PREDICTION OF BREAST CANCER USING HISTOPATHOLOGY IMAGES

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Team - P601

K V L BHAVYA (HU21CSEN0101141)

CH HARIKA (HU21CSEN0101163)

K PRANATHI (HU21CSEN0100954)

K SURYA TEJA (HU21CSEN0101927)

Under the esteemed guidance of

Dr. Figlu Mohanty

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

HYDERABAD

2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

DECLARATION

I hereby declare that the project report entitled "Prediction of Breast Cancer Using Histopathology Images" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No(s)	Name(s)	Signature
HU21CSEN0101141	K V L BHAVYA	
HU21CSEN0101163	CH HARIKA	
HU21CSEN0100954	K PRANATHI	
HU21CSEN0101927	K SURYA TEJA	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled “Prediction of Breast Cancer Using Histopathology Images” is a bonafide record of work carried out by K V L BHAVYA (HU21CSEN0101141), CH HARIKA (HU21CSEN0101163), K PRANATHI (HU21CSEN0100954), K SURYA TEJA (HU21CSEN101927) students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Date :

Dr. Figlu Mohanty	**Dr. A B Pradeep Kumar**	**Dr S Mahaboob Basha**
Project Guide	Project Coordinator	Head of the Department
CSE Department	CSE Department	CSE Department
GITAM Hyderabad	GITAM Hyderabad	GITAM Hyderabad

ACKNOWLEDGEMENT

Our project report would not have been successful without the help of several people. We would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honorable Pro-Vice-Chancellor, **Prof. D. S. Rao**, for providing the necessary infrastructure and resources for the accomplishment of our seminar. We are highly indebted to **Prof. N. Seetharamaiah**, Associate Director, School of Technology, for his support during the tenure of the seminar.

We are very much obliged to our beloved **Prof. S Mahaboob Basha**, Head of the Department of Computer Science & Engineering, for providing the opportunity to undertake this seminar and encouragement in the completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr. A B Pradeep Kumar**, Project Coordinator, Department of Computer Science and Engineering, School of Technology and to our guide, **Dr Figlu Mohanty**, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the project report.

We are also thankful to all the Computer Science and Engineering department staff members who have cooperated in making our seminar a success. We would like to thank all our parents and friends who extended their help, encouragement, and moral support directly or indirectly in our seminar work.

Sincerely,

HU21CSEN0101141	K V L BHAVYA
HU21CSEN0101163	CH HARIKA
HU21CSEN0100954	K PRANATHI
HU21CSEN0101927	K SURYA TEJA

ABSTRACT

Breast cancer is one of the most prevalent and life-threatening diseases affecting women worldwide. Accurate and efficient classification of histopathology images is crucial for early diagnosis and effective treatment planning. This project presents a robust breast cancer classification framework integrating deep learning-based feature extraction with traditional machine learning classifiers. The study utilizes the BreaKHis dataset, which contains microscopic biopsy images categorized as benign or malignant.

In the proposed approach, feature extraction is performed using Convolutional Neural Networks (CNN) and VGG16, two powerful deep learning architectures. To enhance computational efficiency and reduce redundancy, Principal Component Analysis (PCA) is applied for dimensionality reduction while retaining 95% of the variance. Additionally, JAYA optimization is employed to fine-tune feature selection and classifier hyperparameters, further improving model performance. The extracted and optimized features are then classified using machine learning algorithms such as Random Forest (RF), K-Nearest Neighbors (KNN), and Extreme Gradient Boosting (XGBoost).

The performance of different model combinations is evaluated based on metrics such as accuracy, precision, recall, and F1-score. Experimental results demonstrate that integrating deep learning-based feature extraction with traditional classifiers yields improved classification accuracy, with JAYA optimization further refining the results. This study highlights the effectiveness of hybrid deep learning and machine learning approaches in breast cancer diagnosis and contributes to the development of efficient, automated diagnostic tools for histopathology image analysis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Breast cancer is one of the most prevalent and life-threatening diseases among women worldwide. Early and accurate detection of breast cancer plays a crucial role in improving survival rates and facilitating timely treatment. Traditional diagnostic methods, such as histopathological examination by expert pathologists, are time-consuming, subjective, and prone to human error. With advancements in artificial intelligence (AI) and deep learning, automated breast cancer classification systems have emerged as a powerful tool to assist medical professionals in diagnosing cancer with high accuracy and efficiency.

Several classification systems have been proposed for breast cancer diagnosis, leveraging machine learning (ML) and deep learning (DL) techniques. Traditional ML models rely on handcrafted feature extraction methods, such as texture analysis and morphological characteristics, which often fail to capture complex patterns in histopathology images. Deep learning-based approaches, particularly Convolutional Neural Networks (CNNs), have demonstrated superior performance in feature extraction and classification. However, CNNs require extensive computational resources and large amounts of labeled data for effective training. Furthermore, high-dimensional feature representations generated by CNNs can introduce redundancy, leading to inefficiencies in the classification process.

To address these challenges, this study proposes a robust breast cancer classification framework that integrates deep learning-based feature extraction with traditional machine learning classifiers. The framework leverages CNN and VGG16 architectures to extract meaningful features from histopathology images. To mitigate high-dimensionality issues, Principal Component Analysis (PCA) is applied for feature reduction while retaining the most significant information. Additionally, classification is performed using Random Forest (RF), K-Nearest Neighbors (KNN), and XGBoost (XGB) classifiers, which are further optimized using the JAYA optimization algorithm to enhance classification accuracy.

The justification for the title, "Prediction of Breast Cancer Using Histopathological Images", stems from the study’s comprehensive approach. The proposed framework combines the strengths of deep learning for feature extraction and traditional machine learning for classification, ensuring both accuracy and interpretability. By incorporating PCA for dimensionality reduction and JAYA optimization for feature selection and hyperparameter tuning, the system achieves a balance between computational efficiency and classification performance. The study systematically evaluates different model configurations to identify the most effective approach for breast cancer classification.This research aims to contribute to the development of an automated, efficient, and highly accurate breast cancer diagnostic system, assisting pathologists in making more reliable and consistent diagnoses.

CHAPTER 2: LITERATURE REVIEW

Breast cancer prediction using histopathology images has gained significant attention in recent years due to advancements in deep learning and machine learning techniques. Researchers have explored various approaches, including pre-trained convolutional neural networks (CNNs), traditional classifiers, and optimization algorithms, to improve the accuracy and efficiency of breast cancer diagnosis. This section reviews recent studies that have contributed to the field, focusing on methodologies, datasets, results, and limitations.

2.1 PRE-TRAINED CNNS AND TRADITIONAL CLASSIFIERS

Recent studies have explored the integration of pre-trained Convolutional Neural Networks (CNNs) with traditional classifiers for breast cancer classification. Li et al. (2021) [5] proposed a deep feature extraction method that leveraged different CNN levels to classify benign and malignant breast cancer cases. Using AlexNet for feature extraction and classifiers such as Support Vector Machines (SVM), Logistic Regression, and Random Forest, they achieved an accuracy of up to 88.69% on the BreaKHis dataset. The study demonstrated that intermediate and high-level CNN features, when combined with traditional classifiers, enhanced classification performance. However, it was limited to binary classification and did not explore subtype differentiation.

Similarly, Gupta and Chawla (2020) [3] employed pre-trained models, including VGG16, VGG19, Xception, and ResNet50, for feature extraction and classified them using SVM and Logistic Regression. ResNet50 combined with Logistic Regression achieved the highest accuracy of 93.27%. Despite its success, the study was restricted to binary classification and lacked data augmentation techniques.

2.2 TRANSFER LEARNING FOR BREAST CANCER DETECTION

Transfer learning has been widely utilized to enhance breast cancer detection by leveraging pre-trained models. Rana and Bhushan (2023) [7] evaluated seven transfer learning models, including LENET, VGG16, DarkNet53, and Xception, for histopathological image classification. Xception achieved the highest accuracy (83.07%), while DarkNet53 provided the best-balanced accuracy (87.17%). Additionally, comparison with YOLOv3 demonstrated a high accuracy of 96.50%, highlighting the potential of object detection models in medical imaging.

Bayramoglu (2016) [2] developed a magnification-independent classification model using CNNs. The study introduced single-task and multi-task CNNs to classify breast cancer irrespective of magnification levels. The multi-task CNN model attained an accuracy of 82.13% in malignancy classification and 80.10% in magnification classification. Despite its effectiveness, the study was constrained by dataset limitations and the need for fine-tuning.

2.3 ADVANCED ARCHITECTURES AND OPTIMIZATION TECHNIQUES

Several studies have focused on optimizing deep learning architectures for improved classification performance. Spanhol et al. (2016) [8] implemented CNN-based classification using the Caffe framework, incorporating patch extraction and fusion techniques. The study demonstrated that CNNs outperformed traditional texture-based models by 6%, with sum-rule fusion achieving the best results. However, computational cost remained a challenge.
Su et al. (2023) [10] introduced BCR-Net, a deep learning framework that predicts breast cancer recurrence using Multiple Instance Learning (MIL) and attention-based pooling. The model achieved an AUC of 0.775 for H&E-stained whole slide images (WSIs) and 0.811 for Ki67-stained WSIs, emphasizing the effectiveness of MIL and interpretability improvements. However, the model's clinical applicability remains untested, and further validation on diverse datasets is required.

