Prototyping a Privacy-Conscious Dog Monitoring System: Exploring Semantic Segmentation Techniques for Real-Time Video Feed

Allison Aprile
August 2021
aaprile@stevens.edu

ABSTRACT

Image processing and computer vision techniques are prevalent in almost all modern home monitoring systems, backending tasks such as facial recognition and object detection. These features have branched into pet monitoring applications, where developers continue to seek new ways to improve the experience of both the owner and their pet. The lack of privacy is one pitfall of the existing products, which inspired me to create my own privacy-conscious dog monitoring system.

To address the privacy issue, it is necessary to enhance the important pixels of each video frame (namely, the dog) and distort the background pixels. In this paper, I explored various unsupervised and supervised semantic segmentation techniques to classify the pixels of a given image as either belonging to a dog or the background. Then, I evaluated their performance on several color images containing dogs, sourced locally from either my Furbo Dog camera or my iPhone. As expected, the unsupervised techniques (including Global and Adaptive Thresholding, K-Means Clustering, Edge Detection and Linking) could not universally segment the images, while the supervised technique (U-Net [3] trained on the dog subset of the Oxford Pets III [4] dataset) yielded reasonable results. Accordingly, I incorporated this model into my camera scripts, which, given a video frame, enhances the pixels classified as a dog and distorts the pixels classified as background, using Histogram Equalization and Gaussian Blur, respectively. Once the scripts were deployed to my Raspberry Pi, I examined the performance on a live video feed of my dogs. My solution performed decently overall, effectively blurring most of the background pixels while emphasizing the dog. Unfortunately, due to the heavy computation of the model, the application experienced severe lag. Therefore, in future improvements, I plan to utilize the Raspberry Pi's GPU for computation, as well as collect more data for a more precise semantic segmentation.

Tools

The prototype dog camera was built using a selective combination of hardware and software. For the computer component, I chose a Raspberry Pi 4 Model B due to its comparably superior video processor and overall CPU. For the camera, I used the Arducam Raspberry Pi Official Camera Module (V2), with an 8 MegaPixel IMX219 Autofocus Replacement. I decided on the largest resolution available because more pixels would allow for more visual information of the objects, which is important in image processing and computer vision applications. Additionally, the Autofocus Replacement seemed suitable given the frequent movement of my dogs. I connected the camera module to the Raspberry Pi using a ribbon connector, and then secured it with a plastic camera mount. To protect against my dogs, I built a simple camera box out of wood. For software, I flashed the Raspberry Pi with Raspbian, and installed Python 3.6. Majority of the image processing was done on Google Colab using OpenCV 3 and NumPy, and the deep learning model training was completed using TensorFlow 2.2.0.

I. INTRODUCTION

Over the past few years, image processing and computer vision have significantly advanced the benefits of digital media, especially video. By applying these techniques, we can automate mundane tasks and discover hidden patterns, most recently in real-time due to modern computing resources. Thus, image processing and computer vision features, namely object detection, have become desirable in monitoring systems.

Inspiration

Pet monitoring systems, such as Furbo and Companion, use computer vision (in addition to audio and other domains of Artificial Intelligence) to provide owners with real-time updates about their pets, as well as interact with the pets. In particular, Furbo uses object detection to send users SmartAlerts about home emergencies, such as fires or burglaries, and to send a notification when their dog is in the frame. On the other hand, Companion focuses on fine-grained classification of the dog's position, using such to decide on how to engage with the dog. They highlight their use of edge detection, which is image processing technique based on rapid intensity changes throughout an image.

Semantic segmentation is another area of image processing and computer vision that could be successful in dog monitoring systems. The goal of semantic segmentation is to individually classify pixels, leading to more precise understanding of a particular object in an image. For pet cameras like Companion that focus more on the animal's behavior rather than the environment, it could be beneficial to segment any instances of a dog from the background. Not only will this reduce noise in the behavioral classifications, but it could help preserve privacy; we can enhance the quality of the pixels belonging to a dog while independently reducing the quality of the background. While this similar idea has been incorporated into applications like Zoom, it has not yet been introduced in pet monitoring systems.

