Development of Computational Models in Creating Skin Disease Classifiers Using CNN and ResNet Architecture

1. Elaine Faythe Hartono, Sekolah Pelita Harapan Kemang Village, Jakarta, Indonesia (elaine.hartono@student.sph.ac.id)
2. Joanna Natasha Marjono, Sekolah Pelita Harapan Sentul City, Bogor, Indonesia (joanna.marjono@student.sph.ac.id)

Abstract: *This research paper delves into the exploration of various computational models utilized in creating skin disease classifiers, aiming to enhance diagnostic accuracy and optimize potential accessibility to isolated patients. By investigating the different types of computational models employed in this context, this paper aims to shed light on how these models contribute to more accessible and efficient diagnoses of skin conditions, enabling healthcare professionals to provide widespread patient care. Furthermore, the paper will address the challenges encountered in the development of effective skin disease classifiers using computational models, such as data quality issues, model interpretability, and generalizability across diverse populations. Through a comprehensive analysis of these aspects, this research endeavors to advance our understanding of the potential of computational models in improving skin disease diagnosis and ultimately enhancing healthcare outcomes for individuals affected by dermatological conditions. Results showed that ResNet18 demonstrated higher overall accuracy on HAM10000 when compared to the confusion matrix and Grad-CAM visualizations of both models.*

Keywords: *Computational models, Skin disease classifiers, CNN, ResNet, Python*

1. Introduction

Skin diseases, such as acne, alopecia, bacterial skin infections, decubitus ulcers, fungal skin diseases, pruritus, psoriasis, scabies, urticaria, viral skin diseases, and skin cancer lesions, pose a significant public health concern worldwide, with recent years showing a rise in such diseases due to worsening living conditions and lack of access to healthcare [1]. Skin diseases are the fourth most common cause of all human diseases, affecting nearly one-third of the world's population; however, their burden is often underestimated [2]. Skin and subcutaneous diseases lead to profound long-term alterations even after the disease has resolved, affecting not only the physical health but also the mental health and quality of life of the patient, placing a high burden on patients' families and national healthcare systems globally. The burden of skin conditions was high in both high- and low-income countries, indicating that prevention of skin diseases should be prioritized. Hence, widening the knowledge of skin disease epidemiology is critical for policy development and resource allocation, which ultimately leads to disease prevention [1].

However, in this digital age, the development of computational models has innovated the field of dermatology by offering new approaches to skin disease classification. Through the potential concept of integrating A.I with skin disease classification, the prevention of common skin diseases may be accomplished. This research paper utilizes Convolutional Neural Networks (CNN) and Residual

Networks (ResNet) to achieve its findings. CNNs are deep learning architectures that learn directly from data. CNNs are useful for detecting patterns in images and recognizing objects, classes, and categories [3]. Residual Network (ResNet) is a deep learning model designed for computer vision applications. It is a CNN architecture capable of supporting hundreds or thousands of convolutional layers. ResNet uses "skip connections," which allows it to stack multiple identity mappings, skip those layers, and reuse the activations from the previous layer, resulting in improved performance for a larger number of layers [4].

2. Methods and Experimental Details

We have accomplished two models on identifying cancerous skin lesions using Python, incorporating libraries like OpenCV for traditional computer vision and PyTorch that serve as key tools for implementing deep learning. The goal is to adjust and compute the dataset we have chosen into a machine learning model, then evaluate its performance based on accuracy and compare which model has the fastest learning speed.

