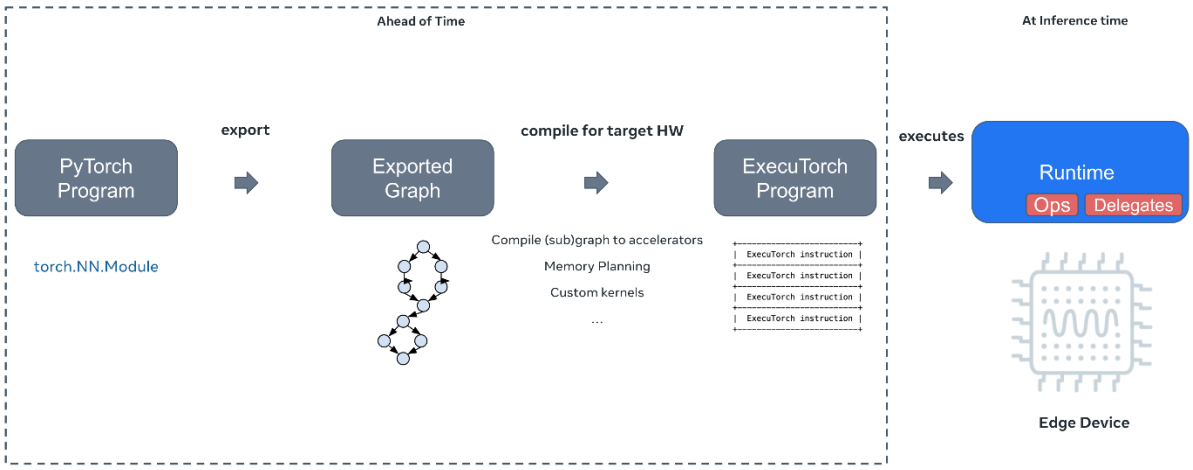


***Figure 30**: High-level overview of the ExecuTorch deployment pipeline. A PyTorch model is exported and compiled into an optimized ExecuTorch program, which is executed at inference time through a lightweight runtime on edge devices. [51]*

One of the core advantages of ExecuTorch lies in its seamless integration with the broader PyTorch ecosystem. The model export pipeline remains consistent with existing PyTorch tools, allowing developers to convert trained models into deployable assets without the need for extensive manual optimization or external conversion tools. This tight alignment between training and deployment simplifies maintenance and ensures reproducibility, especially in scenarios where models are updated frequently or must be adapted for deployment across multiple device types.

In contrast to alternatives such as NCNN, which focuses heavily on low-level optimization for ARM architectures and includes Vulkan-based GPU acceleration, ExecuTorch prioritizes ease of integration and end-to-end consistency within the PyTorch workflow [51].

Although ExecuTorch is a relatively recent addition to the PyTorch ecosystem, having been introduced in 2023, it demonstrates considerable promise for enabling the deployment of sophisticated neural networks on mobile and embedded platforms.

Materials and Methods

2.1 Dataset

The images used for this study were selected from the public archive of the International Skin Imaging Collaboration (ISIC), one of the leading repositories of dermoscopic and clinical images of skin lesions for researchers [53]. The selected images represent pigmented lesions acquired in a clinical setting using dermatoscopy. Images containing lesions with diagnostic features such as color variations and irregular borders were chosen, as these are particularly sensitive to resolution degradation and therefore suitable for evaluating the perceptual and structural improvements offered by super-resolution algorithms. Since the purpose of this study is not to train a new model but to study the feasibility of implementing the Real-ESRGAN model on a mobile device, only three images were chosen.

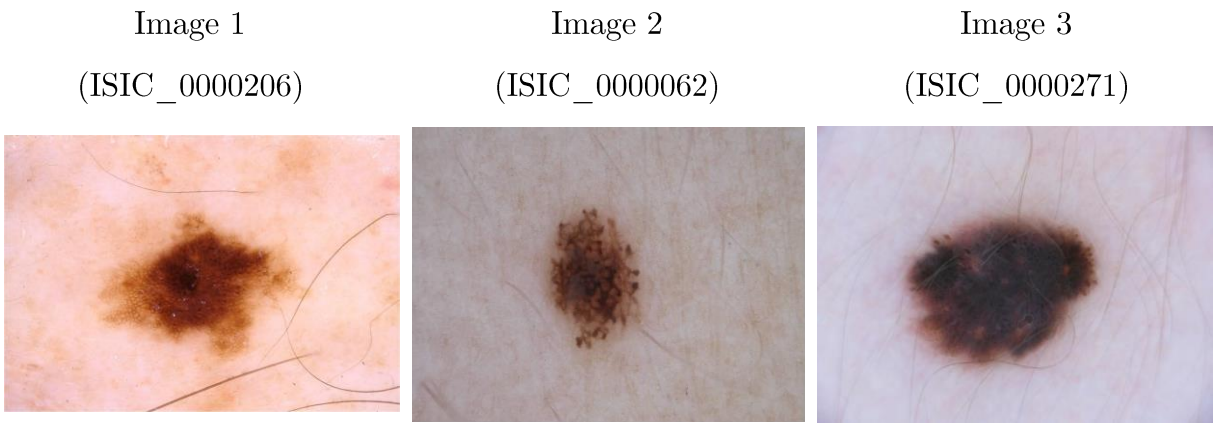


Figure 31: *High-resolution dermoscopic images selected from the ISIC archive. Specifically, the chosen images are ISIC_0000062, ISIC_0000206, and ISIC_0000271.*

In their original form, the images were available in high resolution. To simulate realistic low-quality input, downscaled and degraded versions of the images were generated using an additional degradation pipeline, which included blur and compression artifacts, to better reflect real-world acquisition conditions. This degraded input was then used for inference in all test environments.

The introduction of this pre-processing pipeline allowed the simulation of degradation types observed in practical contexts, such as loss of sharpness and texture, typically due to smartphone camera limitations or imperfect user acquisition. This approach allowed the system to be tested in an application context in line with teledermatology workflows [14].

Figure 32: *Degraded version of the dermoscopic images obtained as an output of the preprocessing pipeline.*

2.2 Image Preprocessing Pipeline

To ensure consistency across different inference environments, all input images were subjected to the exact same pre-processing pipeline, designed to simulate realistic degradations and prepare the data in accordance with the input requirements of the Real-ESRGAN model. This pre-processing was applied in each deployment environment with equivalent parameters and structure. The primary objective of this pipeline was to mimic real-world imperfections typically found in mobile teledermatology, such as blur, compression artifacts, and limited resolution, while standardizing the format for neural network input.

The complete pre-processing pipeline, shown in Figure 33, consisted of the following five steps:

- **Central Cropping**
  The first step consisted in extracting a central square region from the input image. Given that dermoscopic images often contain non-informative background areas, central cropping helped remove peripheral noise and guaranteed spatial consistency across all samples. This step mitigated the influence of peripheral background artifacts and ensured input uniformity across samples.

- **Gaussian Blur**
  Next, a Gaussian blur filter was applied to the image using a 3x3 kernel. Each pixel in the image was replaced with the weighted average of the surrounding pixels, assigning a higher weight to pixels closer to the center. This process reduced the sharpness of the image, particularly around the lesion area. As a result, it simulated the kind of blurring typically introduced by camera movement or minor focus imperfections.
- **Compression Artifact Simulation**
  Lossy compression is a commonly used technique in smartphone image uploads that introduces blocky artifacts and loss of fine details. To replicate these effects, the image underwent a two-step process. First, the blurred image was downscaled by a factor of two, using bilinear interpolation. Then, it was immediately upscaled back to its original size using the same interpolation method. This strategy imitates the lossy reconstruction commonly seen in JPEG algorithms by reducing spatial resolution and then restoring it without recovering high-frequency details.
- **Resize to Fixed Input Dimensions (224×224)**
  After applying the degradation steps, the image was resized to 224×224 pixels, which corresponds to the expected input size of the Real-ESRGAN model used in this project. This same resolution was used in every test environment to keep the comparisons consistent.
- **Normalization and Tensor Conversion**
  The final step consisted in converting the preprocessed image into a format suitable for inference. Pixel values in the RGB channels, originally in the [0, 255] range, were normalized to the [0, 1] interval by dividing each value by 255. The normalized image was then converted into a tensor with shape (1, 3, 224, 224), where the first number corresponds to the batch size, the second to the image channels, and the last two to the height and width, respectively.

By standardizing the degradation pipeline before inference, the study maintained consistent input conditions across all platforms. This approach ensured that any difference in output fidelity could be confidently linked to the backend framework rather than the preprocessing routine. At the same time, introducing imperfections that resemble real-world acquisition brought the test conditions closer to those encountered in everyday mobile health settings.

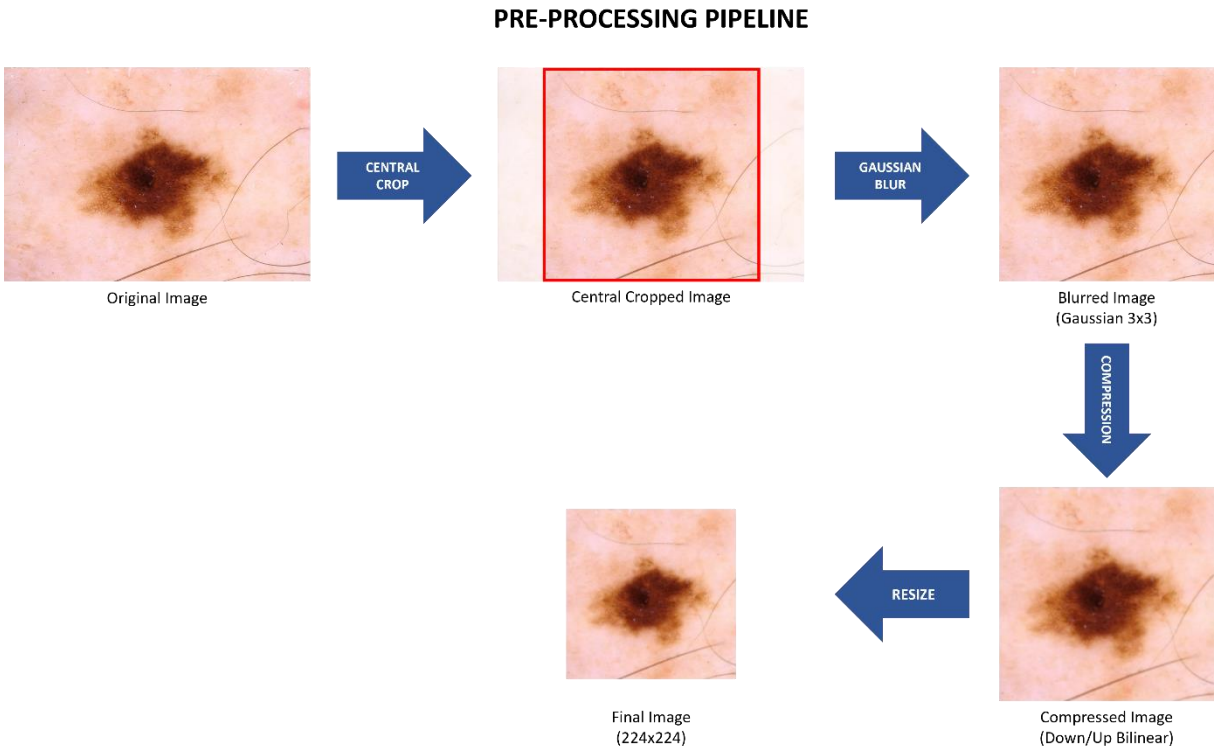


Figure 33: *Preprocessing pipeline applied to the dermoscopic images before inference.*