Problem Statement

For the past few months, I have been building a custom monitoring system for my dogs. In this paper, I will discuss a semantic segmentation-based solution to the privacy concerns. Specifically, I aim to identity and enhance the foreground object (i.e. the dog) and distort the background. First, I propose several unsupervised and supervised semantic segmentation approaches. Then, I implement several of them, namely Basic Thresholding, Edge Detection and Linking, Clustering, and Artificial Neural Networks. Next, I evaluate their respective performances on images procured from a Furbo dog camera and an iPhone camera. Finally, I incorporate the most effective solution into a camera application, which is then deployed to a Raspberry Pi and tested on real-time video feed of my dogs.

II. BACKGROUND

Video Semantic Segmentation

Image segmentation is one of the most important tasks of digital image processing, both playing a key role in a wide variety of applications and challenging researchers. Image segmentation involves partitioning an image into disjoint, homogenous regions based on similar features and attributes, such as color, intensity, and texture. It is a low-level computer vision operation that simplifies the image for higher-level operations, including content-based image retrieval, object detection, and object recognition. There are three types of segmentation subtasks: instance segmentation (annotating multiple objects of the same class as distinct individual instances), semantic segmentation (annotating every pixel of an image with a class label), and panoptic (assigning both a semantic label and instance label to every pixel in an image) [1][2]. **In this paper, I explore various approaches to this semantic segmentation subtask. Given a frame from a continuous video feed, I aim to classify each pixel as belonging to a dog or the background (i.e. not dog). This low-level processing will determine which pixels to enhance and which to distort, thereby accomplishing the high-level task of a more privacy-conscious pet monitoring system.**

Figure 1. Types of Image Segmentation (a) original image; (b) Semantic Segmentation; (c) Instance Segmentation; (d) Panoptic Segmentation

Related Work – Surveys of Image Segmentation Techniques

Image segmentation techniques can be categorized in many ways. For example, segmentation can be global (applied on the entire image) or local (applied on a subset of the image), discontinuity-based (based on areas of intensity discontinuity) or similarity-based (based on dividing pixels into regions having similar pixel properties), or unsupervised (using inherent features of an image to recognize patterns) or supervised (using class labels to train a model to recognize patterns). Several publications, such as "Various Image Segmentation Techniques: A Review" [2], as well as "Image Segmentation Using Unsupervised Techniques" [1] overview and compare popular techniques on the bases of mathematical theory, ideal use case, advantages, and disadvantages.

Unsupervised Segmentation Techniques

Unsupervised segmentation is the classification of pixels using software analysis of inherent pixel qualities, such as intensity and other similarities relative to neighboring pixels. Because human input is not required, unsupervised algorithms are easy to implement and efficient.

Thresholding is the simplest of the unsupervised segmentation method and is often used as a decision function for later supervised methods. It is similarity-based, involving dividing the image pixels according to their intensities. There are three major types of thresholding: 1) global (where the threshold value T is constant throughout the image); 2) variable (where T varies over the images, also known as adaptive thresholding); and 3) multiple (global thresholding with multiple T values). It is typically applied on grayscale images, where the threshold value(s) T are determined by valleys in the grayscale histograms. For color images, it is required to threshold the color channels separately and combine using an AND operation– this is known as called multiband thresholding. Overall, thresholding works best on bimodal images with highly dependent histogram peaks. For that reason, I did not expect it to perform well with my data.

Clustering is another similarity-based segmentation method. It involves dividing neighboring pixels into homogenous clusters, where pixels of a cluster have similar characteristics. It is either hierarchical (using trees to produce a top-down or bottom-up leveling of clusters) or partitional (requiring input parameters and criterion analysis to optimize partitions), and either hard (pixels have one-to-one mappings with clusters) or soft (pixels have one-to-many mappings with clusters). Soft clustering tends to yield more favorable results in image segmentation tasks as its flexibility allows for more natural clustering. Overall, clustering works well in simple images, but determining cluster membership for noisy images can be challenging. That being said, this method is not expected to adequately segment my data.

