CISC3023 Machine Learning Project Report

Image-Based Wound Location Prediction: A Machine Learning Approach

by

Li Linhao DC 427351

Time frame: Nov 19 2024 - Nov 21 2024

Teacher
Prof. Dr. Chen Long

澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

Faculty of Science and Technology
University of Macau

1 Introduction

The accurate prediction of wound locations in animal images is crucial for medical research and diagnosis. This report explores the use of machine learning models—specifically Random Forest, Ridge Regression, and Linear Regression—to predict wound locations in images from animal models. The study investigates the effect of dimensionality reduction techniques, such as Principal Component Analysis (PCA), on model performance. The objective is to assess how these models perform in predicting the coordinates and sizes of wounds and to identify the challenges faced when dealing with complex and irregularly shaped wounds. Performance metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2 scores are used to evaluate the models, and insights from the visualizations of predictions help further analyze model behavior.

2 Two Types of Machine Learning Models Used. Why I Chose Them?

For this project, I selected **Linear Regression** and **Random Forest** as the two machine learning models to predict the wound location in animal model images. These models were chosen for the following reasons:

2.1 Linear Regression

Linear Regression is a straightforward and interpretable model that works well for predicting continuous values. In the context of this project, the goal is to predict four continuous values: the coordinates of the wound center (x, y) and the dimensions of the wound (x_width, y_width). Linear Regression provides a simple approach to model these relationships by finding the best-fitting linear relationship between the input features (extracted from the images) and the output variables (wound location). Its advantages include:

- **Simplicity:** As a relatively simple model, Linear Regression can serve as a baseline to compare the performance of more complex models.
- **Efficiency:** It is computationally efficient, making it suitable for training on the available dataset.

2.2 Random Forest

Random Forest is an ensemble learning method that builds multiple decision trees to make predictions. It is particularly effective for handling complex, high-dimensional data like images. Random Forest was chosen for the following reasons:

- **Non-linear Relationships:** Unlike Linear Regression, Random Forest can capture non-linear relationships between input features and the output variables. Given the complexity of image data and the relationship between pixel values and wound locations, Random Forest is better suited to model these interactions.
- **Feature Importance:** Random Forest can provide insights into the relative importance of different features, helping to identify which parts of the image (or image features) contribute most to accurate predictions.
- **Robustness:** It is less prone to overfitting than individual decision trees, as it averages the results of many trees, leading to more stable predictions.
- **Handling Complex Data:** Random Forest is known for performing well with high-dimensional data, such as the pixel values from the images in this project, without requiring extensive feature engineering.

By using both models, I aimed to leverage the simplicity of Linear Regression while also benefiting from the flexibility and robustness of Random Forest. This combination allows for a comprehensive evaluation of the performance of different machine learning approaches in predicting the wound locations.

3 How do you handle the prediction of 4 outputs?

In this project, the prediction of 4 outputs (x, y, x_width, y_width) is handled by training a machine learning model (Linear Regression or Random Forest) to predict all 4 values simultaneously for each image. Here's how it's done:

3.1 Data Preparation

- The dataset consists of images with 4 corresponding output values: x (x-coordinate), y (y-coordinate), xw (x-width), and yw (y-width).
- Each image is flattened and normalized to serve as the input for the model.
- The outputs are stored together in a single array, named trainOutputs, with each column corresponding to one of the four values (e.g., trainOutputxi, trainOutputyi, trainOutputxw, trainOutputyw).

3.2 Model Training

- The images (flattened and normalized) are used as inputs, and the 4 output values are combined into a single output array (trainOutputs). This output array is transposed so that each row corresponds to a single set of outputs (x, y, xw, yw) for each image.

- A regression model, such as Linear Regression or Random Forest, is trained to predict these 4 values at once.

3.3 Model Prediction

- After training, the model predicts all 4 outputs for each input image in the test set. The predicted values are returned as a set of 4 numbers corresponding to x, y, xw, and yw.
- During evaluation, the predicted outputs are compared to the ground truth values, and errors (e.g., MSE) are calculated for each dimension (x, y, xw, yw).

In summary, the model outputs are not predicted individually but rather as a set of 4 values for each image, with the model being trained to predict these 4 coordinates together.