2.3 Architecture and Configuration of the Real-ESRGAN Model

The Real-ESRGAN model used in this project is based on the SRVGGNetCompact architecture, a compact convolutional network architecture designed for super-resolution tasks. This variant is a lightweight alternative to the original RRDB-based architecture, offering significantly reduced computational load while maintaining good output quality.

Unlike the original Real-ESRGAN architecture, this structure performs upsampling as the last operation, so no convolution is performed in the high-resolution feature space. This strategy thus reduces the computational cost of inference, making it more suitable for execution on hardware with limited computing power [48].

The network accepts a low-resolution RGB image with a size of $3\times224\times224$ as input and returns an enhanced output that is four times larger in both spatial dimensions, with a size of $3\times896\times896$. The $4\times$ upscaling factor, consistent with the standard Real-ESRGAN configuration, was maintained across all implementation environments to enable direct comparison of results and performance.

Structurally, the network begins with a single convolutional layer followed by a PReLU activation. This is followed by 16 convolutional blocks, each using a 3×3 kernel and 64

feature maps. Every block includes its own activation function. At the end of the sequence, another convolutional layer increases the number of channels before upscaling is performed using a pixel shuffle layer. A residual connection is also added by interpolating the input image using nearest-neighbor and summing it with the output. This approach encourages the model to reconstruct only the missing high-frequency details while preserving the overall structure.

Instead of training the network from scratch, this work adopted a transfer learning approach by reusing the pre-trained weights provided in the official Real-ESRGAN repository [24], [48]. Specifically, the realesr-general-x4v3 model, originally trained on generic image datasets, was chosen. Since the goal of this study is not to improve super-resolution performance, but rather to evaluate inference behavior and implementation feasibility in mobile contexts, the pre-trained generator was used as-is, without any fine-tuning. This allowed for a more accurate comparison of backend performance by eliminating variability due to different training regimes.

The exact same pretrained model was used in all backends. The PyTorch model served as the reference and was converted into NCNN format using the PNNX tool, and into ExecuTorch format using the torch.export() function followed by XNNPACK graph partitioning. This consistency ensured that any observed differences in output were due to the inference frameworks themselves, not changes in the model configuration.

2.4 Model Conversion: from PyTorch to NCNN and ExecuTorch

Two different conversion paths were followed to deploy the Real-ESRGAN model on mobile devices: one tailored for NCNN, the other for ExecuTorch. Although both originate from the same PyTorch checkpoint, the export processes differ in format specifications and integration workflows, reflecting the architectural distinctions of the respective runtimes.

The conversion of the model for NCNN deployment began with the original PyTorch .pth checkpoint of the SRVGGNetCompact generator. This file was loaded in a Python environment, and the model was traced using torch.jit.trace with a dummy input tensor of shape (1, 3, 224, 224). Tracing produces a static TorchScript graph by executing the model with a dummy input and recording the sequence of operations activated during forward propagation. The resulting TorchScript graph was then saved as a .pt file.

This file was subsequently processed using PNNX, an open-source toolkit developed specifically for exporting PyTorch models to NCNN format. The pnnx.export() function takes the traced model as input and generates two output files: realesr_general_x4v3.ncnn.param, which defines the network structure, and realesr_general_x4v3.ncnn.bin, which stores the binary weights. Together, these files constitute the NCNN-compatible version of the model [54].

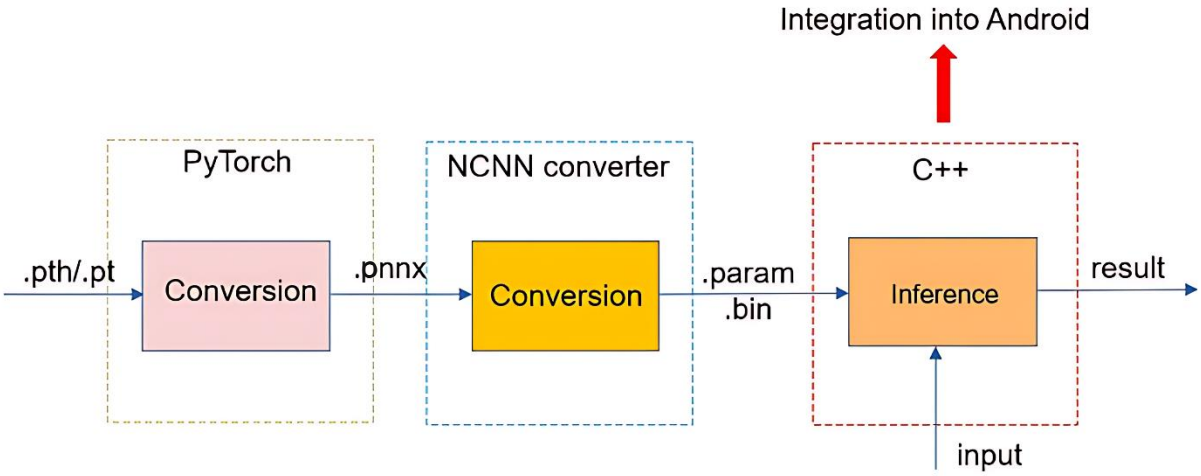


Figure 34: *NCNN conversion and deployment pipeline. [54]*

For deployment with ExecuTorch, the same pretrained PyTorch checkpoint of the SRVGGNetCompact generator was used. Similar to the process followed for NCNN export, the model was traced using a dummy input tensor of shape (1, 3, 224, 224). However, in this case, the torch.export.() function was used, which is part of the PyTorch 2.0 graph export infrastructure and is therefore designed to be fully compatible with Executorch.

Following the export, the resulting graph was passed to to_edge_transform_and_lower, a function provided by the executorch.exir module. This function is responsible for transforming and lowering the exported model graph into a backend-compatible representation suitable for embedded inference.

The conversion pipeline included a partitioning phase, in which the computation graph generated by the export process was segmented so that it could be optimized for a specific target hardware. Executorch offers several optimization backends for specific hardware. In the case of this study, the XNNPACK backend was chosen, which is specific to ARM CPUs such as those found in Android smartphones. As part of the conversion process, ExecuTorch identifies portions of the model (partitions) that are supported for the given backend. These sections are processed by the backend ahead of time to support efficient execution (Delegation). This allows for partial model acceleration when not all model operators are supported on the backend but may have negative performance implications [52].

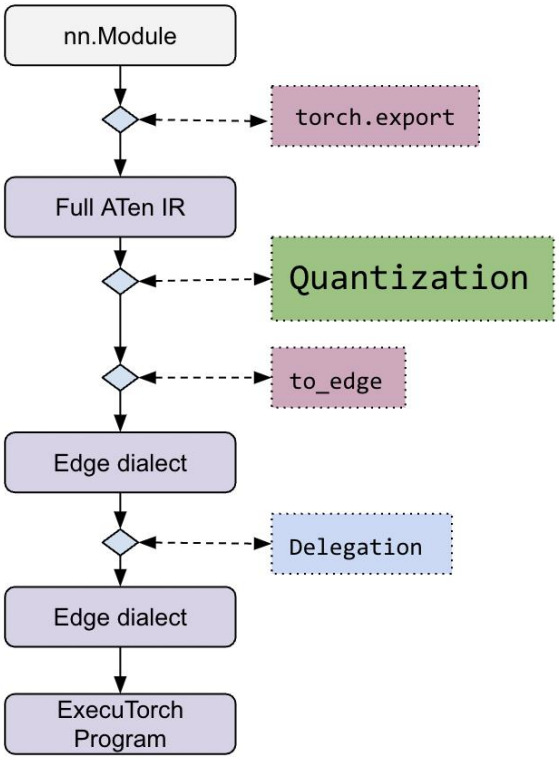


Figure 35: *Executorch conversion and deployment pipeline. [52]*

Following this stage, the model was lowered and translated into the .pte format, a binary representation compatible with the ExecuTorch runtime. This file encapsulates both the network structure and its parameters and can be loaded directly into the mobile application without further preprocessing.

2.5 Android App Development with NCNN and ExecuTorch

In order to test the performance of the Real-ESRGAN model in mobile environments, two Android applications were developed: one based on the NCNN inference engine

with native C++ integration, and the other using ExecuTorch through a Java-based runtime. Despite the different execution backends, both applications were structured to perform the full inference pipeline, starting from image loading, through preprocessing and model execution, to visualization of the super-resolved output.

In both apps, the user selects an image from the device gallery, which is then passed through the custom degradation pipeline, designed to simulate the visual artifacts commonly introduced during smartphone-based image acquisition.

2.5.1 NCNN-Based Application

In the NCNN application, image preprocessing and model execution were implemented in native C++ using OpenCV and the NCNN API. In addition, the native Java interface (JNI) was also used to connect the Android Java code to the underlying C++ inference logic. JNI allows methods to be called across the Java/C++ boundary by linking Java methods to their native counterparts compiled into shared libraries (.so files).

