Problem Statement:

The detection of malaria-infected cells in microscopic images is a critical task in healthcare, as it aids in the timely diagnosis and treatment of malaria. However, manual examination of these images is labor-intensive and prone to errors. Therefore, there is a need for automated systems that can accurately classify malaria-infected cells from uninfected ones.

Goal:

Our goal is to develop a robust deep learning model capable of accurately detecting malaria-infected cells in microscopic images. Additionally, we aim to create a user-friendly application using Streamlit that allows users to upload images for classification. The application will not only provide predictions but also visualize the areas within the images that influence the model's decision-making process, using Explainable AI techniques such as heatmap visualization. By achieving this goal, we aim to improve the efficiency and accuracy of malaria diagnosis, ultimately contributing to better healthcare outcome.

Data Information:

Given dataset was manually split into training and testing datasets, maintaining the distribution of infected and uninfected images in each set. The dataset consists of a total of 27,294 images belonging to 2 classes for training and 264 images belonging to the same 2 classes for validation. Here are further details:

- **Training Dataset:** Contains 27,294 images.
- **Validation Dataset:** Contains 264 images.
- **Classes:** The dataset is divided into 2 classes: infected and uninfected cells.

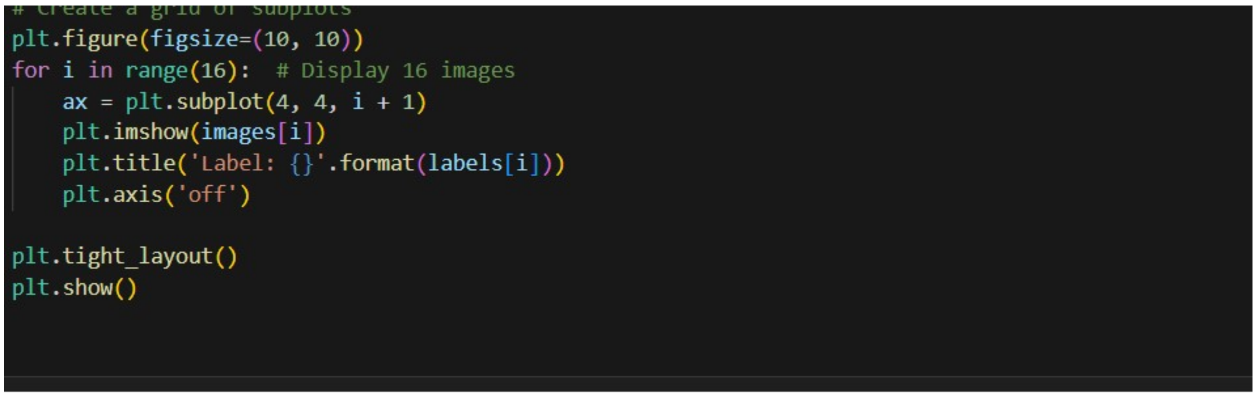
Data Source: https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria

Training and validation datasets are generated using ImageDataGenerator with appropriate preprocessing and augmentation.

Preprocessing:

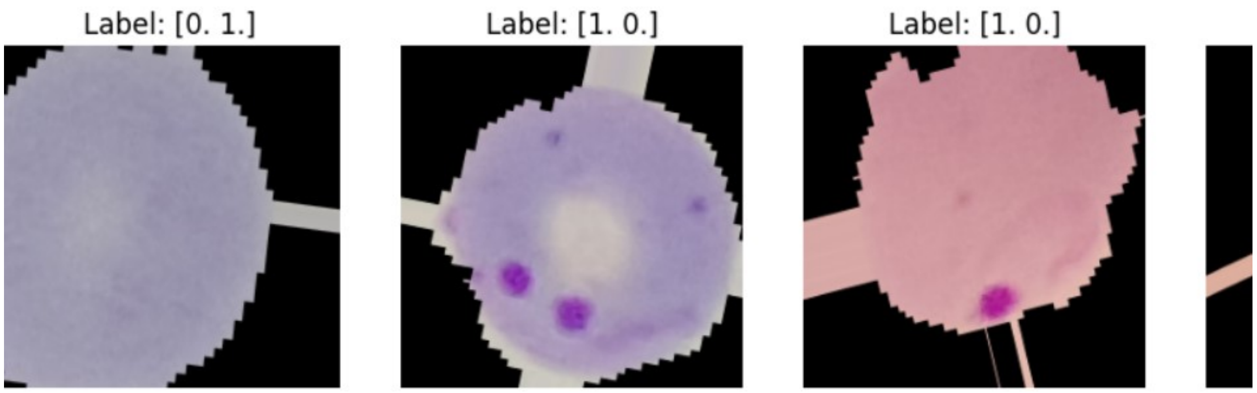
1. **Image Resizing:** All images are resized to a fixed size of 224x224 pixels.
2. **Data Augmentation (Training Set):** Augmentation techniques such as rotation, width and height shifting, shear, zoom, and horizontal flipping are applied to increase the diversity of training samples and improve model generalization.
3. **Data Normalization:** Pixel values of the images are scaled to the range [0, 1].

