Deep Learning 2025

Universitat Pompeu Fabra Barcelona

Final project:

DEEP LEARNING FOR SKIN DISEASE CLASSIFICATION

Team information:

Members:
Yuxiang Jiang, 268449
Pau Reig Vaqué, 269035
Nil Tomàs Plans, 268384

Team code:
Team R

Table of contents:

1. Introduction

Skin cancer affects a significant portion of the global population and, if left undiagnosed, can lead to serious health complications. We believe that a model capable of early identification of cancerous skin diseases can help reduce long-term risks and reduce the chances of misdiagnosis.

In this report we propose a deep learning-based model capable of classifying skin diseases from skin images. Specifically, we address the task of multi-class classification using the *Multiple Skin Disease Image Dataset.*

Our main objectives are: Design and train a lightweight Convolutional Neural Network (CNN) inspired by MobileNetV2, that can be efficiently deployed in low-resource environments. Achieve the classification accuracy of the VGG16 model as described in state-of-the-art research. And minimize classification loss using different techniques such as data augmentation and transfer learning.

2. State of the Art

In recent years, skin disease classification has received significant attention due to its clinical relevance and the growing availability of skin images.

To contextualize our work within the current research, we rely on the article "*State-of-the-art skin disease classification: a review of deep learning models (Jaiyeoba et al., 2025)".*The article analyzes 74 papers published between 2019-2023, and identifies key models, evaluates their performance, and discusses some challenges in the field.

First, the most frequent architectures for skin disease classification are CNN (18), followed by ResNet-50 (9), MobileNetv2 (8), and VGG16 and Inception-v3 with (5 each). While some architectures such as ResNet-50 or Inception-v3, are known for its depth and accuracy, MobileNetv2 stands for its lightweight design, making it suitable for applications with limited resources.

Second, regarding the performance of the different papers analysed, the authors reported an average an accuracy of 86.20% across several deep learning models. This highlights both the effectiveness of these models and the need for further improvement.

Third, the authors found three emerging themes that covered most limitations. First, the authors highlighted low model performance due to small and poor quality datasets, which limits generalization.Second, they identified biases caused by imbalanced datasets. Third, they emphasized the importance of developing applications that allow real-time testing with potential users to discover practical challenges or refine evaluation strategies.

With this literature stated, our project proposes a lightweight CNN inspired by MobileNetv2 that is designed with efficiency in mind, making it suitable for environments with limited computational capacity. However, as in prior studies highlighted in the article, we face major challenges such as limited dataset size and quality. Our goal is to reach a test accuracy close to the best model that uses the same dataset: VGG16 model with 89% accuracy. The model uses transfer learning with pre-trained ImageNet weights, and has over 20 million parameters making it less viable for low resource environments, compared to Mobilenetv2. This objective is pursued while being aware of practical limitations in terms including time and resources.

3. Methodology

3.1. Data Analysis

In this section, we discuss key aspects of the dataset used in this study, including preprocessing steps, data characteristics, and the methodology for splitting the data. First of all, the dataset used is "*Deep Learning for Skin Disease Classification*", published on July 19th, 2024, on the Kaggle platform as an openly available dataset.

The dataset is generally balanced, with approximately 500 images per label. However, Vascular Lesion class is under-represented with only 290 images.
The dataset consists of nine different classes, covering cancerous lesions, precancerous conditions, and benign lesions: Melanoma, Actinic Keratosis, Basal Cell Carcinoma, Dermatofibroma, Nevus, Pigmented Benign Keratosis, Seborrheic keratosis, Squamous Cell Carcinoma, and Vascular Lesion. Additionally, the dataset includes two metadata files with additional information for some image labels. The dataset is composed of real-world images, which inherently introduces noise, such as the presence of hair, air bubbles, microscope lens silhouettes, or varying image dimensions.