4 How do you prepare your data to get a better result?

Data preparation plays a critical role in the performance of machine learning models. For this project, several steps were taken to ensure the data was well-prepared for training, and these include normalization, flattening, splitting, dimensionality reduction, and hyperparameter tuning.

4.1 Normalizing Pixel Values

The image data in this project contains pixel values in the range [0, 255]. To improve the performance of machine learning models, it is essential to normalize these values to a range of [0, 1]. This ensures that all input features are on a similar scale, which is especially important for models that are sensitive to feature magnitude.

```
trainImagesNormalized = np.array(trainImages) / 255.0
```

4.2 Flattening the Images

After normalization, the images are flattened to 1D arrays. This transformation is required because most machine learning models, such as Random Forest Linear Regression, expect the input to be a vector rather than a matrix (2D image).

```
trainImages_flattened =\
    [img.flatten() for img in trainImagesNormalized]
trainImages_flattened = np.array(trainImages_flattened)
```

4.3 Splitting the Data into Training and Validation Sets

To evaluate the model's performance and prevent overfitting, the data is split into training and validation sets. The training set is used to train the model, while the validation set helps assess how well the model generalizes to unseen data. The split is done using an 80-20 ratio, with 80% of the data for training and 20% for validation.

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(
    trainImages_flattened, trainOutputs,
    test_size=0.2, random_state=42
)
```

4.4 Feature Scaling

Some models, like Ridge Regression, perform better when the features are scaled to have a mean of 0 and a standard deviation of 1. **StandardScaler** is applied to standardize the features for both the training and validation sets.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

4.5 Dimensionality Reduction with PCA

To potentially improve model performance and reduce computation time, **Principal Component Analysis (PCA)** is used to reduce the dimensionality of the input features. By keeping only the most significant components, we can retain the variance in the data while reducing the number of features fed into the model.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=20)  # Try reducing dimensions
X_train_reduced = pca.fit_transform(X_train)
X_val_reduced = pca.transform(X_val)
```

4.6 Hyperparameter Tuning with GridSearchCV

To improve model performance, hyperparameter tuning is applied using **GridSearchCV**. This method performs an exhaustive search over a specified parameter grid, evaluating different combinations of parameters using cross-validation. The

goal is to find the optimal set of hyperparameters that results in the best performance.

For example, **RandomForestRegressor** is tuned with parameters like n_estimators, max_depth, min_samples_leaf, and min_samples_split. GridSearchCV helps identify the best combination for these parameters based on cross-validation performance.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

param_grid = {
    'n_estimators': [15, 20, 25, 50],
    'max_depth': [1, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 8],
    'min_samples_split': [2, 5, 10, 20],
}
rf_grcv_model = RandomForestRegressor(random_state=42, n_jobs=-1)
grid_search = GridSearchCV(
    estimator=rf_grcv_model,
    param_grid=param_grid,
    # Negative MSE as scoring metric
    scoring='neg_mean_squared_error',
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1  # Parallel processing
)
# Fit the grid search to the data
grid_search.fit(X_train_reduced, y_train)
```

And, **Ridge Regression** is trained with different *alpha* using GridSearchCV.

```
from sklearn.linear_model import Ridge

ridge_model = Ridge()
# Range of alpha values
param_grid = {'alpha': [0.01, 0.1, 1, 10, 100, 1000]}
grid_search =\
GridSearchCV(ridge_model, param_grid,
    cv=5, scoring='neg_mean_squared_error', verbose=1)
grid_search.fit(X_train_scaled, y_train)
```

5 How to Choose the Parameters of Different Models?

Selecting the right parameters is crucial for optimizing the performance of machine learning models. For this project, we applied a systematic approach to choose the parameters for the Random Forest Regressor and Ridge Regression models using techniques like grid search and cross-validation.

5.1 Random Forest Regressor

Random Forest is a versatile and robust ensemble learning method. To ensure the model performs well, we tuned several hyperparameters:

- **n_estimators:** This parameter defines the number of decision trees in the forest. A larger value can improve performance but increases computation time. We explored values of 15, 20, 25, and 50 to find a balance between accuracy and computational cost.
- **max_depth:** This controls the maximum depth of each tree. Shallower trees reduce overfitting but might underfit the data. We tested depths of 1, 5, 10, and 15 to identify the optimal complexity.
- **min_samples_leaf:** This is the minimum number of samples required to be a leaf node. Smaller values allow the model to capture finer patterns but risk overfitting. We evaluated values of 1, 2, 4, and 8.
- **min_samples_split:** This is the minimum number of samples required to split an internal node. Higher values encourage simpler models. We examined values of 2, 5, 10, and 20.

Tuning Process:

We used **GridSearchCV** to systematically explore all combinations of these parameters. A 5-fold cross-validation was employed to ensure the model generalizes well to unseen data. The performance metric used was **negative mean squared error (MSE)** to minimize prediction errors.