Edge Detection based segmentation is a discontinuity-based segmentation technique, based on the behavior of large changes in intensity over a short frequency. Edges are determined as areas in the image where either 1) the first derivative of intensity (i.e. change in intensity) is greater than a threshold T or 2) the second derivative of intensity has zero crossings. Then, edges are linked, typically using morphological transformations, to form boundaries. This method is especially suitable for images with moderate noise, where a single intensity value cannot provide sufficient

information to determine groupings. However, it does not perform well on images with too many edges; therefore, I do not have high expectations for this method when applied to my data.

The Watershed and Region-Based methods are two other popular segmentation techniques. They are unique because they can be automatic (unsupervised) or manual (supervised). The Watershed method is based on topological interpretation, where a given pixel intensity is treated as a 'basin' that dilates until it reaches another pixel 'basin'. While the results are stable on images similar to those in my dataset, said results dependent on some user input (marking basin areas) and heavy computational time. The Region-Based (Growing) method, which segments an image into regions based on the growing of initial pixel 'seeds', is similar, with stable results depending on user input (initial selection of seeds) and a high time and memory expense. All aspects considered, these two methods are not viable for a real-time segmentation algorithm, and are not tested in this paper.
[1][2][5]

Supervised Segmentation Techniques

Supervised segmentation is the classification of pixels using a combination of software analysis and human input regarding low and high-level image characteristics. After inherent feature extraction, a model is trained to associate certain features with their corresponding classes; this simulates a learning process for decision making [2]. Semantic segmentation requires input of pixel-level class labels for all pixels in an image, and accordingly, a model trained for that task will predict labels for all pixels given an image. Therefore, supervised segmentation segmentation (or pixel-level classification) is more complex and expensive than standard image classification, where one label is output to describe the collective pixels given an image. Nonetheless, the modern deep learning architectures designed for image classification are applicable for the semantic segmentation task.

In the publication, "Image Segmentation Using Deep Learning: A Survey" [6], the authors recommend multiple deep learning model architectures, including fully convolutional networks, graphical convolutional models, encoder-decoder based models, and recurrent neural networks. In the interest of time, I trained adaptations of the fully convolutional networks. Convolutional Neural Networks (CNNs) are state-of-the-art for computer vision applications as they are revolutionary in their architecture and flexible for a variety of tasks. They consist of three layers: 1) convolutional layers (where a filter of weights convolves with an image to extract features); 2) nonlinear activation layers (where an activation function is applied on the feature maps produced by convolutional layers); and 3) pooling layers (which reduce the dimensionality of the feature maps by replacing a small region of the map with a statistical quantifier). CNNs output classification scores, which, given an image and d class categories, is a d-dimensional vector of probabilities. Fully convolutional networks (FCNs) adapt the standard CNN by including only the convolutional layers. This outputs a spatial segmentation map rather than a vector of class probabilities. Because the FCN architecture is designed for the segmentation tasks, as well as trained on typically diverse data, it can recognize and generalize patterns sufficiently despite noisy images. That being said, I expected it to perform very well in my application, although there were concerns about training and real-time prediction complexity.
[2][6]

Data Description

I applied the various techniques on seven different images of my dogs. Five of the images are screenshots from my Furbo, four of which containing a single dog and one containing both. I decided to take predominately screenshots because they best parallel my Raspberry Pi camera quality and dog positioning.

