

CNN-Based Multi-Class Classification of Fungal, Scabies, and Allergic Skin Diseases Using Image Processing

Randi Farmana Putra*, Teresa Sheila Dinda

Universitas Pertamina

Correspondence Email: randi.putra@universitaspertamina.ac.id*

Abstract

Neglected Tropical Diseases (NTDs) affect over a billion people globally, with skin conditions like fungal infections, scabies, and allergies often overlooked due to overlapping symptoms and limited diagnostic resources. To address this, we propose a CNN-based multi-class classification model using image processing techniques to distinguish six skin disease classes: tinea, candidiasis, pityriasis versicolor, scabies, contact dermatitis, and eczema. A dataset of 300 images was curated from DermNet, a credible dermatology resource, and preprocessed via normalization, augmentation, and batch-wise training. The designed CNN architecture achieved 93% testing accuracy, with 92% precision, 95% recall, and 93% F1-score, significantly outperforming benchmark models. By integrating image processing (e.g., noise reduction, flipping) with a 10-layer CNN framework, the model mitigates challenges posed by symptom similarity and dataset limitations. This work aligns with the WHO's 2030 NTD roadmap by offering a scalable tool for early detection and reduced transmission of neglected skin diseases.

Keywords: CNN, multi-class classification, fungal infections, scabies, allergic skin diseases.

INTRODUCTION

Neglected Tropical Diseases (NTDs) are a group of infectious diseases that disproportionately affect impoverished populations in tropical regions. NTDs encompass many illnesses caused by bacteria, viruses, fungi, parasites, and toxins, involving over a billion people worldwide (World Health Organization, 2024). Scabies is one such NTD, and, like other skin conditions such as fungal infections and allergies, these diseases often receive insufficient attention and treatment.

Several factors contribute to the transmission and neglect of skin diseases, including overcrowded living conditions, unsanitary environments, and shared facilities. Additionally, the similar symptoms and visual manifestations of different skin diseases pose challenges to accurate diagnosis (Harahap et al., 2022). This lack of awareness and knowledge regarding skin diseases leads to a high prevalence of these conditions (Wijaya et al., 202). Therefore, tools that can classify skin diseases will help reduce the oversight and spread of these conditions.

Furthermore, significant advancements in Artificial Intelligence (AI) have demonstrated its potential in various fields, including medicine (Esteva et al., 2021). Computer Vision (CV), a branch of AI, enables machines to perceive the world like human eyes. CV encompasses image processing technologies capable of detecting, recognizing, labeling, classifying, and analyzing images. Deep Learning (DL) algorithms address CV problems. Several DL categories apply to visual understanding tasks, including the Convolutional Neural Network (CNN) architecture (Chai et al., 2021).

Previous research has implemented image processing technology with CNN architecture to identify skin diseases. However, the simultaneous classification of fungal infections, scabies, and allergies with CNN architecture remains unexplored. Therefore, developing a model for classifying fungal infections, scabies, and allergies on human skin using CNN architecture is necessary to support the WHO's NTD road map 'together toward the 2030 agenda' and reduce the neglect of skin diseases in Indonesia.

METHOD

Research Design

The following stages detail the research process for developing a classification model for fungal infections, scabies, and allergies on human skin.