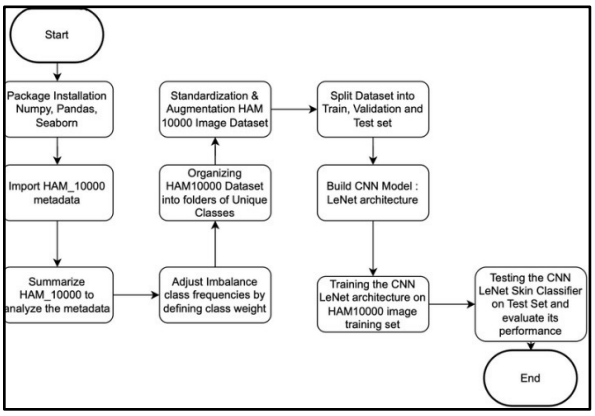


Figure 1. Diagram of the computation process

2.1. Data Collection and Analysis

For this project, we collected the case data and images for skin lesions from the HAM10000 using the Harvard Dataverse online query tool. The dataset comprises of 10,015 dermatoscopic images that was publicly released by the Harvard database in June 2018 to supply reliable training data for the automation of skin cancer lesion classification. In addition to the 10,015 images, a metadata file containing demographic information for each lesion was included, which were verified through histopathology (histo), while the ground truth for other cases was determined through follow-up examination (follow_up), expert consensus (consensus), or confirmation by in-vivo confocal microscopy (confocal). [5] The selection of skin lesions and diseases reflects the prevalence, the case definition and the availability of the data used for the development of the models.

To begin, we first imported the HAM10000 Image Dataset into the code, which contains seven classes of skin cancer lesions: Melanocytic nevi, Melanoma, Benign keratosis-like lesions, Basal, cell carcinoma, Actinic keratoses, Vascular lesions, Dermatofibroma. As numerical representation is required to be usable for machine learning/artificial intelligence model, the classes must first be encoded into categorical outcomes. The following model shows code that uses sckit-learn, a machine learning library, utilizing *LabelEncoder* from *sklearn.preprocessing* package module to encode categorical labels into numerical values.

```
# Write your code here to instantiate Label Encoder Object
le = LabelEncoder()
le.fit(metadata['dx'])
# Write your code here to fit the label encoder object into the outcome column
metadata['label'] = le.transform(metadata['dx'])
print("Classes:", list(le.classes_))
```

```
Classes: ['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']
```

```
# Write your code here to create new column to map the fitted outcome
metadata['label'] = le.transform(metadata['dx'])


metadata.sample(10)
```

	lesion_id	image_id	dx	dx_type	age	sex	localization	dataset	label
4191	HAM_0003067	ISIC_0032084	nv	follow_up	60.0	male	abdomen	vidir_molemax	5
3304	HAM_0000912	ISIC_0026232	nv	follow_up	45.0	female	abdomen	vidir_molemax	5
226	HAM_0000790	ISIC_0032654	bkl	histo	75.0	male	face	vidir_modern	2
3355	HAM_0006270	ISIC_0031606	nv	follow_up	55.0	female	lower extremity	vidir_molemax	5
9774	HAM_0005389	ISIC_0031012	akiec	histo	70.0	male	lower extremity	rosendahl	0
9779	HAM_0003146	ISIC_0029563	akiec	histo	85.0	male	upper extremity	rosendahl	0
5620	HAM_0003975	ISIC_0030098	nv	follow_up	40.0	female	lower extremity	vidir_molemax	5
8985	HAM_0000266	ISIC_0024933	nv	histo	45.0	female	lower extremity	rosendahl	5
8808	HAM_0005017	ISIC_0027461	nv	histo	75.0	female	back	rosendahl	5
1307	HAM_0004579	ISIC_0025603	mel	histo	75.0	male	trunk	vidir_modern	4

Figure 2. Sckit-learn code using LabelEncoder from sklearn.preprocessing package module

Once the metadata has been converted into numerical representation, we must then examine the experimental validity of the HAM10000 dataset in order to detect any potential biases that may occur during machine learning. To do this, a quick Exploratory Data Analysis (EDA) had to be conducted on the metadata. Exploratory Data Analysis (EDA) involves systematically examining and visualizing a dataset to understand its structure, identify patterns, detect anomalies, and gain insights into the underlying relationships among variables. It requires various plotting functions (*plot(), histplot()*) to create visualizations for different aspects of the metadata using the numerical values of the dataset in Figure 2. The following code shows the implemented kernel:

```
fig, axes = plt.subplots(2, 2, figsize=(40, 25))

# Plot 1 - Disease Type
ax1 = axes[0, 0]
value_counts_dx = metadata['dx'].value_counts()
value_counts_dx.plot(kind='bar', ax=ax1)
ax1.set_ylabel('Count', size=50)
ax1.set_title('Disease Type', size=50)
ax1.set_xticklabels(value_counts_dx.index, rotation=45, ha='right', fontsize=30)

# Plot 2 - Sex
ax2 = axes[0, 1]
metadata['sex'].value_counts().plot(kind='bar', ax=ax2, rot=0, fontsize=25)
ax2.set_ylabel('Count', size=50)
ax2.set_title('Sex', size=50)

# Plot 3 - Localization
ax3 = axes[1, 0]
value_counts_localization = metadata['localization'].value_counts()
value_counts_localization.plot(kind='bar', ax=ax3)
ax3.set_ylabel('Count', size=50)
ax3.set_title('Disease Localization', size=50)
ax3.set_xticklabels(value_counts_localization.index, rotation=45, ha='right', fontsize=25)

# Plot 4 - Age
ax4 = axes[1, 1]
sample_age = metadata[pd.notnull(metadata['age'])]
sns.histplot(sample_age['age'], kde=True, ax=ax4)
ax4.set_title('Age', size=50)
ax4.set_xlabel('Year', size=50)

plt.tight_layout()
plt.show()
```

