ECE 498 ICC
Final Project Report
Kanchi Shah (khshah6)
Kushal Goenka (kgoenka2)

Seat Finder

Project Track and Platform

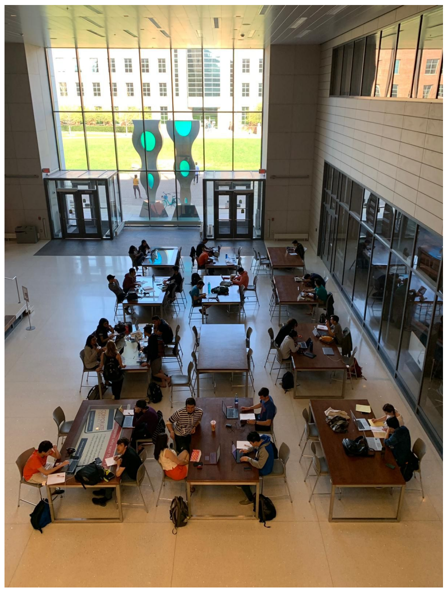
We decided to pursue a project from Track 1: IOT Systems. We plan on utilizing the Raspberry Pi CPU, along with a Raspberry Pi Camera for data collection. We decided against using amazon AWS as it wasn't economical given we wanted to send images at a very frequent rate for the most amount of uptime.

Our project initially focussed on helping students find a lecture hall seat in the least amount of time and with minimal disturbance to their peers. Our system would involve a large array of cameras (just one for demo purposes), positioned at the top of the lecture hall, facing downwards (birds-eye view), such that every seat is accounted for and visible. It would then determine if the seat is occupied or vacant, using function calls from the openCV api. Our system would ideally ignore any objects that are not students (eg: bags, jackets etc). On further consideration, we realised that getting such lecture hall data was proving to be more difficult than we had previously thought and less tangible in the amount of time we had available. We decided to move ahead with a similar iteration of our project. We currently try to predict which seats are vacant and which are occupied in the ECEB atrium.

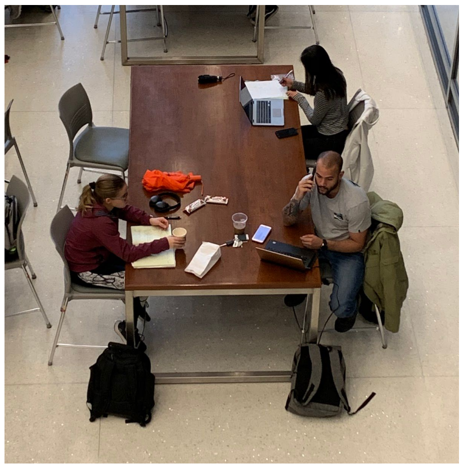
Problem to Solve and Associated Challenges

During midterm/finals week or even during weekends, we noticed that the atrium is extremely busy and we wanted to solve the problem of trying to find a place to study for students. With our platform students will be able to know in real time how many seats are vacant in the atrium and whether they should come to the ECEB to work on their projects or go someplace else. This would save precious time looking for a place to study. Our design is replicable and can be distributed to be used in many different scenarios, and if enough data is available, even lecture halls.

The first challenge that we faced was actually getting pictures collecting data for our training/detection and proof of concept. It was proving to be difficult to get a bird's eye view of the atrium or lecture halls with and without students. We solved this problem by making our model more robust and only taking pictures from a side angle as shown below.

`` 

Secondly we needed a camera resolution that was high enough to resolve all the tables extremely well and actually be able to detect the chairs and people accurately. Since the raspberry pi camera didn't come with such a high resolution, we decided to focus on a single table at a time, as shown below.



Moving on, the next challenge we faced was actually detecting the chairs, tables and people sitting on them. This was particularly difficult due to the angle with which we were taking images. We implemented this by making our own implementation of the popular YOLO object detection paradigm and CNN architecture. Understanding how YOLO works, and implementing the same using PyTorch was another hurdle which we faced in our project.