2.4 GENE EXPRESSION PREDICTION AND TREATMENT RESPONSE

Recent advancements have also focused on predicting gene expression and treatment response using deep learning. Mondol et al. (2023) [6] proposed hist2RNA, a deep learning model that predicts gene expression from histopathology images. Using EfficientNet, RegNet, and DenseNet for feature extraction, hist2RNA achieved a Spearman correlation of 0.82 for gene expression prediction and an AUROC of up to 0.89 for subtype classification. The study demonstrated high computational efficiency but required further validation on diverse patient cohorts.
Hoang et al. (2024) [4] introduced ENLIGHT-DeepPT, an AI framework that imputes transcriptomics (mRNA expression) from histopathology images to predict cancer treatment response. Evaluated on TCGA datasets, the framework achieved an odds ratio of 2.28, outperforming existing models. However, the study lacked computational efficiency analysis and feature importance evaluation.

2.5 RECURRENCE PREDICTION AND CLINICAL APPLICATIONS

Breast cancer recurrence prediction has been a key area of research, with deep learning models showing promise. Su et al. (2023) [10] demonstrated that BCR-Net effectively classifies high-risk patients, with Ki67-stained WSIs achieving a classification accuracy of 79.2%. The study emphasized the importance of intelligent patch sampling and MIL for recurrence prediction. However, clinical validation and integration with additional biomarkers remain necessary.

2.6 LIMITATIONS AND FUTURE DIRECTIONS

While deep learning has significantly improved breast cancer classification, several challenges remain. Most studies rely on the BreaKHis dataset, limiting generalization to diverse clinical settings. Furthermore, computational costs and dataset imbalance continue to pose challenges. Future research should explore multi-magnification fusion, diverse datasets, and advanced optimization techniques to enhance model robustness.

2.7 SUMMARY OF LITERATURE REVIEWED

The reviewed studies highlight the effectiveness of CNNs, transfer learning, and optimization techniques in breast cancer classification. Pre-trained models, when combined with traditional classifiers, improve classification performance. Transfer learning models like Xception and DarkNet53 demonstrate high accuracy, while architectures such as BCR-Net and ENLIGHT-DeepPT show promise in recurrence prediction and treatment response analysis. However, dataset limitations, computational efficiency, and clinical applicability require further exploration to ensure robust and interpretable AI-driven solutions for breast cancer detection and prognosis.

CHAPTER 3: PROBLEM IDENTIFICATION AND OBJECTIVES

3.1 PROBLEM IDENTIFICATION

Breast cancer is one of the leading causes of mortality among women worldwide, and early and accurate diagnosis is crucial for effective treatment. Histopathology image analysis is a widely used diagnostic method, but manual examination by pathologists is time-consuming, subjective, and prone to human error. Traditional machine learning techniques for classification rely on handcrafted feature extraction, which often fails to capture complex patterns in histopathology images. Deep learning-based methods, while highly effective, generate high-dimensional feature representations that can lead to redundancy and increased computational complexity. Additionally, selecting the most relevant features and optimizing classifier performance remains a challenge. Therefore, a robust and efficient classification framework is needed to improve the accuracy and reliability of breast cancer diagnosis.

3.2 OBJECTIVES

To address these challenges, the following objectives are defined:

- Develop a breast cancer classification framework integrating deep learning-based feature extraction with traditional machine learning classifiers.
- Extract features from histopathology images using CNN and VGG16 architectures.
- Apply Principal Component Analysis (PCA) for feature dimensionality reduction while preserving essential information.
- Evaluate classification performance using Random Forest (RF), K-Nearest Neighbors (KNN), and XGBoost (XGB).
- Incorporate JAYA algorithm for optimization of features and fine-tuning of the hyperparameters of the classifier to enhance accuracy.
- Perform comparative analysis of different feature extraction, reduction, and classification combinations to determine the most effective approach.

CHAPTER 4: EXISTING SYSTEM AND PROPOSED SYSTEM

4.1 EXISTING SYSTEM AND ITS DRAWBACKS

Traditional breast cancer classification methods rely on handcrafted feature extraction techniques such as texture analysis (GLCM, LBP), shape descriptors, and statistical measures, followed by classification using machine learning algorithms like Support Vector Machines (SVM) and Random Forest (RF). While these methods have been effective to some extent, they suffer from several limitations:

- Limited Feature Representation: Handcrafted feature extraction fails to capture intricate spatial and hierarchical patterns present in histopathology images, reducing classification accuracy.
- High Variability in Histopathology Images: Traditional methods struggle to generalize across different staining conditions, magnifications, and variations in cell morphology.
- Computational Complexity: Extracting and selecting meaningful features manually is computationally expensive and time-consuming.
- Dependence on Feature Engineering: Performance heavily relies on predefined feature extraction techniques, limiting the adaptability of the model.
- Difficulty in Handling High-Dimensional Data: When deep learning-based features are used, they often result in high-dimensional feature vectors, leading to redundancy and overfitting.
- Lack of Optimization: Most existing systems do not integrate optimization algorithms for feature selection and hyperparameter tuning, leading to suboptimal classification performance

4.2 PROPOSED SYSTEM AND ITS IMPROVEMENTS

The proposed system overcomes these limitations by integrating deep learning-based feature extraction with traditional machine learning classifiers, combined with dimensionality reduction and optimization techniques. The key improvements are:

- Automated Feature Extraction: Instead of relying on handcrafted features, the system extracts deep features from histopathology images using CNN and VGG16 architectures, capturing complex spatial and hierarchical patterns.

- Dimensionality Reduction Using PCA: PCA is applied to eliminate redundant and less informative features while retaining 95% of variance, reducing computational complexity and mitigating overfitting.
- Hybrid Classification Approach: Deep features are classified using machine learning classifiers (RF, KNN, XGB), leveraging the strengths of both deep learning and traditional classification techniques.
- JAYA Optimization for Feature Selection and Hyperparameter Tuning: The JAYA optimization algorithm is integrated to refine feature selection and optimize classifier parameters, further enhancing accuracy.
- Comprehensive Performance Evaluation: The proposed system is evaluated using multiple classification metrics, including accuracy, precision, recall, and F1-score, ensuring a robust and reliable classification model.
- Increased Generalization Capability: Data augmentation techniques (rotation, flipping, rescaling, contrast normalization) are applied to improve model robustness against variations in histopathology images.

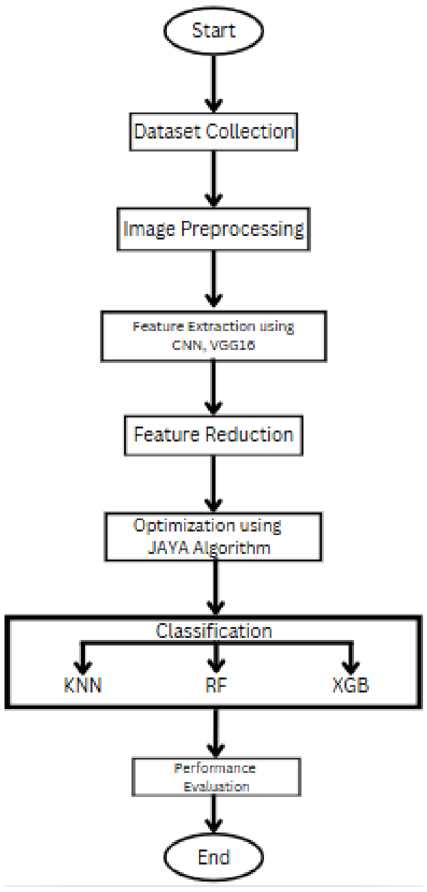


Figure 4.2: Proposed System

CHAPTER 5: SYSTEM ARCHITECTURE

This study presents a robust breast cancer classification framework by integrating deep learning-based feature extraction with traditional machine learning classifiers. The methodology comprises multiple stages:

1. Dataset Preprocessing
2. Feature Extraction using CNN and VGG16
3. JAYA Optimization for Feature Selection and Hyperparameter Tuning
4. Feature Reduction using PCA
5. Classification using Random Forest (RF), K-Nearest Neighbors (KNN), and XGBoost (XGB)
6. Model Evaluation

5.1 DATASET AND PREPROCESSING

The study employs the BreaKHis dataset, which consists of histopathology images categorized as benign or malignant tumors. The dataset is divided into an 80% training set and a 20% testing set, ensuring both classes are well represented.

To enhance model generalization and prevent overfitting, the following data augmentation techniques are applied to training images:

- Rotation within ±15° to account for variations in slide preparation.
- Horizontal and Vertical Flipping to introduce mirror transformations.
- Rescaling within a range of 90% to 110% to simulate different magnifications.
- Adaptive Histogram Equalization to normalize contrast and improve visibility of tissue structures.

These augmentations introduce variability into the dataset, allowing the model to learn robust feature representations, which improves classification performance.

5.2 FEATURE EXTRACTION

Feature extraction is performed using two different deep learning architectures: a custom CNN model and VGG16, both of which extract high-dimensional feature vectors from histopathology images.

5.2.1 Custom CNN for Feature Extraction

A Convolutional Neural Network (CNN) is designed to automatically learn spatial and hierarchical features from the images. The CNN consists of:

- Multiple convolutional layers with 3×3 kernels to extract spatial patterns.
- Batch Normalization to stabilize learning.
- ReLU activation functions for non-linearity.
- Max-Pooling layers to reduce spatial dimensions.
- Fully Connected Layers, where the penultimate layer's activations are extracted as feature vectors.

5.2.2 VGG16 for Feature Extraction

VGG16, a deep CNN architecture pre-trained on ImageNet, is employed to extract higher-level hierarchical features from images. In this study:

- The fully connected layers of VGG16 are removed.
- The last convolutional block's output is extracted as a feature vector.

By leveraging both CNN-based custom feature extraction and pre-trained VGG16 features, the study ensures a comprehensive feature representation of histopathology images.

