**Malaria Parasite Detection on Microscopic Blood Smear Images Using Deep Learning Model and Developing an android app**

**Abstract**

Malaria is a deadly syndrome formed by the Plasmodium parasite that spreads through the bite of infected Anopheles mosquitoes. There are several drugs to cure malaria but its diagnosis is still tedious, time-consuming and costly. Although the existing equipment for detecting malaria gives high accuracy, those are costlier. Keeping this issue in view we develop a deep learning classification model where the blood smear images are used for training which could be incorporated in an android app. The outcome reveals that the proposed model gives 82% accuracy.

**Keywords**

Malaria Detection, Deep Learning, Microscopic Blood Smear Images, Android Application

**Introduction**

Malaria is a major public health concern in many tropical and subtropical regions. Traditional diagnostic methods, such as microscopic examination of blood smears, are labor-intensive and require skilled technicians. The rise of deep learning and convolutional neural networks (CNNs) offers a promising alternative for automating malaria diagnosis. This study aims to develop a CNN-based model for accurate and efficient detection of malaria parasites in blood smear images and to integrate the model into an Android application for easy accessibility.

**Motivation of the Study**

The motivation for this study stems from the need to improve malaria diagnostic methods, particularly in resource-limited settings. Automated systems can significantly reduce the burden on healthcare workers and increase the accuracy of diagnosis. By developing a deep learning model and an accompanying mobile application, we aim to provide a scalable and user-friendly tool that can assist in early detection and treatment of malaria.

**Contributions**

The main contributions of this study are:

1. Development of a CNN-based model for malaria parasite detection in blood smear images.

2. Creation of an efficient pipeline for data preprocessing, model training, and evaluation.

3. Integration of the trained model into an Android application for real-time usage.

4. Extensive evaluation of the model's performance using standard metrics.

5. Documentation and open-source release of the code and trained models for further research and development.

**Related Work**

Several studies have explored the use of machine learning and deep learning techniques for malaria detection. Earlier approaches focused on traditional image processing and machine learning methods. Recent advancements leverage deep learning, particularly CNNs, for improved accuracy. Notable works include the application of CNNs for detecting malaria parasites in thick and thin blood smear images, as well as the development of mobile applications for malaria diagnosis.

**Materials and Methods**

**Dataset Description**

The dataset used in this study comprises microscopic images of blood smears labeled as infected or uninfected with malaria parasites. The images were obtained from publicly available sources and preprocessed to ensure uniformity in size and quality.The dataset comprises a collection of labeled images depicting both uninfected and infected cells. Sourced from kagile Datasets, it contains a total of 1000 images.

**Methodology**

The methodology consists of several key steps, as detailed below:

**Data Preprocessing**

**1. Data Reading and Initial Processing:**

We faced a significant class imbalance issue in the dataset. Class imbalance occurs when the number of instances of one class significantly outnumbers the instances of another class. In our case, among the 1000 images, 839 were labeled as infected, and only 161 were labeled as uninfected. This imbalance can lead to a model that is biased towards the majority class, resulting in poor performance, especially in detecting the minority class.

#### **Our Approach to Mitigating Class Imbalance**

To address the class imbalance problem, we implemented a strategy to reduce the number of infected images while maintaining a representative sample of the infected class. Specifically, we reduced the number of infected images from 839 to 182. This was done by filtering out images with fewer than three infected cells, retaining only those images that had more than three infected cells. This approach helped to create a more balanced dataset, with 161 uninfected images and 182 infected images, resulting in a total of 343 images.

#### **Code Implementation**

The following steps were implemented in our code to achieve this:

1. **Reading and Processing Text Files**:
   * We defined a function **read\_first\_column** to read the first column values from the text files containing the labels. These values indicate whether the corresponding images are infected or uninfected.
   * The function processes each line in the text file, extracting the first column values and storing them in a list.
2. **Categorizing Files**:
   * We defined a function **categorize\_files** to categorize the text files into "uninfected" and "infected" based on the values read from the text files.
   * If all values in a file were "0", the file was classified as "uninfected". Otherwise, the file was classified as "infected", and the count of non-zero values was recorded.
3. **Filtering Infected Images**:
   * After categorizing the files, we filtered the infected images to address the class imbalance. We retained only those infected images with more than three infected cells. This filtering reduced the number of infected images from 839 to 182.
4. **Copying Images to Destination Folders**:
   * We separated the images into different folders based on their labels. Uninfected images were copied to one folder ‘uninfected’, and the filtered infected images were copied to another folder ‘infected’. This structured the dataset, making it ready for further processing and model training..