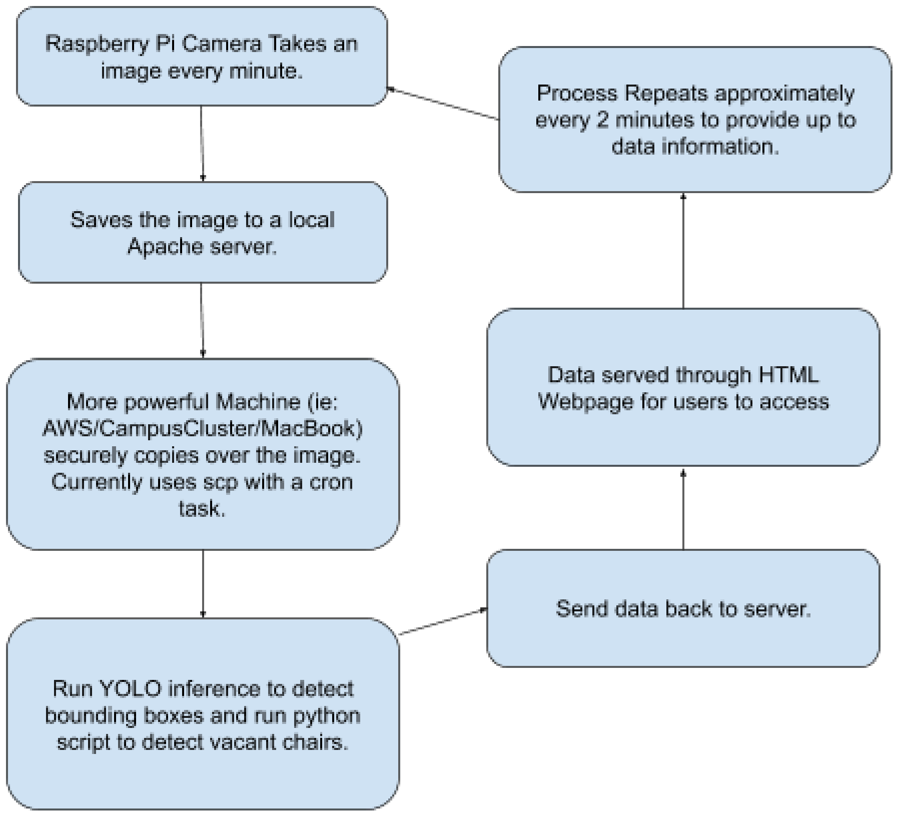
Lastly we faced the challenge of using the Raspberry Pi with the University Guest network to send and receive images. However, we were able to create cron tasks and automate the same result.

Detailed Design Characteristics

Communication Topology

Our current design depends on the fact that the user is on the University network while trying to access the data. In the future we would ideally want to have the data be sent to a server with a static domain name. The flowchart below illustrates the communication flow in this project.

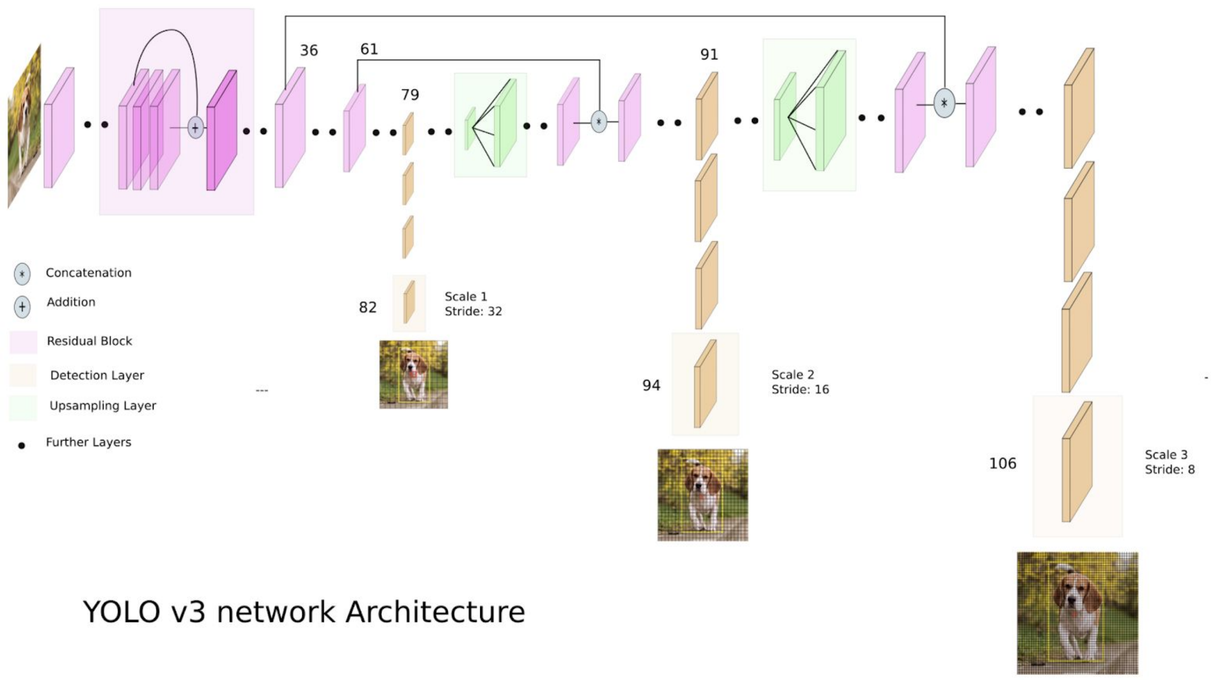
Project Flow Chart


Raspberry Pi Camera Takes an image every minute.
Saves the image to a local Apache server.
More powerful Machine (ie: AWS/CampusCluster/MacBook) securely copies over the image. Currently uses scp with a cron task.
Run YOLO inference to detect bounding boxes and run python script to detect vacant chairs.
Send data back to server.
Data served through HTML Webpage for users to access
Process Repeats approximately every 2 minutes to provide up to data information.