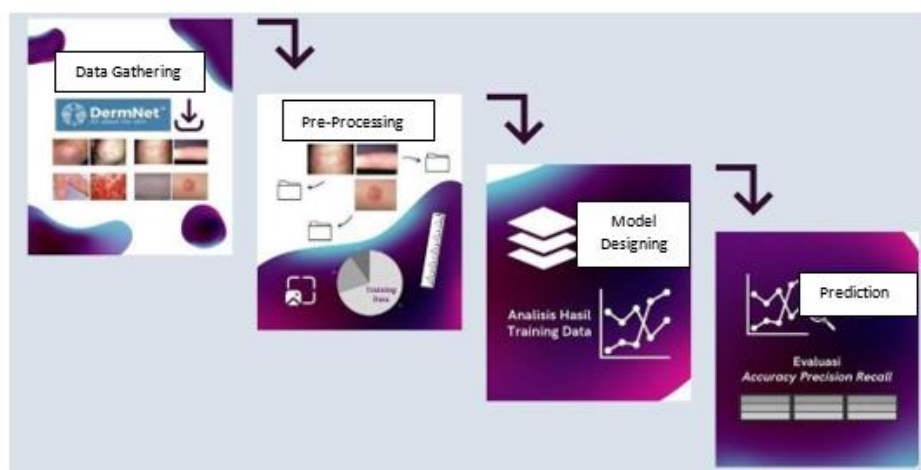


Figure 1. Research Flowchart

Data Collection

In this stage, we gather images of fungal infections, scabies, and allergies from the DermNet website in 2024. DermNet is a dermatology data publisher that provides credible information about skin diseases. DermNet is owned by the DermNet New Zealand Trust and supported by the New Zealand Dermatological Society. The following table presents the results of data collection based on the data search process on the DermNet website.

Table 1. Data Collection Result

Disease Name	Keywords	Data Volume
tinea	"tinea", "tinea corporis", "tinea capitis", "tinea cruris"	50
candidiasis	"candidiasis", "candida", "candidiasis dermnet", "candida skin"	50
pityriasis versicolor	"pityriasis versicolor", "tinea versicolor"	50
scabies	"scabies", "scabies images"	50
contact dermatitis	"contact dermatitis", "contact dermatitis dermnet", "allergic contact dermatitis", "irritant contact dermatitis"	50
eczema	"eczema", "discoid eczema"	50
Total		300

Six folders are set up, each designated for a specific disease. Each downloaded image is sorted into the corresponding folder based on the type of disease. The data from these folders are then loaded into the IDE to run the next stage, data preprocessing.

Data Preprocessing

This stage begins when the required libraries and packages are successfully installed and running. We will use TensorFlow, opencv-python, matplotlib, scikit-learn, and seaborn packages. The preprocessing stage consists of five stages, including:

a. Removing Invalid Data

In this research, data is considered valid when it meets two conditions: it must be read as a numpy array and has the extension jpeg, jpg, bmp, or png. The validity of the data will be verified using the cv2.imread function. The cv function converts the image into a numpy array by representing each pixel as a matrix reflecting its RGB values. If the data can meet the two specified conditions, then the data is considered valid.

b. Loading Data

Data will be loaded into the IDE using the Keras Tensorflow utilization, namely the image_dataset_from_directory function. This function can load data from subdirectories organized within a directory. This setup facilitates smooth preprocessing because the data is automatically labeled according to the subdirectory names. The image_dataset_from_directory function also has several parameters like batch size and image size. In this study, the number of batch sizes is determined to be 30, and the image size follows the default value, 256x256. When data is loaded using the image_dataset_from_directory function, it becomes an object of tf.data. The dataset consists of a collection of data batches. Each batch is a tuple containing two numpy arrays: one array represents the image data, and the other represents the corresponding labels.

c. Determining Data Scale

Once the data is loaded, it forms a multidimensional array where each pixel in every image is represented by three values corresponding to the RGB color channels. These values range from 0 to 255, reflecting the intensity of each color on a scale from 0 to 255. To facilitate the computational process, normalization of these numbers is needed. In this study, each pixel

representation will be divided by 255 to ensure the range of values for the normalized data is between 0 and 1.

d. Dividing Data

Once the data is successfully loaded and rescaled, it will be split into three groups: training data, validation data, and test data. This split is essential for the subsequent data training process. To achieve this, the skip and take functions are used. In this study, the data is divided into a ratio of 70% for training, 20% for validation, and 10% for testing.

e. Data Augmentation

The data augmentation stage occurs before training begins. The augmentation technique functions to increase the variation of image data in a realistic manner. In this study, the training data will go through a flipping function (reflection) both vertically and horizontally.

Data Design and Training

In the third stage, the CNN algorithm will use the Sequential function to form linearly stacked layers. The first layer is the input followed by hidden layers, and finally the outputs. Each parameter includes the layer name, the number of filters, the filter size, the filter movement scale, and the type of activation used.

