Malaria Techincal Paper

Problem Statement: Given the dataset containing images of parasitized and uninfected cells, we should build a model that predicts if the cell is infected or uninfected.

Approach:
Given the problem statement, I could think of two possible approach

1. Using the existing Pre-trained model and implementing Transfer Learning to calculate the new weights for the given dataset. I.e. Using the YOLOv5 State of the art model trained on custom data.
   This method is comparatively easy considering that, all I would have to do is augment the data, i.e. label all the data where there is parasite and train YOLOv5 on the new data set. And build a running Application using Streamlit.

2. Since the above method was considerably easy, I preferred to build a CNN model from scratch to predict the infected cells, and once they predict the infected cells it would then calculate the coordinates using the Binary threshold algorithm and apply a bounding box on the image.
   Then create an application that would take in the cell as input and give out the image with a bounding box as output.

The second application is explained in detail.

The provided code is a Python script that utilizes TensorFlow and Keras libraries to build a convolutional neural network (CNN) for classifying images of malaria-infected and uninfected cells. Below is a detailed summary of each step in the code:

1. Importing necessary libraries:
   - `tensorflow`: The main library for building and training neural networks.
   - `os`: Provides functions for interacting with the operating system.
   - `splitfolders`: A library for splitting a dataset into training, validation, and test sets.
   - `cv2`: OpenCV library for image processing.

2. Setting up the GPU support:
   - `os.environ["CUDA_VISIBLE_DEVICES"] = "-1"`: Sets the environment variable to disable GPU usage. This line ensures that the code runs on a system without GPU support.

3. Splitting the data into train, validation, and test sets:
   - `splitfolders.ratio()`: Splits the images in the input folder into train, validation, and test sets based on the provided ratio (70% train, 20% validation, 10% test).
   - The input folder contains two subfolders: "Parasitized" and "Uninfected," representing the infected and uninfected cell images, respectively.
   - The resulting split datasets are saved in the output folder.

4. Reading and displaying an example image:
   - `cv2.imread()`: Reads an example image from the "Parasitized" subfolder.
   - `cv2.imshow()`: Displays the infected image.
   - `img.shape`: Retrieves the dimensions (height, width, channels) of the image.
   - `cv2.waitKey()`: Waits for a key press to close the displayed image window.

5. Setting the batch size:
   - `BS = 16`: Defines the batch size for training the neural network. It determines the number of samples processed before updating the model's parameters.

6. Creating data generators for training, validation, and testing:
   - `ImageDataGenerator`: Generates augmented images by applying various transformations and normalization.
   - `train_datagen.flow_from_directory()`: Generates training data from the "train" folder, resizes the images to (64, 64) pixels, applies augmentation techniques like shearing, zooming, and flipping, and sets the class mode to binary (infected or uninfected).
   - `val_datagen.flow_from_directory()`: Generates validation data from the "val" folder with similar settings as the training data.
   - `test_datagen.flow_from_directory()`: Generates test data from the "test" folder with the same image size and class mode.

7. Building the CNN architecture:
   - `tf.keras.models.Sequential()`: Initializes a sequential model, allowing the layers to be added in sequence.
   - `cnn.add(Conv2D(...))`: Adds a convolutional layer with 32 filters, a kernel size of 3x3, and ReLU activation. The first layer also specifies the input shape as (64, 64, 3) representing the image dimensions (height, width, channels).
   - `cnn.add(MaxPool2D(...))`: Adds a max pooling layer with a pool size of 2x2 and stride of 2. This downsamples the feature maps to extract dominant features.
   - Two more convolutional and max pooling layers are added with the same configurations to further extract features.
   - `cnn.add(Flatten())`: Flattens the feature maps into a 1D vector for the fully connected layers.
   - `cnn.add(Dense(...))`: Adds a fully connected layer with 64 units and ReLU activation.
   - `cnn.add(Dense(...))`: Adds

the output layer with a single unit and sigmoid activation, which outputs the probability of an image being infected.

8. Compiling and training the CNN:
   - `cnn.compile(...)`: Compiles the model with the Adam optimizer, binary cross-entropy loss, and accuracy as the evaluation metric.
   - `cnn.fit(...)`: Trains the model using the training data (`train`) and validates it using the validation data (`val`). The training is performed for 6 epochs, which means the model will go through the entire training dataset six times.

The provided code is a Streamlit application for detecting malaria in cell images. It uses a pre-trained neural network to classify the uploaded image and identify the infected areas within the image. Below is a detailed summary of each step in the code:

1. Importing necessary libraries:
   - `streamlit`: The main library for creating web applications with Python.
   - `pandas`, `numpy`: Libraries for data manipulation and numerical operations.
   - `cv2`: OpenCV library for image processing.
   - `os`, `joblib`: Libraries for interacting with the operating system and job-related tasks.
   - `tensorflow`, `keras`: Libraries for building and training neural networks.
   - `preprocessing` from `tensorflow.keras`: Provides functions for image preprocessing.
   - `Image` from `PIL`: Allows opening and manipulating images.

2. Setting up the Streamlit application:
   - `st.title()`: Sets the title of the application.

3. Defining the main function:
   - `st.file_uploader()`: Provides a file uploader widget for selecting an image file.
   - If an image file is uploaded, the function proceeds with image processing and classification.

4. Image processing and classification:
   - `Image.open()`: Opens the uploaded image file.
   - `st.image()`: Displays the uploaded image in the application.
   - The uploaded image is resized to (64, 64) pixels, converted to an array, normalized to values between 0 and 1, and expanded to include a batch dimension.
   - The `predict_class()` function is called to classify the image as either "Parasitized" or "Uninfected".
   - The classification result is displayed in the application.

5. Detection of infected areas:
   - The uploaded image is converted to an array.
   - The image is converted to grayscale using OpenCV.
   - A binary thresholding operation is applied to obtain a binary image.
   - Contours are extracted from the binary image using OpenCV's contour detection function.
   - Bounding boxes are calculated around the contours to identify the infected areas.
   - The `add_Bbox()` function is called to draw bounding boxes on the original image.

6. Displaying the result:
   - The image with bounding boxes is displayed in the application.

7. The `predict_class()` function:
   - The pre-trained model is loaded from the file "Malaria_neural_network.h5".
   - The image is passed through the model for prediction.

   - The class with the highest probability is determined, and the corresponding class name is retrieved.
   - The result is formatted as a string.

8. The `get_infected_area()` function:
   - The image is converted to an array.
   - The image is converted to grayscale.
   - A binary thresholding operation is applied to obtain a binary image.
   - Contours are extracted from the binary image.
   - Bounding boxes are calculated around the contours.
   - The `add_Bbox()` function is called to draw bounding boxes on the original image.

9. The `add_Bbox()` function:
   - Draws a bounding box on the image based on the provided coordinates.

10. The main function is executed if the script is run directly.

Conclusion: The model and the app run just fine but the algorithm to generate the bounding box still needs to be optimized to get the exact coordinates of the parasite, which could not be implemented due to lack of time.

Thank you.