This document describes a technical challenge that forms part of the Hectre recruitment process for Spectre back-end engineers.

It should not take more than one working day. Don’t worry if the project isn’t complete by then

- just email back what you have with some notes on where you got to and what’s left to do.

# Background

Spectre AI tool is one of Hectre’s products that utilises machine learning, computer vision, and cloud computing to provide size & colour distributions for fruit bins within seconds from uploaded images/videos.

Here is one example of uploaded images for Kiwifruit with expected diameter lines.



Your task is to design and deploy a backend machine learning solution that could allow the frontend to upload image(s) and save the information to a structured database. The tasks are:

1. Design the solution/service architecture (ideally combining with cloud platform)

My solution

1. Deploy one Python backend API endpoint for uploading one or many images with at least the following information:
   * Image ID (Primary key)
   * Created date (UTC format)
   * Fruit type
   * User email
   * Tenant ID

And save this information into a relational database.

My solution to this objective is given in the files 1) “*initialize\_backendAPI.py*”, 2) “*db\_add\_initial.py*” and 3) “*db\_query.py*”. Basically, the code is run from 1) to 3) as labelled. “*initialize\_backendAPI.py*” utilized the Flask API and SQL Alchemy to create a database and provided a way to input data into the database through parsed arguments. Please excuse my descriptions as I may not be using accurate software development jargon. I also added an additional input argument for the file path of the image. My assumption for this problem was that the input arguments were already available to me through the front-end user interface, and somehow the user interface passed this information to my code. To test the code I used “*db\_add\_initial.py*” which inserted 3 samples of manually defined rows into the database. Basically the 3 samples were manually parsed, while in reality they would be automatically fed into the script. “*db\_query.py*” then did a few get/put/delete methods on the database to check that I defined the database correctly.

In terms of uploading the images, I specified the image path as an input argument and read the image in with open-cv, then saved it into a folder as I thought filesystem is more efficient that SQL databases for storing images. I just used the image provided in the first page as input/output such that the input path was simply my current repository, and the output path was just to a folder I just created inside the repository. However, I’m not entirely sure this is the correct way to do it as it feels kind of wrong (i.e. using open-cv inside Flask to read and write images like this). I also tried some other code at around line 193 of “*initialize\_backendAPI.py*” which attempts to route directly to an interface which receives the data, then stores it into the SQL database. But since I did not have an interface I did not test it so I commented it out.

For the “relational database” part, I could not demonstrate any relationality as there was only basically one table. I am also not entirely sure if the database I defined in SQL Alchemy is relational for this task (i.e. one to many, many to one mapping etc). But the documentation said SQL Alchemy used ORM, so I was not sure if relationality is just inherently present in all databases defined using this library.

1. Demo test with simple Unit test(s)

My solution

1. Document the detailed steps on how to deploy and run the code in Mac/Linux Environment

I’m not sure exactly where I should outline this info so I am doing it here. I will also include it in the README. Most of the work involves installing the dependencies. I also think that you can have all the commands below inside one single bash script so you don’t have to do each step individually. But for now I’m just sticking to what I know works.

1. SSH into the desired server (could be cloud, hpc cluster, or just a local server)
2. Set up virtual environment (or could be container):
   1. mkdir Virtual\_ENV
   2. cd Virtual\_ENV
   3. virtualenv -p /usr/bin/python3.6 Virtual\_ENV

This will then install python/pip and other basic tools

1. Activate virtual environment:
   1. source Virtual\_ENV/bin/activate
2. Install the python dependencies which are in the requirements.txt file that I have provided in the repository:
   1. pip install –r requirements.txt
3. The python code can then be run through commands in the terminal etc:
   1. python sample.py

But make sure to activate the virtual environment in #4 every time you reconnect to the server.

1. Deploy the one of core blocks of solution on cloud with Infrastructure as code (IaC) (In case of designing solution with cloud)

My solution

1. One of Spectre AI features is grading fruit size. Assume that we have to develop a machine learning solution to find the pixel diameter of Kiwifruits. From the visualization image above, can you describe your practical solution:
   * Which kind of ML model should you use? Why?

The primary group of neural networks to be considered is ones designed for instance segmentation. For example Mask R-CNN, SSD, YOLO etc. By performing segmentation of all objects of interest in an image, the subsequent measurements of individual objects can be easily calculated from the mask. The presence of the segmentation masks also allows for easier extraction of fruit measurements in oddly shaped fruits.

Mask R-CNN has several advantages over single-shot methods such as SSD or YOLO. Mask R-CNN utilizes two parts, the convolutional feature extractor, and the region proposal network, both being trainable end-to-end. The convolutional feature extractor produces an abstract representation of the initial input image by mapping it onto a larger number of feature maps as opposed to the 3 feature maps of the initial image (represented by R/G/B channels). The abstract representation then provides more information to the region proposal stage. The regional proposal network then learns potential regions of interest with bounding boxes (defined by 4 coordinates and 1 value for the confidence) which are then segmented by a subsequent network. In this case, the convolutional feature extractor can be re-used for segmentation to avoid additional parameters. The major advantage of Mask R-CNN is the fact that the detection and segmentation are unified into a single framework, making the approach extremely robust given sufficient training data. Prior research has also demonstrated its efficacy on segmenting small objects, which is important for fruit segmentation. The fully trainability of the region proposal network also means computational costs with estimating potential regions of interest are minimized as the network learns this part on its own. However, the major disadvantage is the difficulty in training the network, as several objective costs has to be balanced to obtain a desirable outcome. These optimization objectives include the accuracy of the region proposals (classification loss and bounding-box regression loss), and the accuracy of the final detected objections (also with classification loss and bounding-box regression loss). The network then has to be optimized for accuracy of segmentation in each bounding box, introducing an overlap loss for segmentation. The current state of literature suggests Mask R-CNN is the state-of-the-art in instance segmentation.