In this study, the CNN architecture is designed using the Sequential function and consists of ten layers. The input layer is configured with a shape parameter of (256, 256, 3), representing image data with a resolution of 256x256 pixels and three-color channels (RGB). The first convolutional layer includes 15 filters, each with a size of 3x3, a stride of 1 pixel, and utilizes the ReLU activation function. This is followed by a max pooling layer to reduce the spatial dimensions. The second convolutional layer has 30 filters of size 3x3, also with a stride of 1 pixel and ReLU activation, and is followed by another max pooling layer. The final convolutional layer contains 15 filters with the same configuration as the previous layers and is also followed by a max pooling layer. After these convolutional and pooling stages, a flatten layer is used to convert the 2D output into a 1D vector. This flattened vector is then passed through a fully connected layer with 256 neurons and ReLU activation. Finally, the last fully connected layer consists of six output neurons with a Softmax activation function, which is used for multi-class classification tasks.

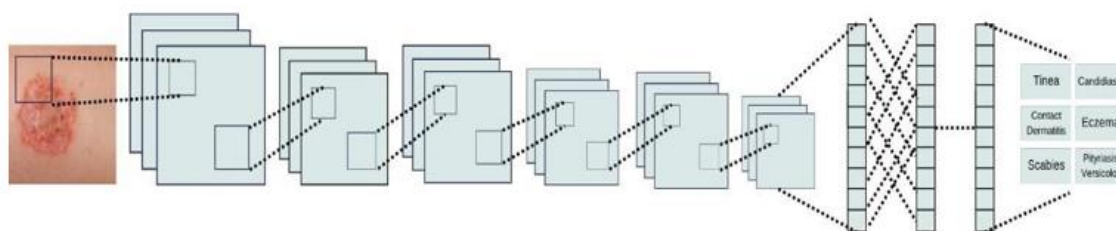


Figure 2. CNN Architecture Illustration

The next step is to run the compile function, which has parameters in the form of optimizer type, loss specification, and metrics to be displayed. In this study, we use the Adam optimizer with a learning rate of 0.0001. The loss specification used is Sparse Categorical Crossentropy, and the metric displayed is accuracy.

Finally, the designed model will train data with the fit function. In the fit function, you can specify the number of epochs, validation data used and the desired callback handling. In

this study, the number of epochs is determined to be 50 (fifty). The results of data training provide information about the trend of loss and accuracy.

RESULT AND DISCUSSION

1. Classification Process of Fungal Infections, Scabies, and Allergies

The classification of fungal infections, scabies, and allergies on human skin will use an image-processing approach and CNN architecture. This process will use the TensorFlow platform and the Keras library as a high-level API. There are several stages in the classification process, including the preprocessing stage, the data design and training stage, and the evaluation stage.

Before starting the preprocessing stage, the required dependencies must be installed, including TensorFlow, opencv-python, matplotlib, scikit-learn, and seaborn. Each dependency has its function, such as designing models, processing image data, and displaying visualizations of data and prediction results. The preprocessing stage itself consists of five steps, namely removing invalid data, loading data, determining data scale, splitting data, and data augmentation.

Invalid Data Removal Stage

This stage begins by running a nested loop to iterate over each image from each owned subdirectory. The goal is to ensure that each image is converted to a numpy array and has the correct file extension. The cv2 library is needed to represent each pixel in the image into a sequence of numbers from the RGB intensity of the image. Then imghdr library checks that the image extension is as desired. Data is considered valid if it meets the purpose of this process. The result of this stage is that all the images owned are valid data, and no data is discarded.

Data Loading Stage

Before this stage starts, the path of the destination directory is declared. This directory contains data confirmed to be valid in the previous stage. In addition, there are setups needed, such as determining the number of images in each batch and setting the seed to ensure consistent training results. The data loading stage will use the `image_dataset_from_directory` function, a built-in function of Tensorflow.

This function loads the data into a dataset as a `tf.data` object. Within this dataset, each batch of data consists of two tuples, namely tuple x as an image representation and tuple y as a label representation. The result of this stage is an output in the form of a description of all the images owned, namely 300 images successfully loaded. The first four images from one batch are displayed to ensure the success of this stage, as shown in Figure 3.