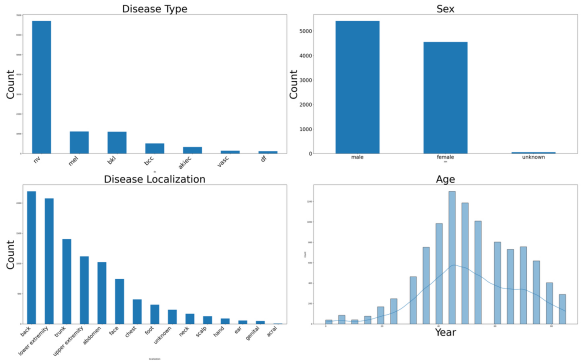


Figure 3. Code and tables used to examine the experimental validity of the HAM10000 dataset

As indicated by Figure 3, the extraction of the metadata using EDA allows a visual representation of the dataset. The gender and age distribution seems to be at balance, meaning that the data comes from a relatively balanced set of male and female with a variety of ages. However, the HAM10000 Image Dataset had unproportionately high occurrences (class imbalance) of the lesion type "Melanocytic Nevi" compared to other types, which was an unusual occurrence for medical datasets. Class imbalances occur when certain classes have significantly fewer instances than others, creating a risk for the machine learning model to be biased towards the majority classes.

To tackle this problem, class weights for different skin conditions needed to be defined and adjusted based on their frequency in the dataset. We assigned higher weights to less frequent conditions using an iterative method by implementing data structure queue with supportive function enqueue and dequeue, enabling the model to prioritize learning from underrepresented classes during training. This approach aims to enhance the model's ability to make accurate predictions across all skin conditions, contributing to a more

balanced and effective classification system.

```
label = ['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']

def estimate_weights_mfb(label):
    class_weights = np.zeros_like(label, dtype=float)
    counts = np.zeros_like(label)
    for i, l in enumerate(label):
        counts[i] = metadata[metadata['dx'] == str(l)]['dx'].value_counts()[0]
    counts = counts.astype(float)
    median_freq = np.median(counts)
    for i, l in enumerate(label):
        class_weights[i] = median_freq / counts[i]
    return class_weights

classweight = estimate_weights_mfb(label)

for i in range(len(label)):
    print(f"{label[i]}: {classweight[i]}")
```

Figure 4. Code used to define and adjust class weights of different skin conditions based on frequency

2.2. Standardization and Optimization of Dataset

To overcome the challenge of limited medical data and improve our model's learning, we used a technique called data standardization and data augmentation. Through PyTorch's function that combines multiple transformations into a single callable object, we adjusted the images' brightness, size, and colors using standard values from a large image database. Since all our images follow a similar pattern, the model will be able to learn at a faster rate with a simpler architecture. Next, we applied data augmentation to our training images. This involved creating variations of our images by flipping them horizontally and rotating them, which assists our model in becoming better at recognizing different skin conditions.

