EasyRFP: An Easy to Use Edge Computing Toolkit for Real-Time Field Phenotyping

Sai Vikas Desai[†1], Akshay L Chandra[†1], Masayuki Hirafuji[2], Seishi Ninomiya[2], Vineeth N Balasubramanian[1], and Wei Guo[2]

[1] Indian Institute of Technology Hyderabad
{cs17mtech11011, akshaychandra, vineethnb}@iith.ac.in
[2] The University of Tokyo, Japan
{hirafuji, snino, guowei}@g.ecc.u-tokyo.ac.jp

Recent advances in deep learning have catalyzed rapid progress in high throughput field phenotyping. Much research has been dedicated towards developing accurate and cost effective deep learning models to capture phenotyping traits such as plant stress, yield and plant growth stages. However, there is a shortage of software tools to promote the usage of such intelligent methods among plant phenotyping practitioners and researchers. To bridge this gap, we developed EasyRFP [1], a Flask back-end, Angular font-end software toolkit which can be interfaced with any commercial GPU enabled micro computer (such as NVIDIA Jetson) and a digital camera.

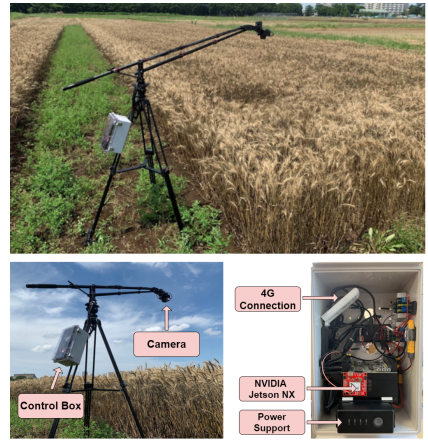

Fig. 1. A typical on-field setup looks as shown above (top). The control box hardware configuration is shown at the bottom.

With camera and control box setup in an appropriate position in the field as shown in Fig. 1, our tool can be used in two modes: (i) Real-Time mode: Capture images and process them in real-time to obtain phenotypic traits, and (ii) Scheduler mode: schedule the system to capture images at fixed time intervals and notify the user with the results obtained by the deep learning model. Real-Time mode is well-suited for situations where instantaneous and on-demand observation of crop is desirable. Fig. 2(a) shows the workflow of real-time mode. The user only have to configure the real-time mode parameters - task, recipient email(s) in the UI before starting the process. Scheduler mode is suitable for periodical inspection of crops at fixed intervals. In this mode, the toolkit runs a long process in the background. Fig. 2(b) shows the workflow of scheduler mode. Here, along with the task, the user is expected to configure the scheduler time parameters - capture interval, notification interval, total time of the process. For instance, the tool can be configured to run a long process, say for 48 hours

†Equal Contribution

while capturing and processing camera input every 1 hour and send the logs via email every 12 hours. Details of where the user can access the results (log files and images captured) outside the UI can be seen both on the logs section in UI and in the email. To achieve this, we wrapped the scheduling and the real-time code in Python's Flask framework (back-end) and then we established a web socket connection to periodically send updates of the progress and results to the front-end. We chose Angular 7 as the front-end framework, which actively listens to the the web socket messages as and when sent from Flask.

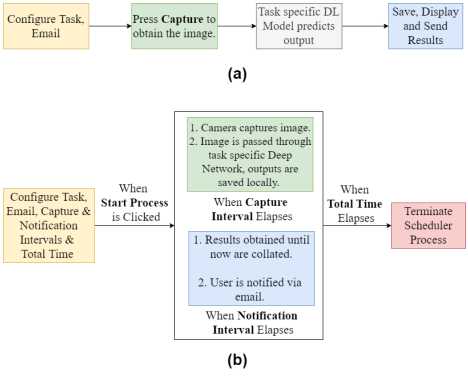

Fig. 2. Workflow of **(a)** Real-Time Mode **(b)** Scheduler Mode

Our toolkit can be seen as a wrapper as it can work with any pre-trained model, provided that its learning framework (ex: Tensorflow, PyTorch, ONNX) is supported by the underlying embedded AI device (NVIDIA Jetson NX). It allows for users to seamlessly add tasks and use their trained models on custom plant phenotyping datasets of their choice. For a custom trained model, users can copy the model to the *tasks/models* directory inside the toolkit's root directory. The user is expected to place a python file, say *task_name.py*, with a class that particularly implements the method *perform_task()*. The class constructor should take an image as input and this method is expected to output specific things for specific tasks. If it is a classification task, the function should return a tuple with the predicted class name and the predicted probability *(class name, pred_prob)*. If it is a detection task, the function must return a list of tuples, where each tuple looks as follows: *(class_name, [x, y, width, height])*. Each tuple corresponds to a single predicted bounding box, consisting of the class name and the bounding box information. Our code currently supports classification and detection tasks but can extend to any task provided the output format is appropriately handled for logs in *api/utils.py*.

We refer the readers to our code repository for code examples, demos, execution details and also for a detailed tutorial on how to integrate new tasks [1]. In the future, we hope to make the tool work on lighter embedding devices such as Raspberry Pi or Arduino UNO, perhaps by limiting the device activity to capturing images and moving the inference and reporting (logging and emailing) modules entirely to a cloud based server.

References

1. Sai Vikas Desai, Akshay L Chandra, M.H.S.N.V.N.B.W.G.: Easyrfp: An easy to use edge computing toolkit for real-time field phenotyping. https://github.com/lab1055/easy-rfp (2020)