YOLO and Object Detection

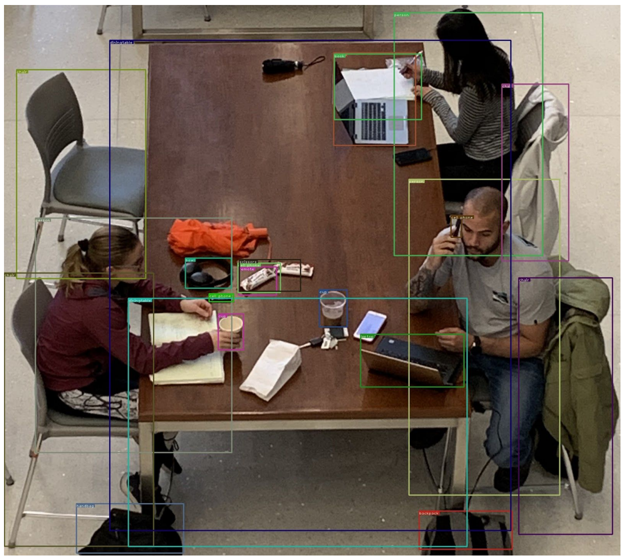
Our inference was conducted through a method known as YOLO - You only look once. We used the Yolov3 model with pre-trained weights, trained on the COCO dataset. The COCO dataset contains over 160K images with labels on over 80 common objects, including dining table, chair, and person. Due to the size of this dataset, we didn't have the compute resources to actually train the model on Chairs, tables and people and generate our own weights. We believe that if we were able to, we would get much better accuracy for our model and detection.

The model was built by following the tutorial from reference 3, using Pytorch rather than Keras to construct the layers. Our network contains 75 convolution layers, with skip connections and upsampling layers, as well as a convolution layers with stride 2 in order to downsample feature maps. The network architecture is illustrated in the diagram below.



https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

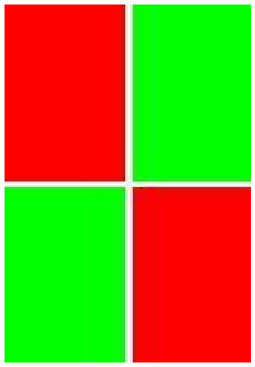
The code for this model is found in yoloLOL.py as well as util.py. Images of a table in the ECEB atrium (left), as well a the same table with bounding boxes drawn from our model (right), are shown below.



Vacancy Determination and Output Format

The model detects objects such as chairs and tables, then outputs the coordinates of the bounding boxes as well as the corresponding labels. We then developed an algorithm to determine which seats are occupied and which are vacant using these coordinates. This was done by creating arrays for the chairs and people, and comparing the Euclidean distance between the center of each chair bounding box and each person bounding box to “assign” a person to a chair. The chairs that were assigned people were determined to be occupied, and the remaining were determined to be vacant. This algorithm is found in final_detection_python.py.

Finally, our system serves this data in a visual grid format, loosely resembling an airplane seating chart. An example where the upper left and bottom right seats are occupied, and upper right and bottom left seats are vacant, is displayed below.



The key features of this project are the ability to receive the image, process it to accurately analyze which seats are vacant, develop an accurate seating chart with this data, and send the data to the user. We automated all of these tasks in the python script camera_script.py.

Design Evaluation

Our design was unique in the sense that we could not find any additional projects trying to solve the problem we are solving in spite of the fact that we think its a very real and important problem to solve. This solution saves many students, such as ourselves, a lot of time trying to find a place to study or work collaboratively. Our design might also be superior to those which are dependent on certain pressure sensitive hardware to be implemented in chairs to detect people, which come with it many challenges and difficulties. The design we have proposed and shown a proof of concept for is highly scalable as well as distributable. With high resolution imaging, which has become easily accessible today, it can be easily implemented in large seating areas in a variety of different situations.

We saw that our YOLO model took approximately 15-20s to actually process the computation inference on a Quad core machine with 16 GB of RAM. However, if we were to improve our model to only be trained on the required objects that we wish to detect, we could reduce the model size considerably while not needing a very powerful processor or the large amount of time and resources it takes to process a single image. It might also be possible to train and conduct the inference real time on the raspberry pi itself.

Overall, our project successfully received the image from the server, processed it, and delivered the user an accurate seating chart.

References

1) https://medium.com/@JohnFoderaro/how-to-set-up-apache-in-macos-sierra-10-12-bca5a5dfffba
2) https://www.pjreddie.com/darknet/yolo
3) https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/
4) https://hackernoon.com/efficient-implementation-of-mobilenet-and-yolo-object-detection-algorithms-for-image-annotation-717e867fa27d
5) http://cocodataset.org/#home
6) https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b