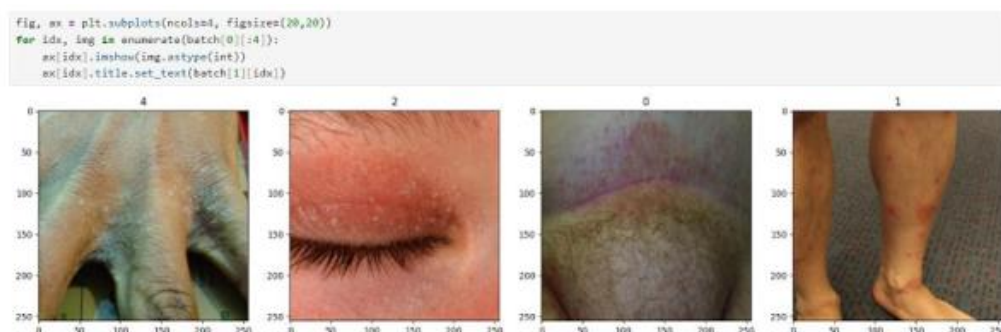


Figure 3. Display Data

Determining Data Scale Stage

This stage is executed by dividing the tuple x in each batch within the dataset by 255. As previously explained, tuple x represents the image, so it consists of a sequence of numbers (numpy array) from the RGB intensity possessed by each pixel in the image. Since RGB intensity ranges from 0 to 255, dividing tuple x by 255 will change the data scale on tuple x from 0 to 1. The results of this stage can be confirmed by showing the maximum value of tuple x in one batch, as shown in Figure 4. The output value of 1 indicates that the data scaling stage was successful.



```
data.as_numpy_iterator().next()[0].max()
1.0
```

Figure 4. Scaling Data Results

Data Division Stage

The data is divided into three groups, namely train data, validation data, and test data with a ratio of 7:2:1. Typically, data is split into training data and test data with a ratio of 7:3 or 8:2. However, datasets with small sizes, such as in this study, require validation data. Validation data can help to analyze training results and monitor overfitting symptoms, making it easier to design an optimal model.

Data Augmentation Stage

Data augmentation is a technique to increase the variation in existing data and reduce the risk of overfitting. In this study, the flipping technique is applied both vertically and horizontally to the training data. Training data with high variation can create a more general model because it has adapted to diversity. Within Tensorflow, a dataset in the form of a `tf.data` object has an augmentation technique approach on the fly. In this approach, augmentation is done in real time as each batch of data is processed or trained. The computational process does not increase significantly while memory consumption becomes efficient.

After the preprocessing stage, the prepared data will move to the data design and training stage. In this stage, the model will be designed using the sequential function from the Keras library. The sequential function is the most used in Keras to plot linear models. The model will be composed of sequential layers. In this research, model design is carried out by arranging 10 (ten) layers, including 1 (one) input layer, 3 (three) convolutional layers, 3 (three) pooling layers, 1 (one) flatten layer, and 2 (two) fully connected layers.

The finished model will go through the compile process. In this process, Adam is chosen as the optimizer function with a learning rate of 0.0001. By default, Adam has a learning rate of 0.001, but setting it to 0.0001 is done to reduce the step size in optimization. Then Sparse Categorical Crossentropy is set as the loss specification and accuracy as the metric to be displayed. This loss function is typically used in multi-class classification problems, particularly in label encoder cases.

In Figure 5, you can see the model summary from the model design results. The input layer will receive images sized 256x256. In the convolutional layer, the image size is reduced since padding is not applied. In the final dense layer, six neurons are set to output the prediction results from the six-class classification.