Other than augmenting and standardizing the image, we also carefully organized our dataset into distinct groups to ensure effective learning. This process involves creating three essential sets: the 'train' set, the 'validation' set, and the 'test' set for evaluating its ability to recognize new, unseen images. The following code defines a custom sampler, *StratifiedSampler*, used for stratified sampling of data. It takes *class_vector* as input, which represents the class labels of the data. The *test_size* parameter determines the proportion of data to be held out for testing. The *gen_sample_array* method generates stratified train-test splits based on the class distribution. The sampler ensures that each split has approximately the same distribution of classes as the original data. It then splits the dataset into training, validation, and test sets using these generated indices. Finally, it prints the number of images in each set, fully rotated and loaded into memory. This enables efficient and controlled feeding of data to the model during the training and evaluation phases.

Figure 5. Images of code and skin conditions arranged specifically to ensure effective learning

2.3 - Learning Approaches towards Data Models

2.3.1. CNN LeNet Model

The Convolutional Neural Network (CNN) LeNet is a class of neural networks designed specifically for image recognition tasks, inspired by the human visual system. The structure of LeNet compromises of several key components: convolutional layers for detecting visual features, max-pooling layers for spatial down-sampling, and fully connected layers for decision-making. The architecture is articulated within the *init* method, while the forward method defines how input data traverses through these layers [3].

The following code sets up the CNN model for artificial learning and incorporates a method for calculating the number of features within a flattened layer. Once the network is defined, an instance called *net* is created, and the entire structure is deployed to a specific computing device, often a Graphics Processing Unit (GPU) for enhanced computational efficiency.

```
num_classes = len(classes)
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, (5,5), padding=2)
        self.conv2 = nn.Conv2d(6, 16, (5,5))
        self.fc1   = nn.Linear(16*54*54, 120)
        self.fc2   = nn.Linear(120, 84)
        self.fc3   = nn.Linear(84, num_classes)
    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2,2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2,2))
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
    def num_flat_features(self, x):
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

net = LeNet()
net = net.to(device)
```

Figure 6. Code to set up CNN LeNet

Before initiating image classification training using the HAM10000 dataset, establishing the loss function and choosing an optimizer with the Cross-Entropy and the Adam functions are crucial. These components play critical roles in the training process, influencing how the model learns and adapts over time. The Cross-Entropy loss defines how much the model's predictions deviate from the ground truth (sample image), which help quantify the model's performance and utilizes the following formula:

$$H(p, q) = -\mathrm{E}_p[\log q]$$ [7]

The minimization of the Cross-Entropy loss indicates a higher accuracy on unseen images. This, in turn, steers the learning process towards producing probability distributions that align closely with the true class distributions in the training data. Defining the Adam optimizer before training ensures that the CNN follows a strategic path during optimization, adjusting its parameters effectively based on the observed gradients.

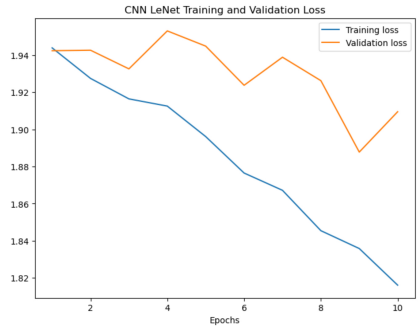
Afterwards, we can start to train the model. The provided code is a training loop for the CNN LeNet using PyTorch. Its primary goal is to train the network over 10 epochs on HAM10000 Image dataset and monitor its performance:

1. Initialization: The code sets the number of training epochs (*num_epochs*) and initializes lists to store training and validation metrics, including accuracy and loss.
2. Training Loop: For each epoch, the code iterates over batches of training data (*train_data_loader*), computes the loss, and performs backpropagation to update the model's parameters. Training metrics such as running loss and accuracy are calculated and loaded for each epoch.
3. Validation: After each epoch, the code evaluates the CNN model on a separate validation dataset (*validation_data_loader*) to assess

its performance on unseen data. Validation metrics, including loss and accuracy, are printed to gauge how well the model generalizes to new data.

4. Monitoring: The code keeps track of training and validation metrics over epochs, storing them in lists for later analysis.

