Open Access

Malaria detection using Convolutional Neural Network

Viviana Greco

Abstract

Malaria is a life-threatening disease caused by Plasmodium parasites and transmitted through mosquito bites. In 2019, over 229 million cases and approximately 400,000 deaths were reported worldwide, with children under five years old being the most vulnerable, accounting for 67% of fatalities. Traditional malaria diagnosis relies on expert examination of blood cells under a microscope. This process is not only tedious and time-consuming but also susceptible to inter-observer variability, leading to potential diagnostic errors. To address these challenges, this project proposes leveraging deep learning techniques to automate malaria detection. Specifically, we utilize a pre-trained VGG16 model, which strikes a balance between reducing both missed infections and unnecessary treatments. By automating the diagnosis, this approach aims to enhance efficiency, accuracy, and accessibility in malaria screening, ultimately contributing to improved healthcare outcomes.

1. Dataset and Preprocessing

1.a. Dataset Download

The dataset consists of labeled images of red blood cells, categorized as either parasitized or uninfected. You can find the dataset on Kaggle.

1.b. Image Preprocessing

To prepare the dataset for model training, we applied several preprocessing steps:

1. Image Resizing: Standardized all images to 64x64 pixels for uniform model input.
2. Normalization: Scaled pixel values to the range [0,1] by dividing by 255, improving model convergence.
3. Image Visualization: Conducted exploratory data analysis by visualizing sample images. Below is a sample:

Published Mar 08, 2025

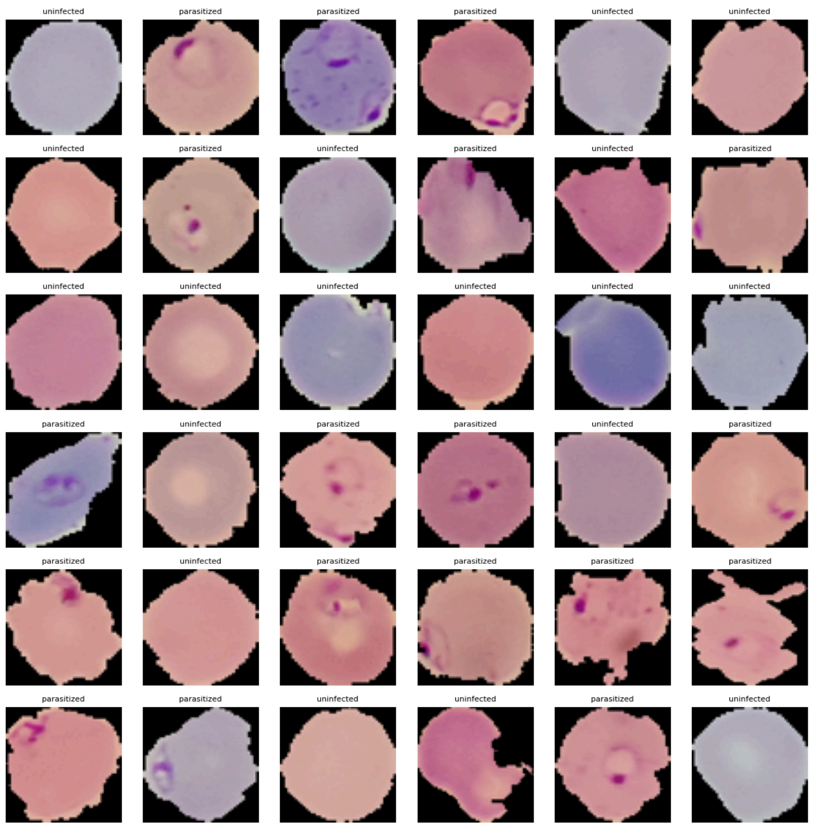


Figure 1: Malaria-infected cells under a microscope.

4. Color Transformations:

- HSV Conversion: Converted images to the HSV color space to enhance the visibility of parasite structures.
- Gaussian Blurring: Applied Gaussian blurring, but it was found to be suboptimal as it reduced feature sharpness. HSV transformation was more effective in highlighting parasitized regions.

5. One-Hot Encoding: Encoded class labels for both training and testing sets, facilitating categorical classification.

2. Model Development

We developed and experimented with five different models to evaluate their effectiveness in malaria detection.

2.a. ***CNN Base Model***

Our initial approach utilized a convolutional neural network (CNN) architecture consisting of three convolutional layers, along with max pooling and dropout mechanisms.

Convolutional Layers

The three convolutional layers each contain 32 filters with a 2×2 kernel size and "same" padding to preserve spatial dimensions.

Pooling Layers Max pooling is applied to reduce the spatial dimensions of the feature maps by selecting the maximum value within each window.

Dropout Layers Dropout is used as a regularization technique to randomly deactivate neurons during training, reducing overfitting.

Dense Layers and Output After the convolutional and pooling blocks, the feature maps are flattened into a one-dimensional vector.

- **Fully Connected Layer:**

A dense layer with 512 neurons is used to learn complex interactions between features.

- **Output Layer:**

A final dense layer with 2 neurons (softmax activation) classifies the images.

All the preivous layers use ReLu (Rectified Linear Unit) activation function.

2.b. ***CNN Model 1***

Model 1 consisted of 2 convolutional layers within a block before applying pooling. This allows the network to learn more complex and abstract features while maintaining the full spatial resolution. Furthermore, it increases the receptive field.

2.c. ***CNN Model 2***

This model incorporates **batch normalization** and **LeakyReLU**:

- Batch Normalization: Normalizes inputs across layers, improving stability and convergence.
- LeakyReLU: Allows small gradients for negative inputs, mitigating dying ReLU problems.

2.d. ***CNN Model 3***

This model integrates data augmentation, applying rotation, width and height shifting, shear (which skews images along an axis), zoom, and flipping to increase variability in the dataset, improving generalization.