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 254, 254, 15)	420
max_pooling2d_6 (MaxPooling2D)	(None, 127, 127, 15)	0
conv2d_7 (Conv2D)	(None, 125, 125, 30)	4,080
max_pooling2d_7 (MaxPooling2D)	(None, 62, 62, 30)	0
conv2d_8 (Conv2D)	(None, 60, 60, 15)	4,065
max_pooling2d_8 (MaxPooling2D)	(None, 30, 30, 15)	0
flatten_2 (Flatten)	(None, 13500)	0
dense_4 (Dense)	(None, 256)	3,456,256
dense_5 (Dense)	(None, 6)	1,542

Total params: 3,466,363 (13.22 MB)
 Trainable params: 3,466,363 (13.22 MB)
 Non-trainable params: 0 (0.00 B)

Figure 5. Model Summary

Next is the data training process using epochs with a total of 50. The number of epochs is determined based on the trend of accuracy results. As shown in Figure 6, the training data accuracy results reached 99%, while validation data accuracy results reached 100%.

```
hist = model.fit(train, epochs=50, validation_data=val, callbacks=[tensorboard_callback])
```

Epoch 42/50	7/7	5s	594ms/step	- accuracy: 0.9058	- loss: 0.2932	- val_accuracy: 0.9667	- val_loss: 0.2028
Epoch 43/50	7/7	5s	650ms/step	- accuracy: 0.9097	- loss: 0.3148	- val_accuracy: 0.9333	- val_loss: 0.1437
Epoch 44/50	7/7	4s	563ms/step	- accuracy: 0.9290	- loss: 0.2608	- val_accuracy: 1.0000	- val_loss: 0.1093
Epoch 45/50	7/7	4s	536ms/step	- accuracy: 0.9493	- loss: 0.1774	- val_accuracy: 0.9333	- val_loss: 0.1294
Epoch 46/50	7/7	4s	546ms/step	- accuracy: 0.9283	- loss: 0.1898	- val_accuracy: 1.0000	- val_loss: 0.0571
Epoch 47/50	7/7	4s	542ms/step	- accuracy: 0.9383	- loss: 0.2422	- val_accuracy: 1.0000	- val_loss: 0.1462
Epoch 48/50	7/7	4s	527ms/step	- accuracy: 0.9190	- loss: 0.2487	- val_accuracy: 0.9000	- val_loss: 0.2668
Epoch 49/50	7/7	4s	538ms/step	- accuracy: 0.9568	- loss: 0.1863	- val_accuracy: 0.9000	- val_loss: 0.3769
Epoch 50/50	7/7	4s	542ms/step	- accuracy: 0.9908	- loss: 0.1214	- val_accuracy: 1.0000	- val_loss: 0.0974

Figure 6. Training Data Result

The results of training can be interpreted by displaying a trend plot of performance. Two trends are evaluated: the loss and accuracy trend. Monitoring accuracy and loss metrics can help identify overfitting. When training accuracy increases while validation accuracy stagnates or decreases, overfitting is indicated. Similarly, if training loss decreases while validation loss increases, it also specifies overfitting.

Ideally, the accuracy values of training and validation increase as the number of epochs increases. Furthermore, the loss values of training and validation decrease as the number of epochs increases. Figure 7 shows that the accuracy trend for both training and validation is escalated, indicating that the model is not overfitting. The loss trend for both the training and validation moves down together, also indicating that the model is not overfitting.