The structure of the Android application package (APK) is as follows:

- **/app/src/main/cpp/native-lib.cpp**

  Contains the native C++ source code, which handles image preprocessing using OpenCV, model loading, and inference execution. It also contains the JNI functions that allow Java code to access the C++ functions.

- **/app/src/main/assets/**

  Stores the model files:
  - realesr_general_x4v3.ncnn.param: model architecture.
  - realesr_general_x4v3.ncnn.bin: model weights.

- **/app/CMakeLists.txt**

  Configures the native build, linking NCNN, OpenCV, and enabling OpenMP for multithreaded inference.

- **/app/src/main/java/MainActivity.java**

  Handles user interaction and calls native methods via JNI. The final output Bitmap is passed back from C++ and displayed in the UI.

- **/app/src/main/res/layout/activity_main.xml**

  Defines the graphical layout of the app. It contains the ImageView components and the two action buttons.

2.5.2 ExecuTorch-Based Application

Unlike the application developed for NCNN, the Executorch app was implemented entirely in Java. Executorch was integrated through the AAR library distributed by the authors via Maven Central, one of the main repositories for Java-developed projects. In this way, pre-processing was handled through Android Bitmap class, while inference was handled through Executorch Java API.

The APK of the project is structured as follows:

- **/app/src/main/java/MainActivity.java**

  It contains all the logic for image loading, degradation, tensor conversion, inference, and output display.

- **/app/src/main/assets/**
  Stores the precompiled model:
  - realesrgan_x4_exported_execu.pte: serialized model in Portable Torch Executable format.

- **build.gradle**

  Includes dependencies to import the executorch-android runtime from Maven Central.

- **/app/src/main/res/layout/activity_main.xml**

  Defines the graphical layout, mirroring the NCNN version with consistent design and functionality. It contains the ImageView components and the two action buttons.

2.6 Inference Benchmarking on Google Colab and Android Devices

To evaluate the inference performance of the Real-ESRGAN model under different deployment conditions, a benchmarking protocol was implemented across three

environments: a cloud-based setting using Google Colab, and two mobile applications running natively on Android devices via NCNN and ExecuTorch, respectively. The objective of this benchmarking was to quantify latency during model execution and assess the responsiveness of each platform under real-world constraints.

In the Google Colab environment, inference was executed using the official PyTorch implementation of Real-ESRGAN. The environment was equipped with a Tesla T4 GPU, and the runtime measurements were collected using Python time module. Specifically, timestamps were recorded immediately before and after the forward pass of the generator model to calculate the execution time in seconds. This setup served as a reference for ideal-case inference conditions with no hardware limitations.

On Android, two separate applications were developed to run inference using NCNN and ExecuTorch. In the NCNN-based implementation, developed in native C++ using the Android NDK, latency was measured using the clock_gettime() function to capture nanosecond-level precision. The timing bracketed the realesrgan.forward() function, which corresponds to the core model inference. For the ExecuTorch implementation, developed in Java, the method System.currentTimeMillis() was used to measure elapsed time between the start and end of the etModel.run() function call, which triggers the execution of the Portable Torch Executable (.pte) model.

In each environment, benchmarking was performed using the same low-resolution dermatological image as input, following identical preprocessing and model loading procedures. To ensure consistency and mitigate the effect of outliers, inference was repeated five times per configuration and the mean execution time was reported. No post-processing steps such as image saving or visualization were included in the measured interval, in order to isolate the pure computational overhead associated with model inference.

Attention was also paid to differences in runtime initialization. For example, NCNN showed minimal startup overhead due to its statically linked libraries and optimized binary structure, while ExecuTorch benefitted from fast model deserialization using the .pte format, despite slightly longer Java-level initialization steps. These aspects were noted during testing but excluded from the timing measurements, which focused exclusively on the inference time.

By conducting this multi-platform benchmarking, it was possible to compare the inference efficiency of each framework in a controlled and reproducible manner. All Android tests were executed on the same device to eliminate hardware variability and ensure fairness in the comparison.

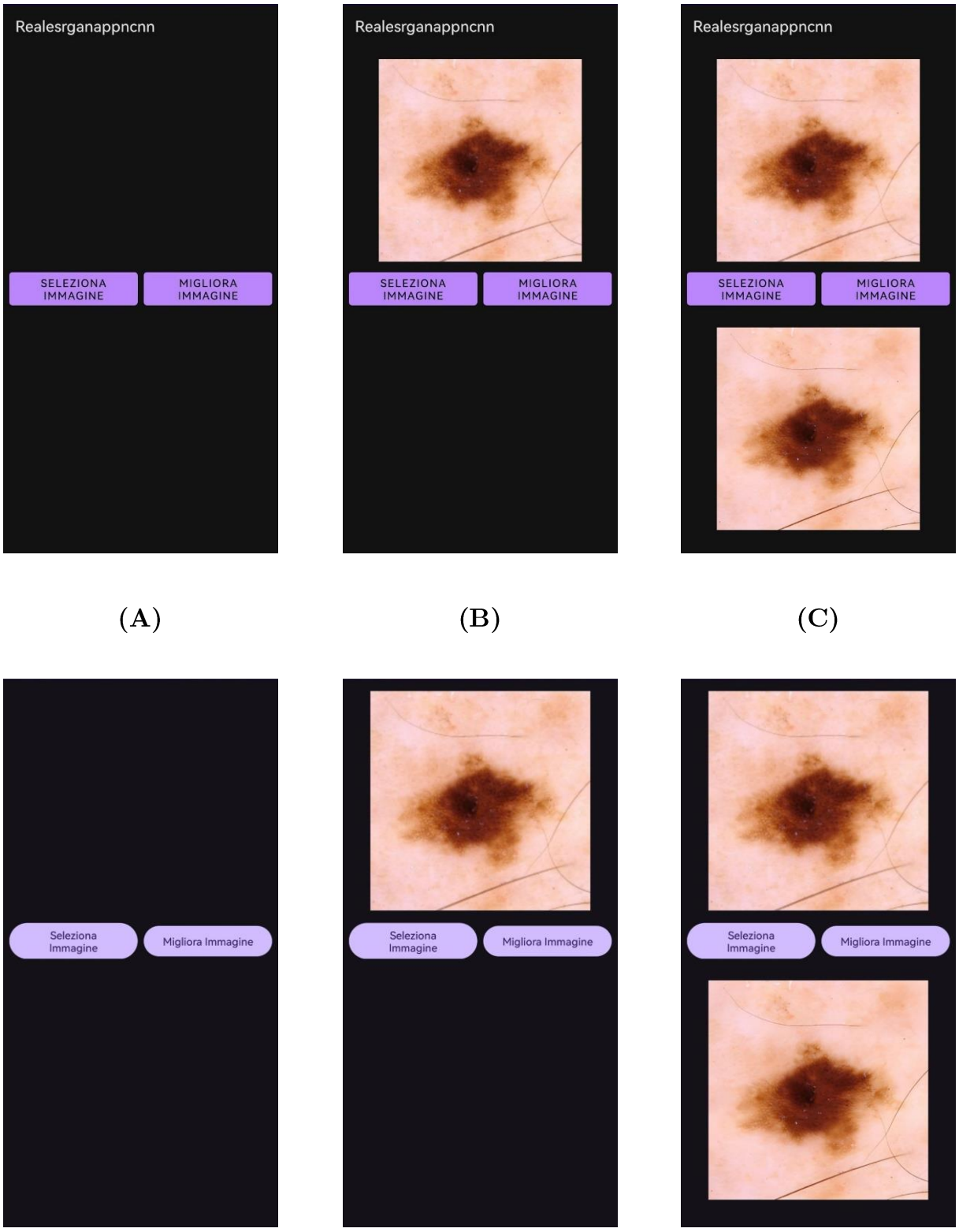


Figure 36: *User interface flow of the two Android applications (NCNN above, ExecuTorch below). Each row shows the main steps performed by the user: (A) app launch, (B) image selection, and (C) inference result after image enhancement.*

Results

3.1 Qualitative Assessment

Once inference was performed on the three images in the ISIC dataset using the three different frameworks, the outputs were displayed side by side to perform a qualitative assessment *(Fig. 37)*.