The code then produces a table detailing the progress of validation loss to provide a learning overview of how accurately the CNN model gauged unseen samples, which is key for the final developments of the analysis in this model.




```
num_epochs = 10
accuracy = []
val_accuracy = []
losses = []
val_losses = []

start_time = time.time()

for epoch in range(num_epochs):
    running_loss = 0.0
    correct_total= 0.0
    num_samples_total=0.0
    for i, data in enumerate(train_data_loader):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        _, predicted = th.max(outputs, 1)
        b_len, corr = get_accuracy(predicted, labels)
        num_samples_total +=b_len
        correct_total +=corr
        running_loss += loss.item()

    running_loss /= len(train_data_loader)
    train_accuracy = correct_total/num_samples_total
    val_loss, val_acc = evaluate(net, validation_data_loader)

    print('Epoch: %d' %(epoch+1))
    print('Loss: %.3f  Accuracy:%.3f' %(running_loss, train_accuracy))
    print('Validation Loss: %.3f  Val Accuracy: %.3f' %(val_loss, val_acc))

    losses.append(running_loss)
    val_losses.append(val_loss)
    accuracy.append(train_accuracy)
    val_accuracy.append(val_acc)

end_time = time.time()
elapsed_time = end_time - start_time
```

Figure 7. Code and graph to provide metric insight to CNN model training

2.3.2 ResNet18 Model

Unlike the CNN model, the ResNet18 architecture is a pre-built model that uses residual blocks, where the input to a layer is combined with the output through a shortcut connection. This shortcut allows the network to directly learn the difference between the input and output of a layer. The residual connections enable the smooth flow of gradients during backpropagation, which could sometimes be absent in training very deep learning. [

To achieve this, we used the torchvision library to load the pre-trained ResNet18 model with weights from ImageNet via *pretrained=True*. The model's original fully connected layer is then adjusted to match the number of classes for the specific task (from 512 to num_classes). Finally, the model is transferred to a specified device (GPU or CPU) for efficient computation. This setup enables the use of ResNet18's feature extraction capabilities while tailoring it to the unique requirements of the classification task.

```
num_classes = len(classes)
net = torchvision.models.resnet18(pretrained = True)

net.fc = nn.Linear(512, num_classes)
net = net.to(device)
```

Figure 8. Code used to define the ResNet18 model

We then assessed the performance of the ResNet model using the validation set. Similar to our method with the CNN LeNet model, a separate validation dataset (*validation_data_loader*) is employed to assess the model's performance on unseen data, gauging validation loss and accuracy. The script also logs above-mentioned metrics, capturing the evolving dynamics of the model. It uses an iteration method through batches of validation data to make predictions using the model, calculating the loss and accumulating the number of correct predictions. Following that, a graph is produced based on the model's progress. These metrics provide and load outcomes into how well the model

generalizes to unseen data, and play a crucial role in the results.


```
num_epochs = 10
accuracy = []
val_accuracy = []
losses = []
val_losses = []

start_time = time.time()

for epoch in range(num_epochs):
    running_loss = 0.0
    correct_total= 0.0
    num_samples_total=0.0
    for i, data in enumerate(train_data_loader):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        _, predicted = th.max(outputs, 1)
        b_len, corr = get_accuracy(predicted, labels)
        num_samples_total +=b_len
        correct_total +=corr
        running_loss += loss.item()


    running_loss /= len(train_data_loader)
    train_accuracy = correct_total/num_samples_total
    val_loss, val_acc = evaluate(net, validation_data_loader)
```