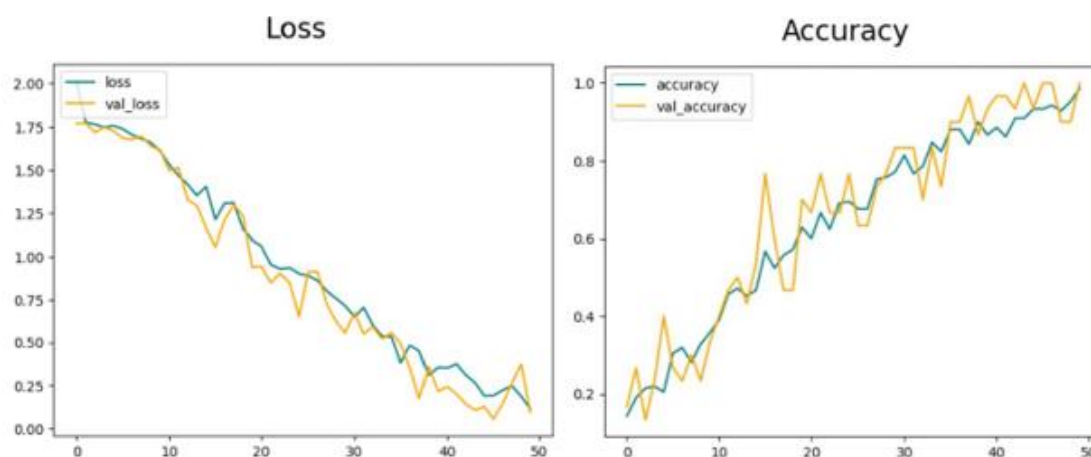


Figure 7. Trend of Accuracy and Loss

After the model has finished training, the next step is the evaluation stage, where predictions take place using the test data. The results of these predictions will be displayed in a classification report using the Scikit- Learn library, as shown in Figure 8. The performance evaluation can be analyzed through the following metrics: an accuracy value of 93%, an average precision of 92%, an average recall of 95%, and an average f1- score of 93%.

```
print(classification_report(y, yhat))
```

	precision	recall	f1-score	support
0	0.88	1.00	0.93	7
1	1.00	0.83	0.91	6
2	0.67	1.00	0.80	2
3	1.00	1.00	1.00	4
4	1.00	0.86	0.92	7
5	1.00	1.00	1.00	4
accuracy			0.93	30
macro avg	0.92	0.95	0.93	30
weighted avg	0.95	0.93	0.93	30

Figure 8. Classification Report

The test data predictions are visualized using a confusion matrix with the Seaborn library, as shown in Figure 9. The matrix reveals two prediction errors: one contact dermatitis image is classified as candidiasis, and one scabies image is classified as eczema. Next, the model was tested with an image of a disease it had never seen before, as shown in Figure 10. The images that are not included in the training, validation, or test were correctly predicted.