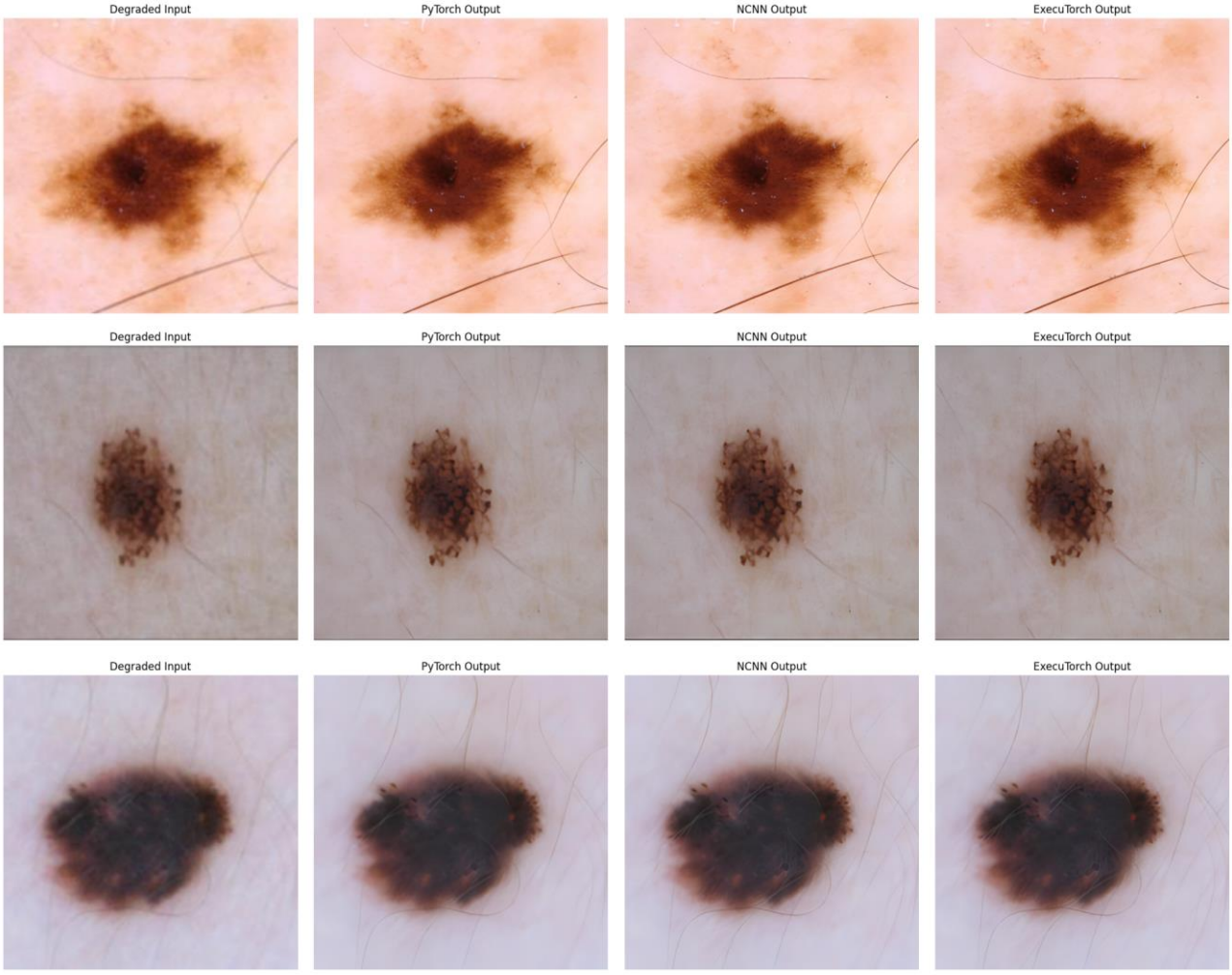


Figure 37: *Qualitative comparison of the outputs obtained across different frameworks. Each row shows the degraded input and the corresponding results for the same image.*

The output of the PyTorch network on desktop was used as ground truth to observe the results obtained on smartphones with NCNN and Executorch. Both frameworks successfully reconstructed high-resolution versions of the input images consistent with the ground truth, preserving the structure and main features of the images themselves.

Small visual differences can be observed with the naked eye, in particular the output image of the NCNN framework appears sharper, especially in the areas around the lesions and in the hairs that appear around them.

Beyond these perceptual differences, none of the results show artifacts or distortions that could compromise diagnostic usability. At this first assessment, the results generated by the two mobile frameworks are qualitatively comparable to the desktop reference. A more rigorous evaluation using quantitative metrics is addressed in the following sections.

3.2 Quantitative Benchmark: Inference Time

The first quantitative metric considered is inference time. Inference time is crucial for determining the potential use of the application by an average user: excessive inference time on mobile devices could make the application less appealing and thus undermine any good qualitative results obtained.

Therefore, to evaluate the performance of the frameworks in terms of speed, only the time taken to perform the inference was measured, without taking into account the preprocessing pipeline or the final conversion from tensor to image.

Four different frameworks were considered for this metric: PyTorch on desktop with GPU, PyTorch on desktop with CPU, NCNN on mobile, and Executorch on mobile.

This choice was made to show the performance loss that a classic desktop framework faces when switching from a GPU backend to a CPU backend. At the same time, it shows the comparison between two frameworks optimized for an Android smartphone and the desktop ground truth.

Table 1 shows the average inference times of the frameworks, calculated by averaging the inference time on each of the three images.

	Inference Time (ms)			
Image ID	PyTorch GPU	PyTorch CPU	NCNN	ExecuTorch
Image 1	3.07	2159.58	1840.03	4146.51
Image 2	3.25	1560.38	1840.66	4228.25
Image 3	3.58	1636.05	1877.15	4237.63

Table 1: *Inference time in milliseconds (ms) for each image across the three deployment frameworks. PyTorch was separated into CPU and GPU executions. Values reflect the average time required to process one image.*

As expected, PyTorch Desktop on GPU proves to be the fastest inference, always taking less than 4 milliseconds (ms) per image.

PyTorch Desktop on CPU still achieves efficient results, taking between 1500 and 2200 ms per image. However, it is important to note that when switching from GPU to CPU, the performance drops dramatically.

Among the two mobile frameworks, the one based on NCNN certainly stands out, taking an average of 1852 ms for a single inference. The least efficient framework among those analyzed is the one based on Executorch, which showed the highest inference latency. The average time required for a single inference was 4204 ms, more than double the average for NCNN.

Based on these results, several observations can be made. First, although PyTorch with CPU showed faster inference times than the most efficient mobile framework, it is important to note that the CPU used by PyTorch is an Intel Xeon CPU typically provided by Google Colab and optimized for PyTorch. The NCNN-based application, as well as the Executorch-based application, were run on a Huawei P20 Pro Android smartphone, which features a HiSilicon Kirin 970 processor with ARM architecture. The latter, in addition to not being optimized for PyTorch, is a much more compact processor than Google Colab server-class processor, which is why there is a performance gap from the outset.

Focusing instead on the comparison between the two mobile frameworks, we note that the NCNN-based application proved to be significantly faster than the Executorch-based one. This increase could be due to the relative immaturity of the ExecuTorch runtime, which is much newer than NCNN and still under development.

Although the latency performance does not match that of PyTorch, both frameworks are well suited for use on mobile devices in scenarios where real-time processing is not essential. In particular, the NCNN-based application shows inference times that are within acceptable waiting criteria for the average user.

3.3 Quantitative Image Quality Metrics

After qualitative evaluation and inference time analysis, the next step is a quantitative analysis of the outputs obtained from the different frameworks. The comparison was made between the output of PyTorch running on CPU and the outputs of NCNN and ExecuTorch running on Android. PyTorch running on GPU was not considered in the quantitative analysis as it was found to be almost identical to the output obtained with CPU, with measurable differences in the order of magnitude of 10^{-6} and therefore negligible.

The metrics chosen for this comparison are Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS).

Mean Squared Error (MSE)

MSE is one of the most traditional parameters for comparing two images, especially when you want to generate an image that has pixel colors consistent with the reference image [55]. This is because MSE measures the mean square difference between the values of the generated image and the values of the ground truth image. Therefore, we calculate the square differences pixel by pixel between the two images.

The formula for this metric is as follows:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

Where x_i and y_i are the pixel intensities of the reference image and the generated image, and N is the total number of pixels.

Peak Signal-to-Noise Ratio (PSNR)

PSNR is one of the most commonly used parameters for evaluating the quality of reconstruction in lossy image transformation techniques (e.g., image compression) [55], [56].

PSNR measures the ratio between the maximum possible power of a signal and the power of the noise affecting the fidelity of its representation. In the context of comparing a generated image and a reference image, PSNR calculates the ratio between the maximum possible pixel value and the mean square error between the two images.

In 8-bit images, the maximum pixel value is 255, and the ratio between the latter and the MSE is measured in decibels (dB). The higher the PSNR value, the higher the quality of the generated image. PSNR values above 30dB are usually associated with a generated image that reproduces the reference image with high fidelity.

PSNR has the great advantage of being a simple metric to calculate and easy to interpret. However, since it is derived from MSE and also focuses on the pixel-by-pixel difference between the two images, PSNR does not take human visual perception into account. For this reason, it is usually accompanied by other metrics

The formula is:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{\text{MSE}}\right)$$

Where MAX_I is the maximum possible pixel value (255 for 8-bit images), and MSE is the mean squared error between the reference and the test image.

Structural Similarity Index Measure (SSIM)

Unlike MSE and PSNR, SSIM is related to the quality and perception of the human visual system (HVS). Instead of focusing on the pixel-by-pixel difference between two images, SSIM attempts to approximate the perceived changes in the image structure. Specifically, SSIM focuses on a combination of three factors: correlation loss, luminance distortion, and contrast distortion [55], [56].

SSIM can be expressed as:

$$SSIM(x,y) = \left[\left(l(x,y)\right)\alpha\left(c(x,y)\right)\beta\left(s(x,y)\right)\gamma\right]$$

In this formula, l is the luminance that compares the brightness between the two images, c is the contrast that compares the dynamic range between light and dark regions of the images, and s captures the structural similarities between the two images by analyzing the local patterns of the pixels. The parameters α, β, and γ are positive weights that determine the influence of each component.

The luminance, contrast, and structure of an image can be expressed separately as:

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

In these expressions, μ_x and μ_y are the mean intensities of the two images, σ_x^2 and σ_y^2 are the variances, and σ_{xy} is the covariance. Constants C_1, C_2, and C_3 are small stabilizing terms used to prevent division by zero or numerical instability. These are usually defined as $C_1 = (k_1L)^2$ and $C_2 = (k_2L)^2$, where L is the maximum possible pixel value (typically 255), and k_1 and k_2 are small constants.

When setting $\alpha = \beta = \gamma = 1$, the SSIM formula simplifies to:

$$SSIM(x,y) = \frac{\left(2\mu_x\mu_y + C_1\right)\left(2\sigma_{xy} + C_2\right)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}$$

The SSIM score ranges from -1 to 1, where a value of 1 indicates perfect structural similarity between the two images. According to the literature, SSIM values above 0.95 are often indicative of perceptually indistinguishable images.

LPIPS – Perceptual similarity of learned image patches

LPIPS is a metric that calculates the perceptual similarity between two images. Unlike the metrics mentioned above, LPIPS uses a convolutional neural network to quantitatively evaluate the similarity between two images.

Essentially, LPIPS calculates the similarity between the activations of two image patches for a pre-trained network. In particular, the differences in feature maps are calculated and spatially aggregated. This technique has been shown to match human perception well [57].

Formally, given two images x and y, LPIPS is computed as:

$$\text{LPIPS}(x, y) = \sum_{l} \frac{1}{H_l W_l} \sum_{h,w} | w_l \odot (\phi_l(x)hw - \phi_l(y)hw)|_2^2$$

Where ϕ_l is the activation from layer l of a pre-trained network (e.g., AlexNet), $H_l W_l$ is the spatial resolution of that layer, and w_l are learned weights that adjust the importance of each channel.

In this work, LPIPS was calculated using the AlexNet backbone, as implemented in the PyTorch lpips library [58]. The value of LPIPS ranges from 0 to 1 in most implementations, with lower values indicating greater perceptual similarity between the two images. Values below 0.1 are indicative of excellent similarity.

Quantitative Evaluation Results

The PSNR values for the three images are shown in Table 2.