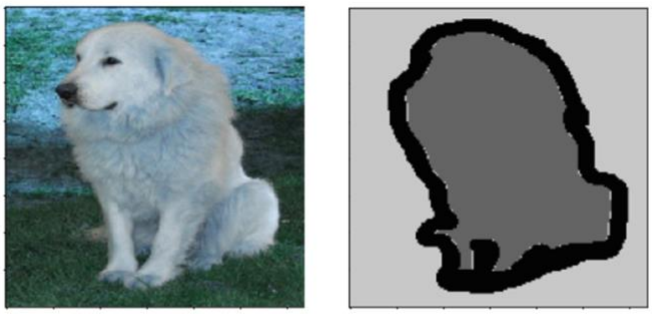


Figure 2. Oxford III-T Pet dataset [4] example of (a) input image; and (b) corresponding pixel-level segmentation mask

The other two images were selected from my iPhone camera role, one containing both dogs close together and the other containing me with one dog. I carefully selected these images so I could thoroughly evaluate each technique's generalization abilities.

Additionally, I modified the Oxford-IIIT Pet Dataset [4] to train the supervised FCN. This dataset contains 7349 images of dogs and cats spanning 37 breeds. Each image is 160x160p and has variation in scale, animal pose, and lighting. The accompanying annotations include species and breed name, tight bounding box Region-of-Interest (ROI) coordinates around the animal's head, and pixel-level segmentation trimaps. More specifically, the pixels are classified as one of three classes, namely foreground (the animal), background, and ambiguous (containing part of the animal boundary, accessories like collars, and fur texture). Because cats are not in my testing scope, I filtered the dataset for only the dog instances. Overall, the final model training dataset included 4990 images of dogs, spanning 25 different breeds. I did a 90/10 training and validation split, yielding 4991 and 499 samples, respectively. Each input image provides the features and the class labels are the corresponding pixel-level segmentation trimap.

III. METHODOLOGY

To determine the semantic segmentation backend for my dog monitoring application, I compared the performance of several of the discussed techniques on my seven testing images. For each technique, I experimented with different algorithms and parameters, and then used human evaluation to determine their capability. I implemented the enhancement, distortion, and unsupervised techniques (Thresholding, K-Means Clustering, Edge Detection and Linking) with the OpenCV library, and the supervised technique (Fully Connected Network) using TensorFlow machine learning framework in addition to OpenCV.

Thresholding

Although I did not expect Thresholding to work well with my images, I wanted to establish a baseline. I decided to convert the images to grayscale and threshold them accordingly, rather than process the color components individually. Then, I applied Gaussian Blur with a 5x5 filter and 0 sigma to remove noise. I then generated the histograms for each image, noticing that most of images had arbitrary peaks and no decisive valleys. This led to undesirable results for global thresholding, with the Truncated and To-Zero types producing the relative best results.

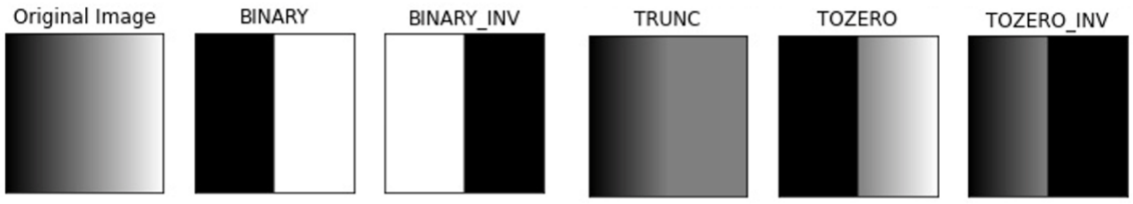


Figure 3. OpenCV thresholding types

I also applied Otsu Global Thresholding (also called Otsu's Binarization), which automatically determines a threshold using variance minimization. Again, I observed insufficient results, which was expected given Otsu's Binarization is ideal for bimodal images.

