Image-Based Dermatological Diagnosis Using Deep Learning

Hugo Alberto Miranda Burnes
The University of Texas at Austin

ABSTRACT
This report explains the design and evaluation of a Deep Learning model for dermatological image classification, using a dataset of over fifteen thousand images labeled across 22 distinct categories of skin diseases and conditions. The iterative optimization process of the Convolutional Neural Network is highlighted in detail in this paper, with the model achieving a peak validation accuracy slightly above 60%. Although the performance did not meet acceptable standards, this study brings a spotlight on the challenges for multi-class dermatological image classification with Artificial Intelligence.

1. INTRODUCTION

In dermatology, skin diseases and conditions in patients can vary drastically, ranging from mild and inoffensive rashes to aggressive and potentially life-threatening types of cancer. Finding and diagnosing these conditions early is key for patients to get effective treatment and better recovery. In recent years, tools and applications leveraging Artificial Intelligence have emerged with the goal of automatically classifying skin images into various disease categories, aiming to assist patients and healthcare professionals by facilitating early detection of potential skin-related issues, enabling timely medical diagnosis and intervention before conditions worsen and become more difficult to treat.

The objective of this report is to provide a detailed account of the use, configuration and training of a Convolutional Neural Network (CNN) deep learning model, using a dataset comprising labeled images of various skin conditions and/or diseases. The main intention is to configure and train a model from scratch and use different techniques to improve its accuracy instead of using any existing pre-trained model to better understand each aspect that makes up Deep Learning models applied to healthcare.

1.1 RELATED WORK

According to the American Academy of Dermatology, up to 50 million people per year are affected by acne in the United States, making it the most common skin condition in this country, where, on the opposite side of the spectrum, the most common type of cancer is skin cancer. Current estimates suggest that around 20% of its population develop skin cancer in their lifetime [1]. For these types of skin conditions and diseases, both mild and severe, high-accuracy deep learning models for dermatology diagnosis have already been widely used in recent years, aiding patients seeking for preliminary diagnosis while waiting for professional advice from dermatologists [2].

Overall, Computer Vision and Pattern Recognition using Deep Learning algorithms have greatly benefited from research and projects in the healthcare field, such as the widely acclaimed U-Net architecture, originally introduced in 2015 for Biomedical Image Segmentation [4], which later proved to be a key breakthrough for Generative AI in the years that followed. Specifically referring to dermatological diagnosis with the help of Artificial Intelligence, a comparable project such as the one proposed in this report was documented in a paper in 2023, *Skin disease detection using deep learning* [3], which intended to identify six prevalent skin diseases using 150,223 images through a Convolutional Neural Network and it managed to obtain an outstanding accuracy of 87%.

2. METHODOLOGY

2.1 WORKFLOW

A quick overview of the methodology for this report can be found in Figure 1. The workflow began with the searching for medical datasets online which eventually led to selecting the *Skin Disease dataset* on Kaggle [5]. After the selection of the dataset and problem at hand to solve, the analysis and model development were carried out within a Jupyter Notebook environment, utilizing commonly used Python libraries in Deep Learning such as Keras, TensorFlow, Numpy and Matplotlib. Following the loading of the skin images, an initial exploratory analysis of the dataset was performed. The dataset was already pre-processed and well-organized, containing no unusable images, which allowed the development process to proceed directly to the iterative configuration and training of the model. Following each training iteration, model performance was evaluated in terms of loss and accuracy to determine whether further adjustments were necessary or whether the pre-trained model had reached satisfactory performance metrics. Once the final model was selected, training logs were examined through TensorBoard for further analysis of the model training process. Finally, a subset of individual images was manually selected to test the model's predictions, thereby providing a practical demonstration of its potential real-world application.