```
param_grid = {
    'n_estimators': [15, 20, 25, 50],
    'max_depth': [1, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 8],
    'min_samples_split': [2, 5, 10, 20],
}
grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42, n_jobs=-1),
```

```
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=5,
    verbose=2,
    n_jobs=-1
)
grid_search.fit(X_train, y_train)
```

Optimal Parameters:

GridSearchCV identified the combination of parameters that resulted in the lowest MSE during cross-validation, which were used for the final model.

5.2 Ridge Rgression

Ridge Regression introduces L2 regularization to reduce overfitting by penalizing large coefficients. The most important hyperparameter for Ridge Regression is:

- **alpha:** This parameter controls the strength of regularization. Smaller values make the model similar to Ordinary Least Squares regression, while larger values increase regularization.

Tuning Process:

We defined a grid of potential values for alpha ($[0.01, 0.1, 1, 10, 100, 1000]$) and used **GridSearchCV** with 5-fold cross-validation to determine the optimal value. The same scoring metric, **negative mean squared error (MSE)**, was used to evaluate the model.

```
param_grid = {'alpha': [0.01, 0.1, 1, 10, 100, 1000]}
ridge_model = Ridge()
grid_search = GridSearchCV(
    estimator=ridge_model,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=5,
    verbose=1
)
grid_search.fit(X_train_scaled, y_train)
```

Optimal Parameter:

The best value of **alpha** was selected based on the GridSearchCV results. This balance ensures that the model is neither under-regularized (overfitting) nor over-regularized (underfitting).

5.3 Additional Considerations

- **Cross-Validation:** For both models, we relied on 5-fold cross-validation to ensure robust performance estimation.
- **Computational Cost:** While tuning parameters like n_estimators or alpha, we considered the trade-off between performance improvement and computational time.
- **Feature Engineering:** Before tuning, techniques such as feature scaling and dimensionality reduction (PCA) were applied to improve model efficiency and avoid numerical instability.

By systematically tuning the parameters for each model, we achieved configurations that minimize errors while ensuring the model generalizes well to unseen data.

6 How Do You Make the Best Model Selection?

To select the best model, I compared the performance of three different algorithms—Random Forest, Ridge Regression, and Linear Regression—on the validation set using their Mean Squared Error (MSE). The MSE values for each model are as follows:

- **Random Forest Regressor:** 1441.84
- **Ridge Regression:** 1027.29
- **Linear Regression:** 1030.05

The primary criterion for selection was the model's MSE on the validation set, as it directly measures the error between predicted and true outputs. Based on this metric, **Ridge Regression** was selected as the best-performing model, achieving the lowest MSE of 1027.29. While Linear Regression's MSE was close, Ridge Regression demonstrated a slight advantage, likely due to its ability to mitigate overfitting by introducing regularization through the alpha parameter.

In contrast, the Random Forest model had a significantly higher MSE of 1441.84, despite attempts to optimize its hyperparameters using GridSearchCV. This could be attributed to the relatively small size of the dataset, where the Random Forest's complexity may have resulted in overfitting or instability.

Additionally, Ridge Regression benefits from the preprocessing steps applied, such as normalization of features, which enhance its performance. Its stability and computational efficiency also make it more suitable for this task compared to the other models.

In conclusion, Ridge Regression with the parameter:$alpha = 1000$ was chosen as the best model for this task due to its superior performance and generalization ability.

7 What is the performance of the two types of machine learning models, how to improve their performance?

7.1 Performance Analysis

