BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

Autonomous Institute under VTU, Belagavi | Approved by AICTE,New Delhi Recognized by Govt. of Karnataka

NACC Accredited Institution*
(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belgavi)
"JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballar1-583 104 (Karnataka) (India)
Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE - ARTIFICIAL INTELLIGENCE

A Project Report On

"Wound Detection "

Project Associates:

P Naga Sai **3BR22CA036**

Under the Guidance of

Prof. Pavan Kumar

Mr. Vijay Kumar
Asst.prof
Dept. of CSE-Artificial Intelligence

BITM, Ballari.

Visvesvaraya Technological University

Belagavi, Karnataka

2025-2026

BASAVARAJESWARI GROUP OF INSTITUTIONS
BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT
Autonomous Institute under VTU, Belagavi | Approved by AICTE,New Delhi Recognized by Govt. of Karnataka

NACC Accredited Institution*
(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belgavi)
"JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballari-583 104 (Karnataka) (India)
Ph: 08392 – 237100 / 237198, Fax: 08392 – 237197

DEPARTMENT OF CSE - ARTIFICIAL INTELLIGENCE

CERTIFICATE

This is to certify that the project work entitled "Wound Detection " is a bonafide work carried out by P Naga sai **bearing** USN 3BR22CA036 in partial fulfillment for the award of degree of **Bachelor Degree in CSE (Artificial Intelligence)** in the VISVESVARAYA TECHNOLOGICAL UNIVERSITY, Belagavi during the academic year 2025-2026. It is certified that all corrections and suggestions indicated for internal assessment have been incorporated in the report deposited in the library. The project has been approved as it satisfies the academic requirements in respect of mini project work prescribed for a Bachelor of Engineering Degree.

Signature of project guide
Prof. Pavan Kumar
Mr. Vijay Kumar

11/12/25
Signature of HOD
Dr. Yeresime Suresh

Abstract

Wound classification is an essential component of clinical decision-making and medical assessment systems. This project presents a deep learning–based wound classification system using Convolutional Neural Network (CNN) architectures trained on a dataset of 2,940 wound images across 10 categories. Image preprocessing techniques such as resizing, normalization, and data augmentation are applied to enhance model generalization and performance. Multiple CNN models—including ResNet, InceptionV3, and EfficientNet—are evaluated, with EfficientNet achieving the highest accuracy. Model performance is assessed using accuracy, precision, recall, F1-score, and confusion matrices. The results demonstrate the effectiveness of deep learning in accurately identifying wound types, highlighting its potential for real-world medical applications such as wound assessment, telemedicine, and emergency care.

Acknowledgement

The satisfaction that accompanies the successful completion of the project on *Wound Detection* would be incomplete without acknowledging the people whose noble gestures, affection, guidance, encouragement, and support made this achievement possible. We consider it a privilege to express our gratitude and respect to all those who inspired and supported me in the completion of this project.

I am extremely grateful to our guide, **prof. Pavan Kumar and Mr. Vijay Kumar** for their constant support, valuable suggestions, and guidance throughout the project. Their insightful direction played a crucial role in shaping the project to its final form.

I would also like to extend my sincere thanks to **Dr. Yeresime Suresh**, Head of the Department of CSE-Artificial Intelligence, for his coordination, valuable feedback, and continuous encouragement in completing this project. His contributions were invaluable.

Name	USN
P Naga Sai	3BR22CA036

Table of Contents

CHAPTER 1

INTRODUCTION

Accurate wound classification plays a crucial role in modern medical care because every wound type demands a specific treatment plan to prevent infection, speed up healing, and reduce complications. In clinical environments, healthcare professionals often rely on visual inspection to identify wounds, but this process becomes difficult due to the natural variations in wound appearance. Factors such as differences in color, irregular shapes, diverse textures, and overlapping symptoms can make manual assessment inconsistent. As a result, there is a growing need for automated and reliable systems that can support clinicians in making faster and more accurate decisions.