	PSNR (dB)		
Image ID	PyTorch vs NCNN	PyTorch vs Executorch	NCNN vs ExecuTorch
Image 1	41.30	39.56	42.42
Image 2	34.05	37.93	38.06
Image 3	45.49	40.84	42.58

Table 2: *PSNR values for each image processed by the three frameworks.*

The PyTorch output image was used as the reference ground truth and compared with the output of the NCNN- and Executorch-based apps. In addition, a comparison between the two mobile outputs was also made.

All pairwise comparisons showed PSNR values around or above 35 dB, which are usually considered indicative of high-fidelity image reconstruction.

Similarly, MSE values remain low in all comparisons.This value supports the conclusion that both mobile frameworks not only reproduce the reference image with high fidelity but also converge towards each other in terms of pixel-level reconstruction.

MSE			
Image ID	PyTorch vs NCNN	PyTorch vs Executorch	NCNN vs ExecuTorch
Image 1	4.82	7.18	3.72
Image 2	25.57	10.47	10.17
Image 3	1.84	5.36	3.59

***Table 3**: MSE values for each image processed by the three frameworks.*

The SSIM scores also remain consistently above 0.98, indicating that the generated images also accurately preserve the structural integrity of the ground truth.

SSIM			
Image ID	PyTorch vs NCNN	PyTorch vs Executorch	NCNN vs ExecuTorch
Image 1	0.980	0.980	0.983
Image 2	0.982	0.984	0.983
Image 3	0.985	0.984	0.987

***Table 4:** SSIM values for each image processed by the three frameworks.*

The LPIPS scores remain below 0.05 in all comparisons. The lowest LPIPS score is found in the comparison between NCNN and ExecuTorch (0.009), reinforcing the idea that the results of both frameworks are nearly indistinguishable from a perceptual point of view.

LPIPS			
Image ID	PyTorch vs NCNN	PyTorch vs Executorch	NCNN vs ExecuTorch
Image 1	0.042	0.044	0.009
Image 2	0.046	0.040	0.019
Image 3	0.046	0.046	0.009

***Table 5:** LPIPS values for each image processed by the three frameworks.*

Overall, the results demonstrate that, while PyTorch remains the gold standard, both mobile frameworks are capable of producing highly accurate and perceptually faithful reconstructions, even under significant hardware constraints. Moreover, the strong convergence between NCNN and ExecuTorch outputs underscores the promise of the latter, despite its relatively recent development and less mature runtime environment.

3.4 Visual Analysis of Images

A visual analysis of the images was also carried out, using pixel intensity histograms and pixel-by-pixel difference maps.

3.4.1 Histogram Comparison

Pixel intensity histograms are a graphical representation showing the distribution of pixels within an image. In particular, they plot the number of pixels present at each intensity level, thus showing information about the brightness, contrast, and color characteristics of the image.

The range of possible pixel values, in this case 0-255, is represented on the X-axis. At the same time, the Y-axis represents the number of pixels that have that specific intensity value.

To plot them, the image is first converted to grayscale and then analyzed pixel by pixel. The intensity level of each pixel is then recorded and reported in the graph.

Pixel intensity histograms were plotted for each of the three reconstructed images, comparing the outputs of Pytorch, NCNN, and Executorch in the same graph.

Image 1 shows three overlapping distributions, with all three frameworks sharing virtually identical peaks. Slight shifts can be observed in the tails of the distributions, particularly at the highest intensity levels, but these represent minimal differences.

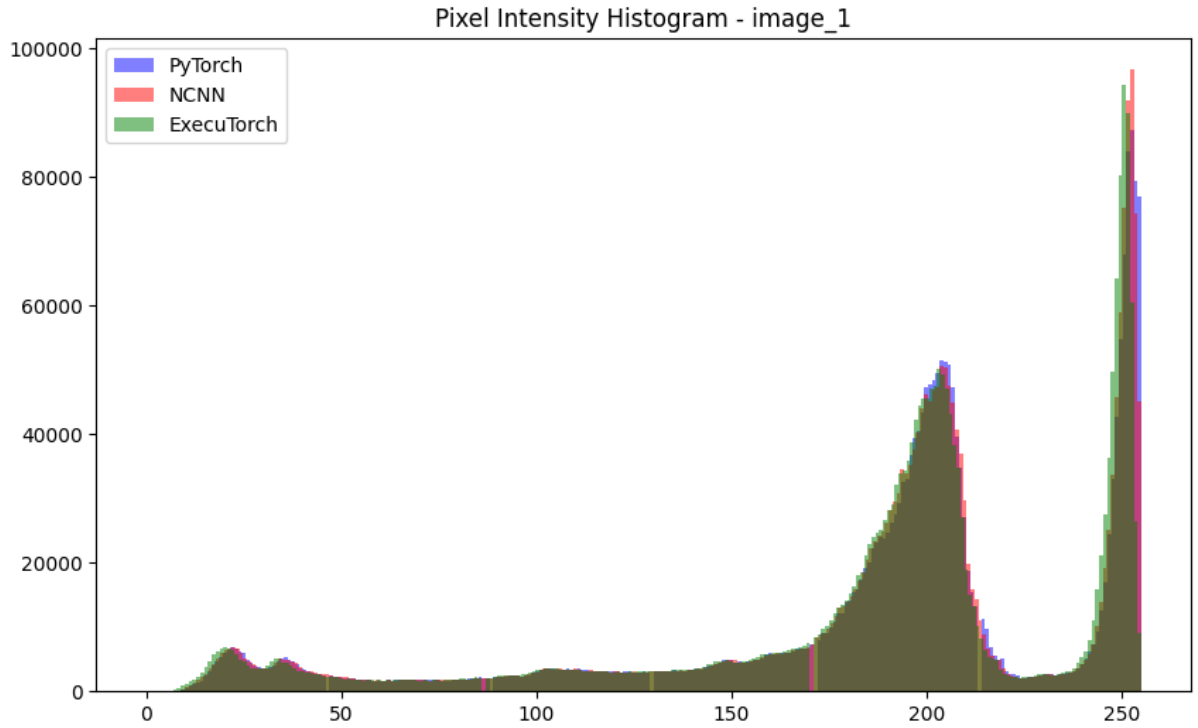


***Figure 38**: Pixel intensity histogram for image 1.*

Image 2 shows two large high-intensity peaks between values 150 and 175. Again, all three outputs reproduce the peaks with almost perfect overlap, showing an almost identical shape and structure.

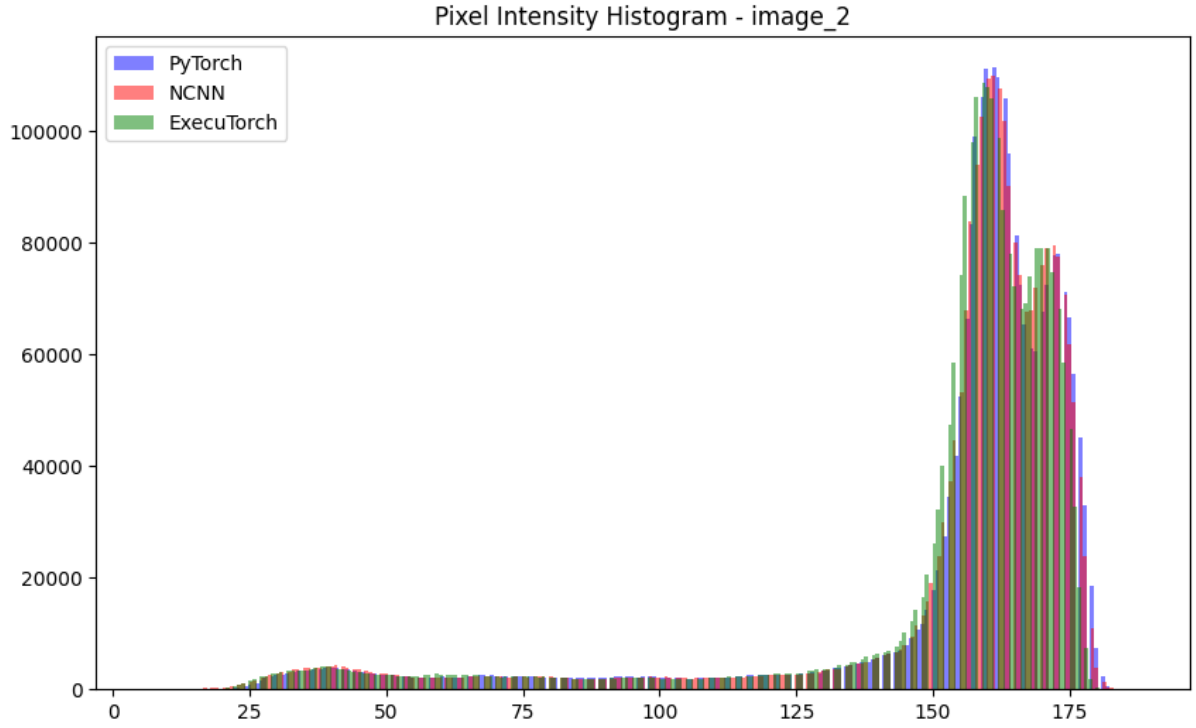


Figure 39: *Pixel intensity histogram for image 2.*

Image 3 shows a bimodal distribution, with a peak near intensity 50 and a second peak near 200. All three frameworks successfully replicate both modes, confirming the consistency observed in the previous two images.