In this project, two types of machine learning models were evaluated: Random Forest and Ridge Regression. Both models were assessed on their ability to predict wound locations using validation and test datasets. The performance metrics for these models are summarized below:

7.1.1 Random Forest Models

Six Random Forest models were trained: one without PCA and five with PCA-reduced datasets (20 components). During validation, the PCA-reduced models showed improved Mean Squared Error (MSE), confirming PCA's effectiveness in reducing redundancy and dimensionality. However, on the test dataset:

- The Random Forest model trained without PCA achieved a **MSE of 1195.41**, **MAE of 26.64**, and an **R^2 score of 0.21**.
- The PCA-trained models had **higher MSEs (1554.92–1671.94)**, **higher MAEs (30.85–31.91)**, and **lower R^2 scores (0.02–0.08)** compared to the non-PCA model.

This performance decline suggests that while PCA improved the training and validation process, it did not generalize well to the test data.

7.1.2 Ridge Regression Models

Ridge Regression models with various regularization parameters (α) were trained. Their performance was consistent and better than Random Forest models:

- The Ridge Regression models achieved **MSEs of 859.47–894.53**, **MAEs of 21.92–22.36**, and **R^2 scores of 0.41–0.42** on the test dataset.
- The simplest Linear Regression model also performed comparably well, with a **MSE of 859.47** and **R^2 of 0.42**.

These results indicate that Ridge Regression and Linear Regression were more robust and generalized better to unseen data.

7.2 Analysis of PCA Impact on Random Forest Models

PCA was used to reduce the dimensionality of the dataset, which improved Random Forest performance during validation but led to worse performance on the test dataset. Despite applying the same PCA transformation to the test dataset, the following issues likely contributed to this decline:

- **Overfitting to PCA-reduced Training Data**: PCA emphasizes variance, which might remove less dominant but still relevant features, leading to suboptimal generalization.
- **Information Loss**: Reducing the dataset to 20 dimensions might not have preserved enough information for effective prediction.
- **Misalignment Between Variance and Target**: PCA focuses on maximizing variance, which does not necessarily align with the prediction task, especially in datasets where important features have lower variance.

7.3 Recommendations for Improvement

To enhance model performance, the following strategies are suggested:

- **Optimize PCA Dimensionality**: Experiment with different numbers of PCA components (e.g., 50 or 100) to balance dimensionality reduction and information retention.
- **Feature Selection**: Consider feature selection methods that prioritize features most correlated with the target variable instead of variance-based methods like PCA.
- **Non-linear Dimensionality Reduction**: Explore advanced techniques like t-SNE, UMAP, or autoencoders, which may better capture non-linear relationships in the data.
- **Hybrid Approach**: Apply PCA selectively to subsets of features while preserving raw features that are critical to the target.
- **Hyperparameter Tuning for Ridge Regression**: While Ridge Regression performed well, additional tuning of the regularization parameter (α) and potential feature engineering might further improve performance.
- **Ensemble Methods**: Combine the predictions of Ridge Regression and Random Forest models to leverage their strengths and minimize weaknesses.

These adjustments aim to address the specific challenges observed in this study while maintaining the generalizability of the models to unseen datasets.

8 What are the samples that are difficult to predict the wound location, do you have any explanation about them?

To better understand the model's performance, we analyzed the samples with the highest and lowest prediction errors. The following steps were taken:

8.1 Identification of Difficult Samples

For each model, we calculated the prediction errors as the difference between the predicted and ground truth wound locations. These errors were sorted, and samples with the highest errors were identified as difficult-to-predict cases. Similarly, samples with the lowest errors were identified as well-predicted cases.