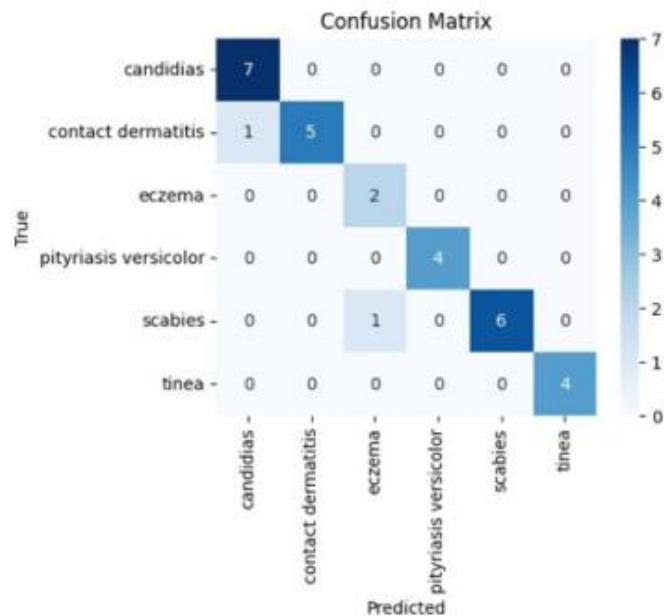


Figure 9. Confusion Matrix

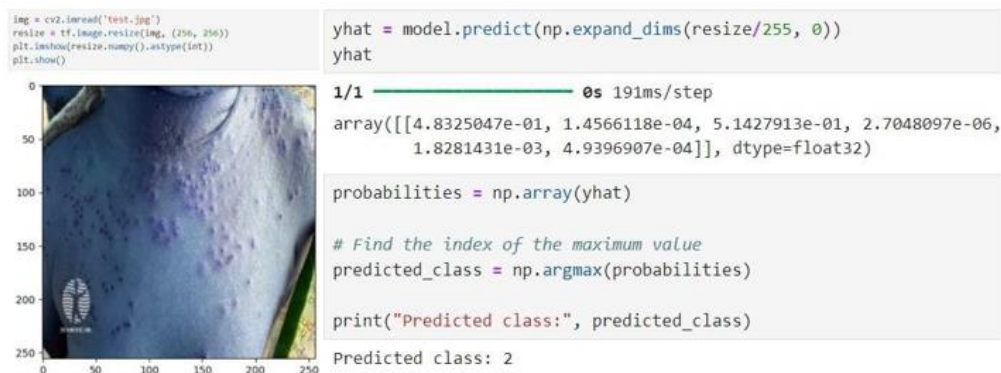


Figure 10. New Image Prediction Results

2. Model Analysis

A benchmark was conducted to evaluate the model's performance by referencing Khaled Elbahnasy's code on the Kaggle platform titled "skin detection acc=98%." The minimum accuracy standard in this study refers to the 98% accuracy achieved by this code. The selection of this code is based on several considerations: it is freely accessible and suitable for general discussion, it applies image processing and a CNN architecture for skin disease classification, it demonstrates a high training accuracy of 98%, it was uploaded in 2023 ensuring its relevance, and it has garnered 9,279 views, indicating substantial interest and engagement from the community.

Khaled Elbahnasy's dataset includes 7,330 images across 23 disease categories. The model he built comprises 26 layers consisting of input, convolution, pooling, flattening, and dense layers. The results of data training on the code reached 98% with 25 epochs. The prediction results from the code consist of an accuracy value of 82%, precision of 83%, recall of 81%, and f1-score of 82%.

The model originally created by Khaled Elbahnasy was adapted by the researcher for use in training the data in this study. Training and testing results using the Khaled Elbahnasy

model are detailed in Appendix Four. The accuracy results of training using Khaled Elbahnasy's model with 50 epochs are very small, only reaching 16%. In contrast to the accuracy value, the model in this study with the same number of epochs achieved 99%. The prediction results using Khaled Elbahnasy's model on the data in this study only reached an accuracy value of 23%, precision of 4%, recall of 17%, and f1-score of 6%. These results are significantly lower compared to those obtained with the model developed in this study. Thus, it is evident that the model developed for this research is more effective for fungal, scabies, and allergy disease datasets.

CONCLUSION

This study successfully developed a CNN-based multi-class classification model for distinguishing fungal infections (tinea, candidiasis, pityriasis versicolor), scabies, and allergic skin diseases (contact dermatitis, eczema) using image processing techniques. The proposed architecture, designed with 10 layers (3 convolutional layers, 3 pooling layers, 2 fully connected layers, and auxiliary layers), demonstrated robust performance in handling the complexity of 6-class skin disease classification. By leveraging DermNet- derived datasets and preprocessing methods such as normalization, data augmentation, and batch-wise training, the model achieved 93% testing accuracy, 92% precision, 95% recall, and 93% F1-score, outperforming benchmark models like Khaled Elbahnasy's architecture on the same dataset.

The integration of image processing (e.g., scaling, flipping, and noise reduction) with the CNN framework proved critical in addressing challenges such as overlapping symptoms and limited dataset diversity. This work aligns with the WHO's 2030 NTD roadmap by providing a scalable tool to reduce diagnostic delays and improve early detection of neglected skin diseases. Future research could expand the model's generalizability through cross-dataset validation and real-world clinical deployment..

REFERENCES

- World Health Organization. (2024). *Control of neglected tropical diseases*.
- P. Harahap, J. Hutahaean, and M. Dewi. (2022). Penerapan Metode Case Based Reasoning Untuk Diagnosa Penyakit Kulit Akibat Virus Eksantema Berbasis Web. *Building of Informatics, Technology and Science (BITS)*, 4(2), 856–866.
- A. R. Mz, I. G. P. S. Wijaya, and F. Bimantoro. (2020). Sistem pakar diagnosa penyakit kulit pada manusia dengan metode dempster shafer. *Journal of Computer Science and Informatics Engineering (J-Cosine)*, 4(2), 129–138.
- A. Esteva et al. (2021). Deep learning-enabled medical computer vision. *NPJ digital medicine*, 4(1), 5.
- J. Chai, H. Zeng, A. Li, and E. W. T. Ngai. (2021). Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6, 100134.
- L. Alzubaidi et al. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of big Data*, 8, 1–74.
- A. Géron. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc.