Building up a data pipeline

In this class, we put the basic building blocks together to decide on how the image files will be obtained, identified, circulated, and how the results will be presented to solve the security problem. We will design the system such that we can process the images on EC2, which will have Pachyderm on the top and images will be stored in S3. The objective of designing such as system is that it will allow us to productionize workflow using codes written to process from a laptop. In the coming sections, I will describe in detail the input, output and the pre-processing, object detection and post-processing stage of the system architecture.

Input

The input for our image detection program is the video recording from the security cameras installed outside doors or building entrance, or parking garage etc. A client can have cameras installed at different parts of the property. Images can be obtained by taking individual frames out of a video. We will not worry about the actual mechanism of how that data will come to us as discussed in class. Each client will have different requirement depending on the purpose of security, hence images may require different treatment to detect threats.

Output

The output of the system has 2 end users: 1. Actual users (Clients), 2. In-company analysts (us)
A consulting company creates a database with dashboard for its customers. But for the class exercise, we can create a bar plot of which locations are giving most alert, how frequently a user is alerted etc.

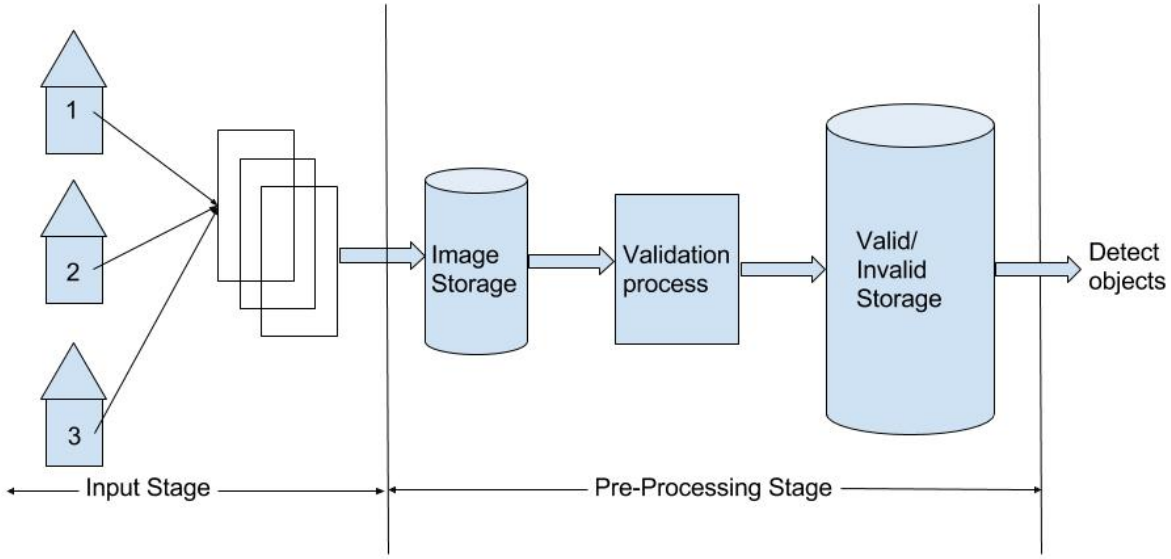
The form of output will be camera image with text. The text will comprise of the camera/ location from which the intruder was detected and the time and date the footage was captured. For the purpose of this project an email will be sent to client using an email marketing service from providers such as SendGrid, Nagios, and PagerDuty. The other options include, notification will be sent from alert service using API call in a JSON file. Notification is meant to alarm the user for

an intruder, so that immediate action can be taken to prevent intruder from causing substantial damage. The other form of output that can be considered is text description of the object instead of an image. Calling the user through automated service will alert the client and advise them on future course of action.

Model Architecture:

Our basic system architecture for project has stages like 'Input of Images – Preprocessing – Detect objects – Postprocessing – API call alert service and bar plot'