5.3 FEATURE REDUCTION USING PCA

Since deep learning models generate high-dimensional feature vectors, Principal Component Analysis (PCA) is applied to reduce dimensionality while preserving the most important information.

The PCA process includes:

- Standardizing the feature vectors to zero mean and unit variance.
- Computing the covariance matrix to identify relationships between features.
- Performing Eigen decomposition to obtain principal components.
- Selecting the top k principal components that retain at least 95% of the variance.

This dimensionality reduction helps:

- Reduce computational complexity.
- Mitigate the risk of overfitting.
- Improve the performance of traditional classifiers.

5.4 JAYA OPTIMIZATION

JAYA is a population-based optimization algorithm that iteratively improves solutions without requiring algorithm-specific control parameters. The algorithm:

- Initializes a population of random parameter sets.
- Evaluates each parameter set using a Random Forest classifier on PCA-transformed features.
- Generates new candidate solutions based on the best and worst solutions in the population.
- Selects the better-performing solutions and updates the population.
- Iterates until convergence or reaching the maximum iterations.

By using JAYA optimization, the most informative feature subset is selected, and optimal hyperparameters are determined, leading to better classification performance.

5.5 CLASSIFICATION MODELS

The extracted feature vectors, after PCA, are classified using Random Forest (RF), K-Nearest Neighbors (KNN), and XGBoost (XGB) classifiers. Random Forest operates by constructing multiple decision trees and aggregating their predictions, which improves classification robustness and reduces overfitting. K-Nearest Neighbors classifies new samples based on their proximity to existing labeled data points, making it a simple yet effective non-parametric classification method. XGBoost, an optimized gradient boosting algorithm, enhances classification accuracy by iteratively improving weak learners while handling complex decision boundaries efficiently.

The CNN and VGG16 feature extraction approaches are evaluated using different classifier combinations, including CNN + RF, CNN + KNN, CNN + XGB, VGG16 + RF, VGG16 + KNN, and VGG16 + XGB, both with and without JAYA optimization. This comparative analysis helps identify the most effective feature extraction classification combination for breast cancer histopathology image classification.

5.6 MODEL EVALUATION

The classification models are evaluated using standard performance metrics:

5.6.1. Accuracy

Measures the proportion of correctly classified images.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

5.6.2. Precision (Positive Predictive Value)

Indicates how many of the predicted malignant cases are actually malignant.

$$\text{Precision} = \frac{TP}{TP + FP}$$

5.1.3. Recall (Sensitivity or True Positive Rate)

Measures the proportion of actual malignant cases correctly identified.

$$\text{Recall} = \frac{TP}{TP + FN}$$

5.1.4. F1-Score

A harmonic mean of precision and recall, balancing the trade-off between false positives and false negatives.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

CHAPTER 6: TOOLS/TECHNOLOGIES USED

The implementation of the breast cancer classification framework involves multiple tools and technologies to ensure efficient data processing, feature extraction, classification, and evaluation. The key tools and their significance are as follows:

6.1. PROGRAMMING LANGUAGE: PYTHON

Python is used due to its extensive support for machine learning, deep learning, and scientific computing. Its libraries simplify data handling, model training, and evaluation.

6.2. DEEP LEARNING FRAMEWORKS: TENSORFLOW & KERAS

- TensorFlow: Provides an efficient platform for building and training deep learning models, particularly CNN and VGG16 for feature extraction.
- Keras: A high-level API of TensorFlow that simplifies model implementation and experimentation.

6.3. MACHINE LEARNING LIBRARIES: SCIKIT-LEARN & XGBOOST

- Scikit-Learn: Used for preprocessing, feature extraction, PCA, and implementing machine learning classifiers like RF and KNN.
- XGBoost: An optimized gradient boosting library for high-performance classification.

6.4. OPTIMIZATION ALGORITHM: JAYA

JAYA is implemented in Python to optimize PCA component selection and classifier hyperparameters, improving classification accuracy without requiring additional control parameters.

6.5. IMAGE PROCESSING & AUGMENTATION: OPENCV & ALBUMENTATIONS

- OpenCV: Used for image loading, preprocessing, and basic transformations.
- Albumentations: Applied for data augmentation (rotation, flipping, rescaling, contrast normalization) to enhance model generalization.

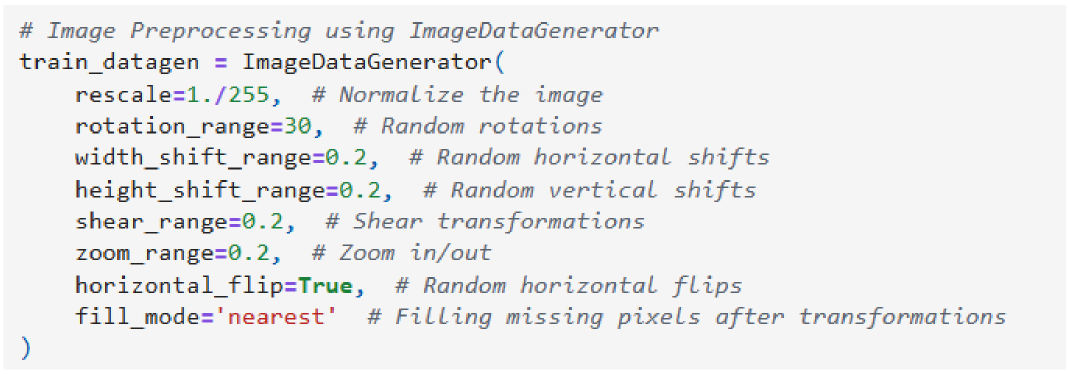
6.6. DATA HANDLING & VISUALIZATION: NumPy, Pandas, Matplotlib, and Seaborn

- NumPy & Pandas: Used for efficient handling and manipulation of dataset features.
- Matplotlib & Seaborn: Used for visualizing dataset distributions, model performance, and comparative analysis.

6.7. DEVELOPMENT ENVIRONMENT: JUPYTER NOTEBOOK & GOOGLE COLAB

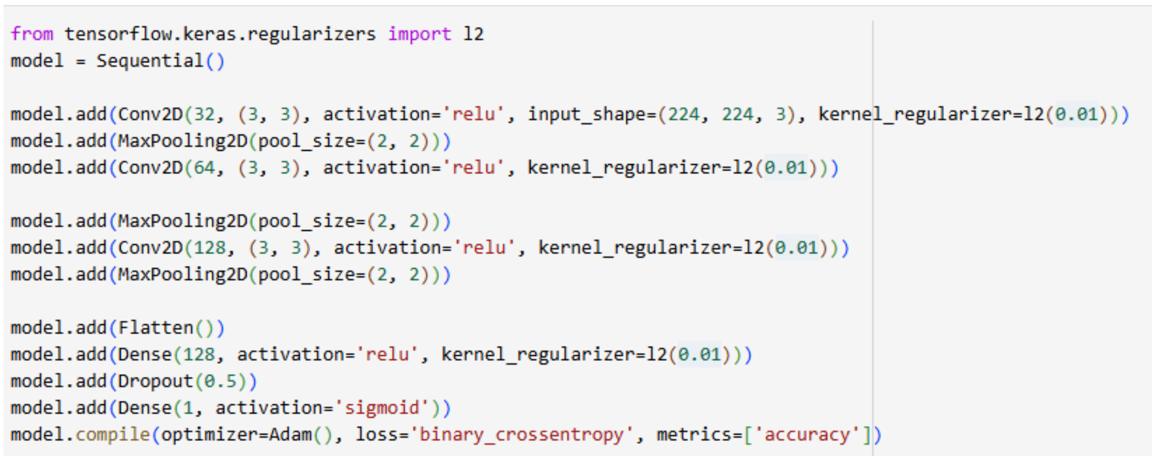
- Jupyter Notebook: Provides an interactive coding environment for step-by-step implementation and debugging.
- Google Colab: Facilitates model training with GPU acceleration for faster computations.

CHAPTER 7: IMPLEMENTATION AND TESTING



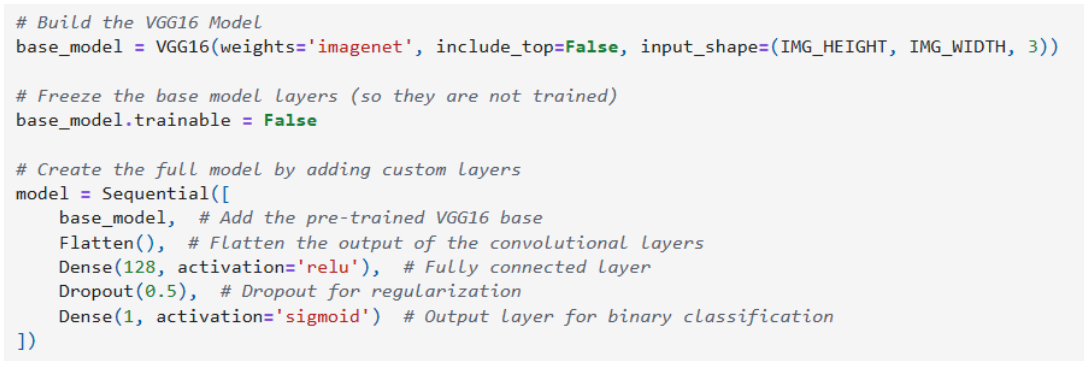
```python
# Image Preprocessing using ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalize the image
    rotation_range=30,  # Random rotations
    width_shift_range=0.2,  # Random horizontal shifts
    height_shift_range=0.2,  # Random vertical shifts
    shear_range=0.2,  # Shear transformations
    zoom_range=0.2,  # Zoom in/out
    horizontal_flip=True,  # Random horizontal flips
    fill_mode='nearest'  # Filling missing pixels after transformations
)
```

