

Good Boiii
Dog Behavior Detection

August, 2022

W251 - Deep Learning in the Cloud and on the Edge

Heather Pieszala, Ethan Duncan, Rishika Pulvender, Valerie Chau

Abstract	2
Background Information	3
Tools Needed	3
How Was it Built	3
Data Gathering & Data Creation	4
Data Gathering	4
Dataset Creation	4
Modeling	Error! Bookmark not defined.
Introduction to the Models Used	6
How the Model was Trained	7
Final Results	9
Architecture	11
Approach	11
Diagram	Error! Bookmark not defined.
Conclusion	12
Challenges	12
What If We Had More Time?	14
References	Error! Bookmark not defined.

Abstract

Ever wonder what your dog is doing when you are not at home or simply not watching? You might say that you have a camera for that. However, does your camera know when your dog is misbehaving and encourage it to engage in “positive” behavior instead? Most likely, it does not. You might also be aware of your dog misbehaving when you leave the room but be unable to determine when. This therefore, makes it difficult to effectively train your dog. This desire to streamline dog training has motivated us to create an end-to-end pipeline that can detect dog behavior and be used to aid in training. Due to time constraints and limited access to “negative” dog behavior data, we have decided to focus on detecting “positive” behavior first. For the purposes of this analysis, “positive” behavior means: sit, lay, stand, and play, with a primary focus on distinguishing “sit” and “other” for the stages of this project. The following study outlines the approach taken to solve this problem.

Background Information

Tools Needed

Hardware:

- Jetson Nano Developer Kit: 4GB
- 128GB Micro SD
- USB MicroSD card reader
- WiFi/Bluetooth card OR ethernet connection and cable
- Power adapter (for Nano)
- 1TB USB3 SSD
- USB Webcam

Software:

- AWS EC2 instance (gd4n.xlarge) - for model training
- AWS EC2 instance (t2.medium) - for the pipeline
- Docker
- Kubernetes
- OpenCV
- Roboflow
- AWS S3 Bucket

How Was it Built

Images were first gathered from the Stanford Dataset and then labeled using Roboflow. The images were used to train a model that we built on an AWS EC2 instance, though any cloud provider instance with a GPU can be used. The weights were then taken and using inference on a Jetson device, it was identified if a dog was sitting. If so, images were sent to an s3 bucket on the cloud.

Full instructions for replication can be found in the following Readme.md on the project GitHub: [\[link here\]](#)

Data Gathering & Data Creation

Data Gathering

We first set out to find a dataset consisting of “bad” dog behavior. This proved to be more challenging than initially anticipated, as we not only wanted to ensure that a range of behaviors were captured but also that we had a variety of breeds to ensure the large breadth of dogs that this model would serve. This data insufficiency shifted our focus to build the piece of our “future pipeline” that focused on “good” dog behavior. While there were a few (though limited) dog pose estimation datasets available, even one synthetic dataset (Sydog), we decided to focus on one used in the study “Who’s a Good Boy? Reinforcing Canine Behavior in Real-Time using Machine Learning,” (Cavey & Stock, 2021). Unfortunately, this study did not provide labeled data. Alternatively, it led us to the title of the dataset used, the Stanford Dogs Dataset.

The Stanford Dog Dataset was originally collected for fine-grain, dog image categorization. This was a challenging problem as certain dog breeds represent near identical features or differ in both color and age, and so it inspired a dataset of diverse breeds. We were able to locate the dataset with annotations on Kaggle linked to the dog breed; however, our interest centered around what the dog was doing in the photos. We therefore traced the Kaggle study to its original source at Stanford University (Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao and Li Fei-Fei, 2011). The entire dataset consisted of over 20,000 images of 120 different breeds. Annotations provided were class labels and bounding boxes. From here, we were able to download the images and create our own labels/annotations in Roboflow to serve our use case, dog behavior. The dataset creation and labeling methodology is discussed next.