To reduce visual noise from the dataset, we've applied an algorithm called DullRazor in order to delete hair from images. However, we're also losing some information when doing this type of preprocessing. More details on the impact of the algorithm will be provided in the Discussion. Additionally, we applied dimensionality reduction to a standard image size of 32x32 and 64x64 pixels. Additionally, in order to deal with the under-represented class, we modified the probabilities that each label is sampled and doubled the probability of that label class achieving a balance between all classes.

We split the dataset into 80% training, 10% validation, and 10% testing. This split ensures balanced representation across sets, facilitates effective hyperparameter tuning, and enables a reliable final evaluation of model performance.

3.2. Models and Optimization

In the first part of our experiments, we designed a lightweight CNN model inspired by the MobileNetV2 architecture.

The architecture of our implementation begins with a standard convolution, followed by a sequence of five basic blocks that consists in a bottleneck depth-separable convolutions with residuals, called Bottleneck residual block, and ends with adaptive average pooling and a fully connected linear layer for classification.
The initial convolution uses a 3x3 kernel with 16 output channels, followed by Batch Normalization and a ReLU6 activation. This first step increases the representation capacity, converting the images RGB (3 channels) to a reasonable number of channels.
As introduced before the basic building block is an inverted residual block with a linear bottleneck. These blocks recreate the MobileNetV2 process and it is based on three steps. The first one consists of a 1x1 convolution, called expansion, which increases the number of channels.
The second step consists of a 3x3 depthwise convolution, which processes the channels independently.

The last layer is a 1x1 projection or pointwise, which compresses the result back to fewer channels. In every step we apply a Batch Normalization. In contrast, we only apply ReLU6 after the expansion and depthwise layers, because in the pointwise layer we have to preserve the information as suggested in the paper.
In addition, we applied residual connections to improve gradients to propagate across multiplier layers and training stability.
The choice of ReLU6 instead of standard ReLU is due to it being more robust for low-precision computations, such as those used in mobile or embedded systems. To reduce the spatial resolution before the final classification, we have used the average pooling.

In order to train and evaluate the model we followed the following methodology:

- Test validation results in order to consider using data pre-processing or “raw” data.
- Apply data augmentation, concretely rotation and flipping to add robustness to our model, generalization and more diverse data.
- We began hyperparameter tuning with 30 training epochs and monitored validation loss. If we observe with the loss function of the validation test that it could still improve we would add 20 epochs more, and successively. Regarding learning rate, we observed that learning rate significantly influenced convergence speed and accuracy, so we experimented with multiple values to determine the optimal one. Same for the weight decay, which we needed that did not limit our weight capacity to learn, while regulating the training values. Also, for each optimizer we analyzed which performed better depending on the cases we tested its performance.
- Final test over one of the best model performance with test set to see its real performance over unseen data.

In the second part of our work, we implemented transfer learning using a pretrained MobileNetV2 model trained on ImageNet (1000 classes).
The main motivation behind this approach was to use the general features learned from a large and diverse dataset, and fine-tune them for our skin lesion classification task.
Transfer learning is especially beneficial in our case because medical image datasets tend to be smaller and more specialized. Starting from pretrained weights allows us to improve convergence speed, require less data, and potentially reach higher performance than training a model from scratch.
Additionally, we chose MobileNetv2 because of its low count of parameters ~2 million, which makes it efficient and lightweight, ideal for our environment and project. Its architecture is built on inverted residual blocks with linear bottlenecks, which helps reduce the number of operations while keeping important spatial information.

4. Experiments

In this section, we present the experimental setup, including the models tested, design decisions evaluated, and the configurations used. We also define our target model and its performance for comparison.

Our target model, VGG19, uses transfer learning with pretrained ImageNet weights and applies extensive data augmentation—including horizontal flipping, random rotations (±30º), and contrast adjustment. It has over 20 million parameters and achieves a test accuracy of ~89%.

4.1. MyEfficientCNN

In this first experiment, we tested the performance of our lightweight CNN model using raw data against the performance using preprocessed data.
We used the Dull Razor algorithm to remove hair from images. Briefly, the algorithm applies morphological operations to generate a mask identifying and removing hairs from the image.
Given that the dataset contains noise, we expected that we could improve the model performance by deleting this noise, in the form of hair. However, applying this technique has some drawbacks such as losing information.