Figure 7.1: Data processing



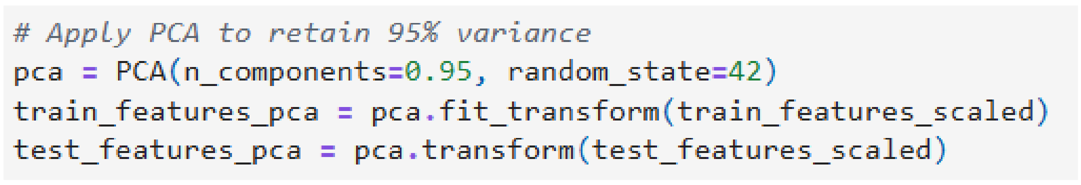
```python
from tensorflow.keras.regularizers import l2
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3), kernel_regularizer=l2(0.01)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.01)))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.01)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
```

Figure 7.2: CNN architecture



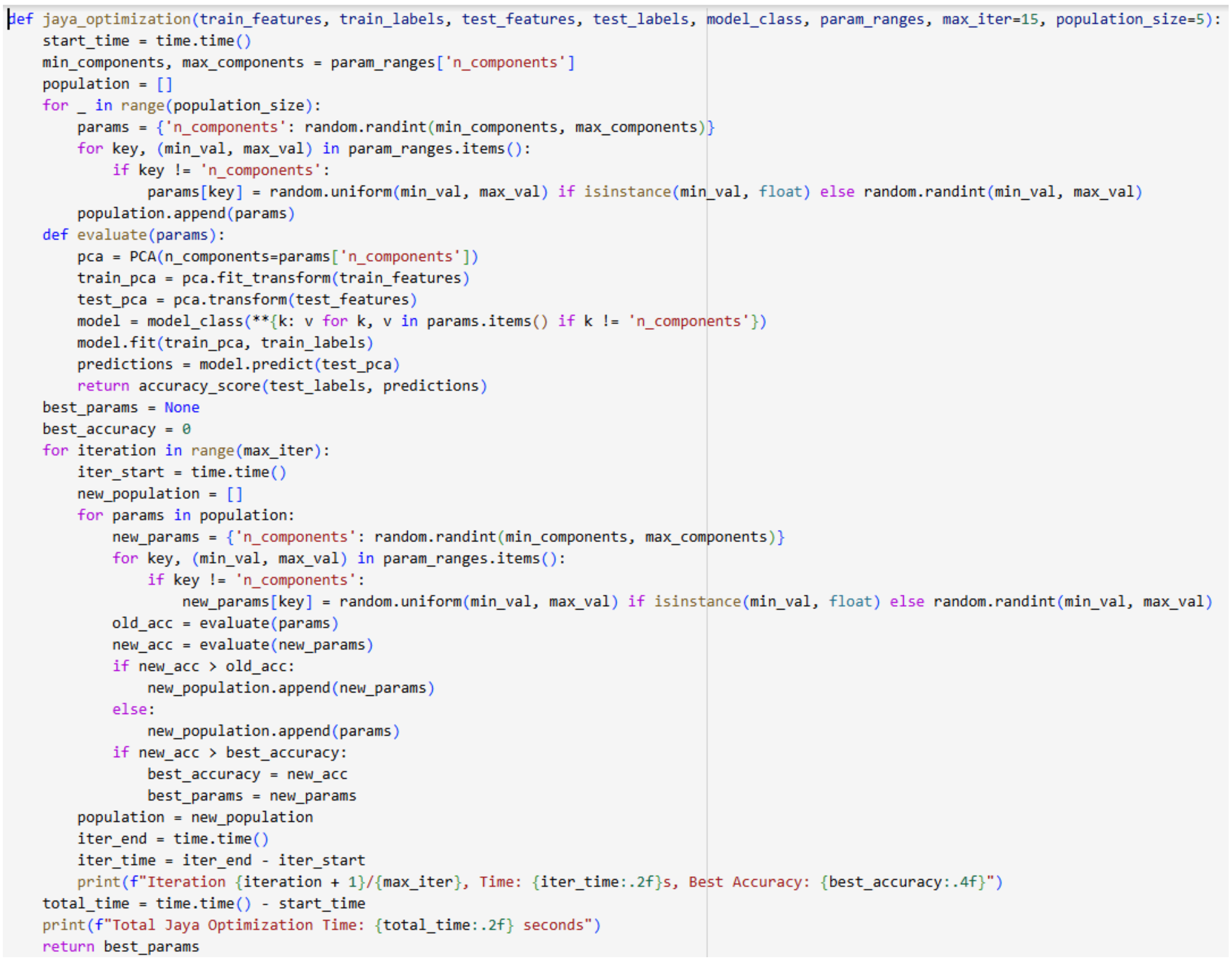
```python
# Build the VGG16 Model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))

# Freeze the base model layers (so they are not trained)
base_model.trainable = False

# Create the full model by adding custom layers
model = Sequential([
    base_model,  # Add the pre-trained VGG16 base
    Flatten(),  # Flatten the output of the convolutional layers
    Dense(128, activation='relu'),  # Fully connected layer
    Dropout(0.5),  # Dropout for regularization
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])
```

Figure 7.3: VGG16 architecture



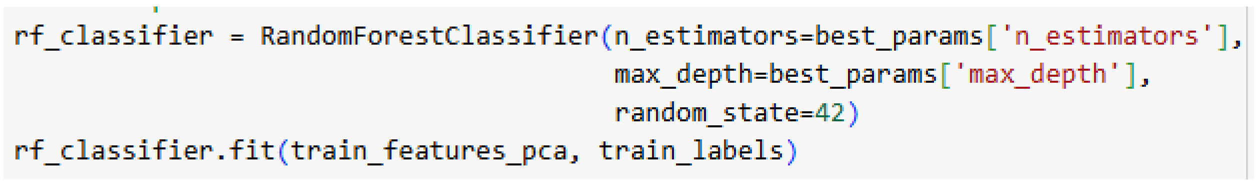
```python
# Apply PCA to retain 95% variance
pca = PCA(n_components=0.95, random_state=42)
train_features_pca = pca.fit_transform(train_features_scaled)
test_features_pca = pca.transform(test_features_scaled)
```

Figure 7.4: Feature Reduction



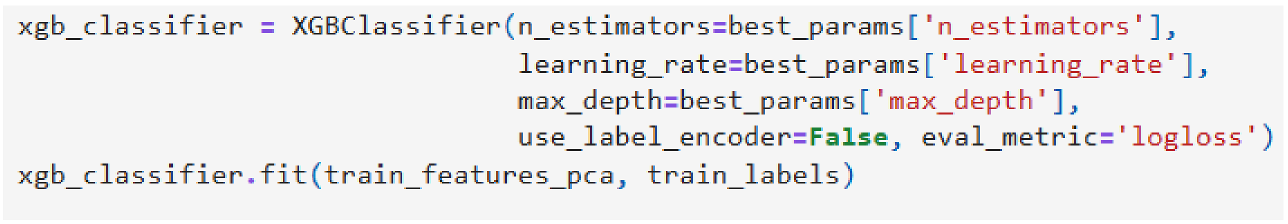
```python
def jaya_optimization(train_features, train_labels, test_features, test_labels, model_class, param_ranges, max_iter=15, population_size=5):
    start_time = time.time()
    min_components, max_components = param_ranges['n_components']
    population = []
    for _ in range(population_size):
        params = {'n_components': random.randint(min_components, max_components)}
        for key, (min_val, max_val) in param_ranges.items():
            if key != 'n_components':
                params[key] = random.uniform(min_val, max_val) if isinstance(min_val, float) else random.randint(min_val, max_val)
        population.append(params)
    def evaluate(params):
        pca = PCA(n_components=params['n_components'])
        train_pca = pca.fit_transform(train_features)
        test_pca = pca.transform(test_features)
        model = model_class(**{k: v for k, v in params.items() if k != 'n_components'})
        model.fit(train_pca, train_labels)
        predictions = model.predict(test_pca)
        return accuracy_score(test_labels, predictions)
    best_params = None
    best_accuracy = 0
    for iteration in range(max_iter):
        iter_start = time.time()
        new_population = []
        for params in population:
            new_params = {'n_components': random.randint(min_components, max_components)}
            for key, (min_val, max_val) in param_ranges.items():
                if key != 'n_components':
                    new_params[key] = random.uniform(min_val, max_val) if isinstance(min_val, float) else random.randint(min_val, max_val)
            old_acc = evaluate(params)
            new_acc = evaluate(new_params)
            if new_acc > old_acc:
                new_population.append(new_params)
            else:
                new_population.append(params)
            if new_acc > best_accuracy:
                best_accuracy = new_acc
                best_params = new_params
        population = new_population
        iter_end = time.time()
        iter_time = iter_end - iter_start
        print(f"Iteration {iteration + 1}/{max_iter}, Time: {iter_time:.2f}s, Best Accuracy: {best_accuracy:.4f}")
    total_time = time.time() - start_time
    print(f"Total Jaya Optimization Time: {total_time:.2f} seconds")
    return best_params
```

Figure 7.5: JAYA algorithm for optimization



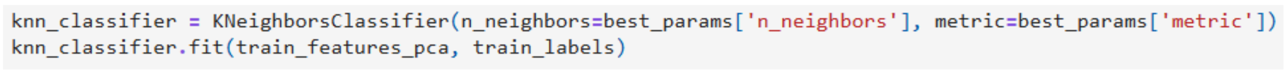
```python
rf_classifier = RandomForestClassifier(n_estimators=best_params['n_estimators'],
                                       max_depth=best_params['max_depth'],
                                       random_state=42)
rf_classifier.fit(train_features_pca, train_labels)
```

Figure 7.6: Random Forest Classifier



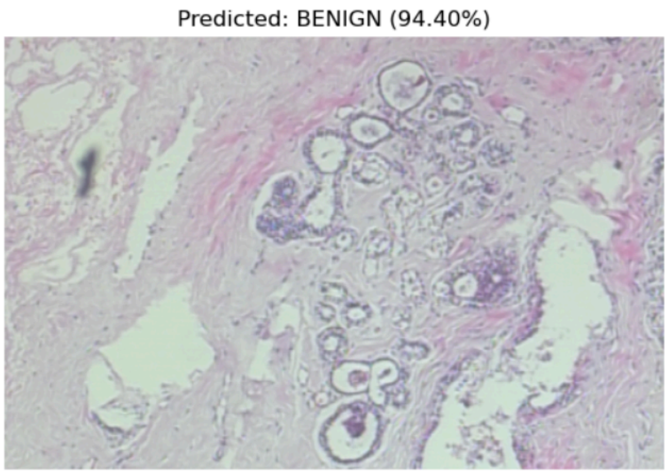
```python
xgb_classifier = XGBClassifier(n_estimators=best_params['n_estimators'],
                               learning_rate=best_params['learning_rate'],
                               max_depth=best_params['max_depth'],
                               use_label_encoder=False, eval_metric='logloss')
xgb_classifier.fit(train_features_pca, train_labels)
```

Figure 7.7: Extreme Gradient Boost Classifier



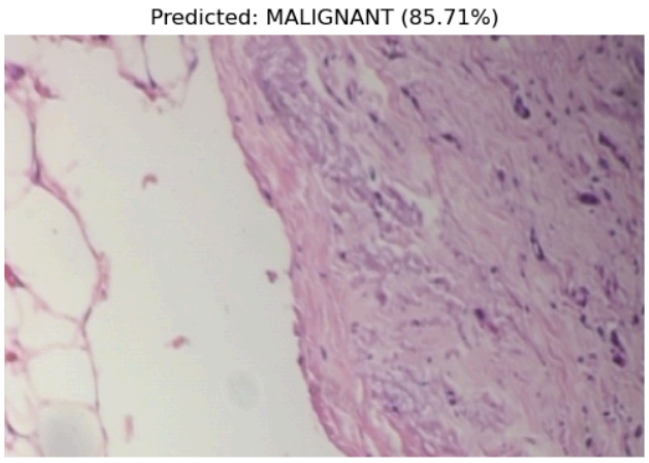
```python
knn_classifier = KNeighborsClassifier(n_neighbors=best_params['n_neighbors'], metric=best_params['metric'])
knn_classifier.fit(train_features_pca, train_labels)
```

Figure 7.8: K Nearest Neighbour Classifier



```
Predicted Class: BENIGN (94.40%)
Class BENIGN: 94.40%
Class MALIGNANT: 5.60%
```

Figure 7.9: Benign image prediction



```
Class Probabilities:
BENIGN: 14.29%
MALIGNANT: 85.71%
```

Figure 7.10: Malignant image prediction

CHAPTER 8: RESULTS AND DISCUSSION

8.1 PERFORMANCE ANALYSIS OF CNN-BASED MODELS

The CNN model was used for feature extraction, followed by classification using RF, KNN, and XGB classifiers. Table 8.1 summarizes the performance of these models in terms of accuracy, precision, recall, and F1-score.

Table 8.1: CNN Performance

Model	Accuracy	Precision (Benign)	Precision (Malignant)	Recall (Benign)	Recall (Malignant)	F1-Score (Benign)	F1-Score (Malignant)
CNN + JAYA + RF	86.37%	0.84	0.87	0.70	0.94	0.76	0.90
CNN + JAYA + KNN	85.85%	0.85	0.86	0.67	0.95	0.75	0.90
CNN + JAYA + XGB	87.18%	0.84	0.88	0.73	0.94	0.78	0.91

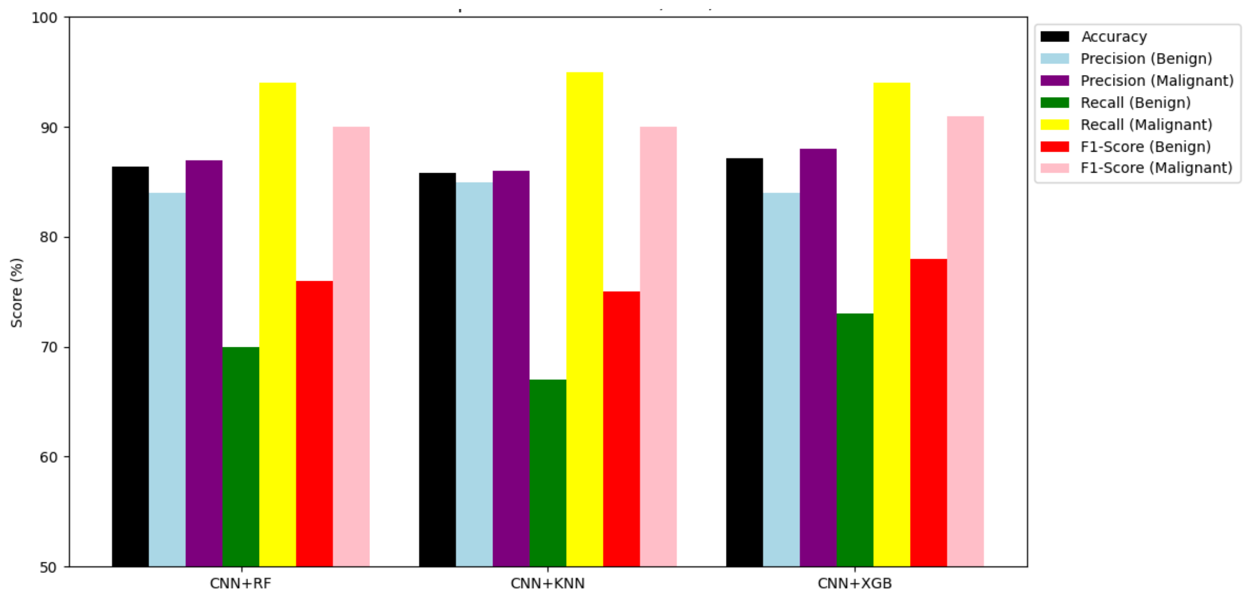


Figure 8.1: CNN Performance

Key Observations:

- CNN+XGB achieved the highest accuracy (87.18%), outperforming the other classifiers.

- CNN+RF (86.37%) and CNN+KNN (85.85%) showed competitive performance, with RF slightly outperforming KNN.
- Precision and recall values varied for benign and malignant classes, with XGB demonstrating better balance across both classes.
- CNN-extracted features contributed to robust classification performance, proving the effectiveness of deep learning-based feature extraction.

8.2 PERFORMANCE ANALYSIS OF VGG16-BASED MODELS

Feature extraction using the pre-trained VGG16 model was followed by classification with RF, KNN, and XGB classifiers. The results are summarized in Table 8.2.

Table 8.2: VGG16 Performance

Model	Accuracy	Precision (Benign)	Precision (Malignant)	Recall (Benign)	Recall (Malignant)	F1-Score (Benign)	F1-Score (Malignant)
VGG16 + JAYA + RF	75.06%	0.88	0.74	0.24	0.98	0.38	0.84
VGG16 + JAYA + KNN	80.68%	0.73	0.84	0.62	0.89	0.67	0.86
VGG16 + JAYA + XGB	83.82%	0.82	0.84	0.62	0.94	0.71	0.89

Key Observations:

- VGG16+XGB achieved the highest accuracy (83.82%) among VGG16-based models.
- VGG16+KNN (80.68%) outperformed VGG16+RF (75.06%), indicating that KNN was better suited for VGG16-extracted features.
- VGG16 models had slightly lower overall accuracy than CNN-based models, suggesting that a custom CNN model tailored to the dataset might be more effective.
- VGG16-based models showed lower recall for benign cases, which may affect their reliability in detecting non-cancerous samples.