Dataset Creation

From the Stanford Dataset, from each of the 120 breeds, we selected 5-10 images of dogs sitting and 5-10 images of dogs doing something other than sitting. Labels were created by drawing a bounding box around the dog(s) in each photo and labeling them as “sit” or “other.” Here we define “sit” as the dog’s bottom on the ground standing on its two front legs. Alternatively, we define “other” as a dog in any other position; this includes laying down, running and jumping. In total, we labeled 1792 images in what will be referred to as the *base dataset*.

To create more images and introduce more variation, we performed permutations on the base dataset, resulting in three main datasets for evaluation. We then ran each dataset through Roboflow’s built in machine learning (what we will refer to as Roboflow pre-check) to get an initial idea of the results. We thought grayscale would have had worse performance than colored images and were surprised at how much of a negative impact the bounding box augmentations had in the Roboflow pre-check model (see table below).

The dataset was split into categories, with 3.8K images (87%) in train, 349 (8%) in validation, and 189 (4%) in test.

Datasets and Roboflow Pre-Check Model Results:

Dataset name	Preprocessing & Augmentations	Model Performance Results (from Roboflow)
Base image augmentation	Auto-Orient: Applied Resize: Stretch to 416x416 Outputs per training example: 3 Flip: Horizontal Crop: 0% Minimum Zoom, 20% Maximum Zoom Hue: Between -50° and +50° Saturation: Between -25% and +25% Brightness: Between -25% and +25% Exposure: Between -25% and +25% Noise: Up to 5% of pixels	mAP: 87.4% Precision: 79.1% Recall: 83.1%
Base image augmentation with bounding box augmentations	Auto-Orient: Applied Resize: Stretch to 416x416 Outputs per training example: 3 Flip: Horizontal Crop: 0% Minimum Zoom, 20% Maximum Zoom Hue: Between -50° and +50° Saturation: Between -25% and +25% Brightness: Between -25% and +25% Exposure: Between -25% and +25% Noise: Up to 5% of pixels Bounding Box: Flip: Horizontal, Vertical Bounding Box: Rotation: Between -15° and +15° Bounding Box: Brightness: Between -25% and +25% Bounding Box: Exposure: Between -25% and +25% Bounding Box: Noise: Up to 5% of pixels	mAP: 48.8% Precision: 47.1% Recall: 59.9%
Base image augmentation with bounding box augmentations & grayscale	Auto-Orient: Applied Resize: Stretch to 416x416 Grayscale: Applied Outputs per training example: 3 Flip: Horizontal Crop: 0% Minimum Zoom, 20% Maximum Zoom Grayscale: Apply to 25% of images Hue: Between -50° and +50° Saturation: Between -25% and +25% Brightness: Between -25% and +25% Exposure: Between -25% and +25% Noise: Up to 5% of pixels Bounding Box: Flip: Horizontal, Vertical Bounding Box: Rotation: Between -15° and +15° Bounding Box: Brightness: Between -25% and +25% Bounding Box: Exposure: Between -25% and +25% Bounding Box: Noise: Up to 5% of pixels	mAP: 42.6% Precision: 43.4% Recall: 57.3%

mAP (mean Average Precision) was nearly two times higher in the base image augmentation vs both the base image augmentation with bounding box augmentation and the base image

augmentation with bounding box augmentation and grayscale (87.4% vs 48.8% and 42.6%). Based on these preliminary findings, the base image augmentation dataset was selected for modeling. The same train, validation, test split was kept for our modeling process. Unfortunately, Roboflow provides no insight into the models or parameters they use to obtain these results.

Modeling

Introduction to the Models Used

Our initial modeling considered 'classification' models suggested by Roboflow. A time-consuming and significant barrier we faced was matching export formats (JSON, XML, TXT, CSV) and more with a pre-trained model like YOLOv5, ResNet34, or EfficientNet.

The first instance of conquering this barrier was the test of the classification models was OpenAI CLIP which utilizes a zero-shot classifier which is used to model with when labeled images are sparse. CLIP is known as one of the most efficient models within its domain and is far more flexible than others in the same domain such as the Transformer Language Model. Upon running the OpenAI CLIP model, we achieved an alarmingly low accuracy of 1.5% (Solawetz). Upon investigating our results, we learned that CLIP trains very well on the images it was fed; since all our images were of dogs, CLIP was more likely than other models to train poorly on any other image.

Taking our learnings from OpenAI CLIP, we elected to experiment and train on top of models that were more familiar to us - from our coursework. We took on ResNet34, EfficientDet as well as YOLOv5 in pursuit of the best model to classify a sitting dog.

Next we took on the 34 layered ResNet architecture which uses deep and hidden layers making it a state of the art classification model. Looking to capitalize on the low error rate deep in the network, we continued to look to ResNet as our answer. We inserted our data into the basic model provided by Roboflow to clear up any preliminary issues we may have with this architecture. To our dismay, we encountered an error with the ImageDataBunch object being pulled from fastai; we investigated alternatives such as ImageDataLoader and using functions comparable to the from_folder() but to no avail.

Parallely, assessment of the YOLOv5 and EfficientDet models were ongoing. These were the two models in which we achieved a successful checkpoint.

When our team initially discussed EfficientDet, we were long down the path of failures. We recognized the similarities between specific coursework and assessments we completed with the EfficientDet architecture and our goals for this project. Additionally, with EfficientDet, we wanted to make use of its ability to balance both accuracy with constraints on resources. Fitting the model on the Jetson, with its limited memory, was a concern that we considered while

choosing which models to train our own model with. Additionally, as exporting the weights in the desired format had already proved to be an obstacle - finding repositories that would output the weights as we wanted was another factor in choosing to train With efficientDet. Note that though we mention 'EfficientDet', we use a pytorch implementation of it, not the original version. Our group did initially train the dataset using Tessellate-Imaging's Monk Object Detection branch's implementation of EfficientDet, but we were unable to produce a weights file. However, this initial trial with the Monk branch's version of EfficientDet gave us a comparison for classification loss and mAP for us to evaluate future models performance. See repository for more details. With EfficientDet ('Yet-Another-Efficientdet-Pytorch'), we were successful in being able to train a model on the custom dataset and both detect and classify sitting dogs when deployed on the Jetson. However, due to time constraints, we would have liked to improve on the model's accuracy and implemented it into our pipeline. Please read further regarding the next steps planned for this approach in *'What If We Had More Time?'*

With YoloV5, we were able to get an output of a .pt & .pth file. While we figured out we could use the .pt file, we continued to investigate a) potential conversions to onnx files and b) how to utilize said onnx file in our detector; we would later succeed in converting to an onnx file. However, the decision was made to utilize the YoloV5 model and tune the hyperparameters to best fit the goals of this project. YOLOv5 was also considered a viable option to train our model with for a multitude of reasons. With EfficientDet, we had relatively high losses. YOLOv5 and EfficientDet are often compared performance wise, but one advantage our group saw with the YOLOv5 model was its automated anchor evolution feature, which recalculates anchors values if the provided anchors do not fit well. With EfficientDet, it is up to the user to provide the correct anchor ratios/scales. Additionally, proceeding to train with YOLOv5 seemed reasonable for our application as its pre-trained weights come in a multitude of sizes. For this project we chose to train our own model with YOLOv5s (small), which only has 7.2 million parameters.

In the end, because the YOLOv5 model produced better evaluation metrics, we decided to build the end-to-end pipeline using the YOLOv5 weights.

How the Model was Trained

- Building cloud instance - g4dn.2xlarge with 1 T storage, deep learning ami
- Which model worked
- What parameters did we change
- What file format worked and how did we integrate it

We were successful in training two different architectures: EfficientDet and YOLOv5.

EfficientDet:

Using a Ubuntu Deep Learning AMI on a g4dn.2xlarge instance, the Yet-Another_EfficientDet-Pytorch repository was cloned and primarily trained according to their tutorials. Images were exported from Roboflow in COCO JSON format, where images were already divided into

training, validation, and testing folders during the image labeling process. A separate file containing the anchor ratio and scales was created, with the following values:

`anchors_scales: [2 ** 0, 2 ** (1.0 / 3.0), 2 ** (2.0 / 3.0)]`

`anchors_ratios: [(1.0, 1.0), (1.4, 0.7), (0.7, 1.)]`

These anchor values can be adjusted, but were chosen as the Yet-Another-EfficientDet-Pytorch creators denoted these as good starter anchors for training on images similar to the COCO dataset.

Multiple training runs were carried out with different hyperparameter adjustments each time.

	Variations
Pretrained weights	D0, D1, D2
EPOCHS (head_only = True/False)	10/65, 30/120
LR (head_only = True/False)	.005 / .001, 0.001/0.0005

See /Archive/efficientdet_model1 and /Archive/efficientdet_model2 for detailed results.

Note that memory issues when trying to train with D1 and D2 weights. D0 weights chosen to train all runs as a result. The D1 and D2 models are supposedly more accurate, but are also much larger than the D0 weights.

Overall, most parameter adjustments on our end, (mostly learning rate and total epoch) still resulted in similar evaluation metrics from run to run. In the end, our best performing model was trained with the 'EfficientDet-D0' weights and with the 'head_only' flag dividing the epochs and learning rate as: training with 'head_only' as True for 30 epochs at))) and training with the 'head_only' flag as False for 120 epochs. These hyperparameters produced a model with the following metrics:

Classification loss = 0.45954.

Regression loss = 0.87002.

Total loss = 1.32956

And its scores for creating the bounding boxes is as follows:

Average Precision (AP) @[IoU=0.50:0.95] = 0.60819

$$\text{Average Recall (AR) @[IoU=0.50:0.95] = 0.68066}$$

Most training runs took several hours to complete. Due to time constraints, not all possible or reasonable hyperparameter combinations were exhausted. With additional time, further refinement of the EfficientDet model could be attempted. Overall loss is on the higher side when compared with the first implementation of EfficientDet (Monk branch). mAP is also higher than 0.5 but could be higher.

Yolov5:

In the table below, we show the different hyperparameter tuning that was done with our Yolov5 model. In summary, the default settings provided very fair results in terms of mAP and recall. Only a slight modification of the default parameters performed best and we moved forward with Modified v1.

Pretrained Weights:	Yolov5's Weights
Hyperparameter Adjustments:	<p>Original: (Epochs: 10, Batch Size: 32, LR: 0.001)</p> <p>Modified v1: (Epochs: 20, Batch Size: 16, LR: 0.0005)</p> <p>Modified v2: (Epochs: 200, Batch Size: 10, LR: 0.001)</p>

Final Results


After 20 epochs, batch size of 16 and using the pretrained weights of Yolo v5s we were able to achieve the following results:

Class	Images	Labels	Precision	Recall	mAP@0.5	mAP@0.5:0.95
all	349	370	0.837	0.854	0.9	0.645
other	349	188	0.849	0.851	0.9	0.634

sit	349	182	0.825	0.857	0.901	0.655
-----	-----	-----	-------	-------	-------	-------

Mean Average Precision (mAP) can be defined as the average of the area under the precision-recall curve. Here we use the Intersection over Union (IoU) thresholds of 0.5 and 0.5:0.95. The threshold goes as follows: if the IoU threshold is 0.5, and the IoU value for a prediction is 0.7, then we classify the prediction as True Positive (TF). On the other hand, if IoU is 0.3, we classify it as False Positive (FP).

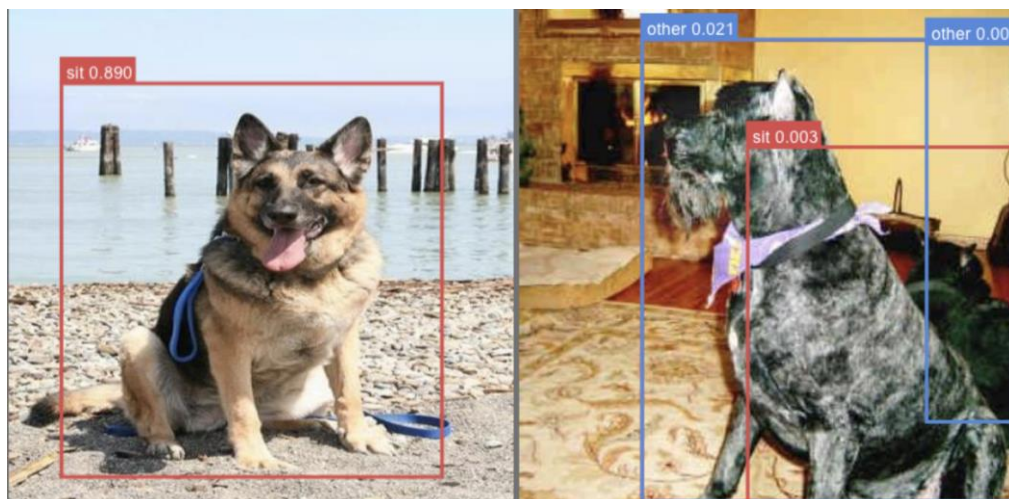
The IoU metric can be visualized by the following diagram:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Precision and Recall are calculated as follows:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

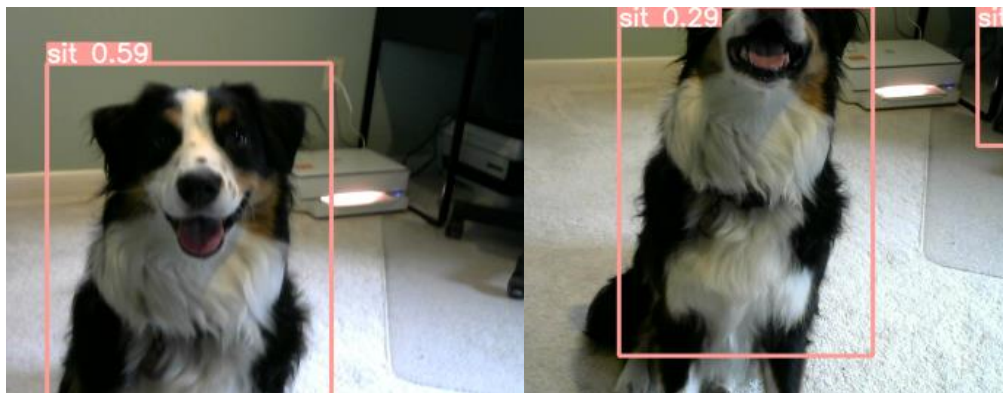
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$



In the images above we have two dog detections, the one on the left is an example of a good detection and the one on the right is an example of a bad dog detection. It is interesting to note

on the image on the right that there is a dog laying down that was identified by the algorithm although it is not very high confidence.

Result note: Note that none of the training data included images of Sage (our live test subject below). The yolov5 inference on the Jetson was very good at detecting when a dog was in the frame. It was mostly (over 50% confident) on the dog's actions, as long as another element was not detected in the frame. The first picture below shows a successful photo of the dog sitting. In the second picture, she is sitting, though another object is detected as a dog and so the confidence decreases. One potential solution we would use to negate this is to either add even more training data to our initial dataset, increase the number of images used in the validation and test datasets, and/or train the model to recognize other objects, outside of just dogs. We may also recommend exploring pose estimation labeling techniques.



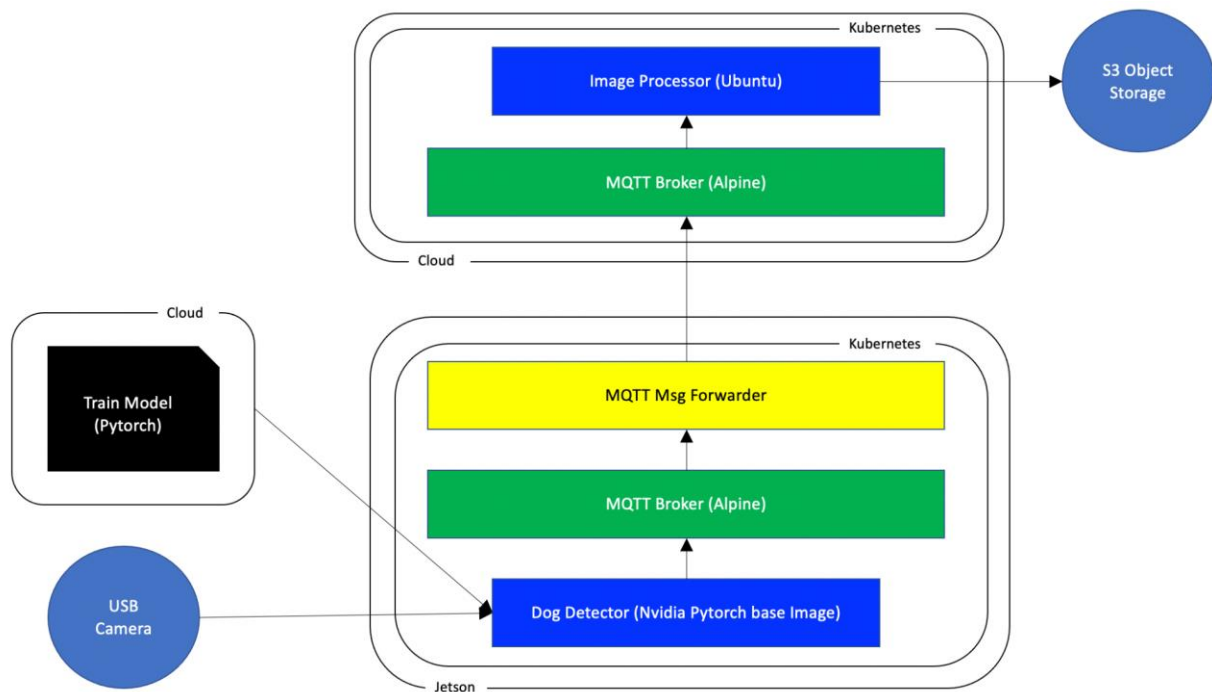
If the dog is sitting, the image is sent to an s3 bucket on the cloud. The eventual goal is to have a device dispense a treat in this scenario. This is showcased in our Future Product Vision section of the paper.

Architecture

Approach

The aforementioned models were a critical piece to the overall IoT pipeline. The biggest challenge we faced was when we tried to inference our model on the Jetson. We thought we had to inference using a .xml or .onnx file, though eventually learned we could inference with the .pt file extension (for the weights) directly, if we built the container with the required nvdia pytorch image.

Diagram



Overall, we trained models on AWS instances, and then took the weights and inferred with those weights on the Jetson. A USB camera is used to detect a dog and that dog image is passed through the dog detector. The dog detector is based on an nvidia pytorch image, which allowed us to use the pytorch (.pt) weights directly on the Jetson. The image is passed through the model, and if a dog sitting is identified, the image is sent as a message through mosquitto to another broker on the cloud. On the cloud, an image processor decodes the image, and sends it to an s3 bucket.

Conclusion

The project presented numerous learning opportunities as we worked through challenges and applied everything we learned in the course to new problems. We are very excited to explore the steps noted in *If We Had More Time* and implemented our future product vision.

Challenges

1. Inferencing on the Jetson

We needed to ensure that the model we were building on the cloud was not too large to inference on the Jetson. After creating our models and obtaining the weights, we ran into issues that presented learning opportunities for using the weights on the Jetson device.

We at first built numerous models on the cloud, only to find that our files may not work with inference with opencv (this was our initial plan). We spent quite a bit of time trying to convert both .pt and .pth files to .xml, only to realize this was not required. We also explored converting

the file extensions to .onnx extensions and performing inference with those. However, we eventually realized that we could use the initial file types directly on the Jetson.