Single-shot methods such as YOLO (you only look once) and SSD (single-shot detection) are much easier to train than mask R-CNN. However, less flexibility is offered as the regional proposals are predefined manually instead of learnt by the network. The method breaks an input image into sub-regions, then optimizes by adjusting the set of estimated bounding box coordinates in each sub-region to the ground truth location of the object of interest. While this allows for the entire processing to be done in one step, the loss function is significantly larger as it is now N x N x (5 x B + C). where N = the number of sub-divisions in each of the x/y axis, 5 is for 4 values defining the bounding box and 1 value defining the confidence, and C is the score for the class prediction. In some ways, this loss function is also hard to optimize. However, the execution speed of such methods is significantly slower, as much inefficiencies are introduced by the repeated computation of the manually defined set of bounding boxes. This may be problematic if the solution needs to be deployed in real-time at a high frame rate.

If the task to be address can be achieved through the use of a bounding box outline on an object of interest, then YOLO/SSD should be considered due to the simplicity in the method. However, if the objective is to eventually obtain accurate measurements of all objects of interest in an image, Mask R-CNN can potentially be more accurate and computationally efficient. A Mask R-CNN with a shallow Res-Net convolutional feature extractor is also fairly lightweight in terms of memory consumption in comparison. This, in addition with using deep neural network compression, may possibly make mask R-CNNs embeddable into apps.

* + Any idea about data, training ML?

The most important aspect of any machine learning pipeline is the datasets. Images should ideally be high resolution, sharp, and contain high contrast. The images would also be fully labelled. Ideally, each fruit (in a bin for example) will have their own independent manually segmented mask e.g. an image with 50 apples has 50 different annotations of each individual apple. The higher the quality of the data label, the more flexibility and potential for future development can be made available, as labels with higher complexities can be manipulated in more ways. Worst comes to worst, a complex label can always be simplified (merge all 50 masks into one mask for example), but a simple mask cannot be made more complex easily. It is therefore imperative to optimize the manual labelling workflow from the very start so that time is not wasted, as human labor is expensive and time-consuming.

The machine learning algorithm would be trained on the largest dataset possible, with the dataset being split into train/validation/test (usually 70%/15%/15% in the case of very large datasets). The validation accuracy is constantly evaluated during every epoch to ensure the algorithm is not over fitting to the training set. Once the accuracy stops increasing on the validation set, the algorithm is then tested on the test set to check the accuracy. The difference between the training accuracy and the validation accuracy is a result of variance in the dataset, and the difference between the validation and testing accuracy is a result of overfitting of the validation set.

Once the algorithm is trained, the network parameters can then be saved and used for initialization in the next training session so backpropagation and stochastic gradient descent does not need to start from scratch. Transfer learning can also be leveraged to extend the algorithm onto different fruits i.e. an initial large apple database can be used to pre-train a network which is then fine-tuned on a much smaller orange dataset resulting in accuracy orange detection despite the small dataset.

Other practical considerations would be the training time and network complexity trade-off, speed of data access during training, systematic methods of hyper-parameter tuning, which are all important in minimizing the computational resources and time taken to develop a satisfactory model. Especially since hyper-parameter tuning is considered a “dark-art” and is very difficult to do efficiently.

* + Any idea about organizing/deploying ML model on production?

minimize size of model

* + Post-process algorithm idea to the get expected diameter size

If the output is a mask of each individual fruit and the fruit is assumed to be spherical, then the diameter of the fruit can easily be extracted by measuring the diameter of the masks. The real diameter can then be calculated by multiplying the number of pixels with the image resolution to obtain a measurement in cm or mm. Multiple diameters can be taken to increase the confidence of the estimation, as this can be vectorized and does not contributed to much more increased computational cost. I will discuss measuring oddly shaped fruits in the section below.

Once the diameters of all fruits in a bin are obtained, a further step for checking can be performed by performing a similarity test between the current distribution obtained versus a collated distribution of all previous measurements for the same fruit. If the distribution is shifted to the right or left, there may be an offset error. If the shape of the distribution is odd (e.g. bimodal when its unimodal), then there may be errors in the masks or method of extracting the pixel diameter.

A more “fancy” approach would be to embed the diameters of the fruit as an additional label in the training datasets. This would be done either fully automatically on known distributions or semi-automatically with human quality control. The Mask R-CNN network can then be trained with an additional branch in the output to predict both the mask and the diameter of each fruit. This loss function is the case would be a regression loss trained with a mean squared error objective. Similarly, if we are only interested in the diameter (and not for example, the volume which estimates the weight), then we can train a single-shot network to simply predict the bounding box of each fruit along with its diameter as a single additional output. However, this would be less scalable to other fruits.

* + How can we best deal with the flat shape of a Kiwifruit?

fit an ellipse shape (provide diagram)

The deliverable product should contain:

* A design solution with diagram(s) and explanations
* A Python application

Python files are all in the repository

* A .git folder with the commit history for the project

Folder provided in repository

* A README that explains how to install and use the project

README.md provided in repository

The project should work on macOS or Linux.

Feel free to document any assumptions made, or ideas for improving the project. Don’t upload the project to a public repo since it might be unfair for other candidates.

# Application notes

* The Python scripts, packages are compatible with Python 3.6, 3.7, 3.8.
* Use Flask API or other blueprint for backend APIs
* Use a PostgreSQL or other structured database, including DB script/code for creation.
* It is better to use ORM (Object Relational Mapper) and a suitable backend design pattern.
* Ideal unit test is Pytest but not limited to this option