```
# Calculate the errors for all predictions
errors = []
for i in range(len(testImages)):
    pred = predictions[i]
    gt = testOutputs[i]
    error = calculate_error(pred, gt)
    errors.append((i, error))

# Sort the errors by ascending order (low error to high error)
errors.sort(key=lambda x: x[1])


# Select well-predicted images (low error) and hard-to-predict images
num_images = 3  # Number of images to display
# First 3 with lowest error
well_predicted_indices = [errors[i][0] for i in range(num_images)]
# First 3 with highest error
hard_predicted_indices = [errors[-i-1][0] for i in range(num_images)]

# Combine indices for mixed display
selected_indices = well_predicted_indices + hard_predicted_indices

for i in selected_indices:
    img = testImages[i]
    pred = predictions[i]
    gt = testOutputs[i]

    # Unpack predictions and ground truth
```

```
pred_x, pred_y, pred_xw, pred_yw = pred
gt_x, gt_y, gt_xw, gt_yw = gt

# Create a plot
fig, ax = plt.subplots(1, figsize=(6, 6))
ax.imshow(img, cmap='gray')  # Display the image

# Draw the predicted rectangle (red)
draw_rectangle(ax, pred_x, pred_y,
    pred_xw, pred_yw, 'red', scale_factor_x)

# Draw the ground truth rectangle (green)
draw_rectangle(ax, gt_x, gt_y,
    gt_xw, gt_yw, 'green', scale_factor_x)

plt.title(f"Prediction (Red): {pred}\nGround Truth (Green): {gt}"
plt.axis('off')
plt.show()
```

8.2 Explanation

The explanation about the poorly-predicted samples is discussed in 9.2

9 Can you visualize the prediction results?

To better understand the performance of the models, we visualized instances from the test dataset that were well-predicted and poorly-predicted. Specifically, we selected three pairs of samples, one pair for each model (Linear Regression, Ridge Regression, and Random Forest), which demonstrate both good and bad predictions. These pairs consist of one image with a well-predicted wound location and one image with a poorly-predicted wound location.

Each selected pair is visualized with the following annotations:

- **Red rectangle:** The red rectangle represents the predicted wound location.
- **Green rectangle:** The green rectangle represents the ground truth.

9.1 Visualization of Well-Predicted and Poorly-Predicted Samples

The following figures display pairs of well-predicted and poorly-predicted samples for each of the three models:

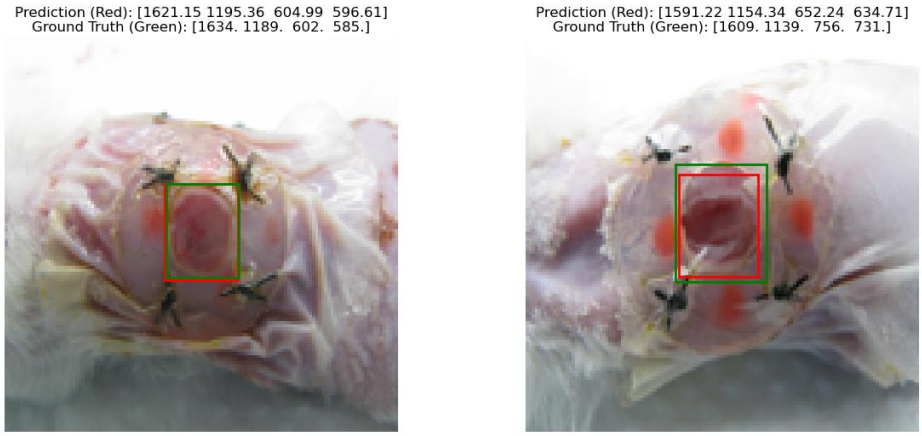

Figure 1: Linear Regression Model: Left - well-predicted sample, Right - poorly-predicted sample. The red rectangles indicate the predicted wound locations, and the green rectangles indicate the ground truth.

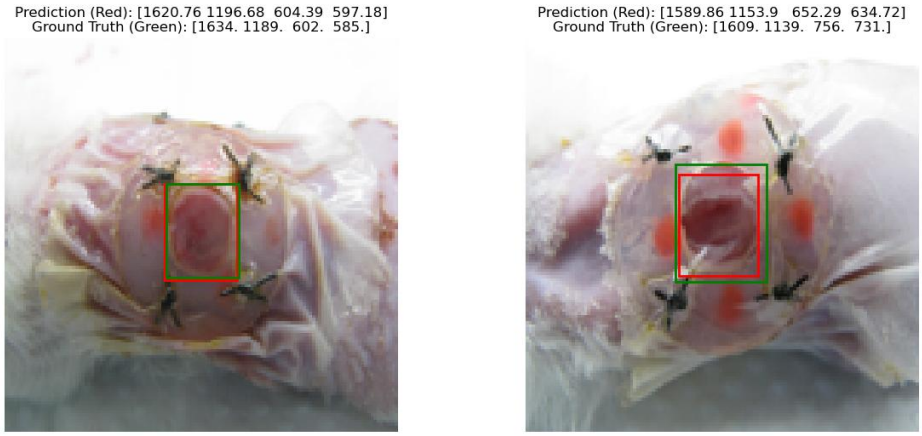

