# Livestalk: Livestock Tracking with Drones, Object Detection, & Data-Centric Techniques

**Whit Blodgett, Jeff Day, Omar Kapur, Ricardo Jénez, Justin Jeng**

School of Information
University of California, Berkeley
*{wwblodge, jrday, omarkapur, rrj, jjeng5}@berkeley.edu*

**Abstract**

The system described in this paper enables livestock ranchers to greatly reduce time and energy spent tracking herds over large areas of land. Using deep learning object detection techniques on drone footage, our system reduces the total time spent tracking cows by a factor of ten while ensuring the highest level of accountability. Beyond modeling, this paper proposes an implementation framework to ensure consistent results and ease of use for a broad range of user profiles.

## 1 Introduction

For many cattle ranchers, tracking their cows and knowing where they are at any given time has always been a challenge. Cow pastures can be expansive and involve hard to reach areas, and knowing when a cow has gone missing is vital. For local ranchers with cattle spanning thousands of acres this presents a huge challenge, increasing an already pressing need to know where their livestock are located. The following system, dubbed "LiveStalk", enables ranchers to quickly generate an interactive real-time map with cattle count and streaming drone imagery across the entire grazing land.

LiveStalk allows a rancher to define a path for the drone to fly and take images, then can process these images in real time and get an accurate count of where all their cattle are with a map view of cattle and their counts in each location.

The main machine learning task in this pipeline is to perform object detection of cows using the aerial imagery from the drone. We use YOLOv5[1], a state of the art object detection model, for its fast training and strong performance. The YOLOv5s model was able to achieve over 0.9 mAP (IoU=0.5) on a hold-out test set, while also being lightweight enough to deploy on our test edge devices (a Jetson Nano and NX). The experimentation approach used in the development process was to keep the model fixed and vary the data, which we collected and annotated using the RoboFlow platform[2]. Ultimately, we were able to settle on a particular drone flight approach that fed consistent images to the model and was able to produce very strong object detection performance with even just a small dataset of over 250 annotated images.

The Livestalk pipeline, as developed, allows a farmer to map their farm, fly the drone and save images to an SD card on the drone. That SD card is read by inserting into an NVIDIA Jetson Nano or NX, where the image detection is done using Yolov5. The original and detected images, along with image metadata are sent via an MQTT client to an MQTT broker in AWS. The data is then processed by another MQTT client on AWS which generates an HTML file stored in S3. That file shows the location of the cows on a map with the count in each location. The farmer can use a web browser to access the data about their cows.

## 2 Data Acquisition

Data was collected using a DJI Mavic Pro drone. We also used a DJI Mavic mini for testing parts of our application pipeline.

### 2.1 Collection

As we worked through this project, it became apparent that in order to capture images of the cattle there were important considerations:

- Could we accurately record cow locations within a pasture?
- How large a field could we image at one time? What height should we fly at?
- Specific to the drone, what camera angle should we use? What height should we fly at? How fast?
- Depending on the height we picked, would the noise affect the movement of the cows?
- Could we stream live images to the cloud or would we have to post-process in 'near real time'?
- Could we reference the coordinates of each image's field of view to provide Ranchers with an understanding of where the cows were being detected geographically?

FIGURE 1: Example images from the initial cow footage collection runs, illustrating some of the different camera angles and elevations initially used.

The first step in answering these questions was to actually fly the drone over a cattle farm and run some test analysis and training. We flew the drone over Connelly Ranch and recorded video of the cattle at various heights off the ground and at different camera angles (Figure 1). The intent was to try and understand what was the most effective way to gather these images.

We took each of the videos, generated images every 2 seconds, and uploaded them to Roboflow in order to annotate the images to train the model.

### 2.2 Annotation

Our initial batch of over 1,100 images were annotated on the Roboflow platform with null images (those without any cows) accounting for 5% of the dataset. In these images we had over 13,600 annotated cows. To avoid data leakage, we had to ensure different drone run videos were partitioned separately into training, validation, and test splits (preferably of different herds of cows). Initial random partitioning all frames across all folds led to data leakage: nearly identical images that were a few seconds apart would leak into validation and test splits and be used to trivially detect cows which led to artificially high validation and test mAP. For all of our annotation efforts, we followed a set of ground rules:

1. Focus on making tight annotations with minimal extra space around the object,
2. Do not include shadows in annotations
3. If at only a recognizable portion (e.g., the head) of a cow is shown on the image, annotate it
4. Annotate cows and calves within a single class (cow)

## 3 Initial Approach

We tested different size YOLOv5 models and ultimately settled on using the smallest-sized YOLOv5s model which demonstrated strong performance in some experiments (see Table 1) and could be easily deployed to smaller edge devices. Until recently, the YOLOv5s was the smallest model in the YOLOv5 family with 7.2M parameters (the YOLOv5n was just released, with only 1.9M parameters). For comparison, the next size up in the YOLOv5 family (the YOLOv5m) contains 21.2M parameters, and we had issues (at times) using that model on the smaller Jetson Nano. Tests done on the dataset with each size YOLOv5 model (n, s, m, l, and x) indicate little benefit to mAP from using the larger YOLOv5 sizes (see Appendix Figure 14). While this analysis could change as the dataset changes, we recommend using the smallest YOLOv5 model since power consumption should be minimized for edge device inference (while there was little benefit in mAP on our data, Appendix Figure 15 shows there is a substantial increase in power consumption as model size increases).

Our models trained on the initial dataset of cows from varying angles did not perform as well as models trained on strictly top-down images (Table 1), despite evidence that object detection models tend to perform better on angled images than top down shots (see evidence from COCO models[3]).. Once we had finalized our image capture approach (described in subsequent sections), we began collecting imagery

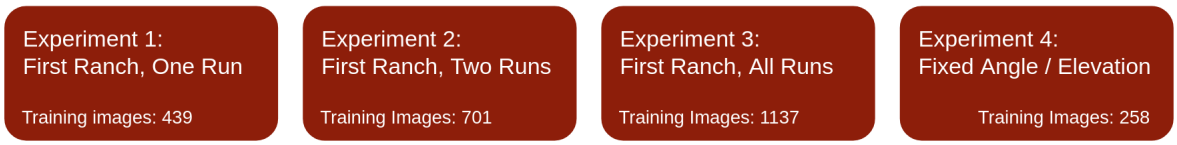


FIGURE 2: Four experiments used to determine the best data collection approach. Each experiment was evaluated against the same holdout test set.

taken from a fixed elevation and fixed camera angle. Throughout the weeks, we continually filmed new cow footage from different pastures and different times of day to boost the size and diversity of our training and evaluation data.

Once we corrected for data leakage, we found the initial model mAP to be under 0.4, far below the optimistic results from our initially biased training split. While many machine learning applications benefit from diversity in data collection to train more robust models that can generalize well, in this case it lowered model performance substantially. Because we had varying heights that the drone flew, the model had difficulty later on in estimating the size of a cow. With angled shots that contain large portions of the pasture, it is also possible to have cows of varying size in the same image.

In this first mixed angle and height dataset, the models had difficulty identifying any cows above a 0.50 confidence level. Given the fact that we would always have a relatively small dataset this was going to be a problem. The important question became, how could we improve imaging on a small dataset?

# 4 A Small Dataset Approach

To take the opposite approach for data collection, we began compiling a smaller dataset of just over 250 images which were all taken at a fixed elevation of approximately 45 meters, and a fixed camera angle of 90 degrees (facing down). The lower fixed elevation prevented the dataset from having a significant variation in cow size, and while the fixed downward angle does not always allow for discernable cow features to be captured, the consistency in size was clearly a benefit for the model (see Figure 9 in the Appendix for example images from this dataset). We would have liked to collect more data with this approach, but given time constraints had to settle for a dataset that was smaller than even what one video at our initial ranch had produced.

In addition to the 250 images we obtained and annotated for the small dataset, we obtained another data on different cows and pastures of 59 images. We used this as a holdout test set to evaluate all four experiments against. This holdout set involved flying at a fixed elevation and angle, as we had determined a method of operationalizing the drone flight that will be covered later. No evaluation had been performed on the holdout test set prior to these experiments.

## 4.1 Experimentation

In order to evaluate performance, we ran four experiments as shown in Figure 2. The first three experiments involve training the model on one, two, and all runs done in the initial ranch footage with varying elevations and angles. The fourth experiment is on the small dataset with a fixed angle and elevation. Each experiment involved training a YOLOv5s model for 100 epochs on the training dataset and evaluating it against the holdout test set. Experimentation was conducted using an NVIDIA RTX 3090 GPU.

## 4.2 Results

The results of each dataset with a YOLOv5s model can be seen in Table 1. Additionally, Figure 3 shows the mAP (with IoU of 0.5) for each experiment during training. The small dataset, which shared consistency only in camera elevation and angle with the holdout test set, clearly performed significantly better than any of the other models trained on datasets with varying elevations and angles.

TABLE 1: RESULTS FOR THE FOUR EXPERIMENTS

| **Category** | **mAP with IoU=0.5** |
|---|---|
| 1) First Ranch, One Run | 0.3629 |
| 2) First Ranch, Two Runs | 0.4429 |
| 3) First Ranch, All Runs | 0.6351 |
| 4) **Small Dataset (fixed angle/elevation)** | **0.9388** |

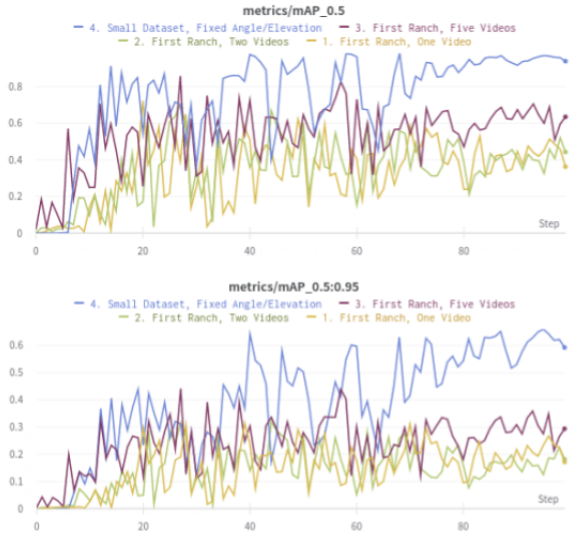


FIGURE 3: Results on the holdout test set for all four experiments. The top figure shows the mean average precision (mAP) with IoU of 0.5. The bottom figure shows the average mAP from IoU of 0.5 to 0.95 in increments of 0.05.

### 4.3 Error Analysis

Error Analysis was performed using both manual inspection of results, as well as an automated analysis based on object detection best practices [4], using the TideCV framework[5]. The automated analysis resulted in classification of errors shown in Figure 4. This categorizes errors into false positive and false negatives, as well as into the following types:

- Classification (Cls) - this does not apply to our dataset since there is only one class
- Localization (Loc) - where a cow was not localized accurately
- Both Classification and Localization (Both) - this is again not applicable with only one class
- Duplicate Detection (Dupe) - where the model finds multiple instances of the same ground truth example
- Background Error (Bkg) - where background was detected as a cow
- Missed Error (Miss) - a false negative not detected by classification or localization errors

As seen in Figure 4, the model did get progressively better in experiments 1, 2, and 3 as the dataset size grew, it began to classify fewer false positives and negatives. However, the experiment 4 results show clearly superior with very minimal false positives or false negatives.

Figure 5 shows an example from Experiment 1, where the model was trained on images with varying elevations and angles. The annotations in this figure show all of the potential cows identified by this model (albeit given a very low confidence threshold of 0.1). In the training images, this model had to learn to detect cows from very far away that look like the specs of grass in Figure 5. There are no actual cows in the figure – all predictions in this image are incorrect.