Using the initial file types (.pt and .pth) was a matter of building the appropriate environment with the docker file. We realized that if we replicated the set-up we had on the Jetson within the docker container, we could successfully infer the weights within the pipeline. We focused on the yolov5 small model first, since we were sure the size was successful on the Jetson device. If we had more time, we would follow the same process using the .pth file extension from our efficientDet model. Some steps of this process include cloning the appropriate repository, installing the required libraries in the docker container, creating a detect.py file to run within the container (inclusive of mqt), and ensuring that the weights and any other required files are appropriately copied into the correct locations to run the inference.

2. Kubernetes issues on ec2:

We ran into a significant issue with our pod deployments stuck in pending on ec2 for both the mosquito broker and the image processor. After further investigation, we realized when using the commands from lab3 to install kubernetes, the version was updated on 7/19. This was found when three of our team members were trying to install kubernetes on fresh instances, and one member had an installation from earlier in the class. The member with kubernetes installed from earlier in class had the earlier version of kubernetes, and did not have issues with pod deployments on the cloud. While we are not sure this is 100% the entire root cause of the kubernetes ec2 issues we faced, and we did take numerous other steps to troubleshoot the issue, this was the most notable difference we found and we were able to deploy with the earlier version.

"Old" kubernetes version information:

```
Client Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.6+k3s1",
GitCommit:"418c3fa858b69b12b9cefbcff0526f666a6236b9", GitTreeState:"clean",
BuildDate:"2022-04-28T22:16:18Z", GoVersion:"go1.17.5", Compiler:"gc",
Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.6+k3s1",
GitCommit:"418c3fa858b69b12b9cefbcff0526f666a6236b9", GitTreeState:"clean",
BuildDate:"2022-04-28T22:16:18Z", GoVersion:"go1.17.5", Compiler:"gc",
Platform:"linux/amd64"}
```

"New" kubernetes version information:

```
Client Version: version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.3+k3s1",
GitCommit:"990ba0e88c90f8ed8b50e0ccd375937b841b176e", GitTreeState:"clean",
BuildDate:"2022-07-19T01:10:03Z", GoVersion:"go1.18.1", Compiler:"gc",
Platform:"linux/amd64"}
Kustomize Version: v4.5.4
Server Version: version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.3+k3s1",
GitCommit:"990ba0e88c90f8ed8b50e0ccd375937b841b176e", GitTreeState:"clean",
BuildDate:"2022-07-19T01:10:03Z", GoVersion:"go1.18.1", Compiler:"gc",
Platform:"linux/amd64"}
```

What If We Had More Time?

As laid out above, we experienced hurdles every step of the way. However each of these obstacles taught us key theoretical and practical concepts that helped guide our next decisions. While YoloV5 proved to work as a POC/MVP for the goal of this project, enhancements for future versions would include the following changes:

1. Add additional images to the dataset that are not just dogs. Per research done in <https://towardsdatascience.com/a-dog-detector-and-breed-classifier-4feb99e1f852>, models need to be trained to understand differences between dogs, humans and other living creatures or non living objects. This addition would be key to improving training of this model.
2. Additionally, we would investigate and commit to pose estimation labeling in order to improve accuracy of classifying sitting versus others. Pose estimation identifies key joints on the body in order to classify the image into a pose.
3. We would add in other classifications for images. In this current project we focused on 'sit' and labeled all other poses 'other'. However with the help of pose estimation, classification of standing, laying and other poses will be possible in a more accurate manner than utilizing a bounding box with classification.