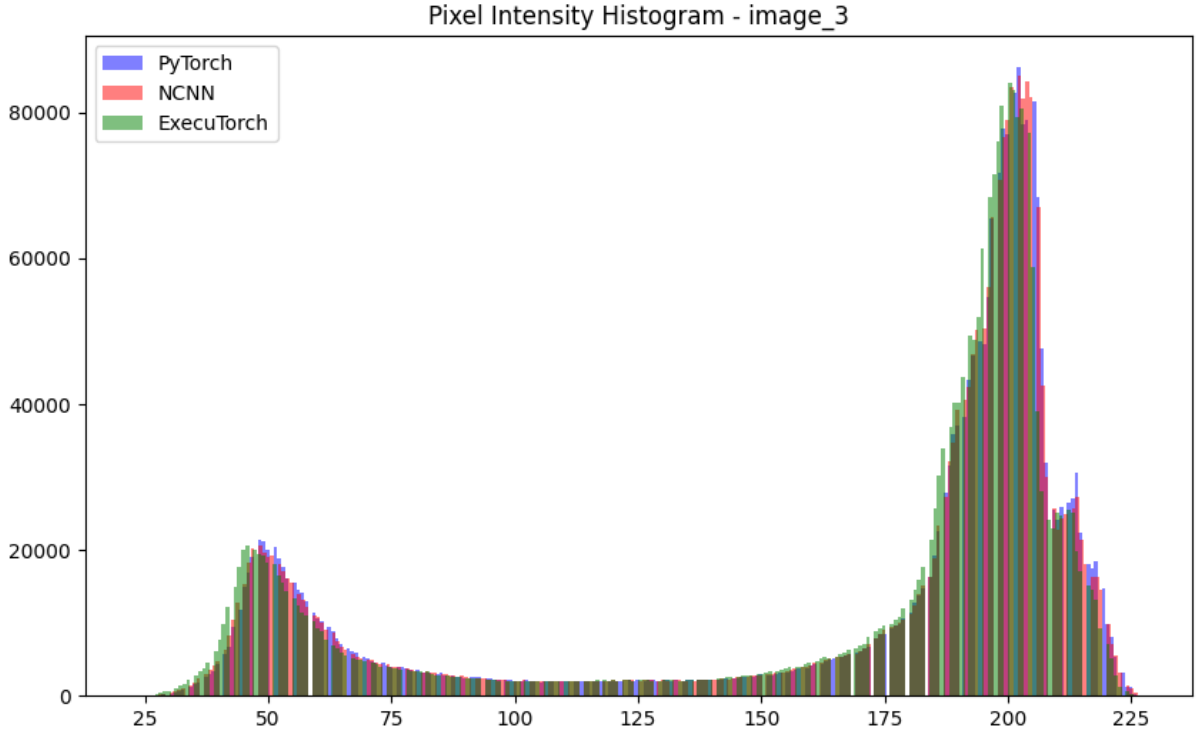


Figure 40: *Pixel intensity histogram for image 3.*

3.4.2 Pixel-by-pixel Difference Maps

Another visual analysis method is pixel-by-pixel difference maps between the two images. These maps provide information on where and how much two images differ at the pixel level.

The formula for calculating difference maps is as follows:

$$D(x, y, c) = |I_{\text{ref}}(x, y, c) - I_{\text{test}}(x, y, c)|$$

where:

$I_{\text{ref}}(x, y, c)$ is the value of the pixel at coordinates (x, y) and in color channel c in the reference image,

$I_{\text{test}}(x, y, c)$ is the corresponding pixel in the test image,

$c \in \{R, G, B\}$,

the result $D(x, y, c)$ is the absolute difference per channel, limited to the range 0-255 to ensure visual compatibility with standard 8-bit displays.

The output of this operation is a new RGB image in which each pixel encodes the magnitude of the difference between the two images at that location.

These maps were generated for three combinations of pairs: PyTorch vs NCNN, PyTorch vs ExecuTorch, and NCNN vs ExecuTorch.

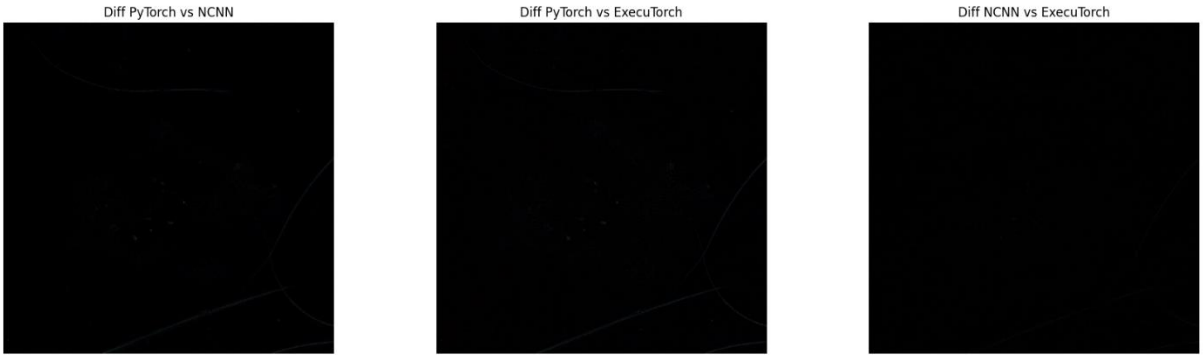


Figure 41: *Pixel-by-pixel difference map for image 1.*

Figure 42: *Pixel-by-pixel difference map for image 2.*

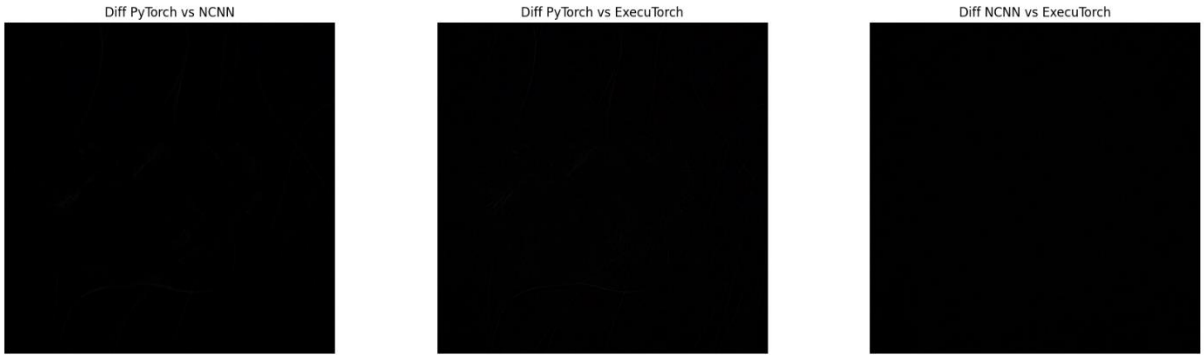


Figure 43: *Pixel-by-pixel difference map for image 3.*

The difference maps show that:

- Differences between the PyTorch gold standard and the two mobile frameworks NCNN and Executorch are minimal and mainly localized around fine structures (e.g., lesion edges or hair),
- The difference map between NCNN and Executorch appears almost completely black, a result consistent with the quantitative analysis and confirming the strong similarity between the outputs of the two frameworks.

3.5 Consistency and Reliability

In addition to qualitative and quantitative assessments, another important aspect to consider is the stability and consistency of the frameworks. The two mobile applications must ensure not only good performance in terms of image enhancement, but also in terms of consistency and reproducibility of results. Each application must ensure that the output images are stable during repeated inferences, without generating unexpected artifacts. To evaluate consistency, inference was repeated several times for each mobile framework using the same degraded inputs.

The outputs of the NCNN-based application remained stable throughout all runs. The pixel values of the generated images remained the same.

In contrast, the Executorch-based application showed significant instability. During multiple runs, the framework alternated between correct outputs and corrupted or even completely black outputs. The images often showed extreme pixel values such as $5.6\,x\,10^{19}$ or $-1.3\,x\,10^{16}$, rendering the images unusable for the purpose.

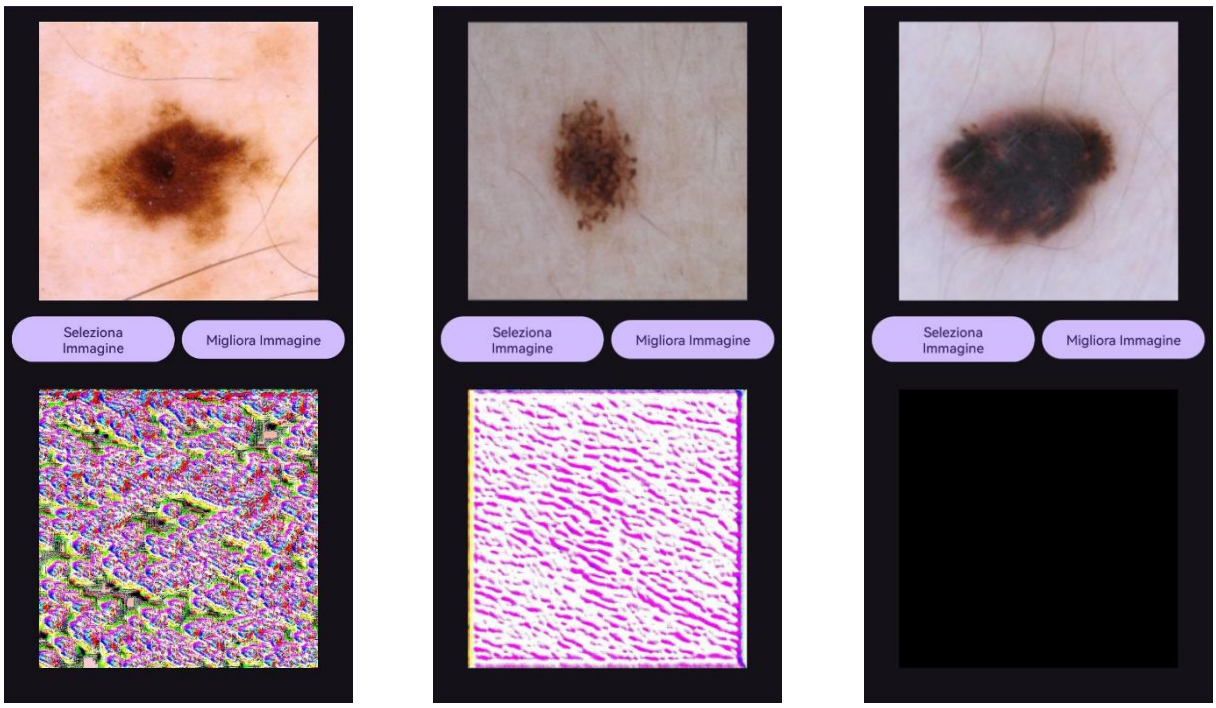


***Figure 44**: Examples of corrupted outputs generated by the ExecuTorch-based application during inference.*

Such application behavior could be explained by the difference in age between the two frameworks. The first release of NCNN dates back to 2017, and since then, several advances have been made in the development of the framework. Over time, the framework has been increasingly optimized for different hardware and backends, and its performance has steadily improved. Executorch, on the other hand, is a relatively new development, having been released for the first time in 2023. Its current version is 0.6, released in April 2025, and it is gradually improving in terms of usability and stability.