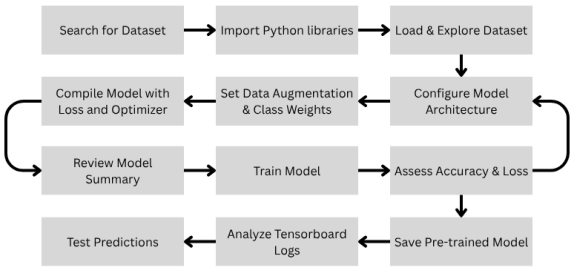

Fig. 1. Methodology workflow chart

2.1 DATASET

The dataset used for this assignment, *Skin Disease Dataset*, is a collection of 15,444 skin disease images categorized into 22 distinct categories or classes: *Acne, Actinic Keratosis, Benign Tumors, Bullous, Candidiasis, Drug Eruption, Eczema, Infestations/Bites, Lichen, Lupus, Moles, Psoriasis, Rosacea, Seborrheic Keratoses, Skin Cancer, Sun/Sunlight Damage, Tinea, Unknown/Normal, Vascular Tumors, Vasculitis, Vitiligo,* and *Warts*. These 256 x 256 images were already splitted in two datasets, one containing 1,546 for testing/validation and the other one with 13,898 images for training which were going to be grouped into batches of 32 images each for the model training process.

2.2 PYTHON LIBRARIES

For this assignment Keras and Tensorflow were the main libraries used to configure and train the CNN model aiming to predict skin diseases on the images. Tensorflow is a library that contains an extensive array of modules that help developers create Machine Learning models and Keras is a Deep Learning API built on top of it which contains different tools for model layers, preprocessing data and images, as well as training and testing the model [6].

3.3 CONFIGURING MODEL ARCHITECTURE

The initial approach for this assignment was to start with a simple CNN Model using Keras and Tensorflow that could be progressively and iteratively improved based on the output performance. Using the Sequential module from Tensorflow, every layer was added in a sequential manner, as its name implies. The Tensorflow modules used for the final model were the following: Conv2D, with increasing filters ranging from 16 to 512, a kernel size of 3x3 pixels and padding set to the same, to cover the edges when downsizing. Batch Normalization which maintains the mean output close to 0 and the output standard deviation close to 1 [7]. A ReLU Activation layer that adds non-linearity to the network and transforms every negative value to zero. MaxPooling2D, which selects the max value from the Convolution matrix, preferred over AveragePooling2D which takes the average of the matrix instead. GlobalAveragePooling2D, taking a global average, was used in the end since it is normally preferred instead of Flatten which simply represents the entire output in a one dimensional vector.

Finally, the Dense layers or otherwise called Fully Connected layers are essential for Deep Learning models and help them learn complex data patterns. Finally, the last Dense layer, the Output layer, is set to return the same number of channels as expected based on the amount of classes required, in this case, skin diseases, followed by a last activation layer, which for this model had to be configured with a Sigmoid function allowing all the classes to represent any given percentage in such a way that they would all add up to 1. With this output, the class with the higher percentage is what the model predicts should be the label assigned to the image in turn.

3.4 DATA AUGMENTATION AND CLASS WEIGHTS

At different stages of the training iteration process, it was evident that the data was a bottleneck in getting to a better validation accuracy. Initially, when the model was overfitting to the data, it essentially meant that with the low variation of images provided, the model was simply memorizing the exact patterns from the training data set, which returned a very high accuracy for predicting images from that dataset split, but when it came to validation data, the accuracy was extremely poor and stagnant. This was solved through data augmentation, a process which after each training epoch, it slightly alters images using filters that could rotate, shift, flip, change the brightness or zoom the images randomly, artificially creating a larger set of patterns for the model to learn. This preprocessing method was done using the *ImageDataGenerator* and *flow_from_directory* modules from Keras using the images directly from the directory, which resulted in segmented data by batches.

The second issue with the data was that it was unbalanced and the classes for skin diseases were unevenly represented by the images, meaning that some classes had significantly more image examples than others. The *compute_class_weight* module from Keras calculates this difference from the source automatically, returning the specific class weight to be used when training the model in order to give each class a fair distribution against the rest of them, allowing the model to have a more generalized learning removing bias towards overrepresented classes (Fig. 2).

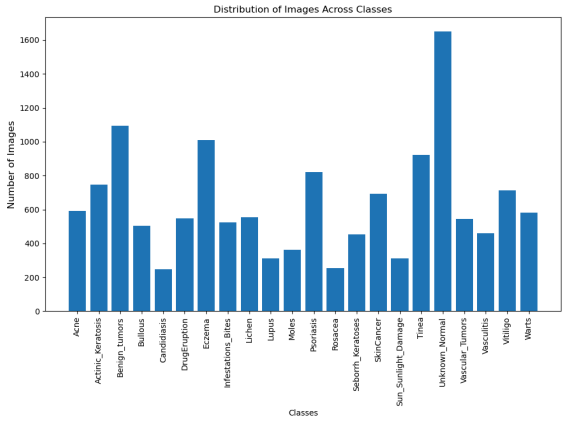

Fig. 2. Visualization plot of the distribution of images across skin disease classes.