Finally, I implemented the Adaptive Thresholding technique, trying both the Arithmetic and Gaussian mean filters. In terms of capturing the dog boundaries, this method performed well; however, it is still not a viable option for my final application as it produced a rather noisy result.
[5]

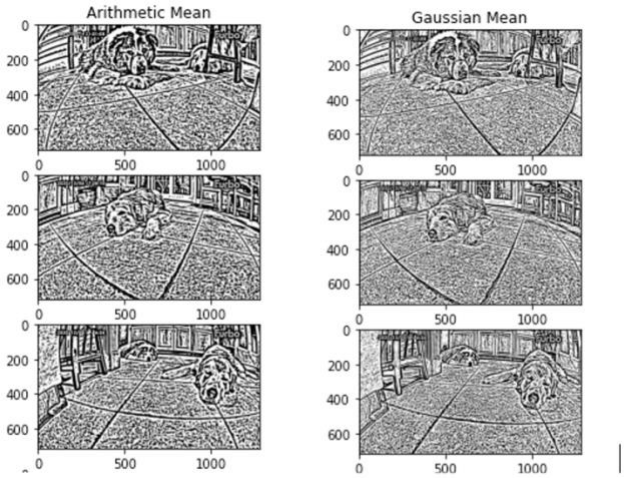


Figure 4. Results of adaptive thresholding on test images

K-Means Clustering

Several papers suggested K-Means Clustering as the Clustering algorithm most suitable for image segmentation [1][2]. It is a hard clustering algorithm, where K centroids are randomly initialized, with K being the number of classes. Each pixel is assigned the class of its nearest centroid, and then the centroids are adjusted as the means of the assigned points. The K-Means algorithm converges when there is little to no change of centroids between iterations. This method is optimized by maximizing intra-cluster similarity and minimizing inter-cluster equality [5]. To implement to, I first applied Gaussian Blur with a 5x5 filter and 0 sigma to remove noise. Then, although I desire K=2 in my final application, I ran the algorithm for K = 2, 3, 4, 5 and 6. I defined the stopping criteria as the first occurrence of either 100 iterations or an intra-cluster similarity of greater than 0.70. The algorithm performed as expected: producing interesting visual effects in terms of spatial regions, but failing to sufficiently segment the dogs. I ultimately attribute these results to lighting conditions and contrast in the dog features (for example, one of my dogs has white and black fur).

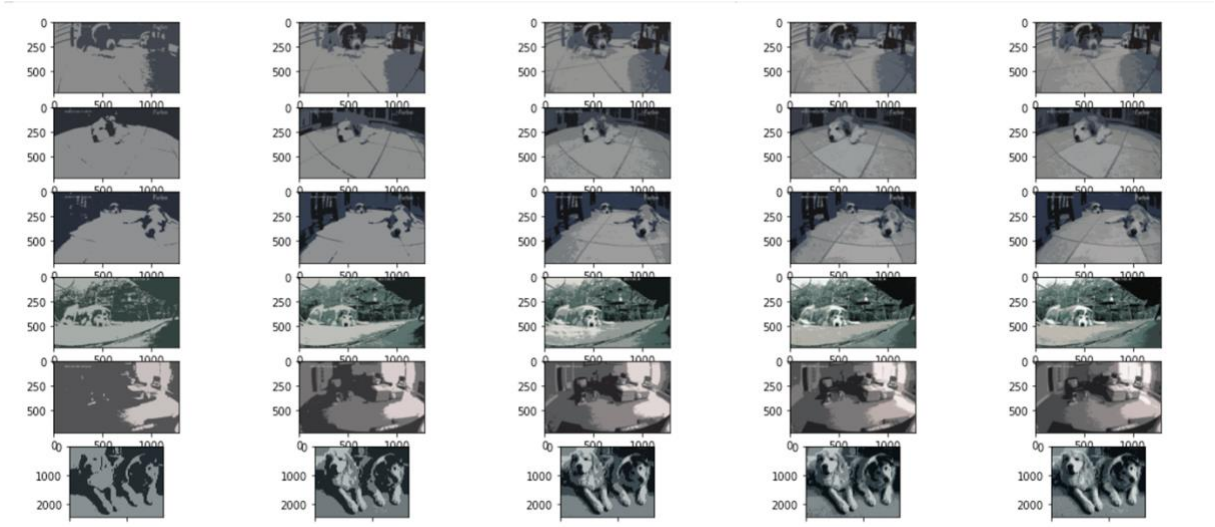


Figure 5. Results of K-Means Clustering on test images for various K values, (a) K = 2; (b) K = 3; (c) K = 4; (d) K = 5; (e) K = 6