Data Visualization:



```python
# Create a grid of subplots
plt.figure(figsize=(10, 10))
for i in range(16):  # Display 16 images
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(images[i])
    plt.title('Label: {}'.format(labels[i]))
    plt.axis('off')

plt.tight_layout()
plt.show()
```



Model Architecture:

The CNN model is constructed using the following layers:

1. **Convolutional Layers:** These layers detect patterns like edges, shapes, or textures in the input images, becoming more complex with each layer. Three convolutional layers with increasing number of filters (16, 32, and 64) and ReLU activation function.

2. **MaxPooling Layers:** They shrink down the information by keeping only the most important features, making the model faster and more efficient. Max pooling layers after each convolutional layer to downsample the spatial dimensions.

3. **Flatten Layer:** Flattens the 3D feature maps into a 1D vector. It takes all the complex patterns detected by the convolutional layers and lays them out flat, ready for the next step.

4. **Dense Layers:** Two fully connected (dense) layers with 500 and 2 units respectively. The last layer uses softmax activation for binary classification. These layers analyze all the flattened patterns and decide which combination of features best represents whether an image is infected or not.

Training and Compilation :

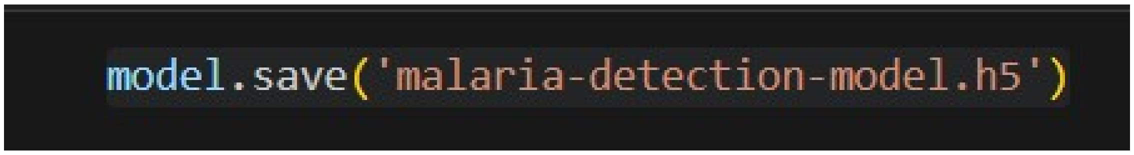
1. Model is trained using above CNN model

2. **Model Compilation:** The model is compiled with Adam optimizer and binary crossentropy loss function.

3. **Early Stopping:** Training includes an early stopping callback to monitor validation loss and stop training if no improvement is observed after a certain number of epochs.

Model Saving:

Once training is completed, the trained model is saved to a file for future use.



```
model.save('malaria-detection-model.h5')
```

Inside app.py

1. **Loading the Trained Model:**
    - We load the pre-trained deep learning model for malaria cell detection using TensorFlow's **tf.keras.models.load_model** function. The model is stored in an HDF5 file format.
2. **Preprocessing Functions:**
    - We define functions to preprocess the uploaded image before passing it through the model.
        - **preprocess_image**: Resizes the image to the required input dimensions of the model (224x224 pixels) and normalizes pixel values to the range [0, 1].
        - **get_heatmap**: Generates a heatmap indicating the areas of the image that influence the model's prediction the most.
        - **overlay_heatmap**: Overlays the heatmap on the original image to visually highlight the influential areas.
        - The areas with high activation are highlighted by overlaying them on the original image. In this case, they are highlighted in red.

The heatmap visualization technique provides insights into how the model analyzes input images, aiding in the interpretation of its predictions. By overlaying the heatmap on the original image, users can visually identify the regions that the model considered most important for its classification decision.

3. **Streamlit Application:**

    - We define the main function (**main**) for the Streamlit application.
    - The title of the application is set as "Malaria Cell Detection App".
    - We prompt the user to upload an image using Streamlit's **st.file_uploader** function.
    - Upon uploading an image, we display it using **st.image**.
    - When the user clicks the "Classify" button, we preprocess the image and pass it through the model to make predictions.
    - If the predicted probability of malaria infection is above a threshold (0.2), we classify the cell as "Infected"; otherwise, we classify it as "Uninfected".
    - We then generate a heatmap using the **get_heatmap** function to visualize the influential regions in the image.
    - Finally, we overlay the heatmap on the original image and display it using **st.image**.