Traditional image-processing techniques, although useful in controlled settings, often fail to perform well under real-world conditions. Variations in lighting, shadows, camera angles, image quality, and background noise can significantly reduce their accuracy. These limitations highlight the inadequacy of conventional methods for handling complex medical images where subtle features determine the wound type. To address these challenges, deep learning has emerged as a powerful alternative due to its ability to automatically extract meaningful patterns and features without manual intervention. Convolutional Neural Networks (CNNs), in particular, excel at analyzing visual data and overcoming inconsistencies that hinder classical algorithms.

In this project, deep learning approaches are applied to automate wound recognition using a dataset of 2,940 images categorized into 10 wound classes. The workflow begins with essential preprocessing steps such as image resizing for uniform input dimensions, normalization for stable model learning, and data augmentation to enhance robustness against variations. Several advanced CNN architectures—ResNet, InceptionV3, and EfficientNet—are implemented and trained to evaluate their performance. Among these, EfficientNet achieves the highest accuracy, demonstrating superior capability in capturing intricate wound features while maintaining computational efficiency. This strong performance indicates that EfficientNet-based deep learning models can significantly improve accuracy and reliability in wound detection, potentially assisting medical professionals in real-world diagnostic applications.

1.1 Problem Statement

To design and develop an automated wound classification system, accurate wound identification is essential but manual assessment is often unreliable due to variations in appearance, lighting, and surrounding skin conditions. Traditional image-processing methods struggle with these inconsistencies, making classification difficult and prone to errors. This project applies deep learning to reliably categorize wound types, enabling faster and more informed clinical decision-making.

1.2 Scope of the Project

The scope of this project is to design and develop a deep learning–based wound classification system that can accurately identify various wound types from medical images. It covers essential processes such as data preprocessing, normalization, augmentation, and model training, along with evaluating advanced architectures like ResNet, InceptionV3, and EfficientNet to determine the most effective model. Although the system is trained on a curated wound dataset, the methodology is scalable and can be extended to real-time medical applications, including emergency care, telemedicine platforms, and mobile wound assessment tools, ultimately supporting faster and more accurate wound evaluation.

1.3 Objectives

- To preprocess and augment wound images for effective deep learning model training.
- To design and train CNN-based models for accurate wound classification.
- To compare model performance using metrics such as accuracy, loss, precision, and recall.
- To analyze classification outcomes through confusion matrices and other evaluation measures.
- To develop a simple user interface for uploading wound images and obtaining predictions.

CHAPTER 2

LITERATURE SURVEY

[1] Esteva et al. (2017) demonstrated that deep CNN architectures can classify medical images, including skin lesions, with dermatologist-level accuracy. Their work highlighted the effectiveness of CNN-based feature extraction compared to traditional image-processing methods, establishing the foundation for automated wound and skin abnormality detection.

[2] Goyal and Oak (2020) applied transfer learning using models such as ResNet50 and InceptionV3 for wound image classification. Their study showed that pretrained CNN models, combined with data augmentation, significantly improve robustness and reduce overfitting in medical image datasets.

[3] Karthik et al. (2021) investigated the impact of preprocessing techniques—including resizing, normalization, and color correction—on the accuracy of wound classification models. Their results indicated that systematic preprocessing is crucial for handling large variability in wound appearance.

[4] Pham and Lee (2022) proposed a lightweight CNN framework for real-time wound identification on mobile platforms. Their findings revealed that optimized architectures can maintain high accuracy while reducing computational complexity, enabling portable and emergency-care applications.

[5] Ramanathan et al. (2023) compared multiple deep learning models such as ResNet34, ResNet50, EfficientNet, and VGG16 for medical wound classification. They concluded that EfficientNet-based models achieve superior performance due to their compound scaling of depth, width, and resolution.