Edge Detection and Linking

Of all the unsupervised techniques, I expected Edge Detection to yield relatively favorable results. I implemented four Edge Detection algorithms, namely Laplacian, Sobel, and Canny. I tested all but the Canny filter on the Gaussian Blur noise-reduced images; the Canny filter incorporates noise removal inherently. The Laplacian Edge Detector is the simplest to apply as it only has one kernel, calculating second order derivatives. While it successfully detected the dog edges, it captured too much noise and textures that would complicate high-level segmentation. The Sobel Gradient filter significantly better, instead taking the gradient magnitude of two filters (one for each of the horizontal and vertical directions). I tried computing the gradient magnitude with both the standard formula (the square root of the sum of squared horizontal and vertical gradients) and the lighter formula (the sum of the absolute value of the horizontal and vertical gradients), with no noticeable difference. This method captured less noise and generated a result comparable to the Canny filter.

Given an image, the Canny filter first reduces noise, then computes the magnitude gradient using Sobel kernels. Then, it uses Non-Maximum Suppression (a mapping function based on the change in neighboring gradient magnitudes) to remove false edges. Finally, it incorporates Hysteresis Thresholding (a comparison function using two threshold values) to determine whether pixels are suppressed or included in the final edge map. Canny filtering involves setting upper and lower threshold values, which can be complicated given the real-time processing of my image application. Therefore, I implemented Automatic Canny detection by setting sigma to 0.33 (as established by computer vision researchers), and using such with the maximum and minimum pixel intensities to scale the median of all the pixel intensities. While this method still did capture noise, in six of the seven instances it detected the distinct edges of the dog.

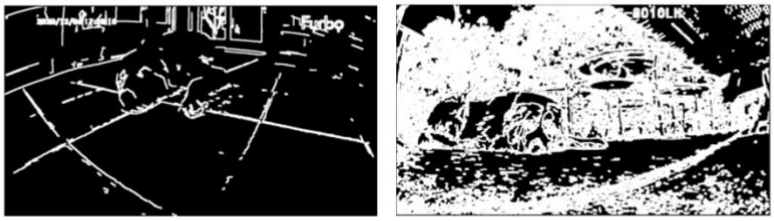


Figure 6. Results of Canny Edge Detection with Morphological Gradient + Contour Detection Edge Linking, (a) clean result; (b) noisy result

To complete the image segmentation, the detected edges must be linked. For this, I first tried the simple close and gradient morphological transformations, with the gradient operation producing discernable results for most of the images. I also tried Contour Detection following the morphological gradient in an attempt to extract the distinct objects. However, overall, this proved to be a unsuccessful method, which caused me to switch to a supervised approach to segmentation.
[5]

Fully Convolutional Network

To implement the Fully Convolutional Network (FCN), I looked to recent research in the biomedical field, where deep learning is topical in the task of segmentation. In "U-Net: Convolutional Networks for Biomedical Image Segmentation" [3], an adaptation of the FCN architecture is detailed. U-Net builds on the FCN, modifying the layers for a contracting path (to capture context) and a symmetric expanding path (for precise localization). The contracting path has repeated applications of the same convolutional block structure found in the FCN, ultimately doubling the number of feature channels at each downsampling step. Every step in the expansive path consists of upsampling the current feature map, followed by transposed convolutional blocks to halve the number of feature channels. At the end, the feature maps are concatenated, followed by additional convolutional layers, and resulting in a mapping of a 64-dimensional component feature vector to a number of classes-dimensional vector. In total, it consists of 23 convolutional blocks. It is revolutionary in several ways: 1) the upsampling operators increase the number of feature channels, in turn increasing the resolution of the output, which is necessary for adequate localization in the following expanding path; 2) it can be effectively trained on smaller datasets with augmentation, which alleviates some of the tedious labeling efforts, consequentially alleviating the big data issues faltering deep learning tasks; 3) it has a fast prediction time on GPU, which is useful for real-time segmentation.
[3]