4. **Error Handling:**

    - We wrap the main functionality within a **try-except** block to handle any errors that may occur during image processing or prediction.

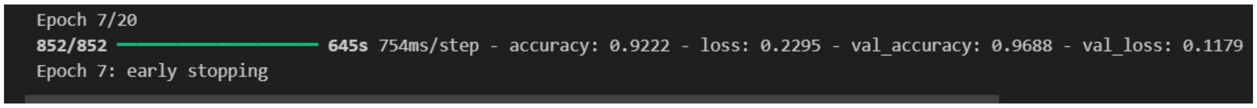
Requirements.txt:

The requirements.txt file contains all the dependencies required by the project to run successfully. Each line in the file specifies a package name and its version number, indicating the exact versions of the dependencies needed for the project.

In this code, **pip install -r requirements.txt** command is used to install all the dependencies inside a virtual environment **malaria-env**

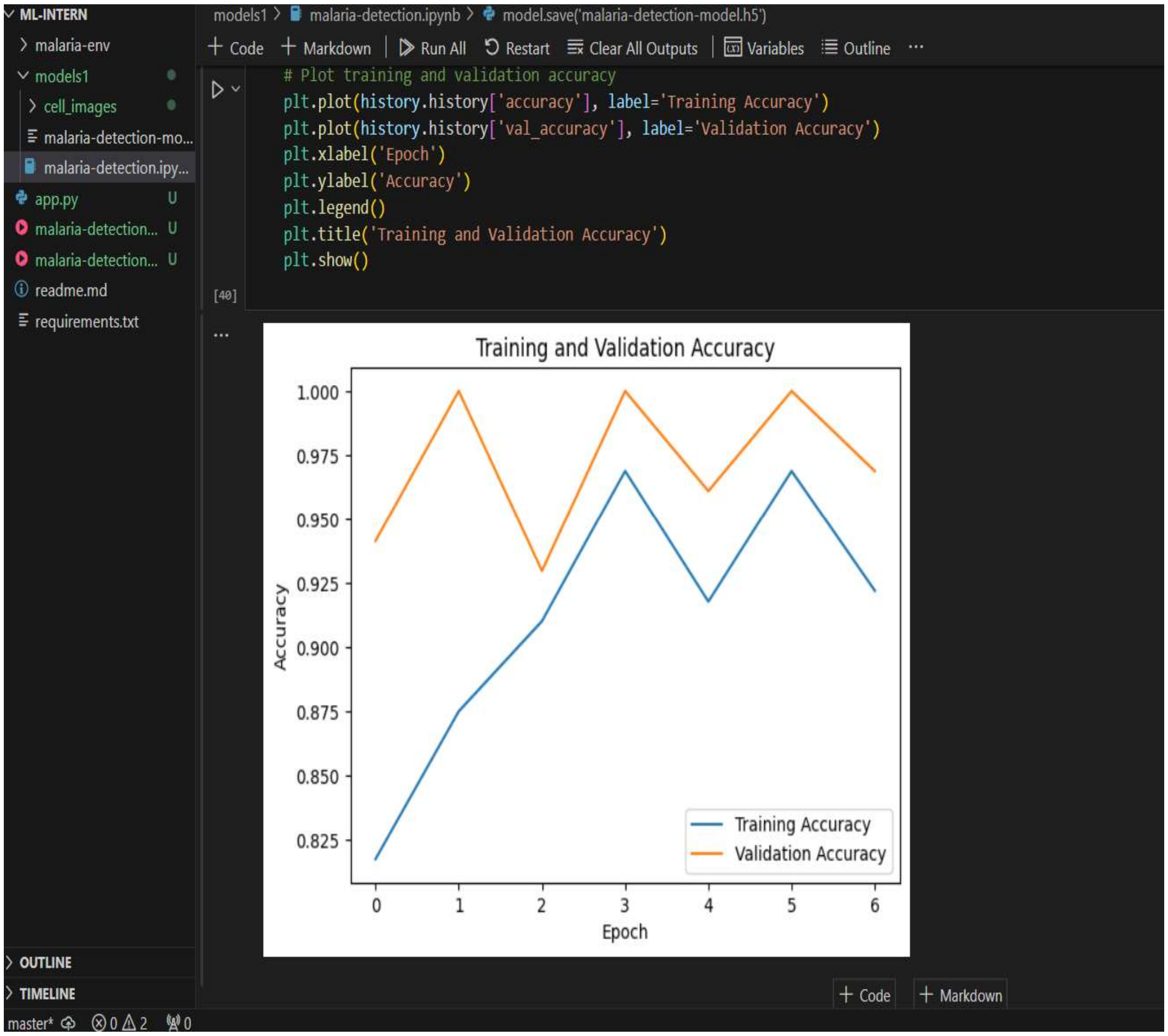
Model’s Performance:

CNN model’s output:



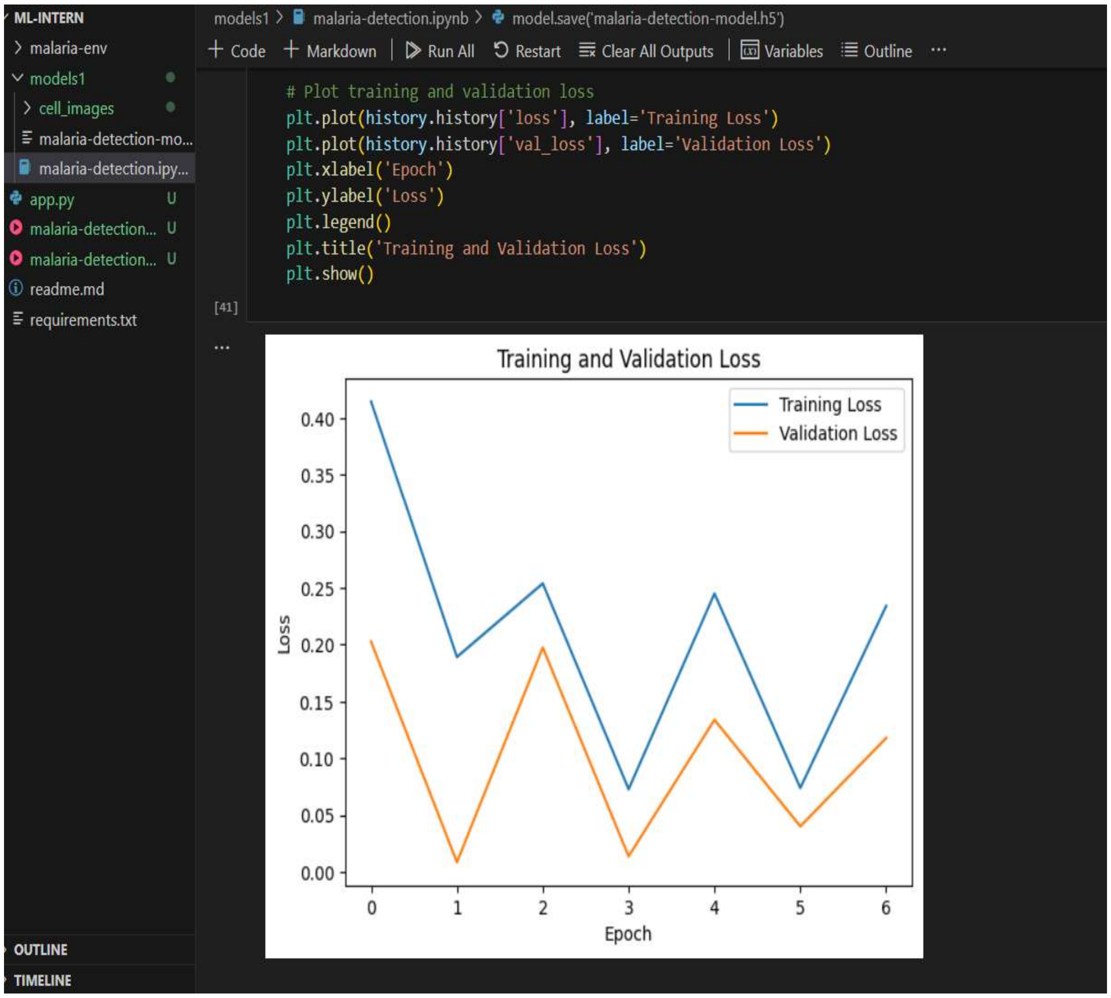
```
Epoch 7/20
852/852 ━━━━━━━━━━━━━━━━━━━━ 645s 754ms/step - accuracy: 0.9222 - loss: 0.2295 - val_accuracy: 0.9688 - val_loss: 0.1179
Epoch 7: early stopping
```

Training and Validation Accuracy:



```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```

Training and Validation Loss:



```
# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```