Additionally, as will be explained in Discussion, we observed a clear overfitting on the current model. To try to solve overfitting, we propose adding data augmentation to the training set, aiming to add robustness to our model. Specifically, we applied Random Rotation over ± 15º, and Horizontal Flipping with probability 0.5.
Also, we adjusted the sampling probabilities to address class imbalance, specifically doubling the sampling rate of the under-represented “Vascular Lesion” class. With these variations, we aim to avoid overfitting, and reduce the validation loss which consequently will improve the validation accuracy.
We tested several hyperparameter configurations to improve model generalization and performance. These included:

- Optimizers: SGD with 0.9 momentum, Adam, AdamW.
- Learning rate: 0.001
- Weight decay: 10^{-4}, 10^{-5}
- Epochs: 30, 50, 70

With these configurations, we aimed to find the optimal hyperparameters that can boost the performance of our model, while taking care to not overfit the model.

After testing different combinations of optimizers and hyperparameters, we converged to two strategies that we believe could improve our lightweight CNN.
The first strategy involves dynamically adjusting the weight decay during training, starting with a lower weight decay and increasing it after epoch 40 to reduce overfitting and help convergence.
The second strategy builds on the first: we use the Adam optimizer for the initial 10 epochs for faster convergence, then switch to SGD to enhance generalization in later stages.

4.2. Transfer learning

In the second part of the project, through transfer learning experiments, we aimed to improve the outputs obtained from different configurations of MyEfficientCNN.

The steps we followed to implement the transfer learning technique were:

- Understand the documentation and procedure to load the pretrained model.
- Apply the same data augmentation methods of above.
- Test the model on the training and validation sets by unfreezing the fully connected layer and observing its performance.
- First, we performed feature extraction, freezing all 18 layers except the fully connected layer. Then, we progressively fine-tuned the model by unfreezing two layers at a time and considering the impact it had on the performance while accurately observing any signs of overfitting and controlling the number of parameters used.

- To fine-tune the hyperparameters, we followed the same procedure as above. We tested various optimizers, learning rates, and weight decay values, aiming to find a balance between leveraging pretrained knowledge and allowing the model to adapt specifically to our dataset without overfitting.
- We then evaluated the best fine-tuned model on the test set to assess its performance on unseen data.

5. Results

5.1. MyEfficientCNN

Table 1: 1st experiment, testing MyEfficientCNN model

Model definition, #epochs	**Data augmentation**	**Preprocessing**	**Optimizer**			**Loss**		**Accuracy (%)**	
			opt_type	**learn. rate**	**weight decay**	**Train**	**Val**	**Train**	**Val**
MyEfficientCNN, 30	None	None	SGD_m (0.9)	0.001	10^{-4}	0.83	1.54	71.11	50.73
MyEfficientCNN, 30	None	Yes	SGD_m (0.9)	0.001	10^{-4}	0.78	1.43	72.6	50.24

Table 2,3,4: 2nd experiment: Adding data augmentation & balance between classes

MyEfficientCNN, 30	Flip, Rotation	SGD_m (0.9)	0.001	10^{-4}	1.12	1.32	59.63	54.88
MyEfficientCNN, 50		SGD_m (0.9)	0.001	10^{-4}	0.99	1.26	63.99	51.22
MyEfficientCNN, 50		Adam	0.001	10^{-4}	0.60	1.37	78.54	58.05

MyEfficientCNN, 50	Flip, Rotation	SGD_m (0.9)	0.001	10^{-5}	0.98	1.29	65.36	54.88
MyEfficientCNN, 70 (64x64)		SGD_m (0.9)	0.001	10^{-5}	1.02	1.27	63.65	56.34
MyEfficientCNN, 50		Adam	0.001	10^{-5}	0.70	1.41	75.22	55.12
MyEfficientCNN, 50		AdamW	0.001	10^{-5}	0.65	1.33	76.90	57.56