Real-ESRGAN is a complex neural network model that has not yet been implemented using Executorch, which is why the latter is probably not yet able to handle all layers of the network in a stable manner.

The NCNN-based application has therefore proven to be a stable and reliable solution for immediate deployment, ensuring excellent performance and deterministic inference execution for all images.

Conclusions

4.1 Results Analysis

This study provided a comprehensive overview of the implementation of the Real-ESRGAN super-resolution model on mobile devices using two different frameworks, NCNN and Executorch. When compared with the PyTorch ground truth, both frameworks demonstrated the ability to generate high-resolution images of comparable quality, preserving the structure and characteristics of the original input.

The differences observed between the outputs were subtle and mostly limited to localized details such as hair edges or lesion contours, without introducing any perceptual artifacts that might hinder medical interpretability.

With regard to inference time, the results revealed a clear distinction between desktop and mobile frameworks. As expected, the gold standard represented by PyTorch desktop proved to be the fastest, both on GPUs and CPUs.

In the comparison between the two mobile frameworks, NCNN consistently outperformed Executorch, with the former maintaining inference times around 1850 ms, while the latter took more than twice as long, settling around 4200 ms.

The quantitative comparison between the smartphone frameworks and the PyTorch reference confirmed the high fidelity of both mobile outputs. Both frameworks generated outputs with PSNR values above 35 dB, SSIM scores above 0.98, and LPIPS values below 0.05. Furthermore, when compared to each other, NCNN and Executorch demonstrated strong convergence of outputs, as evidenced by the value of 0.009 obtained in the LPIPS metric.

Visual analysis further confirmed the excellent results obtained with the quantitative metrics. Pixel intensity histograms showed almost complete overlap between the outputs of NCNN and Executorch and the PyTorch ground truth. Furthermore, pixel difference maps also showed minimal variations between PyTorch and the two mobile frameworks. The strong convergence between NCNN and Executorch was also confirmed by the pixel difference map of the outputs, which was almost completely black, confirming nearly identical outputs at the pixel level.

The most important issue that emerged from the tests was stability. While the NCNN-based application produced stable outputs in repeated inferences, the Executorch-based app showed several corrupted or completely empty outputs. This instability is likely due to the relatively early stage of development of the Executorch runtime, which is why the NCNN runtime currently represents a more stable and deployment-ready solution.

The results of this study therefore confirm that both NCNN and Executorch are highly viable solutions for the mobile deployment of advanced deep learning models, particularly the Real-ESRGAN model. Despite its excellent and promising performance results, Executorch still requires further refinement to ensure robustness and stability in real-world applications. At its current stage of development, the more mature and established runtime of NCNN represents a stable and reliable solution for immediate deployment.

4.2 Improvements and Future Developments

The results obtained in this study confirmed that both NCNN and Executorch offer a valid solution for implementing deep learning models such as Real-ESRGAN on mobile devices. Despite their excellent performance, there is still considerable room for improvement in several areas, as well as ideas for future developments.

System-level improvements

From a technical point of view, the most critical aspect of the implementation was the instability in inference demonstrated by Executorch. Improving the robustness and stability of the Executorch framework is certainly a priority, but given the stage of development the framework is still at, this issue could be addressed and resolved in future updates. Such updates could also lead to an improvement in inference times, another weakness shown by Executorch in comparison with NCNN.

In addition, support for higher input resolutions could be added to both applications to further improve the visual quality of the images.

Integration with classification models

A promising direction to follow after the results obtained in this study is the integration within the mobile application of a classification model that performs inference on dermatological images after their upscaling with the Real-ESRGAN model.

With this in mind, the Real-ESRGAN model could be part of a pre-processing pipeline that applies super-resolution to images that are then fed into a classifier that categorizes the lesions, distinguishing between benign and malignant, for example. There are several image classification models that are well suited for mobile implementation, such as MobileNetV3 or the more recent EfficientNet-lite.

This could result in a comprehensive application that accompanies the end user from the moment the photograph is taken to an initial screening, which could prove to be an excellent support and second opinion tool for dermatologists and healthcare professionals.

Improvements to user experience and usability

Another aspect to consider for future developments is the user experience and the graphical interface of the application. User experience and usability are key aspects to consider when developing mobile applications, especially in the case of telemedicine applications, which can appear outdated and unattractive.

To increase usability, the mobile app could incorporate features such as a progress bar showing the status of the inference, the ability to interactively compare "before and after" images, and an archive showing the history of images analyzed in the past.

Explainability and clinical integration

The term AI explainability refers to the ability to understand and interpret the decisions made by artificial intelligence models. In the clinical setting, this issue is very important in order to allow physicians to fully exploit the potential of AI tools and make the best possible decision for the health of the patient. In the application developed, the outputs obtained from any classification models could be accompanied by saliency maps or attention heat maps to help healthcare professionals understand which regions of the images most influenced the decisions of the model.

Privacy and federated learning

In the European Union (EU), the privacy of health data in telemedicine applications is regulated by the General Data Protection Regulation (GDPR), which classifies all data used by mHealth apps as "sensitive." For this reason, future versions of the application will need to ask the end user for consent to use their data and employ security measures to safeguard it from unauthorized use and disclosure.

An interesting implementation could also be the use of federated learning. This strategy would allow the model to be trained on data processed by each individual user of the application after anonymizing it. The model is improved and tuned on the device by processing the data locally, without sharing it with servers or other devices. This strategy would allow the model to continuously improve its accuracy across different demographic groups and skin types, while preserving the privacy of end users.

Scalability and implementation scenarios

A complete version of the application, ranging from image enhancement to lesion classification, could be tested in large-scale screening initiatives in a subsequent phase. Operating entirely locally on the device allows for practical implementation in contexts such as remote clinics or large-scale public health campaigns.

Testing in these contexts would make it possible to verify the real clinical impact and usefulness of such a system in supporting diagnostic workflows and improving access to dermatological care.

4.3 Conclusions

This study explored the feasibility of implementing a super-resolution model for improving dermatological images on mobile devices. In particular, the use of two different frameworks, NCNN and Executorch, allowed an advanced model such as Real-ESRGAN to be used for inference on an Android smartphone using the CPU of the device.

The study demonstrated that such implementation is not only possible but also highly effective and promising. Comparison between PyTorch on desktop and mobile applications based on NCNN and Executorch highlighted the ability of these frameworks to come close to the performance of the gold standard PyTorch, despite using a mobile CPU with lower computational power.

The mobile implementation of deep learning models, once considered impractical due to the limited resources of smartphones, now appears to be a viable solution for spreading the use of artificial intelligence-based tools in the healthcare sector, particularly in telemedicine.

Systems such as the one proposed in this study show great potential as solutions that could support physicians in their daily practice and improve access to early diagnosis for patients.

Bibliography

[1] R. J. Hay *et al.*, «The Global Burden of Skin Disease in 2010: An Analysis of the Prevalence and Impact of Skin Conditions», *J. Invest. Dermatol.*, vol. 134, fasc. 6, pp. 1527–1534, giu. 2014, doi: 10.1038/jid.2013.446.

[2] X. Du-Harpur, F. M. Watt, N. M. Luscombe, e M. D. Lynch, «What is AI? Applications of artificial intelligence to dermatology», *Br. J. Dermatol.*, vol. 183, fasc. 3, pp. 423–430, set. 2020, doi: 10.1111/bjd.18880.

[3] «Skin cancer rate U.S. by gender 1999-2021», Statista. https://www.statista.com/statistics/663943/skin-cancer-incidence-rate-in-us-by-gender/

[4] Z. Apalla, A. Lallas, E. Sotiriou, E. Lazaridou, e D. Ioannides, «Epidemiological trends in skin cancer», *Dermatol. Pract. Concept.*, vol. 7, fasc. 2, apr. 2017, doi: 10.5826/dpc.0702a01.

[5] U. Leiter *et al.*, «Incidence, Mortality, and Trends of Nonmelanoma Skin Cancer in Germany», *J. Invest. Dermatol.*, vol. 137, fasc. 9, pp. 1860–1867, set. 2017, doi: 10.1016/j.jid.2017.04.020.

[6] «Epidemiology of Skin Cancer in 2024», in *Skin Cancer - Past, Present and Future*, IntechOpen, 2025. doi: 10.5772/intechopen.1008698.

[7] «Melanoma of the Skin - Cancer Stat Facts», SEER. https://seer.cancer.gov/statfacts/html/melan.html

[8] J. Jaworek-Korjakowska e P. Kleczek, «eSkin: Study on the Smartphone Application for Early Detection of Malignant Melanoma», *Wirel. Commun. Mob. Comput.*, vol. 2018, fasc. 1, gen. 2018, doi: 10.1155/2018/5767360.

[9] B. Domingues, J. Lopes, P. Soares, e H. Populo, «Melanoma treatment in review», *ImmunoTargets Ther.*, vol. Volume 7, pp. 35–49, giu. 2018, doi: 10.2147/itt.s134842.

[10] X. Wu, M. A. Marchetti, e A. A. Marghoob, «Dermoscopy: Not just for Dermatologists», *Melanoma Manag.*, vol. 2, fasc. 1, pp. 63–73, feb. 2015, doi: 10.2217/mmt.14.32.

[11]A. Finnane, K. Dallest, M. Janda, e H. P. Soyer, «Teledermatology for the Diagnosis and Management of Skin Cancer: A Systematic Review», *JAMA Dermatol.*, vol. 153, fasc. 3, p. 319, mar. 2017, doi: 10.1001/jamadermatol.2016.4361.

[12]F. Veronese *et al.*, «The Role in Teledermoscopy of an Inexpensive and Easy-to-Use Smartphone Device for the Classification of Three Types of Skin Lesions Using Convolutional Neural Networks», *Diagnostics*, vol. 11, fasc. 3, p. 451, mar. 2021, doi: 10.3390/diagnostics11030451.

[13]A. Börve *et al.*, «Smartphone Teledermoscopy Referrals: A Novel Process for Improved Triage of Skin Cancer Patients», *Acta Derm Venereol.*

[14]S. Bunyaratavej, P. Jirawattanadon, C. Sereeaphinan, S. Wongdama, S. Sombatmaithai, e C. Leeyaphan, «Mobile Device Digital Photography for Teledermatology Consultation: Real-Life Situations», *Siriraj Med. J.*, vol. 75, fasc. 12, pp. 871–879, dic. 2023, doi: 10.33192/smj.v75i12.264488.