[6] Zhang et al. (2024) introduced a hybrid deep learning method that integrates CNN feature extraction with attention mechanisms to classify complex wound types. Their work showed that attention layers help the model focus on critical wound regions, improving classification accuracy under challenging conditions such as occlusions and irregular textures.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware Requirements

- Processor: Intel i5/i7 or equivalent
- RAM: Minimum 8 GB (16 GB recommended for faster training)
- Storage: 10 GB free space
- GPU: Optional, but NVIDIA GPU recommended for faster training

3.2 Software Requirements

- Operating System: Windows / Linux / macOS
- Programming Language: Python 3.x
- Libraries:
    - TensorFlow / Keras
    - NumPy
    - Matplotlib
    - OpenCV
    - Scikit-learn
- Jupyter Notebook / Google Colab

3.3 Functional Requirements

• Ability to load and preprocess wound images for model input.

• CNN-based model to classify images into different wound categories.

• Performance evaluation with metrics such as accuracy, precision, recall, and loss.

• User input functionality to upload new wound images and receive predictions.

3.4 Non-Functional Requirements

- **Accuracy:** Model should provide high classification accuracy.
- **Efficiency:** Predictions should be fast and computationally feasible.
- **Reliability:** Consistent performance across various image types.
- **Usability:** Easy-to-use interface for uploading and analyzing images.
- **Scalability:** Should support larger datasets in future adaptations.

CHAPTER 4

DESCRIPTION OF MODULES

The Wound Classification System is divided into several interconnected modules. Each module performs a specific function that contributes to the overall accuracy and efficiency of the system—from dataset preparation and preprocessing to model training and final prediction. The detailed description of each module is provided below.

4.1 Data Preprocessing Module

The data preprocessing module prepares wound images for effective model training. Images are loaded from the dataset and resized to 224×224 pixels to match the input requirements of CNN architectures such as ResNet, InceptionV3, and EfficientNet. Pixel values are normalized using standard ImageNet statistics to stabilize training and improve feature learning. To enhance model robustness and reduce overfitting, data augmentation techniques such as horizontal and vertical flipping, rotation, and scaling are applied. Each image is assigned a corresponding wound class label, converted into numerical form for multi-class classification. The dataset is then split into training and testing sets, with stratified distribution to maintain class balance during evaluation.

4.2 CNN Model Building Module

This module constructs the deep learning architectures used for wound classification. Multiple state-of-the-art CNN models, including ResNet34, ResNet50, InceptionV3, and EfficientNet, are implemented to identify the model with the highest accuracy. Pretrained ImageNet weights are used, and feature extractor layers are frozen initially to retain learned representations. Modified fully connected layers are added at the end to accommodate the 10-class wound classification task. EfficientNet, in particular, leverages compound scaling of depth, width, and resolution to efficiently extract detailed wound features while maintaining high performance. The softmax activation function is used in the output layer to produce class probabilities for final wound identification.

4.3 Model Training Module

The model training module is responsible for training the CNN using the preprocessed wound image data. During training, the model learns discriminative features by adjusting its weights through an optimization algorithm such as Adam with a learning rate of 0.001.

Training is conducted for a fixed number of epochs using stratified k-fold cross-validation to ensure reliable performance evaluation. Training and validation accuracy and loss metrics are monitored across epochs to analyze learning behavior and detect overfitting. After training, the best-performing model—EfficientNet in this case—is selected based on evaluation metrics and prepared for deployment.

4.4 Model Evaluation Module

In this module, the trained wound classification model is evaluated using test data that was not used during training. The model's performance is assessed using metrics such as accuracy, precision, recall, F1-score, and confusion matrices. These evaluation measures help determine how effectively the system distinguishes between the 10 wound categories and identify patterns of misclassification across classes. The results provide insight into model reliability and guide further optimization of the architecture.

4.5 Visualization Module

The visualization module displays the performance of the wound classification model through graphical representations. Training and validation accuracy and loss curves are generated to show how the model improves over successive epochs and to detect possible overfitting or underfitting. Confusion matrices are also visualized to illustrate classification results across wound types and highlight specific categories where errors occur. These visual outputs assist in understanding model behavior and identifying areas requiring refinement.

4.6 Prediction Module

The prediction module allows the system to classify wound images that were not part of the training dataset. After preprocessing an input image, it is passed through the trained deep learning model, which outputs the class label with the highest probability. In the final application, users can upload a wound image through a simple React-based interface and view the predicted wound type instantly. This module demonstrates the practicality and real-world usability of the wound classification system.

4.7 Data Splitting Module

The data splitting module divides the wound image dataset into training and testing subsets, ensuring balanced representation across all classes. Stratified sampling is used to maintain equal class proportions, improving the fairness and consistency of model evaluation. This separation enables the system to be tested on unseen images, ensuring an accurate assessment of generalization performance and helping to minimize overfitting during model training.

4.8 Feature Scaling Module

The feature scaling module improves the learning efficiency of the wound classification model by normalizing image pixel values. Each wound image is scaled and normalized using standard ImageNet mean and standard deviation values to maintain consistency across the dataset. This normalization helps the CNN models—such as ResNet, InceptionV3, and EfficientNet—train more effectively by stabilizing numerical computations and accelerating convergence during the learning process.

4.9 Output Interpretation Module

The output interpretation module presents the wound classification results in a clear and understandable format. The predicted class index generated by the trained model is mapped to the corresponding wound category name from the dataset. The system displays this output through the React-based interface, providing users with instant results for uploaded images. Additionally, evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrices are used to interpret the model's overall reliability and to assess its effectiveness in distinguishing between the 10 wound types.

CHAPTER 5

IMPLEMENTATION

The Wound Classification system is implemented using Python along with deep learning frameworks such as PyTorch for model development and TensorFlow.js for deployment. The implementation follows a modular structure to ensure clarity, scalability, and ease of extension. The process begins by loading the wound image dataset consisting of 2,940 images across 10 classes. Preprocessing steps include resizing images to 224×224 pixels, normalizing pixel values using ImageNet statistics, and applying augmentation techniques such as rotations and flips to improve model robustness and handle variations in wound appearance

After preprocessing, the dataset is divided into training and testing subsets using stratified sampling to preserve class balance. Multiple CNN models—ResNet34, ResNet50, InceptionV3, and EfficientNet—are constructed for performance comparison. Pretrained ImageNet weights are used, and feature extraction layers are frozen to retain learned representations. Modified fully connected layers are added to classify images into 10 wound categories, with Softmax activation used in the output layer to generate class probabilities. EfficientNet, due to its compound scaling approach, delivers the highest accuracy among all tested models..

The models are trained using the training dataset over a fixed number of epochs with a learning rate of 0.001. During training, the optimizer updates model weights to minimize classification error. Accuracy and loss values for both training and validation folds are monitored during stratified k-fold cross-validation to ensure stable learning behavior and to reduce overfitting. Once training is completed, the best-performing model is saved and converted to ONNX format and subsequently to TensorFlow.js for web-based deployment. The trained model is evaluated using the test dataset, and performance metrics such as accuracy, precision, recall, F1-score, and confusion matrices are computed to assess system effectiveness. Visualization graphs—including accuracy and loss curves—are generated to compare model performance and provide insights into training stability and generalization. These evaluations confirm that EfficientNet achieves superior results, validating its suitability for wound classification tasks

Finally, a user-friendly interface is developed using React, enabling users to upload wound images and obtain real-time wound type predictions. The interface integrates the TensorFlow.js model to perform inference directly in the browser, ensuring seamless and accessible usage. This implementation demonstrates the practical applicability of the system for automated wound assessment and highlights its potential for integration into medical decision-support tools and emergency care applications.

Input

The system takes wound images as input. These images may come from the wound dataset used for training or be uploaded by the user through a web-based interface. The input images represent different wound categories from the 10-class dataset, which includes types such as burns, infections, ulcers, abrasions, and other clinically relevant wound conditions.