MyEfficientCNN, 70	Flip, Rotation	SGD_m (0.9)	0.001	10^{-5} and 10^{-4}	0.93	1.28	65.97	57.81
MyEfficientCNN, 50		Adam+SGD(0.9)	0.001		0.82	1.13	70.50	60.73

Figure 1: Graphical representation of both loss and accuracy functions for train and test sets

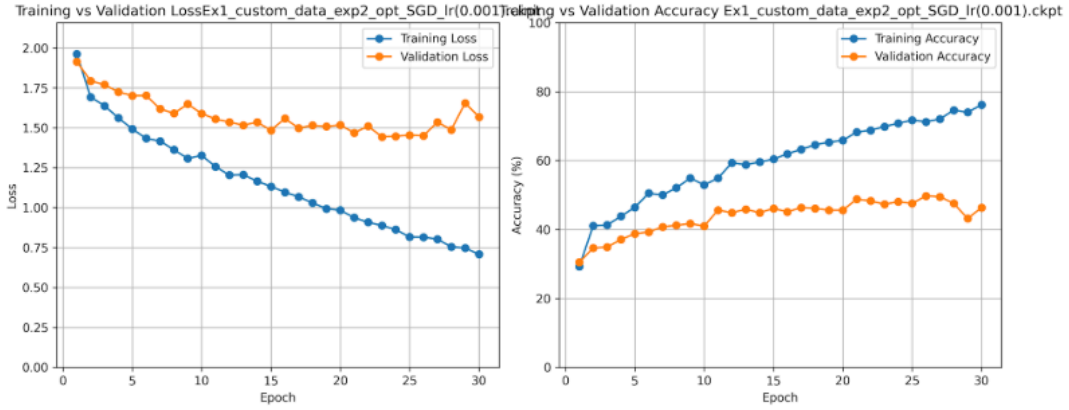


Figure 2: Graphical representation of loss and accuracy functions, 2nd case Table 2

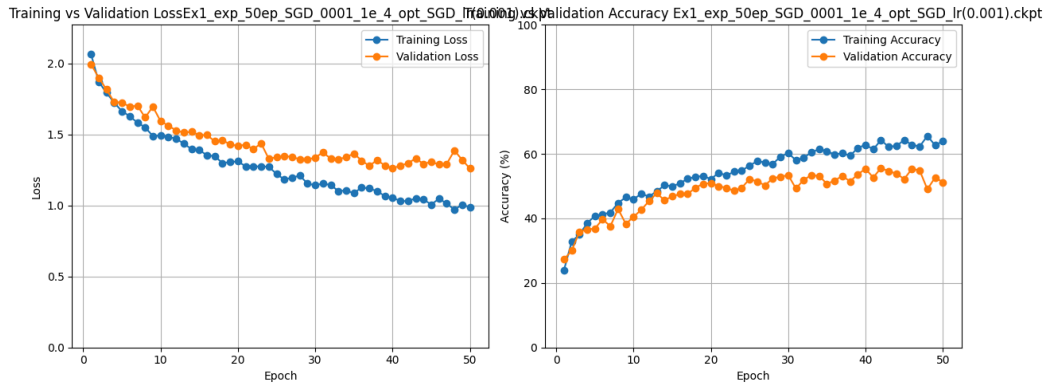
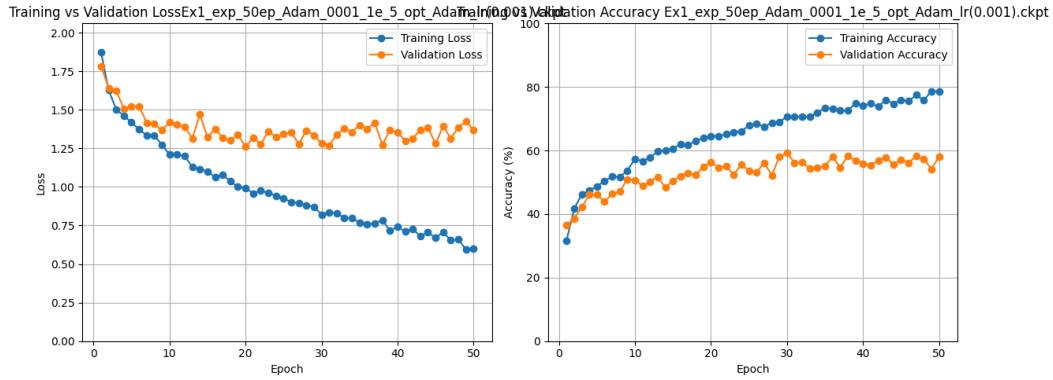