3.5 COMPILING MODEL WITH LOSS AND OPTIMIZER

The loss function was SparseCategoricalCrossentropy from Keras. In contrast with the CategoricalCrossentropy module which would have returned an array of 22 booleans with only a true boolean value for the index that represented the predicted label, SparseCategoricalCrossentropy simply returns that index as an integer, which in this case represents the class for the predicted skin disease. The optimizer used for this task was Adam (Adaptive Moment Estimation) with varying learning rates. This optimizer is preferred for image

classification tasks as it works quite well with large amounts of data.

3.6 REVIEWING MODEL SUMMARY

When reviewing the Model Summary, it could confirm the input shape would be X and the output shape would be Y.The model had a total of 1,976,438 parameters with a size of 7.54 MB. Out of that total of 1,974,422, more than 99%, were trainable parameters and only 2,016, less than 1%, representing non-trainable parameters.

3.7 TRAINING MODEL AND ASSESSING ACCURACY AND LOSS

In total ten training attempts were made due to time and equipment constraints. These training processes ranged between 5-7 minutes per epoch, resulting in most processes running over 10 hours, with the final 200-epoch training taking over 20 hours for completion. This table shows a brief summary of how the model was configured at each attempt and what were the results obtained after training it.

ATTEMPT #1

Results: Overfitting the data with nearly 99% accuracy for training data but stuck at around 30% accuracy for validation.

Key Configuration Highlights:
- 3 - 2D Convolution Networks using ReLU and Max Pooling
- Flatten followed by 1 Dense layer with ReLU
- Output Dense layer for 22 channels with Softmax Activation
- 20 Epoch training

ATTEMPT #2

Results: The model was still overfitting but the accuracy for training data decreased at around 70%.

Key Configuration Highlights:
- Batch Normalization
- L2 Regularizer at 0.001
- One extra Dense layer
- Dropout at 0.3 for both hidden Dense layers
- 100 Epoch training

ATTEMPT #3

Results: Overfitting was not present anymore since the validation accuracy and loss were now following roughly the same trends for the training accuracy, but it was overall very low at around 35%.

Key Configuration Highlights:
- Data Augmentation
- 50 Epoch training

ATTEMPT #4

Results: This configuration got similar results to the last attempt, but the training data accuracy increased to around 45%, while the validation accuracy followed close to 40%.

Key Configuration Highlights:
- 5 - 2D Convolution Networks using ReLU, Batch Normalization and Max Pooling
- Adam optimizer learning rate changed from 0.01 to 0.0001
- 50 Epoch training

ATTEMPT #5

Results: Training data accuracy improved to around 60% but the validation accuracy trailed behind at around 50%.

Key Configuration Highlights:
- 6 - 2D Convolution Networks using ReLU, Batch Normalization and Max Pooling
- Adam optimizer learning rate changed from 0.0001 to 0.001
- 100 Epoch training

ATTEMPT #6

Results: This attempt was interrupted at 50 epochs due to showing very similar results to the last attempt at this point.

Key Configuration Highlights:
- Changed architecture order to Conv2d => BatchNorm => ReLU => MaxPool
- Dense layer at 512 channels
- Removed second Dense layer
- Dropout reduced to 0.15
- 100 Epoch training

ATTEMPT #7

Results: The training data accuracy increased slightly at around 63% but the validation data accuracy seemed to be worse than before at the 45-50% range.

Key Configuration Highlights:
- Switched to using less initial layers (4 - Conv2d => BatchNorm => ReLU => MaxPool) but with increasing number of filters after each layer (32, 64, 128, 256)
- Switched Flatten for GlobalAveragePooling2D
- Tested with Dropout at 0.5
- Used Class Weights for the unbalanced data
- Adam optimizer learning rate changed back from 0.001 to 0.0001
- 100 Epoch training

ATTEMPT #8

Results: This attempt was interrupted at 25 epochs due to showing a trend of worse results to the last attempt at this

point, stopping at 30% training data accuracy.