**2. Dataset Split:**

The dataset is split into training and testing sets to facilitate model training and evaluation. An 80-20 split ratio is used, where 80% of the images are used for training and 20% for testing. The appropriate subdirectories for each class ("uninfected" and "infected") are created within the training and testing folders to maintain a structured directory layout.

**Model Training**

**1. Data Augmentation:**

Data augmentation techniques are applied to the training set to increase data diversity and improve the model's generalization ability. Techniques such as rescaling, rotation, width and height shifts, shear, zoom, and horizontal flips are used. The testing set images are rescaled without additional augmentation to ensure consistency during evaluation.

**2. CNN Model Architecture:**

A convolutional neural network (CNN) is constructed to classify the images. The network consists of several convolutional layers, max-pooling layers, and dense layers. ReLU activation functions are used for the intermediate layers, and a softmax layer is used for the final classification. Dropout layers are included to prevent overfitting by randomly deactivating a fraction of neurons during training.

In the model, a Convolutional Neural Network (CNN) architecture is implemented for image classification using TensorFlow and Keras. The CNN follows a systematic structure that processes input images through several layers to learn and extract features for classification.

At the beginning of the network, the input layer is designed to handle images of size 224x224 pixels with 3 channels representing RGB colors. This forms the initial input data for the CNN.

Following the input layer are convolutional layers, which are responsible for extracting features from the input images. The first convolutional layer employs 32 filters of size 3x3 and uses the Rectified Linear Unit (ReLU) activation function. These filters help in capturing basic features such as edges and textures from the images. After each convolutional layer, a MaxPooling layer with a pool size of 2x2 is applied to reduce the spatial dimensions, aiding in reducing computational complexity and potential overfitting.

The network further refines the learned features through additional convolutional layers. The second convolutional layer comprises 64 filters of size 3x3 with ReLU activation, while the third convolutional layer deepens the feature extraction process with 128 filters of size 3x3 and ReLU activation. These deeper layers can capture more complex patterns and nuances within the images.

After the convolutional layers, a flatten layer is used to reshape the 3D output into a 1D vector, preparing it for input into the fully connected (dense) layers. These dense layers are crucial for learning high-level features and patterns from the extracted features. The first dense layer consists of 512 neurons with ReLU activation, followed by a Dropout layer with a dropout rate of 0.5. This dropout layer aids in preventing overfitting by randomly deactivating a fraction of neurons during training. Finally, the output layer is a dense layer with 2 neurons (equal to the number of classes in binary classification) and a softmax activation function, producing probabilities for each class.

**3. Compilation and Training:**

The model compilation involves configuring the model for training. The Adam optimizer with a learning rate of 0.001 is used, along with categorical cross-entropy loss (suitable for multi-class classification) and accuracy as the evaluation metric.

To enhance the model's robustness and generalization, data augmentation techniques such as rotation, shifting, shearing, zooming, and flipping are applied to the training data. This augmentation exposes the model to variations in the input data, improving its ability to generalize to unseen samples.

To prevent overfitting during training, an early stopping mechanism is employed. The early stopping callback monitors the validation loss and restores the best weights when the validation loss stops improving, with a patience of 5 epochs.

**Training and evaluation :**

The training and evaluation phase involves training the model for 30 epochs using the augmented training data and evaluating its performance on the test data. The `fit` method is utilized along with the early stopping callback for training, and the model is saved after training as 'malaria\_class\_augmented.h5'. Evaluation metrics such as a classification report and accuracy score are computed using the test data to assess the model's performance.

Overall, this CNN architecture is designed to effectively classify images into two categories using a combination of convolutional layers, dense layers, data augmentation, and regularization techniques to enhance performance and prevent overfitting.