Figure 3: Graphical representation of loss and accuracy functions, 3rd case Table 2.



5.2. Transfer learning

Table 5: Table with results of different experiments with flip and rotation (data augmentation), Adam optimizer, learning rate = 0.001 and weight decay 10^{-5}.

Experiments: #epochs and layer/s unfreezed	Loss		Accuracy (%)		Avg. Recall (%)
	Train	Validation	Train	Validation	
Feature extraction: 15 epochs, fc	1.30	1.30	56.07	57.32	56.82
Fine-tune 1: 15 epochs, 2 layers	0.54	1.10	81.61	64.39	64.88
Fine-tune 2: 15 epochs, 6 layers	0.16	0.87	95.15	73.65	71.92
Fine-tune 3: 10 epochs, 8 layers	0.25	0.83	92.1	73.41	73.03

Figure 4: Plots of loss and accuracy functions for feature extraction experiment

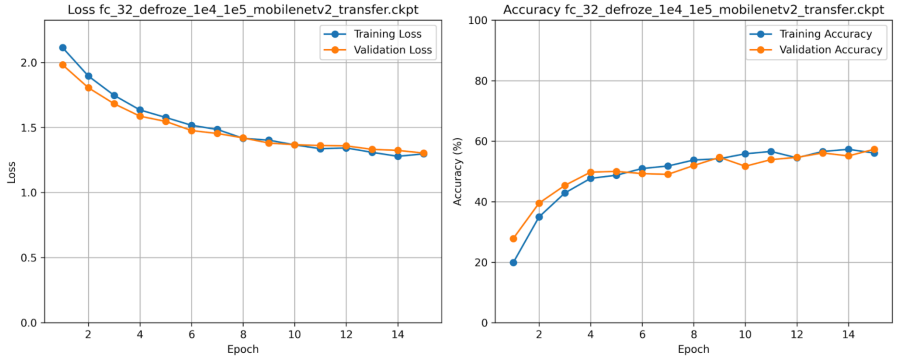


Figure 5: Plots of loss and accuracy functions for Fine-tune 3 experiment

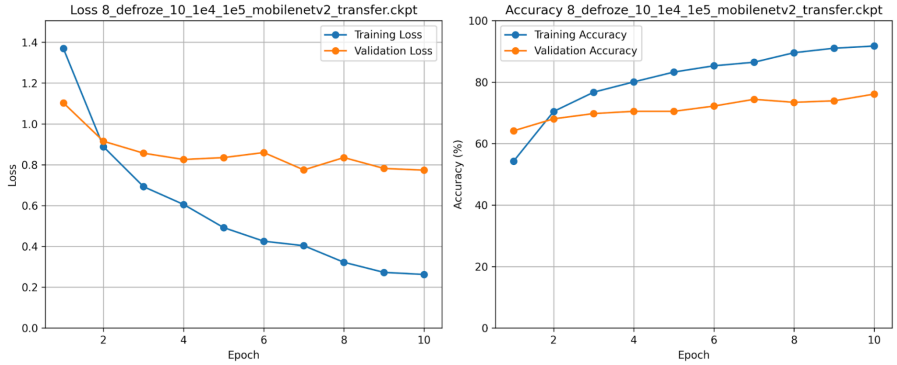
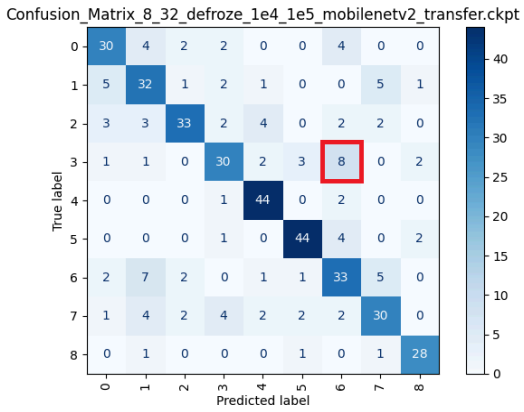