Key Configuration Highlights:
- Removed Dropout
- Adam optimizer learning rate changed from 0.0001 to 0.01
- Used default L2 Regularizer
- 100 Epoch training

ATTEMPT #9

Results: The training data accuracy went back up to around 70% while the validation data accuracy remained closer to 50%, but both trends showed projections of growth at greater epochs.

Key Configuration Highlights:
- 6 - CNN Layers increasing from 16 to 512 filters
- L2 Regularizer at 0.001
- Added padding to the CNNs
- Added back Dropout at 0.25
- Added one more Dense layer with 256 channels
- 100 Epoch training

ATTEMPT #10

Results: Training data accuracy finalized above 90%, while the validation data accuracy deviated even more from the training data accuracy than the previous attempt, oscillating around 60%, which ended up being the highest validation accuracy obtained for this assignment.

Key Configuration Highlights:
- Training started using the same pre-trained model as the previous attempt but simply added 200 more epochs to review results for prolonged training with the same conditions for a total of 300 epochs.

4. RESULTS

4.1 FINAL MODEL ACCURACY AND LOSS

As stated previously, the last training attempt, *Attempt #10*, picked up where *Attempt #9* left off by simply keeping training the same model for a larger amount of epochs. The ninth attempt ran for 100 epochs, taking the training data accuracy to cross the 70% mark while the validation data accuracy ranged from 45-50% (Fig. 3). The last attempt started at basically those same metrics for both datasets, taking the training data accuracy to above 90% but for the validation data it became pretty much stagnant, with an accuracy that barely broke the 60% mark at some epochs (Fig. 4).

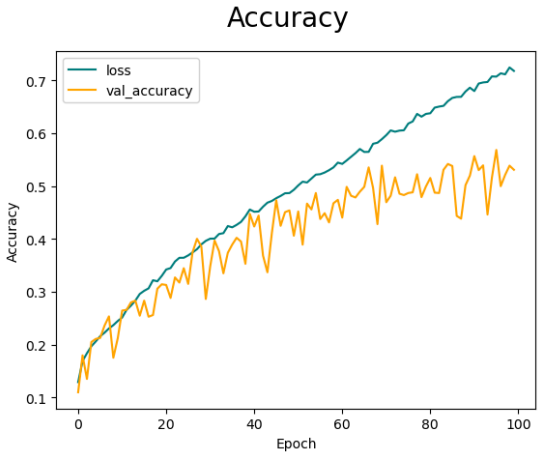

Fig. 3. Accuracy plot for 100 epochs during training in Attempt #9.

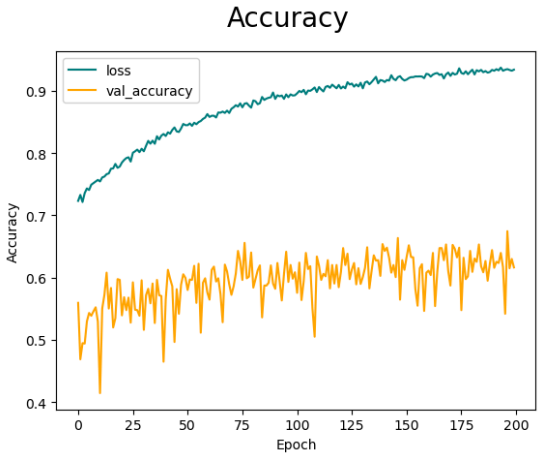

Fig. 4. Accuracy plot for 200 epochs during training in Attempt #10 following initial training in Attempt #9.

As for the loss, the ninth attempt brought it down to around 1.0 for training data, but the validation data rarely got past 2.0 at some epochs (Fig. 5). For the tenth and final training attempt, the training data loss was less than 0.5, but the loss for validation data got stuck in the 1.5-2.5 range through those extra 200 epochs, having even a higher variance than the previous attempt (Fig. 6).

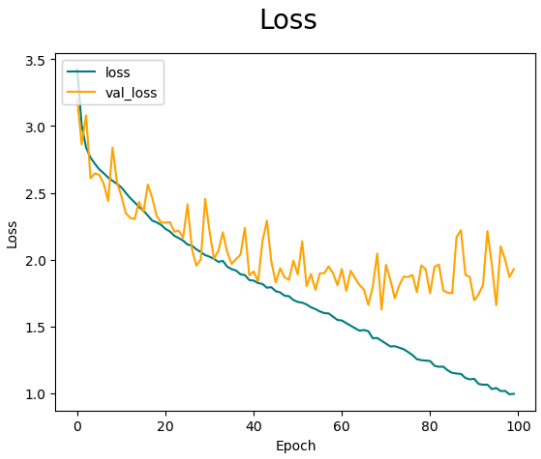

Fig. 5. Loss plot for 100 epochs during training in Attempt #9.

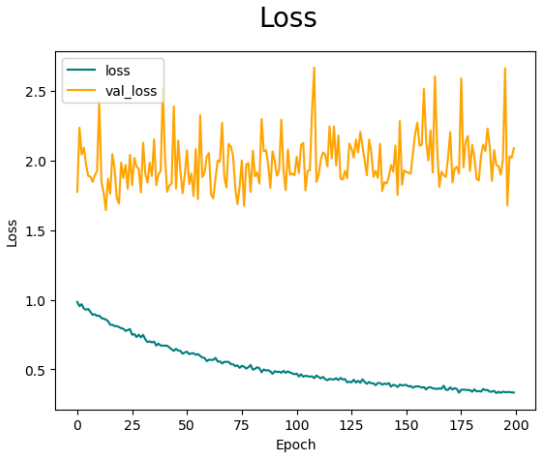

Fig. 6. Loss plot for 200 epochs during training in Attempt #10 following initial training in Attempt #9.

4.2 ANALYZING TENSORBOARD LOGS

We can examine clearly the journey the model training had for the validation data by reviewing the recorded logs in Tensorboard (Fig. 7). There were a couple of thresholds that were not able to be crossed for the most part, where the loss rarely came below 2 and the accuracy was always below 60% until the final training attempt when it finally broke that barrier through a considerably larger number of epochs.

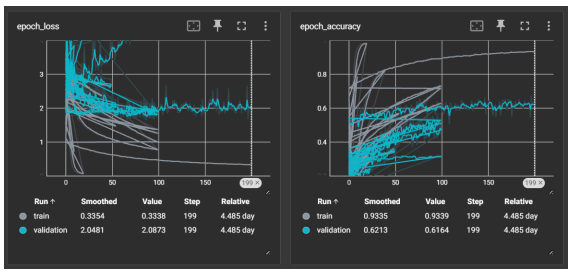

Fig. 7. Tensorboard logs for all the training processes attempts.

4.3 TESTING PREDICTIONS

Below are three examples of the model predictions where we can analyze its output more closely. As explained before, the last layer of the model had 22 channels, each for a skin disease class, and it used a Sigmoid activation function that takes the results for the probabilities for any particular image to be labeled as one of those 22 different classes and turns it basically into a percentage, where the sum of all the values should equal to 1. The final predicted label is simply the label with the highest percentage of probability, which also means the confidence metric, but there were some cases where the model was not as sure as others, and gave different percentages to different skin disease classes. In the first example (Fig. 8), we can see the model had a wrong prediction, Skin Cancer, with a confidence of 66.50%, but the other closest label, with around 30%, was the correct prediction, Infestations Bites. The last two images are examples where the model was basically a 100% sure about its prediction, but only the case where it predicted *Psoriasis* was the correct answer (Fig. 9), while the image with *Candidiasis* was incorrectly identified *Warts* instead (Fig. 10).

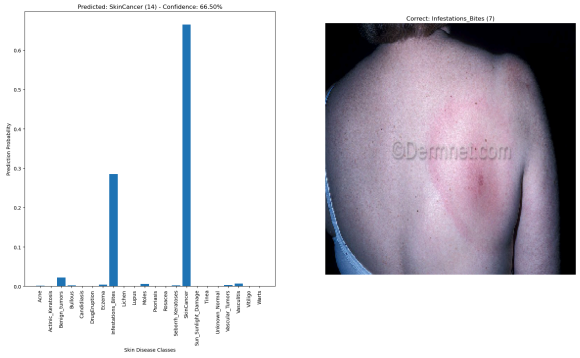

Fig. 8. Prediction with mixed results for the image of skin with Infestations Bites.