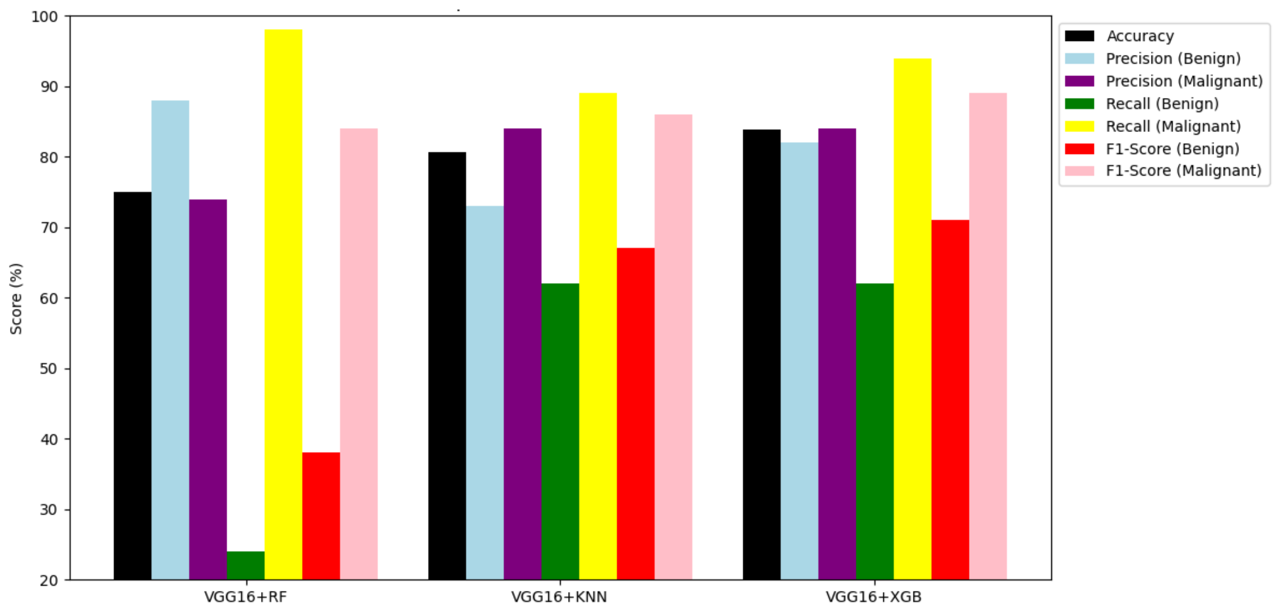


Figure 8.2: VGG16 Performance

8.3 COMPARATIVE ANALYSIS OF OVERALL PERFORMANCE

To understand the overall model effectiveness, we compared all CNN- and VGG16-based models. Figure 8.1 illustrates the accuracy comparison.

KEY FINDINGS:

- CNN-based models consistently outperformed VGG16-based models in classification accuracy.
- CNN+XGB (87.18%) was the best-performing model, followed by CNN+RF (86.37%) and CNN+KNN (85.85%).
- VGG16-based models had lower accuracy, with VGG16+RF performing the worst (75.06%).
- The feature extraction method plays a crucial role in classification performance, with custom CNN models providing better feature representation than VGG16 in this study.

8.4 IMPACT OF JAYA OPTIMIZATION

The JAYA optimization algorithm was applied to refine feature selection and hyperparameters, leading to improved classification performance. Table 8.3 compares the accuracy before and after applying JAYA.

Table 8.3: Accuracies before and after JAYA

MODEL	ACCURACY WITHOUT JAYA	ACCURACY WITH JAYA
CNN + RF	85.9%	86.37%
CNN + XGB	86.83%	87.18%
CNN + KNN	85.61%	85.85%
VGG16 + RF	68.62%	75.06%
VGG16 + XGB	81.32%	83.82%
VGG16 + KNN	72.1%	80.68%

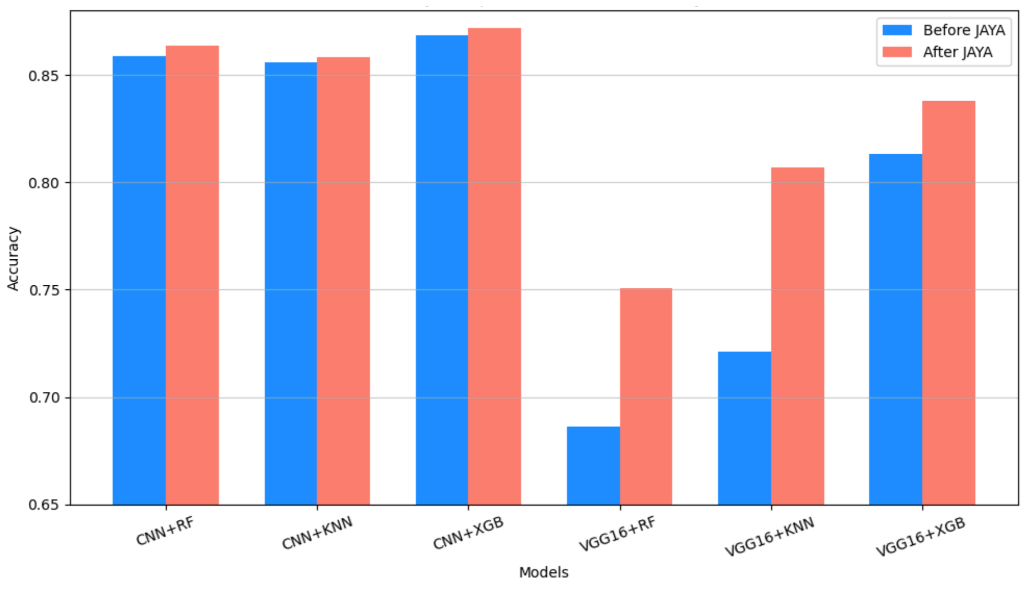


Figure 8.3: Accuracies before and after JAYA

KEY IMPROVEMENTS AFTER JAYA OPTIMIZATION:

- CNN+RF accuracy increased from 85.90% to 86.37%, showing slight enhancement.
- VGG16-based models saw significant improvement, with VGG16+RF increasing from 68.62% to 75.06% and VGG16+KNN from 72.10% to 80.68%.
- The highest improvement was observed in VGG16+KNN, where accuracy increased by nearly 8.5% after applying JAYA.
- JAYA optimization proved to be effective, especially for models that initially had lower accuracy, such as those using VGG16 features.

8.5 CHALLENGES AND LIMITATIONS

Despite achieving promising results, the study has certain limitations:

- Dataset size: The BreaKHis dataset, although widely used, could benefit from additional data to improve model generalization.
- Class imbalance: Some models showed lower recall for benign cases, indicating a potential need for better class balancing techniques.
- Feature selection impact: While PCA helped reduce feature dimensionality, in some cases, it slightly impacted classification performance, emphasizing the need for careful feature selection.

8.6 SUMMARY OF FINDINGS

The results indicate that:

- CNN-based models perform better than VGG16-based models for this dataset.
- XGB was the most effective classifier, achieving the highest accuracy in both CNN and VGG16 models.
- JAYA optimization significantly improved model performance, especially for VGG16-based approaches.
- Deep learning-based feature extraction combined with traditional classifiers provides a robust solution for breast cancer classification.

CHAPTER 9: CONCLUSION AND FUTURE SCOPE

9.1 CONCLUSION

This project successfully implemented and evaluated deep learning and machine learning techniques for the classification of breast cancer histopathological images using the BreaKHis dataset. A hybrid approach combining CNN/VGG16-based feature extraction with machine learning classifiers such as Random Forest (RF), K-Nearest Neighbors (KNN), and XGBoost (XGB) was explored. The study also incorporated Principal Component Analysis (PCA) for dimensionality reduction and the JAYA optimization algorithm to enhance classification performance.

The results demonstrated that integrating deep learning-based feature extraction with traditional classifiers improves classification accuracy. Among all models, CNN+XGB with JAYA optimization achieved the highest test accuracy of 87.18%, highlighting the importance of feature selection and hyperparameter tuning in improving model performance. The study also confirmed that JAYA optimization significantly enhances accuracy compared to conventional hyperparameter tuning methods.

The key conclusions drawn from this study are as follows:

- Hybrid deep learning and machine learning models improve breast cancer classification accuracy compared to standalone deep learning or traditional classifiers.
- CNN-based feature extraction provides superior results compared to VGG16-based feature extraction for this dataset.
- PCA reduces feature dimensionality, improving computational efficiency while maintaining classification performance.
- JAYA optimization effectively selects hyperparameters, improving model accuracy and generalization.
- The approach developed in this study can assist pathologists in early breast cancer detection, reducing subjectivity in diagnosis.

9.2 FUTURE SCOPE

While this study achieved promising results, there are several areas for future improvement:

- Expanding the dataset: Training on larger and more diverse datasets can improve model generalization.
- Exploring advanced deep learning models: Investigating deeper architectures like ResNet, EfficientNet, and Vision Transformers can enhance feature extraction capabilities.
- Real-world clinical application: Extending this approach for whole-slide histopathology image analysis can improve its applicability in clinical settings.