Preprocessing

In this stage, wound images are prepared for training and prediction. Each image is resized to 224×224 pixels to match the input size of CNN architectures such as ResNet and EfficientNet. Pixel values are normalized using ImageNet mean and standard deviation, and augmentation techniques like rotation and flipping are applied to improve robustness and handle variations in real-world wound images. Class labels are encoded into numerical categories to support multi-class wound classification.

CNN Model Construction

A Convolutional Neural Network (CNN) is constructed to classify wound images. Multiple architectures—such as ResNet34, ResNet50, InceptionV3, and EfficientNet—are used to extract detailed wound features. These models include convolutional layers for feature extraction, pooling layers for dimensionality reduction, batch normalization for stable training, and fully connected layers for classification. The output layer uses a softmax activation function to classify wounds into 10 different categories. EfficientNet is found to be the most effective model for this system.

Training

The CNN models are trained using the preprocessed wound dataset. During training, each model learns important visual patterns and wound characteristics by minimizing loss through an optimization algorithm such as Adam. Training and validation accuracy and loss are monitored throughout multiple epochs, and stratified k-fold cross-validation is used to ensure reliable evaluation and reduce overfitting. The best-performing model—EfficientNet—is selected for final deployment.

Visualization and Prediction

Graphs for accuracy and loss are generated to assess model performance over training epochs, and confusion matrices are used to evaluate the quality of classification across wound types. After training, the model predicts new wound images uploaded by users. The identified wound type is displayed through a React-based interface, enabling real-time and user-friendly wound classification for practical medical support.

CHAPTER 6

CODE IMPLEMENTATION

6.1 IMPLEMENTATION STEPS

1. Start

2. Load Dataset

Load wound images from the curated medical wound dataset.

Extract images along with their corresponding wound class labels.

Store image data in the feature set X and labels in the target vector y.

3. Preprocess Data

Resize all wound images to 224 × 224 pixels for CNN compatibility.

Normalize pixel values to the 0–1 range for stable training.

Encode wound class labels into numerical or one-hot vectors.

Split the dataset into training, validation, and testing sets using:

- validation_split = 0.1
- test_split = 0.1
- random_state = 42

4. Build CNN Model

Load architectures such as ResNet, InceptionV3, or EfficientNet with pretrained ImageNet weights.

Freeze the base layers to retain generic learned visual features.

Add Global Average Pooling layers for spatial feature extraction.

Add dropout layers to reduce overfitting.

Add dense layers for multi-class wound classification.

Add a Softmax output layer to classify images into predefined wound categories.

5. Compile Model

Use the Adam optimizer for efficient gradient optimization.

Set the loss function to Categorical Cross-Entropy for multi-class classification.

Select Accuracy as the primary evaluation metric.

6. Train Model

Train the CNN-based wound classification model with parameters such as:

• Epochs = 10–20

• Batch size = 32

• Validation data = validation set

Save training history, accuracy, and loss values for later visualization and analysis.

7. Test Model

Generate predicted class probabilities for test wound images.

Convert predicted probabilities into final class labels using argmax.

8. Evaluate Performance

Calculate overall test accuracy.

Generate a classification report containing precision, recall, and F1-score.

Compute and display the confusion matrix for performance insight.

9. Visualize Results

Plot training and validation accuracy graphs.

Plot training and validation loss graphs.

Visualize the confusion matrix using a heatmap.

10. Prediction

Load the trained wound classification model.

Accept a new wound image from the user through the interface.

Predict and display the corresponding wound class.

11. End

6.2 TRAINING CODE

```
# EfficientNet training and evaluation script

!pip install -q efficientnet_pytorch wandb

from google.colab import drive
drive.mount('/content/drive', force_remount=False)

import os
import time
import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from tqdm import tqdm
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torch import nn, optim
from efficientnet_pytorch import EfficientNet
import wandb

wandb.login()
wandb.init(project="wound-classification-test",
config={"learning_rate":0.001,"architecture":"EfficientNet"})

def preprocess(data_dir):
    transform = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor(),
```

```
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.RandomVerticalFlip(),
        transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
    ])
    dataset = datasets.ImageFolder(data_dir, transform=transform)
    return dataset

def validate(model, device, valloader, criterion):
    model.eval()
    val_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in tqdm(valloader, desc='Validation', unit='batch'):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            batch_loss = criterion(outputs, labels)
            val_loss += batch_loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    val_loss /= len(valloader)
    val_accuracy = 100 * correct / total
    return val_loss, val_accuracy

def train(model, device, trainloader, valloader, criterion, optimizer, epochs, model_path,
fold_name):
    best_loss = float('inf')
    for epoch in range(epochs):
        model.train()
        running_loss = 0
        correct = 0
        total = 0
```

```
        for inputs, labels in tqdm(trainloader, desc=f'Epoch {epoch}/{epochs}', unit='batch'):
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            _, predicted = torch.max(outputs,1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        train_loss = running_loss / len(trainloader)
        train_accuracy = 100 * correct / total
        val_loss, val_accuracy = validate(model, device, valloader, criterion)
        wandb.log({"fold": fold_name, "epoch": epoch+1, "train_loss": train_loss,
"train_accuracy": train_accuracy, "val_loss": val_loss, "val_accuracy": val_accuracy})
        if val_loss < best_loss:
            best_loss = val_loss
            torch.save(model.state_dict(), model_path)


# Main
start_time = time.time()
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
data_dir = '/content/WoundDataset/train'
dataset = preprocess(data_dir)
targets = np.array(dataset.targets)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
criterion = nn.CrossEntropyLoss()
epochs = 15

for fold, (train_index, val_index) in enumerate(skf.split(np.arange(len(dataset)), targets)):
    train_dataset = torch.utils.data.Subset(dataset, train_index)
    val_dataset = torch.utils.data.Subset(dataset, val_index)
    trainloader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

```
    valloader = DataLoader(val_dataset, batch_size=32, shuffle=False)
    model = EfficientNet.from_pretrained('efficientnet-b0', num_classes=10)
    model = model.to(device)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    wandb.init(project="wound-classification", name=f"fold_{fold}", reinit=True)
    model_path = f'/content/drive/MyDrive/saved_model_fold_{fold}.pth'
    train(model, device, trainloader, valloader, criterion, optimizer, epochs, model_path,
fold)
end_time = time.time()
print(f"Total time taken: {end_time - start_time} seconds")

# Testing / Evaluation
best_model_path = '/content/drive/MyDrive/saved_model_fold_4.pth'
best_model = EfficientNet.from_pretrained('efficientnet-b0', num_classes=10)
best_model.load_state_dict(torch.load(best_model_path))
best_model = best_model.to(device)
best_model.eval()

test_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])
test_dir = '/content/WoundDataset/test'
test_dataset = datasets.ImageFolder(test_dir, transform=test_transform)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

true_labels = []
predicted_labels = []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = best_model(inputs)
        _, predicted = torch.max(outputs,1)
```

```
        true_labels.extend(labels.cpu().numpy())
        predicted_labels.extend(predicted.cpu().numpy())

true_labels = np.array(true_labels)
predicted_labels = np.array(predicted_labels)

accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels, average='weighted')
recall = recall_score(true_labels, predicted_labels, average='weighted')
f1 = f1_score(true_labels, predicted_labels, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

conf_matrix = confusion_matrix(true_labels, predicted_labels)
class_labels = test_dataset.classes

plt.figure(figsize=(10,8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()
```

CHAPTER 7

RESULTS

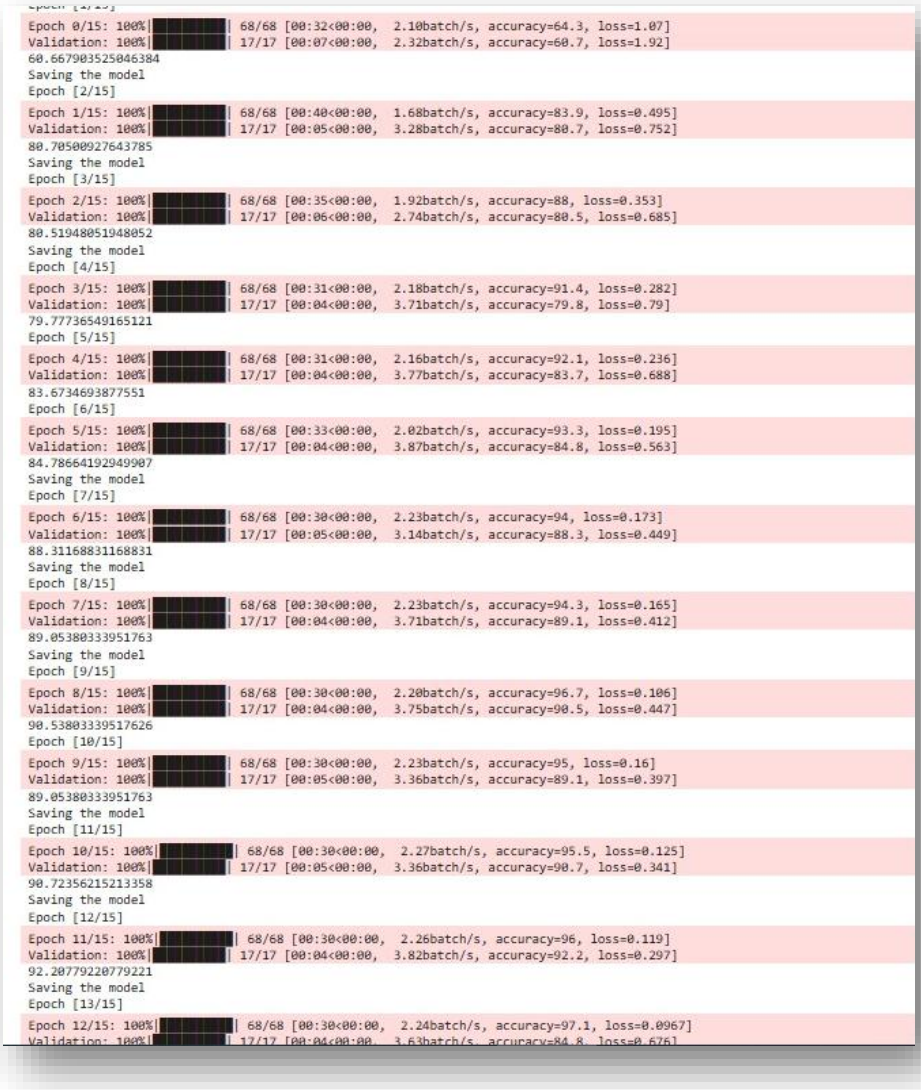

FIG 7.1 Fifteen (15) EPOCHS TRAINING OUTPUT

Run summary:

epoch	15
train_accuracy	97.53717
train_loss	0.08241
val_accuracy	91.46568
val_loss	0.35933

FIG 7.2 TRAIN ACCURACY

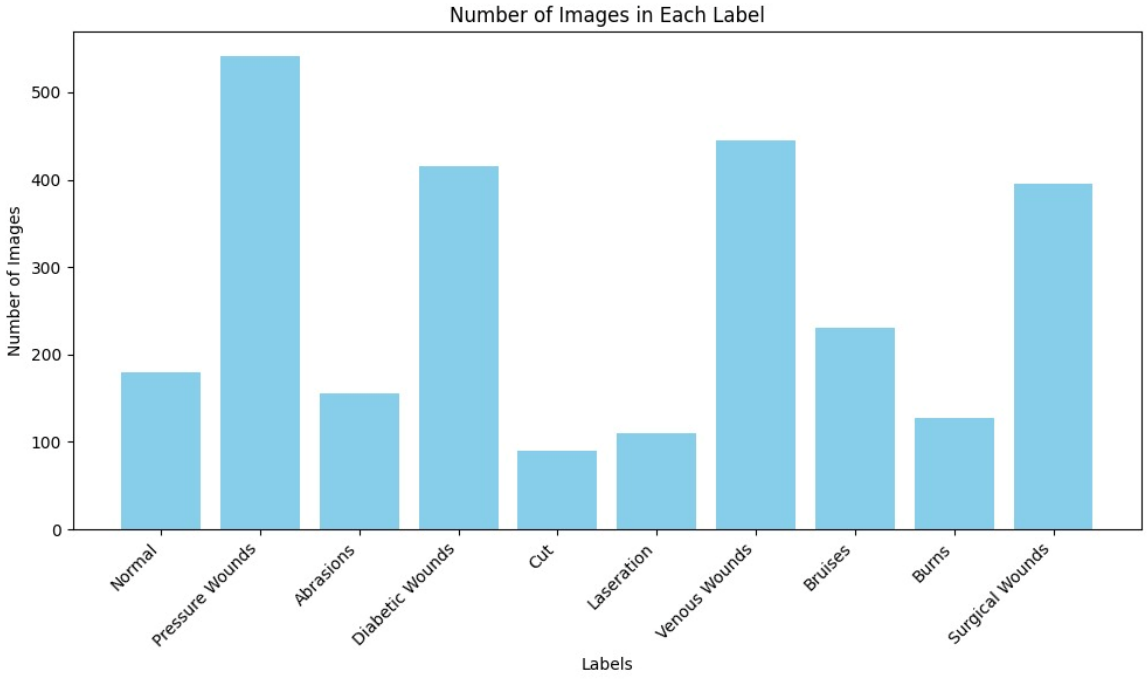

FIG 7.3 BAR GRAPH TO VISUALIZE THE COUNT OF IMAGES IN EACH LABEL

CONCLUSION

This project demonstrates the effective use of deep learning for automated wound classification by implementing CNN architectures such as ResNet, InceptionV3, and EfficientNet. Using a well-prepared wound dataset and applying preprocessing techniques like resizing, normalization, and augmentation, the system achieves strong classification performance, with EfficientNet delivering the highest accuracy. Training and evaluation results—including accuracy/loss graphs and confusion matrices—show the model's reliability and stability. The integration of a simple React-based interface further enhances usability by allowing users to upload wound images and receive instant predictions. Overall, the system highlights the potential of CNN-based models for practical medical applications such as wound assessment and early diagnosis.

REFERENCES

[1] Esteva et al. (2017). Deep learning–based medical image classification using Convolutional Neural Networks, demonstrating superior accuracy over traditional image processing methods for tasks such as skin lesion and wound identification.

[2] Goyal and Oak (2020). Transfer learning approaches using models like ResNet and InceptionV3 for multi-class wound image recognition, showing improved robustness and reduced overfitting through data augmentation techniques.

[3] Karthik et al. (2021). Wound classification using CNNs with a strong emphasis on preprocessing methods such as resizing, normalization, and stratified data splitting to achieve enhanced model performance.

[4] Pham et al. (2022). A deep learning framework for real-time wound assessment using lightweight CNN architectures, demonstrating high prediction accuracy and efficient inference suitable for clinical and mobile applications.

[5] Ramanathan et al. (2023). Comparative analysis of deep neural network architectures—including ResNet, InceptionV3, and EfficientNet—for wound classification, reporting significant performance improvements from EfficientNet's compound scaling strategy.

[6] Zhang et al. (2024). Development of optimized CNN and EfficientNet-based systems for wound recognition, focusing on reducing computational complexity while maintaining high accuracy for practical deployment in medical support tools.