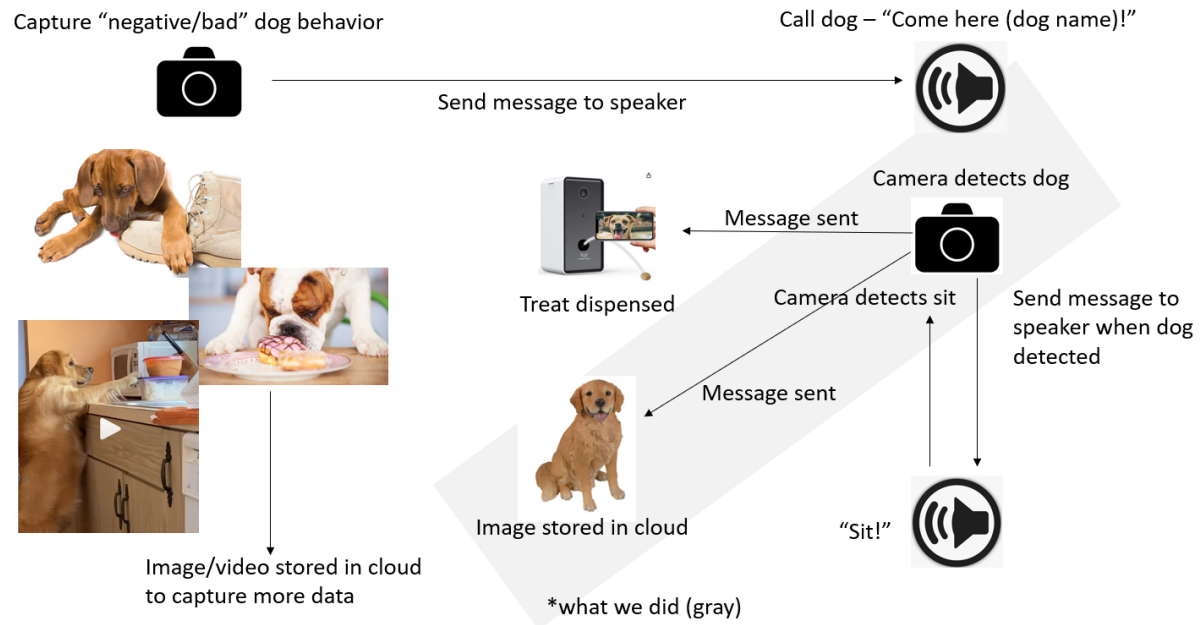
In addition to the changes above, that we recommend in this piece of the pipeline, we also have a desire to extend the pipeline into an end to end training pipeline for dogs. To make this a reality, the main challenge is collecting a robust dataset of "bad/negative" dog behavior. One idea we have, to obtain this, is to work with dog social media influencers (dog owners have Instagram accounts for their pets) and have the dog accounts run campaigns with slogans like "We all know social media makes us seem like the best dogs in the world, though we know that's not true. We're bad, too! Send us your naughtiest dog clips for a chance to be featured on our page." Note, the marketing will need to be massaged. An opportunity may also exist to collect similar public data through apps like YouTube or TikTok, which are historically video focused channels.

Example videos of "Luke" stealing food on his instagram page. These are the types of videos we want to acquire via social media channels to construct a dataset. We may have to start with a specific camera angle, or identify a dog in a frame with a shoe, at first.

https://www.instagram.com/lukethegoldenchild/?utm_source=ig_embed&ig_rid=fab91e29-2914-4972-8f76-01dc5fb44bde

See future product vision chart on the next page.

Future Product Vision



References:

Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao and Li Fei-Fei. **Novel dataset for Fine-Grained Image Categorization.** *First Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. [\[pdf\]](#)

Cavey, Tom & Stock, Jason. "Who's a Good Boy? Reinforcing Canine Behavior in Real-Time using Machine Learning." Cornell University: arxiv. January 21, 2022. Link: [\[2101.02380v2\]](#)
[Who's a Good Boy? Reinforcing Canine Behavior in Real-Time using Machine Learning \(arxiv.org\)](#)

Solawetz, Jacob. "How to Try CLIP: OpenAI's Zero-Shot Image Classifier." *Roboflow Blog*, Roboflow Blog, 8 Jan. 2021, <https://blog.roboflow.com/how-to-use-openai-clip/>.