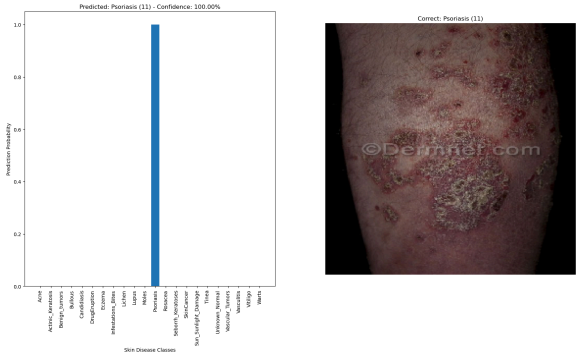

Fig. 9. Prediction with correct results for the image of skin with Psoriasis.

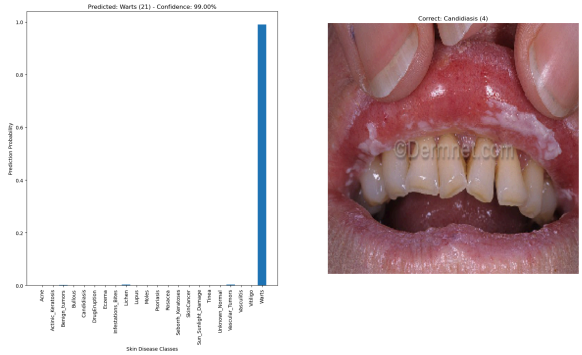

Fig. 10. Prediction with incorrect results for the image of skin with Candidiasis.

5. CONCLUSION

For medical image analysis it could be said that for a prediction accuracy percentage to be decent or acceptable it should range somewhere 70 to 80%, depending on the exact task at hand. With this in mind, it is correct to imply that the model trained for this assignment failed the mark at ~60%. The main reason for this could be the amount of data available per class, meaning trying to categorize this amount of images into 22 unique classes using Deep Learning algorithms presents a bigger challenge and perhaps it was too ambitious, however the goal for this assignment, as it name implies, was to present a high risk.

This approach could be further explored using either a considerably larger dataset or by reducing the number of image classes, potentially removing some specificity and clustering different skin diseases into more general categories. This would allow for similar initial conditions comparable to those in the paper mentioned in the introduction, *Skin Disease Detection Using Deep Learning* [3].

Furthermore, utilizing readily available pre-trained models, such as *ResNet* or *EfficientNet*, would be a more logical route to obtain better accuracy and loss statistics by focusing the effort on fine-tuning instead of designing and training a model from scratch. This method would provide models with acceptable standards that could help existing patients in real-life scenarios seeking preliminary dermatological diagnosis.

REFERENCES

[1] Anon. Skin cancer. Retrieved August 10, 2025 from https://www.aad.org/media/stats-skin-cancer
[2] Venkatesh, K.P., Raza, M.M., Nickel, G. et al. Deep learning models across the range of skin disease. npj Digit. Med. 7, 32 (2024). Retrieved August 4, 2025 from https://doi.org/10.1038/s41746-024-01033-8.
[3] Syed Inthiyaz, Baraa Riyadh Altahan, Sk Hasane Ahammad, V Rajesh, Ruth Ramya Kalangi, Lassaad K. Smirani, Md. Amzad Hossain, Ahmed Nabih Zaki Rashed, Skin disease detection using deep learning, Advances in Engineering Software, Volume 175, 2023, 103361, ISSN 0965-9978, Retrieved August 4, 2025 from https://doi.org/10.1016/j.advengsoft.2022.103361.
[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. (May 2015). Retrieved August 4, 2025 from https://doi.org/10.48550/arXiv.1505.04597.
[5] Prashant Kumar Mishra. 2024. Skin disease dataset. (November 2024). Retrieved August 4, 2025 from https://www.kaggle.com/datasets/pacificrm/skindiseasedataset
[6] Anon. Keras: The high-level API for tensorflow: Tensorflow Core. Retrieved August 10, 2025 from https://www.tensorflow.org/guide/keras.
[7] Keras Team. Keras Documentation: Batch normalization layer. Retrieved August 10, 2025 from https://keras.io/api/layers/normalization_layers/batch_normalization/