Figure 6: Confusion matrix test set with Fine-tune 3 hyperparameters



6. Discussion

All results will be discussed in this section.

6.1. MyEfficientCNN

After observing the results obtained from the first experiment described above, Table 1, we can clearly state that there is no significant difference on using raw data, and data preprocessed by the algorithm Dull Razor due to both test loss and accuracy only differ by some decimals. Additionally, after implementing the algorithm to the data, we observed that some images still contained hair, which made us question the effectiveness of the algorithm in certain images, particularly when the hair is lighter in color, affecting the algorithm overall performance. Therefore, the raw data will be used for the following experiments and cases.
Moreover, the results clearly show overfitting of the current model, likely due to the dataset size and lack of data leading to being unable to generalize well to different data.

After stating and observing overfitting, we tested several scenarios in order to find the best way to boost model performance, by analyzing Tables 2- 4 and Plots 2-4.
For each of the scenarios we applied data augmentation and we clearly saw the positive impact it had on the performance, by reducing loss values and increasing accuracy, as the model improved its robustness.

For Table 2, we can clearly observe overfitting with the Adam optimizer case, also by analysing plot 3, where clearly train functions are improving while the validation functions are getting stacked at 55% accuracy and 1.35 at loss.
However, when using the optimizer Stochastic Gradient Descent (SGD), both loss and accuracy curves tend to follow the same shape while not differentiating too much,reaching a validation loss of 1.26 and 51% validation accuracy for 50 epochs. Progressively the model improves, but both train and validation seem to get stuck, possibly indicating that the model is not deep enough to capture all dependencies and characteristics.

Regarding Table 3 we aimed to improve the results obtained in Table 2 by reducing the weight decay, by which the value could be restricting the model complexity to learn. We observed some improvement on the SGD with momentum case, increasing by 3 points the accuracy, but the loss function would still not decrease, leading to having to search for other strategies to solve it. Regarding other cases on the table, Adam continued to overfit, as well as other optimizers such as AdamW. Additionally, we tested SGD with momentum on images resized to 64×64 and observed slightly better results, although training was more time-consuming—approximately 2 minutes longer per epoch compared to 32x32 images.

From Table 4, we can observe an improvement in terms of loss and accuracy of both experiments, which seems to have improved the model performance, while still getting stuck with minimal improvement over epochs. We obtained 1.13 on loss validation and over 60% accuracy on validation set. Those symptoms, as we said before, could be due to the model architecture's lack of depth and data, a constraint we were aware of from the beginning.

Finally, experiments in which additional depth was added to the model using more Inverted Residual blocks only led to further overfitting. This was surprising, as we expected that a deeper architecture would enhance the model’s capacity to learn meaningful features. Using

the best model obtained through all the experiments, we assess its performance in a test set and find out that test results and val results are similar, which reveals that the model is reliable.

6.2. Transfer learning

Analyzing the results from Table 5 and Figures 4–6, we can clearly assess the relevance of transfer learning compared to our custom model, as well as the importance of the number of layers unfrozen.

In the first experiment, we used feature extraction as a starting point to explore which hyperparameter combinations could lead to the best performance. As previously explained, we ran different configurations, and those which proved to be the most efficient where: 15 epochs, learning rate = 0.001, weight decay =10^{-5} and Adam optimizer, obtaining a 57% accuracy and 1.3 loss. Some of these were slightly adjusted after identifying the best-performing model to try to maximize performance.
At this point, given that the objective of transfer learning is to boost the results, and since we still did not observe any improvement, we began fine-tuning the model (as seen in Table 5) till achieving the best performance with 8 layers unfrozen and 1.9 million parameters (Fine-tune 3), for 10 epochs (we applied early stopping on the number of epochs) and a 73% of validation accuracy and 0.83 of loss.
From this point, we observed that unfreezing additional layers did not lead to meaningful improvements, resulting in unnecessary computational cost.