Figure 2: Ridge Regression Model: Left - well-predicted sample, Right - poorly-predicted sample. The red rectangles indicate the predicted wound locations, and the green rectangles indicate the ground truth.

9.2 Insights from Visualizations

The visualizations reveal the following patterns:

- **Well-predicted Samples:** These tend to have a clear and distinct wound area, where the colors and shapes outside the wound are relatively pure and uniform. The four red dots used for location marking are clearly visible and

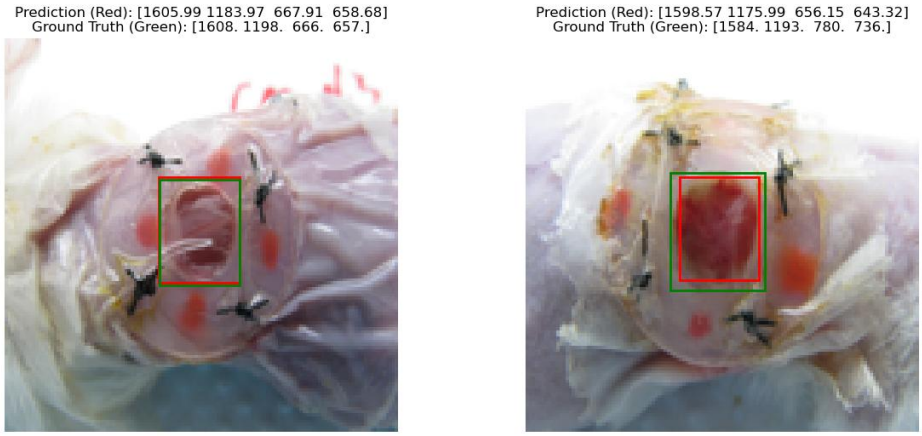

Figure 3: Random Forest Model: Left - well-predicted sample, Right - poorly-predicted sample. The red rectangles indicate the predicted wound locations, and the green rectangles indicate the ground truth.

well-aligned, making the models more successful in detecting and predicting the wound.

- **Poorly-predicted Samples:** These typically involve smaller or irregularly-shaped wounds. The area outside the wound tends to have more chaotic textures, such as yellow medication residues, which make the background less distinguishable. Additionally, the four red dots used for location marking are less clearly visible and may even be partially missing, leading to poor model performance.

9.3 Future Directions

Based on the insights from these visualizations, we propose the following strategies to improve model performance:

- Data augmentation techniques, such as rotating, scaling, and adding noise, to simulate challenging real-world scenarios and improve model robustness.
- Incorporating additional features like edge detection or texture analysis to aid in identifying small or irregular wound locations.
- Fine-tuning the models by training them on a larger dataset with more diverse examples to improve generalization.

10 Conclusion

This study evaluated Random Forest, Ridge Regression, and Linear Regression models for predicting wound locations in animal images, with a focus on the impact of PCA.

The Random Forest model trained without PCA performed better than the PCA-trained models on the test dataset, suggesting that PCA may have removed critical features. Ridge Regression models, with MSEs between 859.47 and 894.53, outperformed Random Forest models, indicating better generalization.

The visualizations showed that well-predicted samples had clear and distinguishable wounds with pure colors and shapes, while poorly-predicted samples involved small, irregular wounds and chaotic textures, making them difficult to detect.

In conclusion, Ridge Regression models demonstrated the best performance. Future work should focus on refining PCA, exploring advanced feature extraction methods, and testing deep learning models to improve accuracy, especially for complex cases.