Figure 9. Code and graph to provide metric insight to ResNet18 model training

3. Results & Discussion

3.1 Confusion Matrix

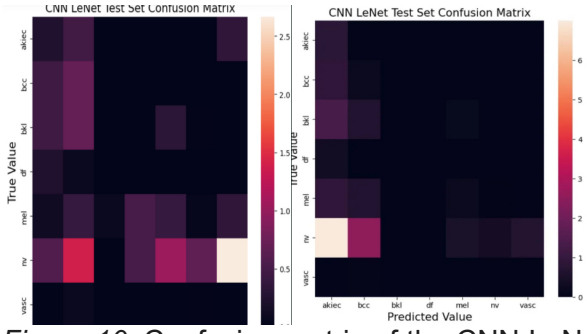


Figure 10. Confusion matrix of the CNN LeNet model

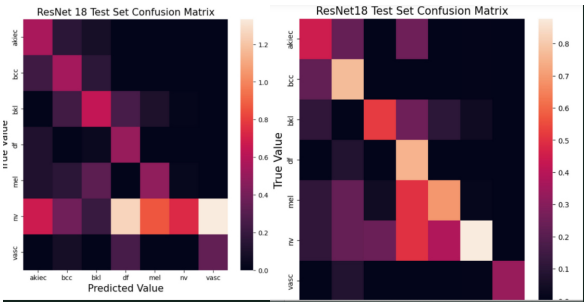


Figure 11. Confusion matrix of the ResNet18 model

The confusion matrices of the CNN LeNet and ResNet18 models can be seen in Figures 7 and 8. A confusion matrix is a table that summarizes the accuracy of the model's predictions, showing the number of true positive, true negative, false positive, and false negative predictions for each class. This gives a detailed breakdown of the model's classification accuracy and potential areas for misclassification across classes. The confusion matrix consists of diagonal and off-diagonal cells. The diagonal cells show the number of correct predictions in each class. The lighter or higher the value in these cells, the more accurately the model classifies instances from that class. Off-diagonal cells represent misclassification. Darker or lower values in these cells indicate instances where the model made errors [5]. To generate each confusion matrix, both CNN LeNet (net) and ResNet18 (net) are applied to a test dataset (*test_data_loader*), and the resulting predictions are compared with the actual labels.

Figure 7 shows that the CNN LeNet model performs best at identifying classes nv and bcc because they have the highest values in their diagonal cells, as indicated by their bright color in comparison to the rest of the class. However, many of the off-diagonal cells are darker or have lower values, indicating that the model made mistakes. As illustrated in Figure 8, ResNet18 outperforms the initial CNN

LeNet model, as evidenced by brighter diagonal cells. This demonstrates how the model more accurately classifies the image provided to it based on the actual conditions. The ResNet18 model also performed exceptionally well in classifying the nv class, as evidenced by the brightest cell color compared to the rest of the class. The brighter off-diagonal cells also indicate fewer mistakes were made in comparison to the CNN LeNet model. Overall, this shows that the ResNet18 model works more efficiently with fewer errors compared to the CNN LeNet model when classifying skin conditions.

3.2 Grad-CAM Visualizations

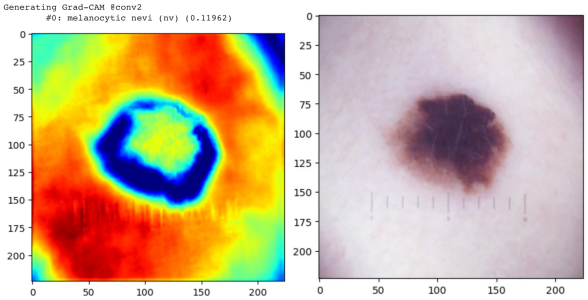


Figure 12. Grad-CAM visualizations of the CNN LeNet model

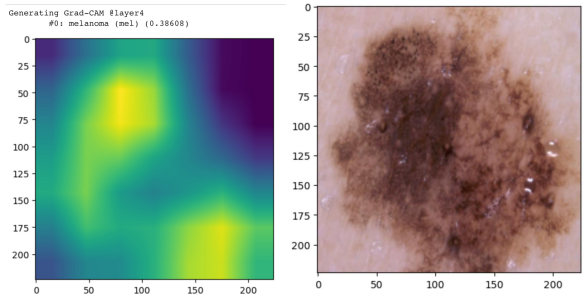


Figure 13. Grad-CAM visualizations of the ResNet18 model

The confusion matrices of the CNN LeNet and ResNet18 models can be seen in Figures 9 and 10. GradCAM is a visualization technique for interpreting the decision-making process of CNNs, particularly in image classification tasks. It generates heatmaps, highlighting regions of an input image that strongly influence the model's predictions. It helps highlight which features the model focuses on during classification. GradCAM enhances the explainability and trustworthiness of CNN-based image classification models [6]. The images on the left in both figures are Grad-CAM visualizations created for the specified target layer ("conv2") and target class. This visualization is created using the Grad-CAM technique on a heat map-style visual. It highlights the regions of the original image that made the most significant contribution to the model's prediction for the specified class. The warmer the color, the more important that region is to the model's ability to predict the image's classification.