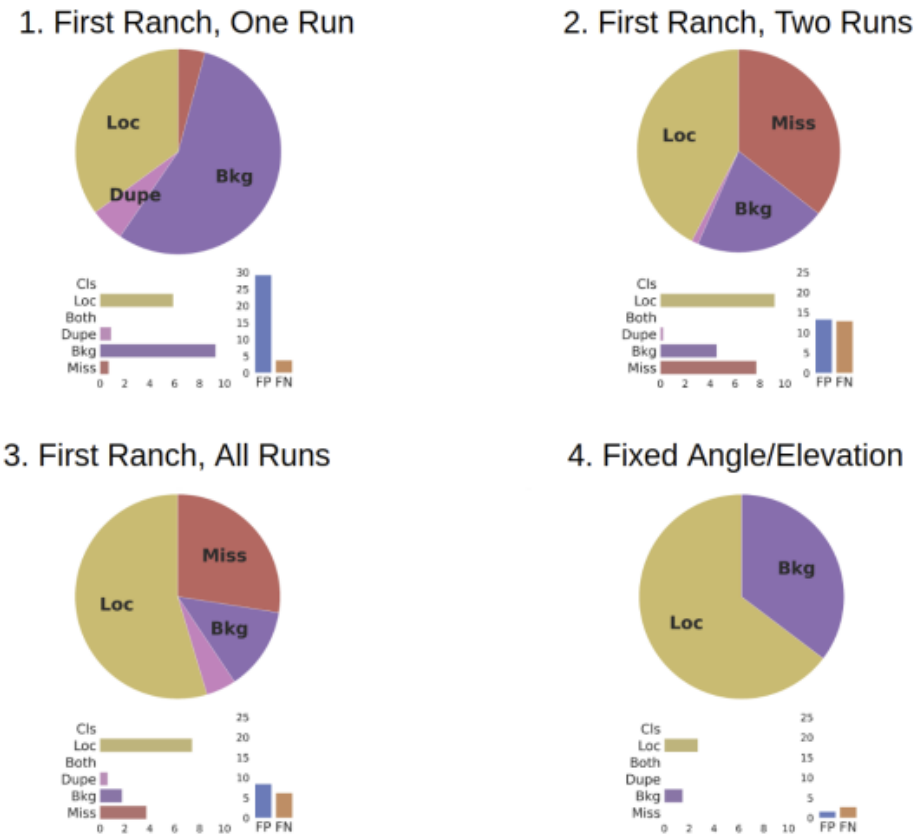


FIGURE 4: Error analysis on the holdout test set for the four experiments using TideCV.

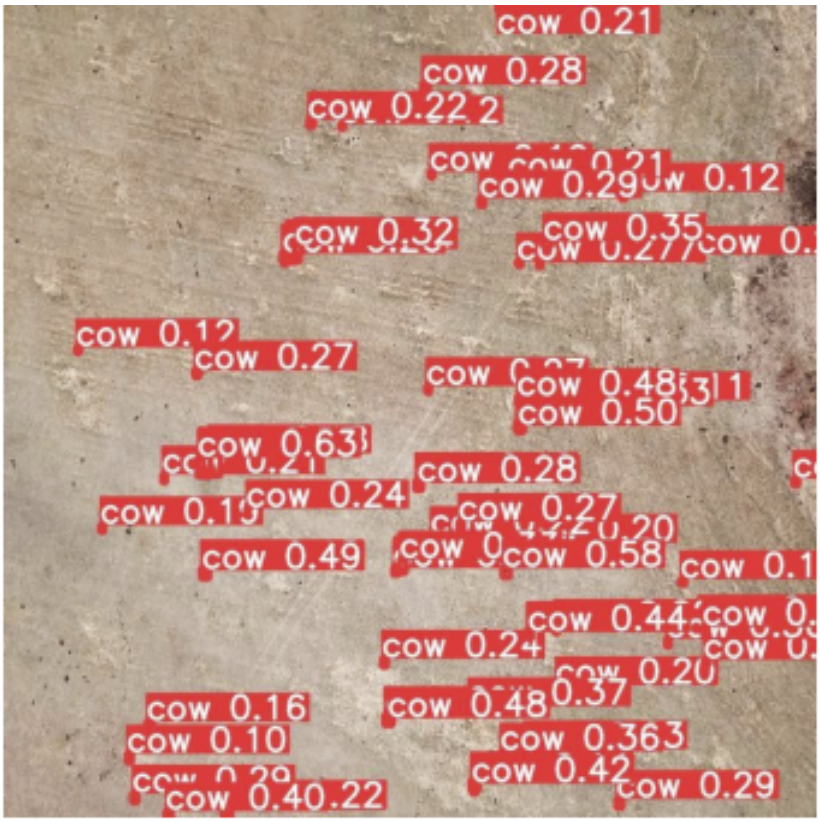


FIGURE 5: Detection results from the first experiment, which involved training on images; the model detected many small objects such as dark patches of grass as cows. There are no actual cows in this image. (Note: any detection with confidence $> 0.1$ shown in photo).

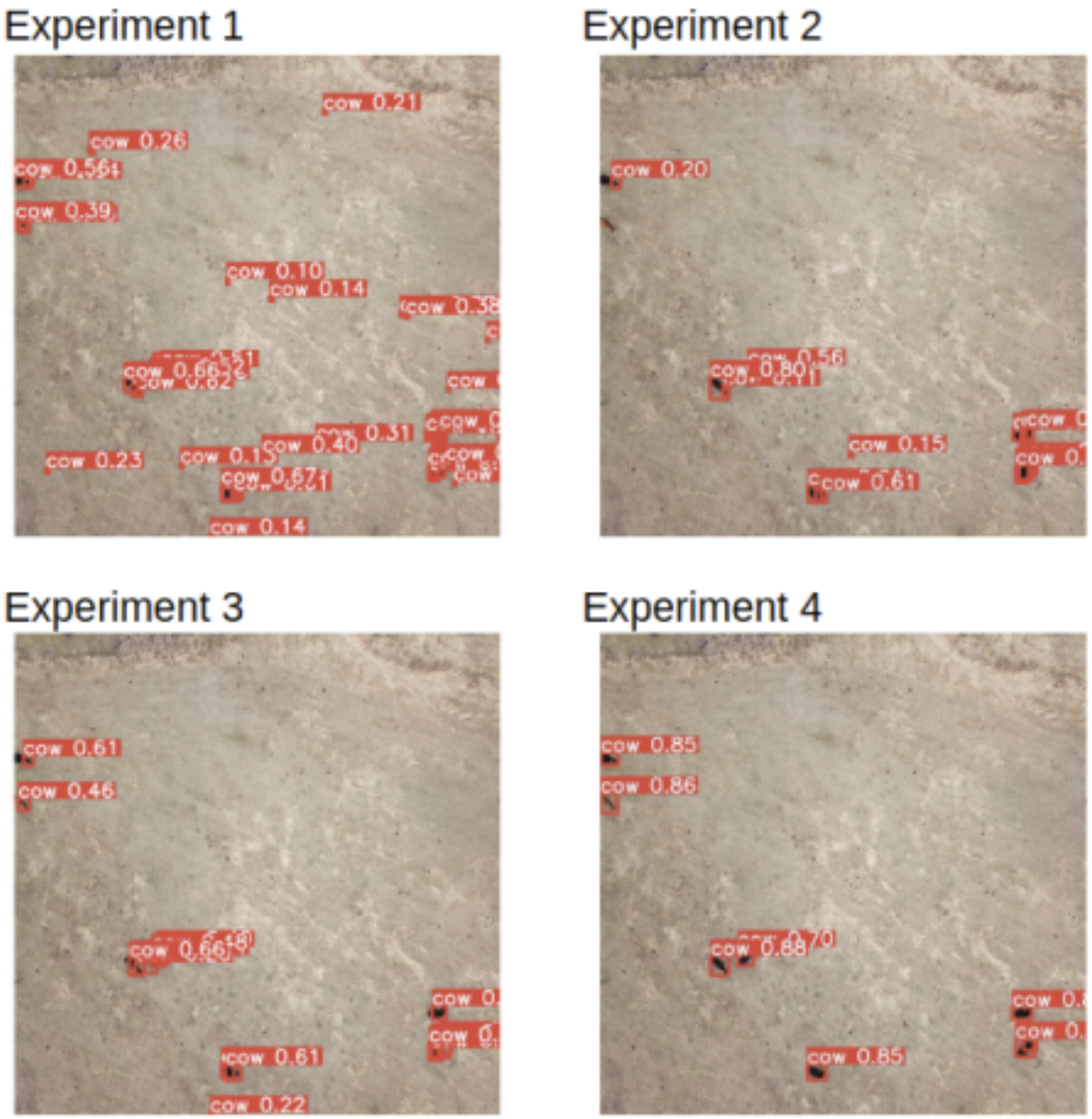


FIGURE 6: Tiled images are the same test holdout image shown, with detection results (with confidence threshold $> 0.1$) for each of the four experiments showing progressive improvement with a reduction in False Positives and increase in confidence scores for True Positives.

The images tiled in Figure 6 all show the same image from the holdout set, which contains 7 actual cows. Each subsequent experiment performs progressively better at reducing false positives, with Experiment 4 performing the best.

## 5 Design of an Image Pipeline

As our team looked to improve cow detection model performance, we simultaneously looked to develop a pipeline that would extract model counts and positional data at the edge and stream the information to a graphical user interface on the cloud for real-time cow tracking. To ensure we could realize our end goal, we first needed to figure out how to geographically reference each image's field of view to provide ranchers with the geographic reference of each image.

### 5.1 Georeferencing the Drone Image

To ensure we could eventually reference and plot the cow detections on a map, we had to ensure geographic drone metadata was accessible. This led to a long effort of trying different approaches to acquire the current drone location and it's height while streaming video from the drone (our original preferred method - using RTMP).

We eventually discovered that it was close to impossible to get real time image data with telemetry. If we didn't have this information we couldn't tell where cows were, nor could we determine if we were continually identifying the same cows. The first part of this mystery was solved by finding a mechanism to fly the drone in a consistent fashion over our target area. After reviewing open source and implementing an application using our drone manufacturer DJI's

SDK, we decided on commercial drone software that could allow us to map an area while taking video or images. This software package, Dronelink (Figure 7), allowed us to map a flight area and specify how the drone would fly over that area. The height, speed, direction, the position of the camera and other pertinent details to getting consistent images.

However, we still had one problem. How do we determine the location of the drone and the imagery field of view while it flew over the farm? The drone had high quality geospatial positioning using GPS, but getting to that data seemed almost impossible.

Once we decided that we would take a series of stills as the drone flew over the farm, we looked more closely at the image output. Each image had metadata attached to it, called EXIF, (EXchangeable Image Format), that has most of the information we needed, but not all of it. The information specific to the drone camera such as the camera yaw, pitch, roll, and relative altitude were missing. Then we realized that DJI included this information in XML format as an addendum to the metadata on the jpeg image.

DJI's abundance of positional data of the camera sensor allowed us to calculate the field of view using a number of geometric relations. Our implementation is available in the `test_map` directory of our github. We had the beginnings of what we would consider a viable pipeline.

### 5.2 The Image Pipeline

We knew that we could now develop and deploy a process for flying over a farm and detecting and counting the cows. The steps were:

1. Map a path with DroneLink Web Interface. Input the geo area to fly the drone, the path that the drone would take, the height of the drone, the image capture (pictures every 2 seconds), and the direction of the camera.

2. Download the mapped path to the drone when on location, and send it off to map the farm.

3. Collect images from the SDK card on the drone and upload them to our image capture area on one of our NVIDIA Jetson devices (Nano or NX) (in the true final version we would have implemented a DJI application to send images off the drone in real time – perhaps the next project).

4. Run the images in sequence through image detection based on Yolov5 on the Jetson device. Using a MQTT client, take the original image, the image with the detection bounding boxes, labels from the detection process, and send them to an MQTT broker on AWS.

5. The MQTT broker on AWS receives the message, and another MQTT client gets notified of the incoming image.

6. We extract the EXIF and XML data from the original image and send this data along with the count of the cows to our map generation software which generates an HTML page that can be posted with an update to the location of the cows in bounding boxes appropriate for each image it receives.

7. Display the html page with click through "thumbnails" of the output of the image detection software to verify that the cows were correctly identified.

8. While this pipeline requires removing the SD card from the drone and inserting it into the Jetson device in step 3, this is still a useful proof of concept approach to demonstrate the potential capability. Using a more professional model such as a DJI Matrice drone, the same approach could be applied through the onboard SDK[6], where the drone would be able to execute this pipeline with model inference performed onboard.

### 5.3 Technical Considerations

We implemented the image pipeline utilizing Docker containers with relevant custom Python files and libraries running on our NVIDIA Jetson devices and an AWS machine. See Figure 16 for a diagram of the whole pipeline. The pipeline consisted of five containers:

1. Image Reader and Cow Counter: This container has all necessary libraries and custom Python files to perform image detection and send relevant data to an MQTT broker. The python file monitors a user-specified folder on the Jetson device, runs new images through the Yolov5 image detection model, and packages the original, annotated, and labeled cow locations into a single JSON message. To monitor a directory, we utilize the watchdog library. Once a new image is detected, the image byte data passes to Yolov5 for image detection. When the image detection process finishes and returns the annotated image and labels, the raw image data, raw annotated image data, and count of the number of cows detected are packaged in a single JSON object, dumped into a string, and sent to the MQTT broker. Because the JSON object containing relevant data cannot hold raw byte data, the image data for both images is encoded to base 64 ASCII characters and then decoded to a string using UTF-8.

2. MQTT Broker: This container holds the Mosquitto message broker, based on the lightweight Alpine Linux OS. An instance of this container runs on both the Jetson device and the AWS instance. After publishing a message to a specific topic on these brokers, subscribers are notified and can appropriately process the message.


FIGURE 7: Images showing a cow detection flight plan using drone link (top) and the view from a drone as it executes a flight plan (bottom).

3. MQTT Forwarder: This container holds a custom Python file that subscribes to the MQTT broker on the Jetson device and forwards any messages received to the MQTT broker on the AWS instance. The container requires two key arguments: the IP address of the AWS instance to forward messages and the topic for which to subscribe on the Jetson MQTT broker.

4. Image Processor and Map Generator: This container holds a custom Python file that subscribes to the MQTT broker on the AWS instance, extracts annotated image data, saves it to an Amazon S3 bucket, and generates the map of bounding boxes of detected cows. The container uses s3fs-fuse to map a local directory to an S3 bucket. The python file extracts the annotated image bytes, original image bytes, and cow count from any JSON messages generated by the image reader. It then saves the original and annotated images into the mapped local directories to write them to S3. Next, the file uses the map generation Python file to extract relevant EXIF data from the original image. The EXIF data and the previously extracted cow counts are appended to a global data frame holding all previously processed images. The map generation code uses this data frame to regenerate an HTML file holding the map with bounding boxes and cow counts. Some essential libraries used in map generation include pyproj, folium, and geopandas.

# 6 Conclusion

The team initially started with a broad concept: Could you use a drone and identify livestock from the air and make it feasible for farmers on the order of minutes, be able to identify the location of their cows and count them without ever leaving the farmhouse. We can clearly state that this is indeed possible and we feel within the technical capability of a farmer to use, since the flying of the drone is push button and the imaging process almost so.

## 6.1 Key Learnings

**Small datasets can be effective.** One of the most surprising results in this process for us is simply how effective a small corpus was when consistent in camera elevation and angle to the test dataset. With not a very large set of images, but with control of the camera angle, the height and the quality of the image, we were able to generate models that had a mAP of greater than 90%. An important part of this was also consistency and attention to annotation, maintaining tight bounding boxes around each cow and following consistent rules when annotating across a team. With more time, we would have created a larger dataset with the same consistency, and likely would have encountered more challenges including occlusion and other animal classes. Through our experimentation, we learned that a small dataset that is consistent in drone imaging to what the model will see in production can be much more effective than a larger dataset with more diversity that will not be experienced in production.

**Narrowing the focus for a model can improve results.** Our initial approach involved attempting to train a model that was generalized for a variety of angles and heights, and our results improved dramatically when we reduced the variability of the objects to be detected. Taking this lesson further, for an implementation where we would want - or need - to use images with different camera heights and angles, we might choose to have multiple models to maintain a narrow scope for each model (e.g., a model that is implemented when the image taken is from a 90-degree angle, and a different model used when an image is at a 45-degree angle).

**Integration opens up applications.** The other thing we learned was the power of integration of open source software and deep learning coupled with amazing processors at the edge, like the NVIDIA Jetson Nano and Xavier NX, it is possible to do sophisticated image detection close to the source of the images and provide value to people who would perhaps have never considered the use of cutting edge image recognition and drones to solve a problem that has been around since the cowboys roamed the prairies.

## 6.2 Future Work

This project highlights the ability to do more with the integration of imagery and detection using sophisticated drones. We would like to continue to improve the ease with which it is possible to gather imagery and smooth the pipeline to make it simpler and easier to use. We could perhaps consider the fact that with the right drone we could do all the image detection on the drone itself and that drone would only send back reports about the cows it sees and be a cowboy in the sky. Yeehaw!

# 7 References

## References


[1] *Ultralytics YOLOv5 (GitHub)*. Last accessed 4 December 2021. URL: https://github.com/ultralytics/yolov5.

[2] *Roboflow*. Last accessed 4 December 2021. URL: https://roboflow.com/.

[3] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[4] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. "Diagnosing Error in Object Detectors". In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 340–353. ISBN: 978-3-642-33712-3.

[5] Daniel Bolya et al. "TIDE: A General Toolbox for Identifying Object Detection Errors". In: *CoRR* abs/2008.08115 (2020). arXiv: 2008.08115. URL: https://arxiv.org/abs/2008.08115.

[6] *DJI Onboard SDK*. Last accessed 4 December 2021. URL: https://developer.dji.com/onboard-sdk/.

# 8 Appendix

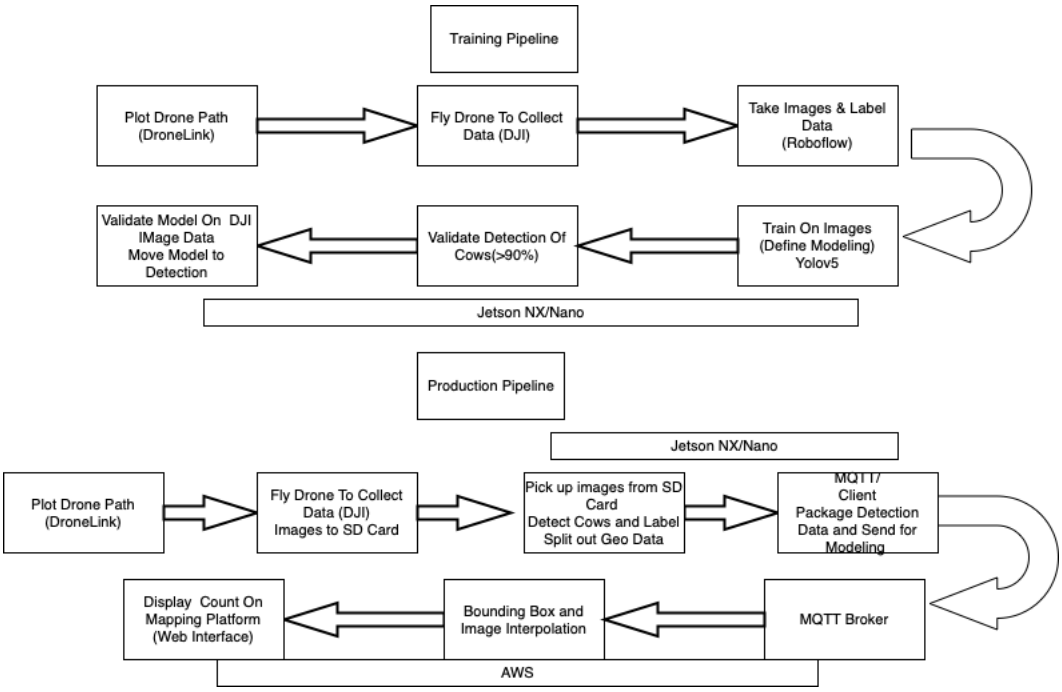


FIGURE 8: Training and Pipeline Flow

FIGURE 9: Sample of the images taken for the small dataset (consistent angle and elevation), annotations are ground truth.

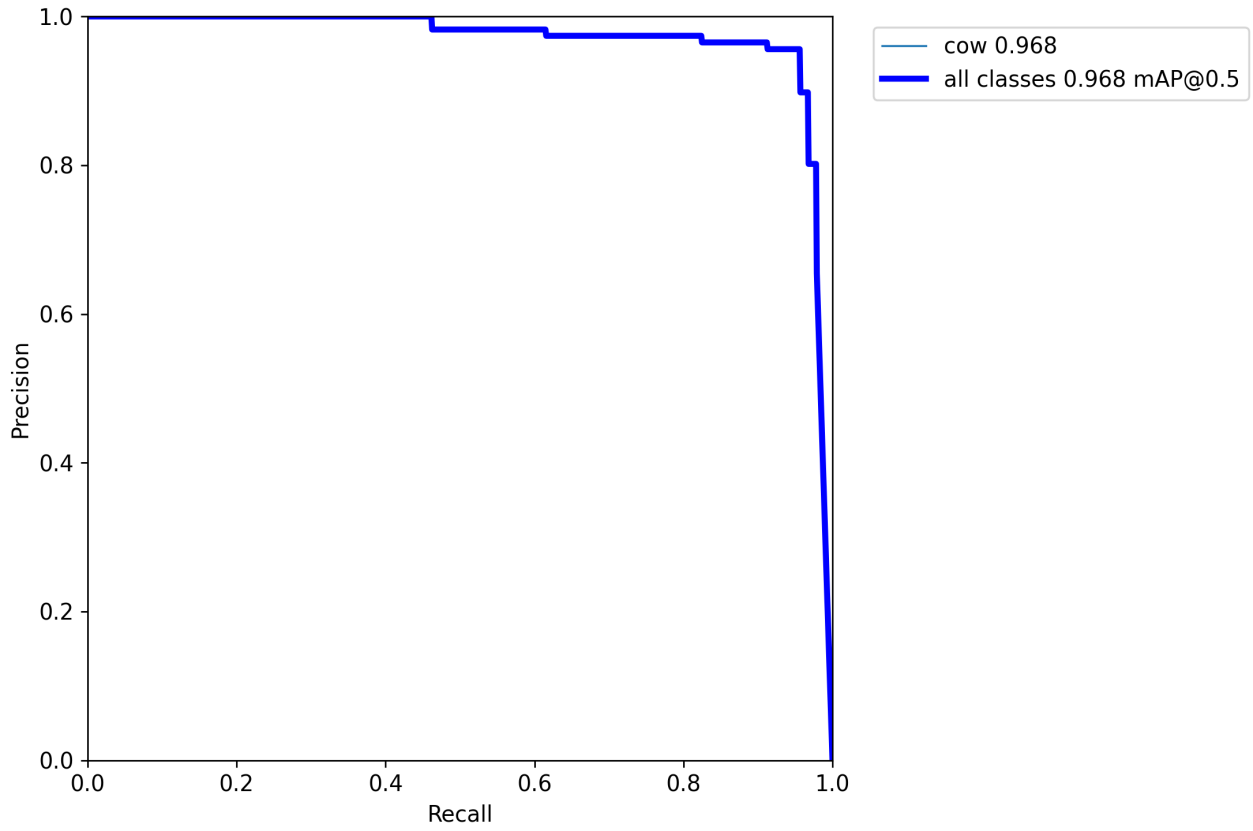


FIGURE 10: Precision-Recall curve for Experiment 4 (Small Dataset with fixed angle and elevation).

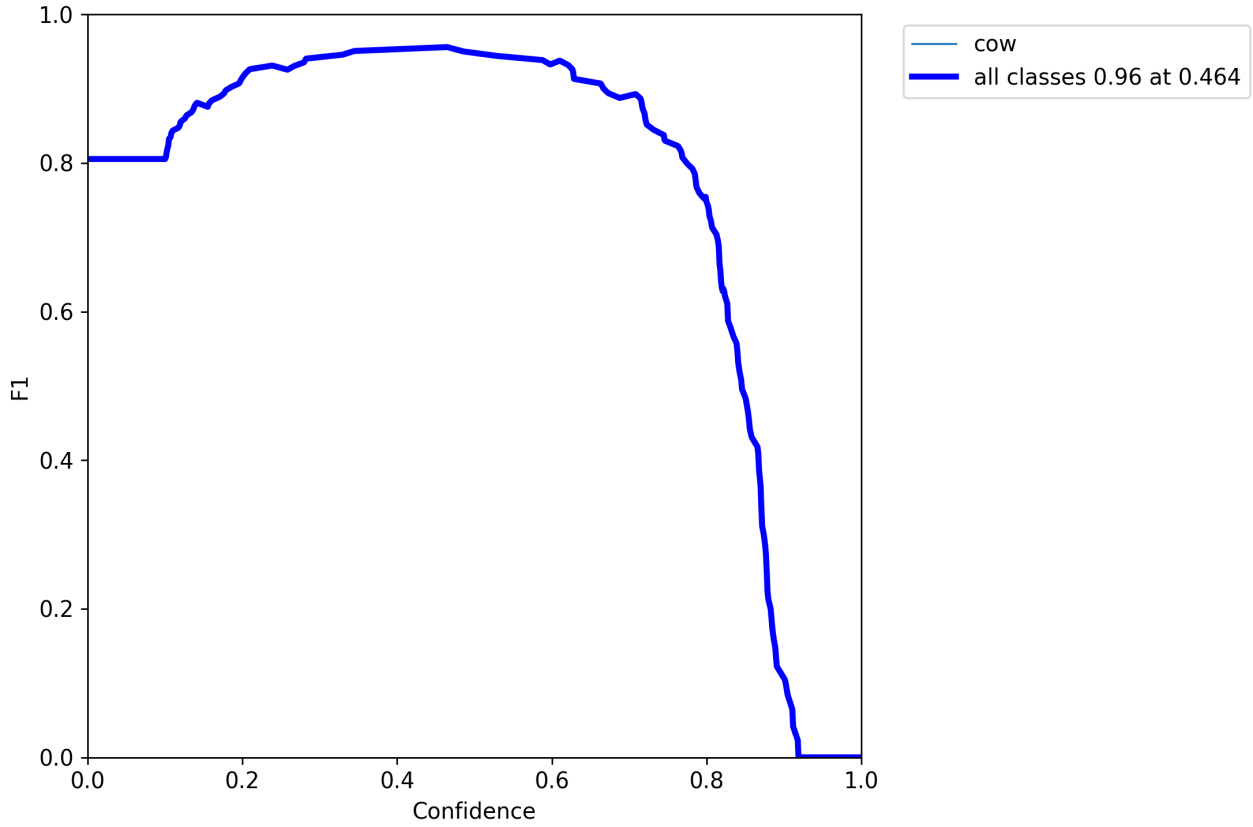


FIGURE 11: F1 curve for Experiment 4 (Small Dataset with fixed angle and elevation).

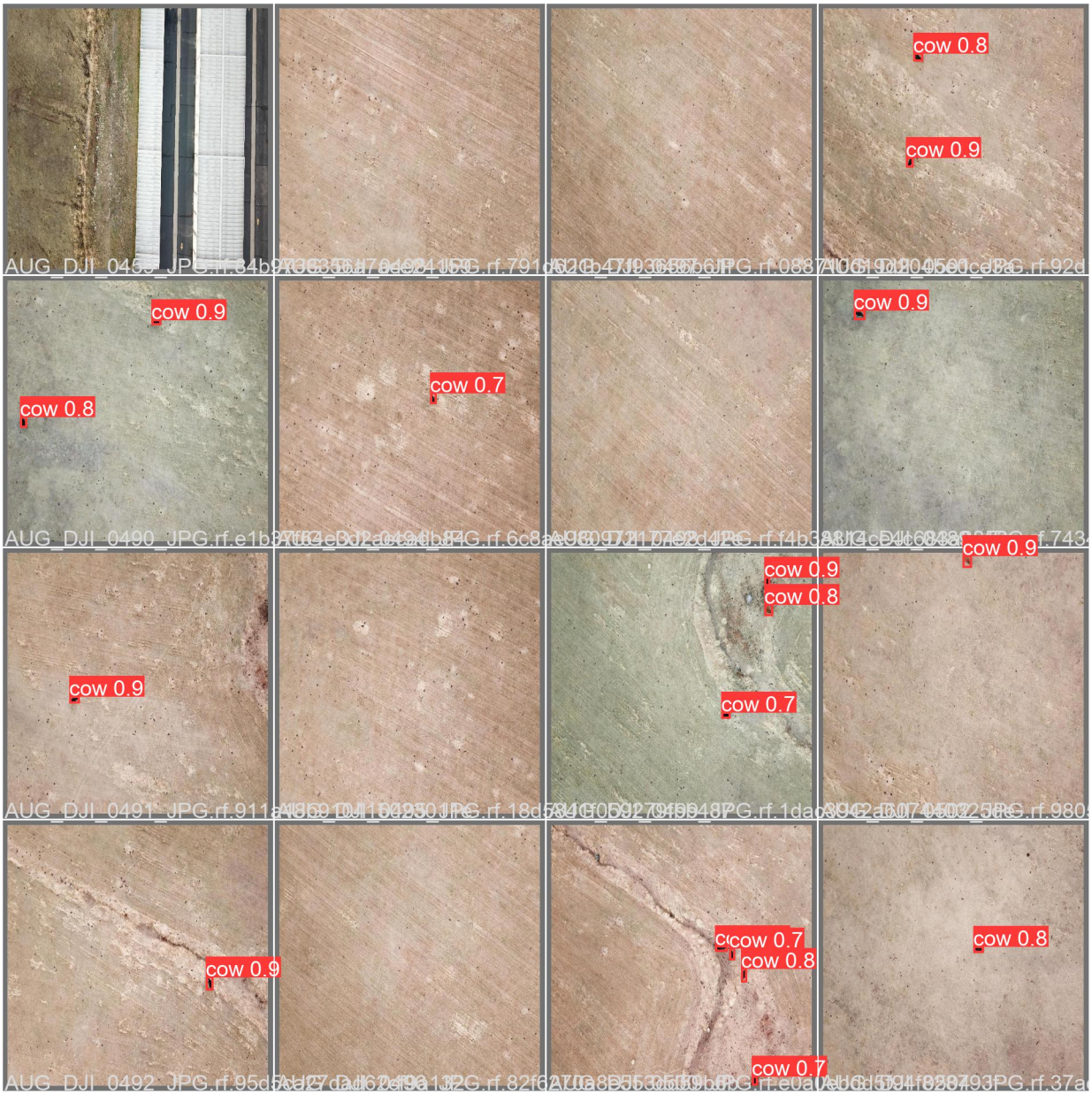


FIGURE 12: Mosaic of batch of validation images with predicted boxes (and confidence scores) from Experiment 4.

FIGURE 13: Holdout test set images with ground truth annotations (blue boxes) and predictions (red boxes) with confidence scores (red text).

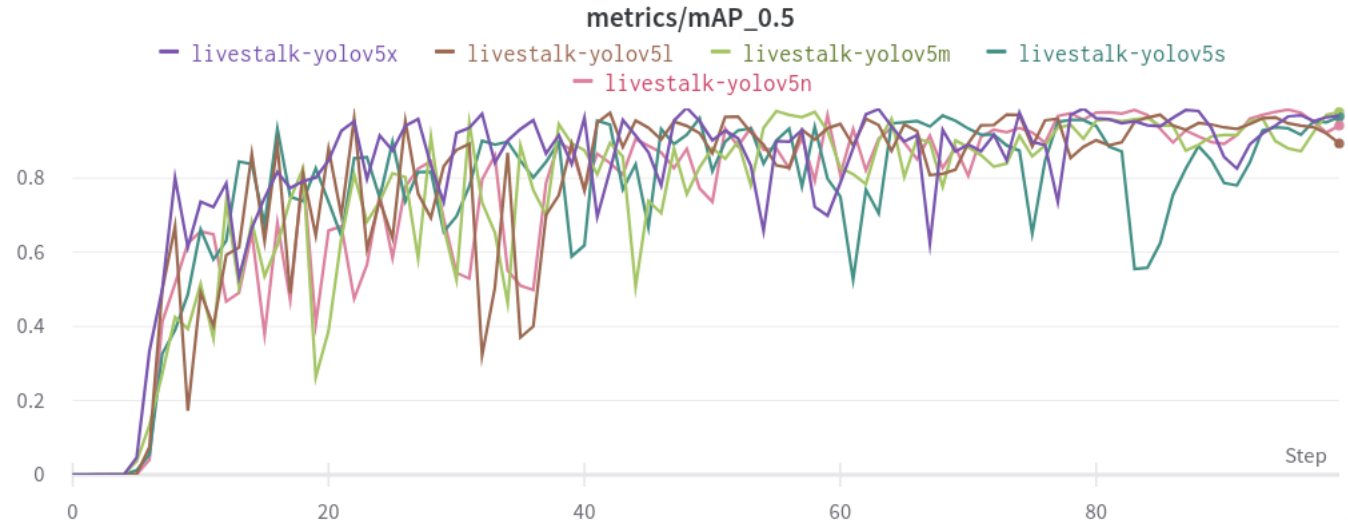


FIGURE 14: Comparison of different size YOLOv5 models trained on the Experiment 4 dataset (fixed angle and elevation) and evaluated against the holdout test set.

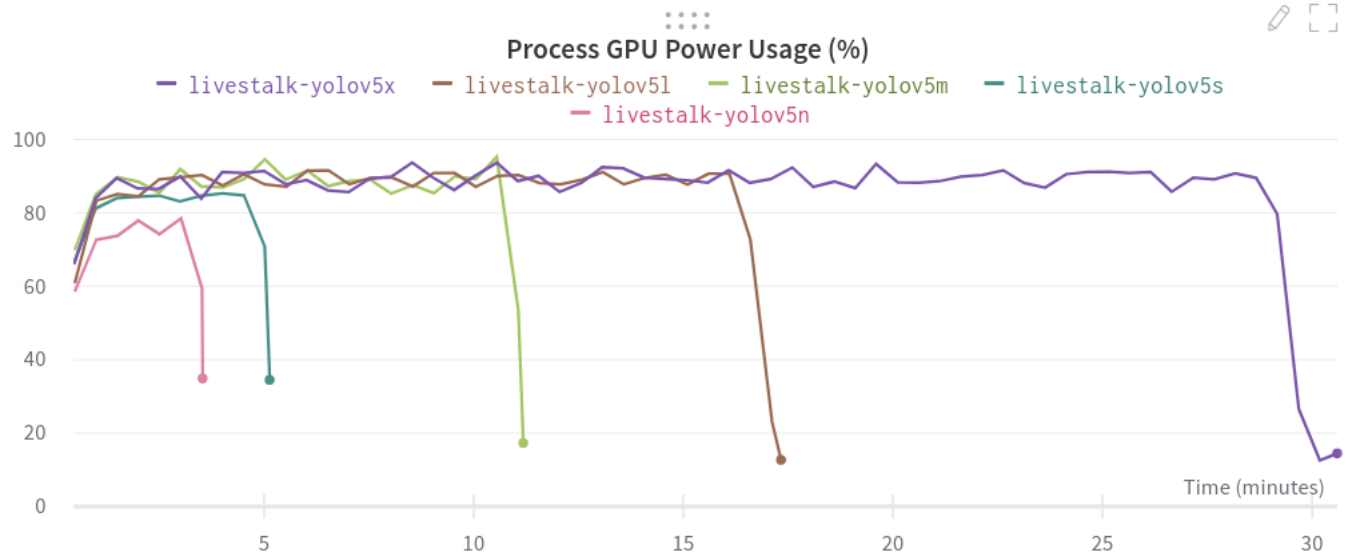


FIGURE 15: Comparison of GPU power usage for different size YOLOv5 models trained on Experiment 4.

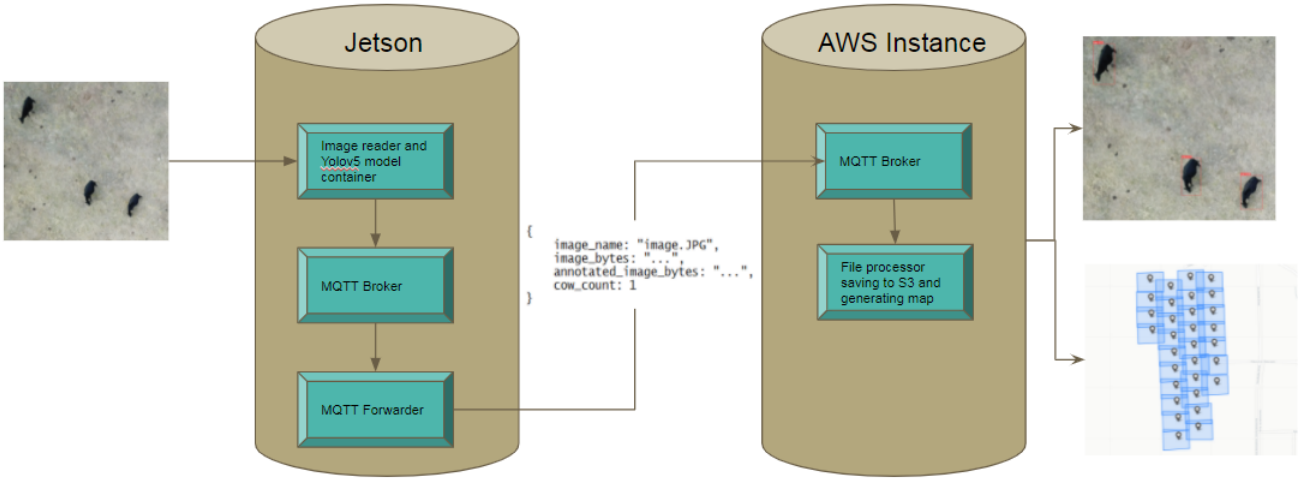


FIGURE 16: Image Pipeline Diagram.