2.e. ***CNN Model 4***

This model utilizes **pre-trained VGG16**, leveraging transfer learning to enhance performance with a smaller dataset.

3. Common Elements Across Models

Callbacks for Model Training

To enhance training efficiency and guard against overfitting, two key callbacks were implemented during the training process of all models:

1. **Early Stopping** to monitor the validation loss during training and stop the process if it does not improve for 2 consecutive epochs.
2. **Model Checkpoint** to save the best version of the model during training based on the lowest validation loss.

Model Training

All models were trained using:

- Batch Size: 32
- Validation Split: 20% of training data
- Epochs: 20 (early stopping enabled)
- Verbosity: 1 (for real-time progress monitoring)

4. Model comparison

- **Base model**
  Overall accuracy = 97% / val accuracy = 98.7%
  Recall class 0 = 98%
  Recall class 1 = 97%
  FN = 23 - FP = 44

- **Model 1**
  Overall accuracy = 98% / val accuracy = 97.8%
  Recall class 0 = 97%
  Recall class 1 = 98%
  FN = 37 - FP = 25

- **Model 2**
  Overall accuracy = 97% / val accuracy = 99.6%
  Recall class 0 = 100%
  Recall class 1 = 95%
  FN = 4 - FP = 64

- **Model 3, data augmentation**
  Overall accuracy = 95% / val accuracy = 93.7%
  Recall class 0 = 99%
  Recall class 1 = 91%
  FN = 12 - FP = 119

- **Model 4, pre-trained model**
  Overall accuracy = 99% / val accuracy = 98.5%
  Recall class 0 = 99%
  Recall class 1 = 99%
  FN = 17 - FP = 17

The final proposed model specification is the pre-trained VGG16 Model (Model 5). VGG16 is a deep convolutional neural network with 16 weighted layers (13 convolutional and 3 fully connected layers) pretrained on the ImageNet dataset. The model architecture includes 5 blocks with convolutional layers and 5 max pooling layers to reduce the spatial dimensions of the feature maps, and 3 fully connected laters to perfom high-level reasoning based on the features extracted by the convolutioanl base.

```
vgg = VGG16(include_top=False, weights='imagenet', input_shape=(64, 64,
3))
```

We specified `include_top=False` to allow the model to be used for feature extractions for a new task. Therefore we removed the original 3 fully connected layers. The model uses pre-trained weights from ImageNet and accepts input images of size 64x64 with 3 color channels.

```
transfer_layer = vgg.get_layer('block5_pool')
```

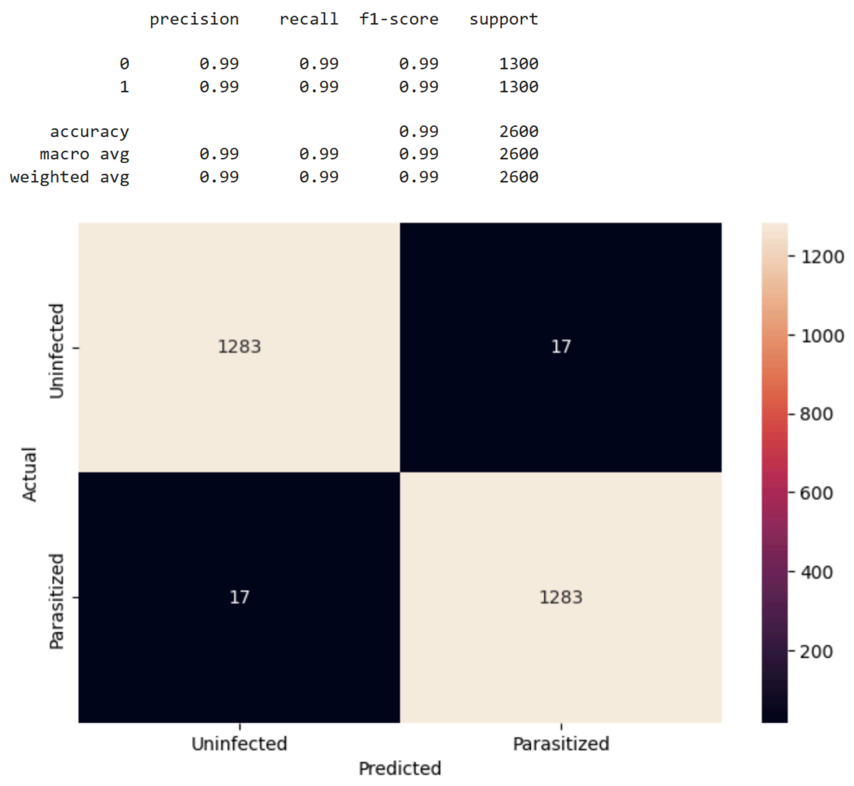
We selected `block5_pool` as the transfer layer. The features output by this pooling layer are used as input to custom classification head. We choose block_5 as it outputs high-level, abstract features.

```
x = Flatten()(transfer_layer.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)

pred = Dense(2, activation='softmax')(x)
```

In our custom classification head, we flatten the features, apply a dense layer with ReLU activation, use dropout and batch normalization for regularization. Finally, output predictions via a dense layer with 2 nodes (one for each class) using softmax activation.

```
history4 = model4.fit(train_images, train_labels, batch_size = 32,
callbacks = callbacks, validation_split = 0.2, epochs = 20, verbose = 1)
```



```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1300
           1       0.99      0.99      0.99      1300

    accuracy                           0.99      2600
   macro avg       0.99      0.99      0.99      2600
weighted avg       0.99      0.99      0.99      2600
```

Figure 2: Classification Report and Confusion Matrix of the pre-trained VGG16 model.