With the hyperparameters fine-tuned on Fine-tune 3, we evaluated the model on the unseen test set, and obtained slightly improved results: 0.82 loss, 73.79% accuracy and 74.38% recall, indicating that the model is reliable.
Additionally, we analyzed the confusion matrix in Figure 6 to evaluate label classification accuracy and identify common misclassifications (highlighted with a red box). For example, Nevus is sometimes misclassified as Squamous Cell Carcinoma.

Note that in this second part of the project, we focused on recall as a key metric to evaluate how well the model classifies each image within its respective class.

Finally, we have seen that adding transfer learning to our data significantly improved the performance obtained if we compared to the model designed, which clearly adds importance to the size of the dataset, and the number of parameters used.

6.3. Future work

Future work to improve performance could involve using a larger pretrained model (e.g., ResNet), which would increase the number of available parameters. Additionally, image processing techniques such as segmentation or the use of alternative color spaces could be explored
The dataset could also be enhanced by adding extra labels; for example, a label indicating whether an image shows healthy skin, or use the metadata files found on the data.
Furthermore, the impact of skin markers on classification results could be investigated, as well as the application of advanced techniques to address image noise.

7. Conclusions

We have seen that our implementation does not fully achieve the targeted results, but it delivers good performance considering we handle 9 classes and use approximately only 100,000 parameters.

Applying transfer learning with MobileNetV2 and fine-tuning the model significantly improves the results compared to our lightweight CNN. With just over 2 million parameters, the results become promising and could be acceptable, especially in contrast with the 89% accuracy obtained by VGG16, which uses around 20 million parameters, nearly ten times more than our implementation.
Therefore, we can conclude that we successfully designed a lightweight model, and implemented a transfer learning model that meets our initial objectives: balancing performance and computational cost to enable efficient deployment in low-resource environments.

Finally, we believe that implementing the enhancements discussed in the previous section, such as using larger pretrained models or advanced preprocessing techniques, could further improve the performance of this transfer learning approach without compromising efficiency.

8. References

[1] BlueDokk/Dullrazor-algorithm: Pre-processing technique called DullRazor for the detection and removal of hairs on dermoscopic images. (2025). Retrieved May 24, 2025, from GitHub website: https://github.com/BlueDokk/Dullrazor-algorithm

[2] Jaiyeoba, O., Ogbuju, E., Ataguba, G., Jaiyeoba, O., Omaye, J. D., Eze, I., & Oladipo, F. (2025). State-of-the-art skin disease classification: a review of deep learning models. Network Modeling Analysis in Health Informatics and Bioinformatics, 14(1). https://doi.org/10.1007/s13721-024-00495-w

[3] mobilenet_v2 — Torchvision main documentation. (n.d.). https://docs.pytorch.org/vision/main/models/generated/torchvision.models.mobilenet_v2.html

[4] Parvathaneni Naga Srinivasu, Jalluri Gnana SivaSai, Ijaz, M. F., Akash Kumar Bhoi, Kim, W., & Kang, J. J. (2021). Classification of Skin Disease Using Deep Learning Neural Networks with MobileNet V2 and LSTM. Sensors, 21(8), 2852–2852. https://doi.org/10.3390/s21082852

[5] Shakya, M., Patel, R., & Joshi, S. (2025). A comprehensive analysis of deep learning and transfer learning techniques for skin cancer classification. Scientific Reports, 15(1). https://doi.org/10.1038/s41598-024-82241-w

[6] Singh, P. (2023). Multiple Skin Disease Detection and Classification. Retrieved June 7, 2025, from Kaggle.com website: https://www.kaggle.com/datasets/pritpal2873/multiple-skin-disease-detection-and-classification