First half of our model is described below



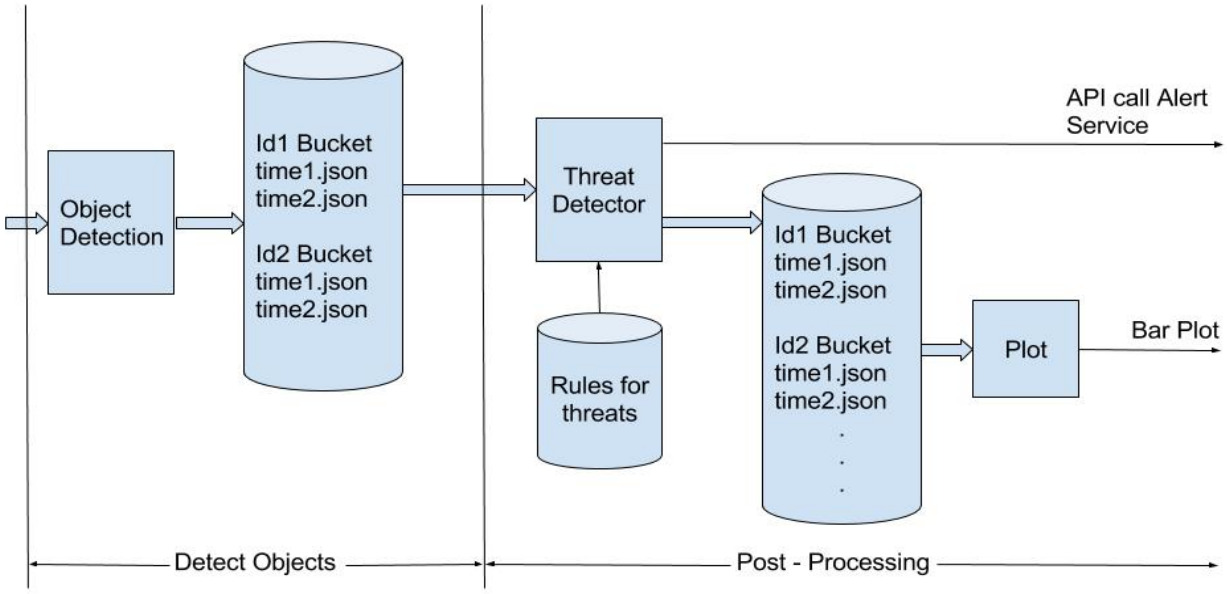
Input Stage:

There must be an agreement on what format of image is coming, so that it becomes easy to process the image. We would require standardizing the size of image, format etc. The images will be grouped and named according to each individual user and their device so that they can be uniquely identified. The format for names can be the 'id_timestamp.jpg', the 'id' will contain the user code and device code. The timestamp will be in UTC format, which will help us maintain consistency across different time zones.

Pre-Processing Stage:

The images are all stored in a database on cloud. In our project, format of image file and quality of file is like a constraint. So, some sort of validation on properties of image: whether file is jpeg, following some standards on quality etc. needs to be performed. All the images must have a consistent format, good resolution and good tag. Then some images will not be meeting standards, we can store them separately as a group of invalid images. The valid images can be separated and stored accordingly, this is to ensure that we get valid objects at the output. We can later explore the source of invalid images in our system and correct them when required. For e.g. If a camera is malfunctioning, it can be replaced. Once this is done, we can then move the standardized images to the object detection stage.

Second half of our model is described below



Detect Objects:

The images fed to the Object Detection process will at output consist of class of object, individual probabilities of each detected object on image in a 'json' file format. These files will be named according to the 'timestamp' format and lumped according to 'ids' – so that the process of threat detection will be smoother as groups are processed with 'for loop'. The Object detection will be done with the help of TensorFlow algorithm in Python on AWS EC2 instance. As discussed before, we are using TensorFlow because it is open-source, portable, flexible, and has object detection

API. Different classes of objects will be identified at the output of this stage for images in different buckets.

Threat Detection:

We won't combine Threat detector with Object detection process as we can personalize and classify threats as defined by users. This approach is particularly good for scaling, provides flexibility, visibility of the classes and threats detected by the system. So, clients with different needs can be served with the same architecture and won't require big changes in codes at Object Detection block, instead a small tweak of threat category will be required in Threat Detection block. The purpose of security cameras will determine what is threat/ non-threat for each client. So, we need to write threat for each client in a 'threat_userid' json file which will be stored in 'Rules for Threat' database. The Threat detector output would use an API call alert service to send a text message/ email/ in-app notification to the client. For the purpose of the project, we will use email. We would follow the principle of Continuous Integration and Continuous Deployment. We can use these approaches in developing the software model for detecting images and improving the way we provide alerts to our clients. The code can be continuously updated on repository like Github, where code can be verified, and incremental updates can be made. The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production.

This architecture will not process historic data of all images, but process only new images. Pachyderm will do de-duplication, thus there will be no duplicates that are stored on a static storage cloud as Amazon S3. We will use Pachyderm to schedule our workflows, to process the images that we will receive from unique 'id's. S3 will be used only to store the historical files, say for the last 1 year from all the clients, due to regulatory requirements. S3 will be best suited for this purpose because it has object storage capabilities. At the output of this project, we would like to see the bar plot of number of alerts at different locations. This will help the company analyst understand any anomalies in threat detection and correct them as soon as they arise.

Ideally, the consulting firm would create an analytical dashboard for its client which is improved upon incrementally.