[15]W. Gouda, N. U. Sama, G. Al-Waakid, M. Humayun, e N. Z. Jhanjhi, «Detection of Skin Cancer Based on Skin Lesion Images Using Deep Learning», *Healthcare*, vol. 10, fasc. 7, p. 1183, giu. 2022, doi: 10.3390/healthcare10071183.

[16]S. B. Mukadam e H. Y. Patil, «Skin Cancer Classification Framework Using Enhanced Super Resolution Generative Adversarial Network and Custom Convolutional Neural Network», *Appl. Sci.*, vol. 13, fasc. 2, p. 1210, gen. 2023, doi: 10.3390/app13021210.

[17]P. Pasquali *et al.*, «Teledermatology and its current perspective», *Indian Dermatol. Online J.*, vol. 11, fasc. 1, p. 12, 2020, doi: 10.4103/idoj.idoj_241_19.

[18]J. J. Lee e J. C. English, «Teledermatology: A Review and Update», *Am. J. Clin. Dermatol.*, vol. 19, fasc. 2, pp. 253–260, apr. 2018, doi: 10.1007/s40257-017-0317-6.

[19]A. Bains, A. Alam, S. Singh, A. Budania, S. Patra, e A. Bhardwaj, «Teledermatology Services during COVID-19 Pandemic: Experience of a Tertiary Care center in Western India», *Indian Dermatol. Online J.*, vol. 13, fasc. 4, pp. 487–492, lug. 2022, doi: 10.4103/idoj.idoj_1_22.

[20]L. Tognetti *et al.*, «Teledermatology in 2020: past, present and future perspectives», *Ital. J. Dermatol. Venereol.*, vol. 156, fasc. 2, mag. 2021, doi: 10.23736/s2784-8671.21.06731-6.

[21]T. M. de Carvalho, E. Noels, M. Wakkee, A. Udrea, e T. Nijsten, «Development of Smartphone Apps for Skin Cancer Risk Assessment: Progress and Promise».

[22]A. S. Jahn *et al.*, «Over-Detection of Melanoma-Suspect Lesions by a CE-Certified Smartphone App: Performance in Comparison to Dermatologists, 2D and 3D Convolutional Neural Networks in a Prospective Data Set of 1204 Pigmented Skin Lesions Involving Patients' Perception», *Cancers*, vol. 14, fasc. 15, p. 3829, ago. 2022, doi: 10.3390/cancers14153829.

[23]A. Börve e C. Sandberg, «Mobile teledermoscopy—there's an app for that!».

[24]C. Janiesch, P. Zschech, e K. Heinrich, «Machine learning and deep learning», *Electron. Mark.*, vol. 31, fasc. 3, pp. 685–695, set. 2021, doi: 10.1007/s12525-021-00475-2.

[25]D. Shen, G. Wu, e H.-I. Suk, «Deep Learning in Medical Image Analysis», *Annu. Rev. Biomed. Eng.*, vol. 19, fasc. 1, pp. 221–248, giu. 2017, doi: 10.1146/annurev-bioeng-071516-044442.

[26]Z. Li, K. C. Koban, T. L. Schenck, R. E. Giunta, Q. Li, e Y. Sun, «Artificial Intelligence in Dermatology Image Analysis: Current Developments and Future Trends», *J. Clin. Med.*, vol. 11, fasc. 22, p. 6826, nov. 2022, doi: 10.3390/jcm11226826.

[27]Y. J. Lee, C. Park, H. Kim, S. J. Cho, e W.-H. Yeo, «Artificial intelligence on biomedical signals: technologies, applications, and future directions», *Med-X*, vol. 2, fasc. 1, dic. 2024, doi: 10.1007/s44258-024-00043-1.

[28]J. Heaton, «Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618», *Genet. Program. Evolvable Mach.*, vol. 19, fasc. 1–2, pp. 305–307, giu. 2018, doi: 10.1007/s10710-017-9314-z.

[29]*A diagram of a convolutional neural network (CNN) architecture.* 2025. https://commons.wikimedia.org/wiki/File:Convolutional_Neural_Network.png

[30]A. Esteva *et al.*, «Dermatologist-level classification of skin cancer with deep neural networks», *Nature*, vol. 542, fasc. 7639, pp. 115–118, feb. 2017, doi: 10.1038/nature21056.

[31]P. Rajpurkar *et al.*, «CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning», 25 dicembre 2017, *arXiv*: arXiv:1711.05225. doi: 10.48550/arXiv.1711.05225.

[32]I. J. Goodfellow *et al.*, «Generative Adversarial Networks», 10 giugno 2014, *arXiv*: arXiv:1406.2661. doi: 10.48550/arXiv.1406.2661.

[33]A. Dash, J. Ye, e G. Wang, «A review of Generative Adversarial Networks (GANs) and its applications in a wide variety of disciplines -- From Medical to Remote Sensing», 1 ottobre 2021, *arXiv*: arXiv:2110.01442. doi: 10.48550/arXiv.2110.01442.

[34]K. Borys *et al.*, «Explainable AI in medical imaging: An overview for clinical practitioners – Saliency-based XAI approaches», *Eur. J. Radiol.*, vol. 162, p. 110787, mag. 2023, doi: 10.1016/j.ejrad.2023.110787.

[35]V. Dumoulin e F. Visin, «A guide to convolution arithmetic for deep learning», 11 gennaio 2018, *arXiv*: arXiv:1603.07285. doi: 10.48550/arXiv.1603.07285.

[36] deepsense ai Ochman Marcin, «Region of Interest Pooling and Region of Interest Align explained», deepsense.ai. https://deepsense.ai/blog/region-of-interest-pooling-explained/

[37]S. Ram, S. Vinoth, R. N. Gopalakrishnan, A. A. Balakumar, L. Kalinathan, e T. A. J. Velankanni, «Leveraging Diverse CNN Architectures for Medical Image Captioning: DenseNet-121, MobileNetV2, and ResNet-50 in ImageCLEF 2024».

[38]N. Reddy, A. Rattani, e R. Derakhshani, «Comparison of Deep Learning Models for Biometric-based Mobile User Authentication», in *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, Redondo Beach, CA, USA: IEEE, ott. 2018, pp. 1–6. doi: 10.1109/btas.2018.8698586.

[39] J. Zhou, «SRGAN based super-resolution reconstruction of power inspection images».

[40] H. Ye, L. Liang, G. Y. Li, e B.-H. Juang, «Deep Learning-Based End-to-End Wireless Communication Systems With Conditional GANs as Unknown Channels», *IEEE Trans. Wirel. Commun.*, vol. 19, fasc. 5, pp. 3133–3143, mag. 2020, doi: 10.1109/twc.2020.2970707.

[41] Sung Cheol Park, Min Kyu Park, e Moon Gi Kang, «Super-resolution image reconstruction: a technical overview», *IEEE Signal Process. Mag.*, vol. 20, fasc. 3, pp. 21–36, mag. 2003, doi: 10.1109/msp.2003.1203207.

[42] O. S. Ci, «Image Super Resolution: A Comparison between Interpolation & Deep Learning-based Techniques to...», HTX S&S COE. https://medium.com/htx-s-s-coe/image-super-resolution-a-comparison-between-interpolation-deep-learning-based-techniques-to-25e7531ab207

[43] «Single-Image Super-Resolution: A Benchmark», in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2014, pp. 372–386. doi: 10.1007/978-3-319-10593-2_25.

[44] C. Dong, C. C. Loy, K. He, e X. Tang, «Image Super-Resolution Using Deep Convolutional Networks», 31 luglio 2015, *arXiv*: arXiv:1501.00092. doi: 10.48550/arXiv.1501.00092.

[45] C. Ledig *et al.*, «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network», 25 maggio 2017, *arXiv*: arXiv:1609.04802. doi: 10.48550/arXiv.1609.04802.

[46] X. Wang *et al.*, «ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks», 17 settembre 2018, *arXiv*: arXiv:1809.00219. doi: 10.48550/arXiv.1809.00219.

[47] X. Wang, L. Xie, C. Dong, e Y. Shan, «Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data», 17 agosto 2021, *arXiv*: arXiv:2107.10833. doi: 10.48550/arXiv.2107.10833.

[48] Xintao, *xinntao/Real-ESRGAN*. Python. https://github.com/xinntao/Real-ESRGAN

[49] Q. Zhang *et al.*, «Benchmarking of DL Libraries and Models on Mobile Devices», 6 luglio 2022, *arXiv*: arXiv:2202.06512. doi: 10.48550/arXiv.2202.06512.

[50] H. Ni e The ncnn contributors, *ncnn*. (giugno 2017). C++. https://github.com/Tencent/ncnn

[51] M. Gschwind, «LLMs Everywhere: Acceleration from Servers to Mobile Devices in the Age of Generative AI», 2024, doi: 10.13140/RG.2.2.28175.29606.

[52] «ExecuTorch Documentation — ExecuTorch 0.6 documentation». https://docs.pytorch.org/executorch/stable/index.html

[53] «ISIC | International Skin Imaging Collaboration», ISIC. https://www.isic-archive.com

[54] «Deployment of PyTorch Model Using NCNN for Mobile Devices — Part 1 | by Huili Yu | Medium». https://medium.com/@freshtechyy/deployment-of-pytorch-model-using-ncnn-bceff5d846b0

[55] Zhou Wang, A. C. Bovik, H. R. Sheikh, e E. P. Simoncelli, «Image quality assessment: from error visibility to structural similarity», *IEEE Trans. Image Process.*, vol. 13, fasc. 4, pp. 600–612, apr. 2004, doi: 10.1109/tip.2003.819861.

[56] A. Hore e D. Ziou, «Image Quality Metrics: PSNR vs. SSIM», in *2010 20th International Conference on Pattern Recognition*, Istanbul, Turkey: IEEE, ago. 2010, pp. 2366–2369. doi: 10.1109/icpr.2010.579.

[57] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, e O. Wang, «The Unreasonable Effectiveness of Deep Features as a Perceptual Metric», 10 aprile 2018, *arXiv*: arXiv:1801.03924. doi: 10.48550/arXiv.1801.03924.

[58] R. Zhang, *richzhang/PerceptualSimilarity*. (15 luglio 2025). Python. https://github.com/richzhang/PerceptualSimilarity