Malaria Cell Classification And Development of Streamlit Application for Classification

By

Shridhar S Benni

Student at MSIS pursuing AIML

Introduction:

Image classification is a common task in the field of computer vision, which involves training a model to classify images into predefined categories or classes. Streamlit is an open-source Python library that allows developers to create interactive web applications for data exploration and visualization. In this report, we will discuss the implementation of image classification in Streamlit, including the necessary steps and considerations.

Dataset Preparation:

The first step in implementing image classification is to prepare the dataset. This involves collecting a large number of images and organizing them into labeled categories. The dataset should be diverse and representative of the classes you want to classify. It is crucial to have a sufficient number of images per class to ensure a robust and accurate model. Dataset can be accessed by this link https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria this dataset has two folders namely “Parasitized” and “Uninfected”, ipynb file in models folder will guide us how to load the data in the code for model training

Model Training:

Once the dataset is prepared, the next step is to train a model for image classification. Popular deep learning frameworks such as keras or TensorFlow can be used for this purpose. I am using Keras sequential model. This model is train on the above data set.

During training, it is essential to split the dataset into training and validation sets. The training set is used to optimize the model's parameters, while the validation set is used to evaluate the model's performance and tune hyperparameters. It is crucial to prevent overfitting by applying techniques such as early stopping and model checkpoints. I have used callbacks (early stopping and model checkpoints) while training the model because the data set is large. By using callbacks, model has achieved 90 percent accuracy in less time.

Integration with Streamlit:

Once the model is trained and evaluated, it can be integrated with Streamlit to create an interactive web application for image classification. The following steps outline the process:

Install the necessary packages: Begin by installing Streamlit and other required libraries using pip or conda.

Build the user interface: Use Streamlit's intuitive API to design the user interface. This may include elements like file upload buttons, image displays, and result visualization.

Load the trained model: Load the trained model weights and architecture using the deep learning framework of your choice.

Preprocess input images: Before making predictions, preprocess the input images according to the requirements of the trained model. This may involve resizing, normalization, or other transformations.

Make predictions: Utilize the loaded model to make predictions on the input images. The predicted classes can be displayed alongside the images or in a separate section of the Streamlit application.

Visualize results: Streamlit offers various visualization capabilities. You can display the input images, their predicted labels, and any additional information or metrics you want to showcase.

Deploy the application: Finally, deploy the Streamlit application to a server or cloud platform so that it can be accessed by users. Services like Heroku or AWS can be used for deployment.

Conclusion:

Implementing image classification in Streamlit provides a user-friendly and interactive platform for showcasing and utilizing trained models. By following the steps outlined in this report, you can create an effective image classification application that allows users to upload images and obtain real-time predictions. Streamlit's simplicity and flexibility make it a powerful tool for developing and deploying machine learning applications.