#### **Model Evaluation**

**Performance Metrics**:

The model's performance is evaluated using accuracy, precision, recall, and F1-score metrics. Accuracy measures the proportion of correctly classified images, while the classification report provides detailed precision, recall, and F1-score for each class. These metrics help assess the model's effectiveness in distinguishing between infected and uninfected blood smear images.

**Visualization**:

The training history, including training and validation accuracy and loss, is plotted to visualize the model's performance over epochs. These plots provide insights into the model's learning process, indicating how well it generalizes the unseen data and whether it overfits the training data.

#### **Model Deployment**

**Model Conversion**:

To deploy the trained model on mobile devices, it is converted to TensorFlow Lite format. TensorFlow Lite is optimized for performance on mobile and embedded devices, making it suitable for real-time inference in the Android application.

**Android Application Development**:

The TensorFlow Lite model is integrated into an Android application, allowing users to upload blood smear images and receive instant predictions. Image preprocessing steps such as resizing and normalization are implemented within the app to prepare input images for the model.

**User Interface**:

A user-friendly interface is designed for the Android app, enabling easy interaction. Users can upload images through the app, and the app displays the predicted class (infected or uninfected) based on the model's inference. This interface provides a convenient and accessible tool for malaria detection.

**Evaluation Metrics**

The evaluation metrics provide valuable insights into the performance of the image classification model.

**Accuracy :** Accuracy is a fundamental metric that measures the overall correctness of the model's predictions. In this case, the model achieves an accuracy of 82.86%, indicating that it correctly classified approximately 83 out of every 100 images.

**Precision :** Precision focuses on the accuracy of positive predictions, specifically the ratio of true positive predictions (correctly identified infected or uninfected images) to the total predicted positives. A precision of 0.86 for infected images and 0.80 for uninfected images indicates that when the model predicts an image as infected or uninfected, it is correct around 86% and 80% of the time, respectively.

**Recall :** Recall, also known as sensitivity or true positive rate, measures the model's ability to correctly identify positive instances among all actual positives. A recall of 0.81 for infected images and 0.85 for uninfected images suggests that the model captures around 81% and 85% of infected and uninfected images, respectively.

**F1-score :** F1-score is the harmonic mean of precision and recall, providing a balanced assessment of the model's performance. The F1-score of 0.83 for infected images and 0.82 for uninfected images signifies a good balance between precision and recall, indicating that the model is effective in both identifying positive cases accurately and minimizing false positives.

The classification report provides a detailed breakdown of precision, recall, and F1-score for each class (infected and uninfected), along with support indicating the number of samples in each class. The macro average and weighted average give an overall view of the model's performance across classes, with both averaging around 0.83, reflecting consistent performance across the classification task.

Overall, these metrics collectively demonstrate that the model has achieved a good level of accuracy and balance in identifying infected and uninfected images, making it a reliable tool for malaria classification.

**Experimental Setup**

The experiments were conducted using a TensorFlow-based environment with GPU acceleration. The dataset was split into training and testing sets, and the model was trained for a specified number of epochs with early stopping based on validation loss. The training process involved fine-tuning hyperparameters such as learning rate and batch size.

**Results and Discussion**

The experimental results demonstrate the effectiveness of the CNN-based approach for malaria detection. The model achieved high accuracy and showed robustness across different validation sets. The performance metrics indicate that the model can reliably distinguish between infected and uninfected blood smear images.

The integration of the model into an Android application allows for real-time usage, providing a valuable tool for healthcare professionals in the field. The app's user interface is designed to be intuitive, enabling users to upload images and receive instant predictions.

**Conclusion**

This study presents a comprehensive approach to malaria parasite detection using deep learning. The developed CNN model and the Android application offer a scalable solution for automating malaria diagnosis. Future work includes expanding the dataset, refining the model architecture, and conducting field trials to validate the system's effectiveness in real-world scenarios.

The results indicate that deep learning can significantly enhance malaria diagnosis, potentially saving lives by enabling early and accurate detection. The open-source release of the code and models encourages further research and collaboration in this critical area of healthcare.