REFERENCES

- [1] Araújo, T., Aresta, G., Castro, E., Rouco, J., Aguiar, P., Eloy, C., Polónia, A., & Campilho, A. (**2017**). *Classification of Breast Cancer Histology Images Using Convolutional Neural Networks.* PLoS One, **12(6)**, **e0177544**. https://doi.org/10.1371/journal.pone.0177544
- [2] Bayramoglu, N., Kannala, J., & Heikkilä, J. (**2016**). *Deep Learning for Magnification Independent Breast Cancer Histopathology Image Classification.* Proceedings of the **2016 23rd International Conference on Pattern Recognition (ICPR)**, **2440-2445**. https://doi.org/10.1109/ICPR.2016.7900001
- [3] Gupta, K., & Chawla, N. (**2020**). *Analysis of Histopathological Images for Prediction of Breast Cancer Using Traditional Classifiers with Pre-Trained CNN.* Procedia Computer Science, **167**, **878-889**. https://doi.org/10.1016/j.procs.2020.03.427
- [4] Hoang, D.-T., et al. (**2024**). *A Deep-Learning Framework to Predict Cancer Treatment Response from Histopathology Images through Imputed Transcriptomics.* Nature Cancer, **5**, **1-13**. https://doi.org/10.1038/s43018-024-00793-2
- [5] Li, X., Li, H., Cui, W., Cai, Z., & Jia, M. (**2021**). *Classification on Digital Pathological Images of Breast Cancer Based on Deep Features of Different Levels.* Mathematical Problems in Engineering, **2021**, **1-13**. https://doi.org/10.1155/2021/8403025
- [6] Mondol, R. K., Millar, E. K. A., Graham, P. H., Browne, L., Sowmya, A., & Meijering, E. (**2023**). *hist2RNA: An Efficient Deep Learning Architecture to Predict Gene Expression from Breast Cancer Histopathology Images.* Cancers, **15(9)**, **2569**. https://doi.org/10.3390/cancers15092569
- [7] Rana, M., & Bhushan, M. (**2023**). *Classifying Breast Cancer Using Transfer Learning Models Based on Histopathological Images.* Neural Computing and Applications. https://doi.org/10.1007/s00521-023-08484-2
- [8] Spanhol, F. A., Oliveira, L. S., Petitjean, C., & Heutte, L. (**2016**). *Breast Cancer Histopathological Image Classification Using Convolutional Neural Networks.* Proceedings of the **2016 International Joint Conference on Neural Networks (IJCNN)**. https://doi.org/10.1109/ijcnn.2016.7727519
- [9] Sudharshan, P. J., Petitjean, C., Spanhol, F., Oliveira, L. E., Heutte, L., & Honeine, P. (**2019**). *Multiple Instance Learning for Histopathological Breast Cancer Image Classification.* Expert Systems with Applications, **117**, **103-111**. https://doi.org/10.1016/j.eswa.2018.09.049
- [10] Su, Z., Niazi, M. K. K., Tavolara, T. E., Niu, S., Tozbikian, G. H., Wesolowski, R., & Gurcan, M. N. (**2023**). *BCR-Net: A Deep Learning Framework to Predict Breast Cancer Recurrence from Histopathology Images.* PLOS ONE, **18(4)**.
- [11] Dataset: https://www.kaggle.com/datasets/ambarish/breakhis

ANNEXURE 1 (SOURCE CODE)

1. CNN Implementation

```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import os
train_dir = r'K:\PROJECT\ORGANISED DATASET\train'
test_dir = r'K:\PROJECT\ORGANISED DATASET\test'
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='training'
)
validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
)
from tensorflow.keras.regularizers import l2
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3),
kernel_regularizer=l2(0.01)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.01)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.01)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
history = model.fit(
    train_generator,
    epochs=13,
    validation_data=validation_generator,
    callbacks=[early_stopping]
)
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
model.save('cnn_model.h5')
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
```

2. VGG16 Implementation

```python
train_dir = r'K:/PROJECT/ORGANISED DATASET/train'
test_dir = r'K:/PROJECT/ORGANISED DATASET/test'
IMG_HEIGHT = 224
IMG_WIDTH = 224
BATCH_SIZE = 32
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    shuffle=True
)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    shuffle=False
)
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))
base_model.trainable = False
model = Sequential([
    base_model,
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy',
metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=test_generator,
    callbacks=[early_stopping]
)
model.save("C:/Users/K M SASTRY/Desktop/PROJECT/vgg16_model.h5")
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

3. CNN+RF+PCA / CNN+KNN+PCA / CNN+XGB+PCA (Base code)

```python
cnn_model = load_model(r"C:\Users\K M SASTRY\Desktop\CAPSTONE
PROJECT\cnn_model.h5")
cnn_model = tf.keras.Model(inputs=cnn_model.input,
outputs=cnn_model.get_layer('flatten').output)
for layer in cnn_model.layers:
    layer.trainable = False
train_dir = r"K:\PROJECT\ORGANISED DATASET\train"
test_dir = r"K:\PROJECT\ORGANISED DATASET\test"
train_datagen = ImageDataGenerator(rescale=1.0 / 255.0)
test_datagen = ImageDataGenerator(rescale=1.0 / 255.0)
train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=(224, 224), batch_size=32, class_mode='binary', shuffle=False
)
test_generator = test_datagen.flow_from_directory(
    test_dir, target_size=(224, 224), batch_size=32, class_mode='binary', shuffle=False
)
def extract_features(generator, model):
    features = model.predict(generator, verbose=1)
    labels = generator.classes
    return features, labels
train_features, train_labels = extract_features(train_generator, cnn_model)
test_features, test_labels = extract_features(test_generator, cnn_model)
print(f"Number of features before PCA: {train_features.shape[1]}")
scaler = StandardScaler()
train_features_scaled = scaler.fit_transform(train_features)
test_features_scaled = scaler.transform(test_features)
pca = PCA(n_components=0.95, random_state=42)
train_features_pca = pca.fit_transform(train_features_scaled)
test_features_pca = pca.transform(test_features_scaled)
print(f"Number of features after PCA: {train_features_pca.shape[1]}")
```

4. RF Classifier

```python
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(train_features_pca, train_labels)
```

```
test_predictions = rf_classifier.predict(test_features_pca)
accuracy = accuracy_score(test_labels, test_predictions)
print(f"Test Accuracy with PCA: {accuracy:.4f}")
print("Classification Report with PCA:")
print(classification_report(test_labels, test_predictions,
target_names=test_generator.class_indices.keys()))
```

5. **KNN Classifier**

```python
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors': [3, 5, 7, 9, 11, 15, 21], 'weights': ['uniform', 'distance']}
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(train_features_pca, train_labels)
best_knn = grid_search.best_estimator_
print(f"Best K value: {grid_search.best_params_['n_neighbors']}, Weights:
{grid_search.best_params_['weights']}")
test_predictions = best_knn.predict(test_features_pca)
accuracy = accuracy_score(test_labels, test_predictions)
print(f"Test Accuracy with PCA: {accuracy:.4f}")
print("Classification Report with PCA:")
print(classification_report(test_labels, test_predictions,
target_names=test_generator.class_indices.keys()))
```

6. **XGB Classifier**

```python
xgb_classifier = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    use_label_encoder=False,
    random_state=42
)
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}
grid_search = GridSearchCV(xgb_classifier, param_grid, cv=3, scoring='accuracy',
n_jobs=-1, verbose=1)
grid_search.fit(train_features_pca, train_labels)
best_xgb = grid_search.best_estimator_
print(f"Best Parameters: {grid_search.best_params_}")
print("XGBoost model trained using CNN features with PCA.")
test_predictions = best_xgb.predict(test_features_pca)
accuracy = accuracy_score(test_labels, test_predictions)
print(f"Test Accuracy with PCA & XGBoost: {accuracy:.4f}"
print("Classification Report with XGBoost:")
print(classification_report(test_labels, test_predictions,
target_names=test_generator.class_indices.keys()))
```

7. **VGG16+RF+PCA / VGG16+KNN+PCA / VGG16+XGB+PCA (Base code)**

Python
```python
vgg16_model = load_model(r"C:\Users\K M
SASTRY\Desktop\PROJECT\vgg16_model.h5")
vgg16_model = tf.keras.Model(inputs=vgg16_model.input,
outputs=vgg16_model.get_layer('flatten').output)
for layer in vgg16_model.layers:
    layer.trainable = False
train_dir = r"K:\PROJECT\ORGANISED DATASET\train"
test_dir = r"K:\PROJECT\ORGANISED DATASET\test"
train_datagen = ImageDataGenerator(rescale=1.0 / 255.0)
test_datagen = ImageDataGenerator(rescale=1.0 / 255.0)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
def extract_features(generator, model):
    features = model.predict(generator, verbose=1)
    labels = generator.classes
    return features, labels
train_features, train_labels = extract_features(train_generator, vgg16_model)
test_features, test_labels = extract_features(test_generator, vgg16_model)
print(f"Train features shape: {train_features.shape}")
print(f"Test features shape: {test_features.shape}")
print("Before PCA, train features shape:", train_features.shape)
print("Before PCA, test features shape:", test_features.shape)
pca = PCA(n_components=0.95)
train_features_pca = pca.fit_transform(train_features)
test_features_pca = pca.transform(test_features)
print(f"After PCA, train features shape: {train_features_pca.shape}")
print(f"After PCA, test features shape: {test_features_pca.shape}")
```

8. RF Classifier

```python
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(train_features_pca, train_labels)
print("Random Forest training complete with PCA.")
test_predictions = rf_classifier.predict(test_features_pca)
accuracy = accuracy_score(test_labels, test_predictions)
print(f"Test Accuracy with PCA: {accuracy:.4f}")
print("Classification Report with PCA:")
print(classification_report(test_labels, test_predictions,
target_names=test_generator.class_indices.keys()))
```

9. KNN Classifier

```python
knn_classifier = KNeighborsClassifier(n_neighbors=7, weights='distance',
metric='minkowski', p=2)
knn_classifier.fit(train_features_pca, train_labels)
```