The image on the right in both figures is the original image of the skin diseases used. Through comparing the ground truth label of the actual image with the prediction made in the Grad-CAM in both the CNN LeNet and ResNet18 models, both models are able to correctly predict the skin condition. As seen in both figures, the Grad-CAM is able to represent the shape of the skin conditions through the differences in colors.

4. Conclusion

It can be concluded from the data collected that ResNet18 produced higher overall accuracy on HAM10000 than CNN LeNet when used to create skin disease classifiers. This difference is most apparent when comparing the two models ' confusion matrix. It's clear that the ResNet18 model achieved more accurate results compared to its CNN LeNet model counterpart. This could be seen through the brighter diagonal cells, which indicate that the model classifies the image provided to it based on the actual

conditions more accurately. The CNN LeNet model performs visibly worse due to having significantly lower values for its diagonal cells. However, when comparing the Grad-CAM visualizations of both models, they performed similarly, as both models correctly predicted the skin conditions. However, ResNet18 was able to highlight the center parts of the image in warmer colors, showing that it can identify important 'zones' more efficiently and precisely, as seen in Figure 13. The CNN model on the other hand highlighted the surrounding skin, rather than the actual lesion, demonstrating a less precise identification. Therefore, ResNet18 has shown to have more accurate results overall on HAM10000 and should be used in comparison to CNN LeNet when classifying skin diseases. However, some limitations of the research could be the limited samples of medical data on the skin illnesses, which may lead to a narrower and less accurate scope of identifiable diseases by the training model. As seen in Figure 7 and Figure 9, both models experienced fluctuating rates of validation accuracy, despite the training loss decreasing. The lack of variety in the samples provided may have led to the models learning the samples, rather than the characteristics of identifying the illnesses. Furthermore, being able to achieve an adequate accuracy with the models would require a substantial amount of time and budget. Hopefully, ResNet architecture could be used more often as a learning model to assist healthcare professionals in a way that the public can access anywhere at all times (i.e. an app, etc.). This way, we can gain more information on skin diseases to tackle the public health concerns they cause worldwide.

5. Acknowledgment

We acknowledge and thank our supervisor, Kak Stefanus, who has guided us throughout the process of our research paper, including data collection. We would also like to extend our acknowledgement and thanks to Putranegara Riauwindu for helping to develop the code for this project.

6. References

[1] Yakupu, Aobuliaximu, et al. "The Burden of Skin and Subcutaneous Diseases: Findings from the Global Burden of Disease Study 2019." *Frontiers in Public Health*, U.S. National Library of Medicine, 17 Apr. 2023, www.ncbi.nlm.nih.gov/pmc/articles/PMC10149786/.

[2] Flohr, C., and R. Hay. "Putting the Burden of Skin Diseases on the Global Map." *Wiley Online Library*, British Journal of Dermatology, 5 Feb. 2021, onlinelibrary.wiley.com/doi/full/10.1111/bjd.19704.

[3] "What Is a Convolutional Neural Network?" *Convolutional Neural Network*, MathWorks, www.mathworks.com/discovery/convolutional-neural-network.html#:~:text=A%20convolutional%20neural%20network%20. Accessed 13 Apr. 2024.

[4] "ResNet: The Basics and 3 ResNet Extensions." *Datagen*, Datagen, 23 May 2023, datagen.tech/guides/computer-vision/resnet/#.

[5] Kulkarni, Ajay. "Confusion Matrix." *ScienceDirect*, Data Democracy, 2020, www.sciencedirect.com/topics/engineering/confusion-matrix#:~:text=A%20confusion%20matrix%20represents%20the,by%20model%20as%20other%20class.

[6] Reiff, Daniel. “Understand Your Algorithm with Grad-CAM.” *Medium*, Towards Data Science, 12 May 2022, towardsdatascience.com/understand-your-algorithm-with-grad-cam-d3b62fce353.

[7] Anisimova, Inara. “Cross-Entropy in ML.” *Medium*, UnpackAI, 4 January 2021, https://medium.com/unpackai/cross-entropy-loss-in-ml-d9f22fc11fe0#:~:text=Cross%2Dentropy%20can%20be%20calculated,*%20log(Q(x))