```
test_predictions = knn_classifier.predict(test_features_pca)
accuracy = accuracy_score(test_labels, test_predictions)
print(f"Test Accuracy with PCA and optimized KNN: {accuracy:.4f}")
print("Classification Report with PCA:")
print(classification_report(test_labels, test_predictions,
target_names=test_generator.class_indices.keys()))
```

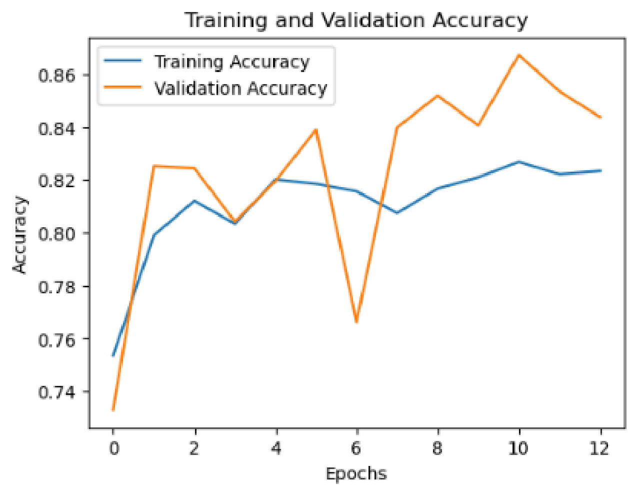
10. XGB Classifier

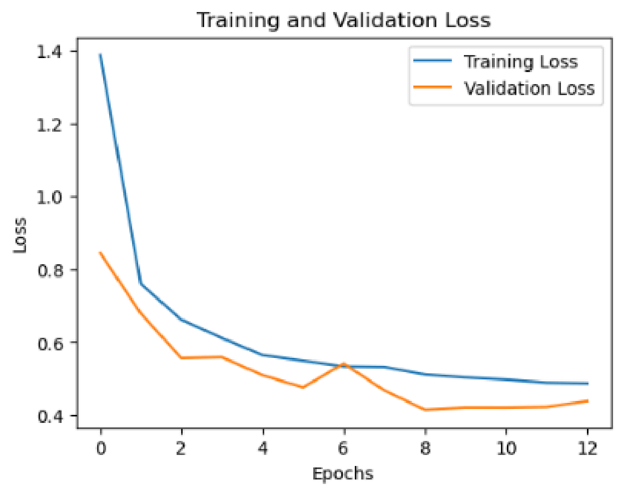
```python
xgb_classifier = xgb.XGBClassifier(
    n_estimators=300, max_depth=6, learning_rate=0.05, subsample=0.8,
    colsample_bytree=0.8, reg_lambda=1, objective='binary:logistic',
use_label_encoder=False, eval_metric='logloss'
)
xgb_classifier.fit(train_features_pca, train_labels)
test_predictions = xgb_classifier.predict(test_features_pca)
accuracy = accuracy_score(test_labels, test_predictions)
print(f"Test Accuracy with PCA and XGBoost: {accuracy:.4f}")
print("Classification Report with PCA:")
print(classification_report(test_labels, test_predictions,
target_names=test_generator.class_indices.keys()))
```

11. JAYA Algorithm for optimization

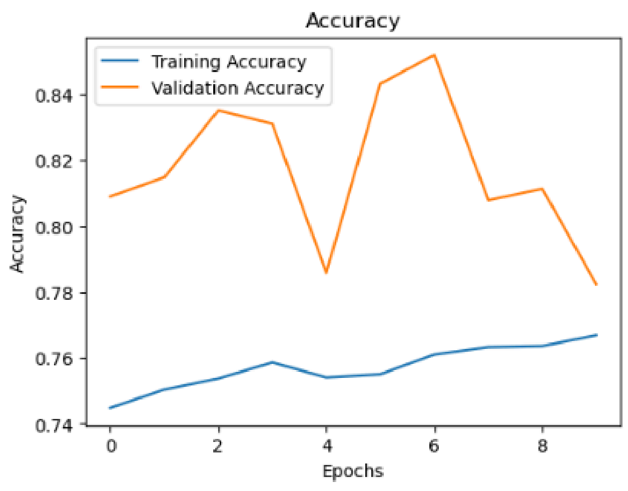
```python
def jaya_optimization(train_features, train_labels, test_features, test_labels, max_iter=10,
population_size=5):
    min_components, max_components = 100, 400
    min_estimators, max_estimators = 50, 300
    min_max_depth, max_max_depth = 5, 50
    population = []
    for _ in range(population_size):
        params = {
            'n_components': random.randint(min_components, max_components),
            'n_estimators': random.randint(min_estimators, max_estimators),
            'max_depth': random.randint(min_max_depth, max_max_depth),
        }
        population.append(params)
    def evaluate(params):
        pca = PCA(n_components=params['n_components'])
            train_pca = pca.fit_transform(train_features)
        test_pca = pca.transform(test_features)
        rf = RandomForestClassifier(n_estimators=params['n_estimators'],
max_depth=params['max_depth'], random_state=42)
        rf.fit(train_pca, train_labels)
        predictions = rf.predict(test_pca)
        return accuracy_score(test_labels, predictions)
    best_params = None
    best_accuracy = 0
    total_start_time = time.time()
    for iteration in range(max_iter):
        iter_start_time = time.time()
        new_population = []
        for params in population:
            new_params = {
                'n_components': random.randint(min_components, max_components),
                'n_estimators': random.randint(min_estimators, max_estimators),
                'max_depth': random.randint(min_max_depth, max_max_depth),
            }
            old_acc = evaluate(params)
            new_acc = evaluate(new_params)
            if new_acc > old_acc:
                new_population.append(new_params)
            else:
                new_population.append(params)
            if new_acc > best_accuracy:
                best_accuracy = new_acc
                best_params = new_params
        population = new_population
        iter_end_time = time.time()
        iter_time = iter_end_time - iter_start_time
        print(f"Iteration {iteration + 1}/{max_iter}, Best Accuracy: {best_accuracy:.4f}, Time
Taken: {iter_time:.2f} seconds")
    total_end_time = time.time()
    total_time = total_end_time - total_start_time
    print(f"Total Optimization Time: {total_time:.2f} seconds")
    return best_params
```

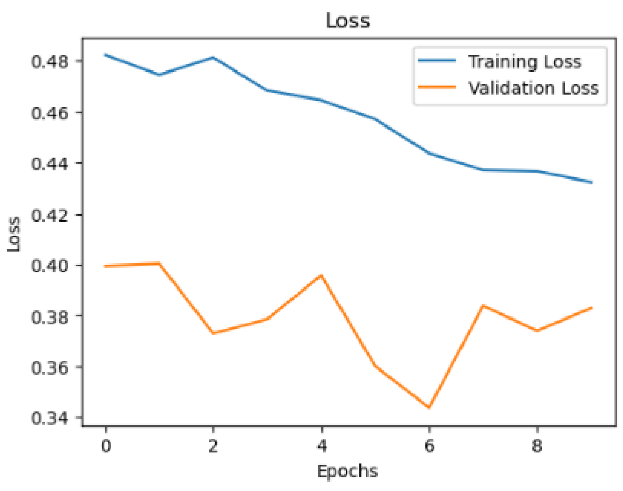
ANNEXURE 2 (OUTPUT SCREENS)





Training and Validation-Loss and Accuracy- CNN





Training and Validation-Loss and Accuracy- VGG16

```
Number of features before PCA: 86528  Number of features after PCA: 389
```

Features count before and after PCA- CNN

```
Number of features before PCA: 25088  Number of features after PCA: 3354
```

Features count before and after PCA- VGG16

```
Optimized PCA components: 195
Feature count before PCA: 86528
Feature count after PCA: 195
```

CNN+RF Feature count after JAYA

```
Optimized PCA components: 124
Feature count before PCA: 86528
Feature count after PCA: 124
```

CNN+KNN Feature count after JAYA

```
Optimized PCA components: 344
Feature count before PCA: 86528
Feature count after PCA: 344
```

CNN+XGB Feature count after JAYA

```
Optimized PCA components: 135
Feature count before PCA: 25088
Feature count after PCA: 135
```

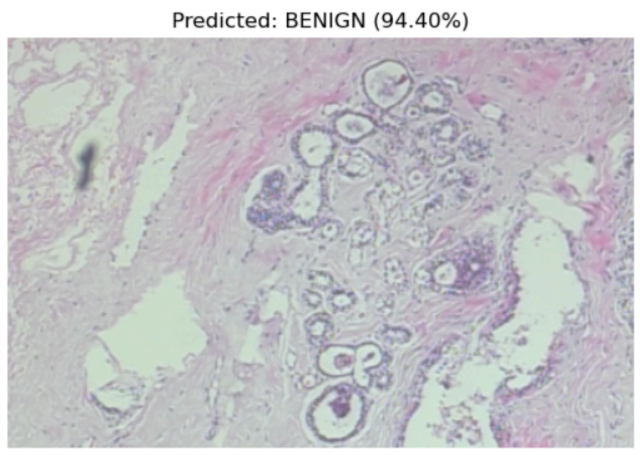
VGG16+RF Feature count after JAYA

```
Optimized PCA components: 150
Feature count before PCA: 25088
Feature count after PCA: 150
```

VGG16+KNN Feature count after JAYA

```
Optimized PCA components: 108
Feature count before PCA: 25088
Feature count after PCA: 108
```

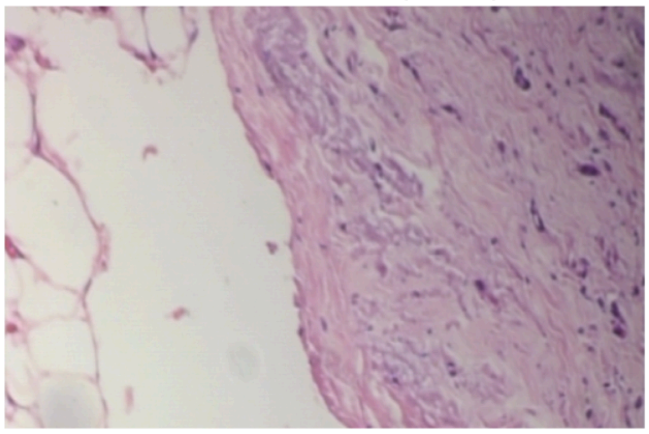
VGG16+XGB Feature count after JAYA



```
Predicted Class: BENIGN (94.40%)
Class BENIGN: 94.40%
Class MALIGNANT: 5.60%
```

Benign image prediction

Predicted: MALIGNANT (85.71%)



```
Class Probabilities:
BENIGN: 14.29%
MALIGNANT: 85.71%
```

Malignant image prediction