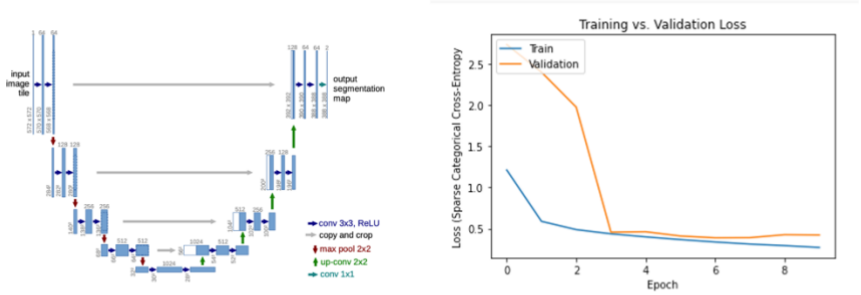


Figure 7. (a) U-Net architecture [3]; (b) Training vs. Validation Loss over Epoch

Because the architecture is quite complex, I referenced a tutorial for the implementation. I trained the model on my modified Oxford-IIIT Pet dataset, compiling with the RMSProp optimizer and Sparse Categorical Cross-Entropy loss. It trained for 10 epochs (completing in approximately five minutes on GPU), resulting in 0.2723 training loss and 0.4235 testing loss. I used human evaluation on both random validation samples from the Oxford-IIIT Pet dataset and my test images. In some instances, it had trouble detecting multiple dogs or picked up background pixels, but overall, it is very usable for my dog monitoring system.

Pixel Enhancement and Distortion

Given a segmentation map produced by the U-Net [3], I processed the dog pixels and not-dog pixels accordingly. First, I enhanced the contrast of the input image using Histogram Equalization, which equalizes intensity distribution. Because this operation is done in the single channel grayscale space, I first converted it to the YUV color space and equalized the Y and U channels independently, then converted it back to the RGB space [5]. Next, I made a copy of the image and applied Gaussian Blur, using a 25x25

kernel and a sigma equal to 100 to blur, rather than simply reduce noise, in the image. This also could have been done by transposing displaced copies of the image, but the Gaussian Blur operation is more convenient and efficient. Then, using the segmentation map, I replaced the dog pixels with their corresponding equalized pixels, and replaced the not-dog pixels with their corresponding blurred pixels. Assuming that the segmentation map was accurate, this produced the desired effect.

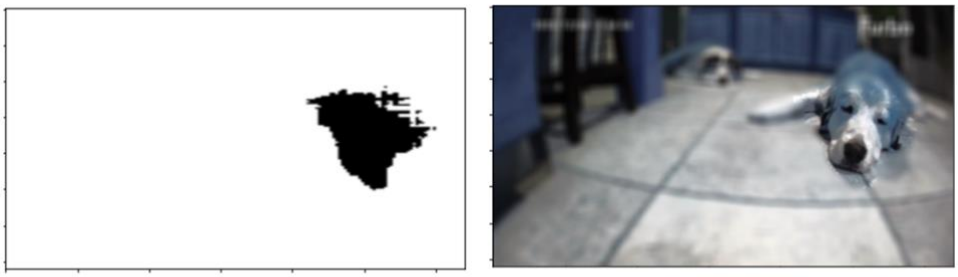


Figure 8. (a) Segmentation map produced by U-Net [3]; (b) Final image output

IV. RESULTS

After selecting the image segmentation technique and designing the enhancement/distortion functions, I incorporated the code into a Picamera script and deployed it to the Raspberry Pi. The script first loads the FCN weights, initializes the camera resolution as 640x480p, and sets the frame per second sampling rate as one. My camera module supports 1080x720p resolution and a frame rate of up to 64 fps, but due to the time complexity, I kept it lower. Then, as the camera captures a continuous sequence of video, I sample a frame, process it, and then display it in a window (which updates the image every second, per the frame rate). The processing steps are as follows: 1) perform histogram equalization to improve contrast; 2) run the image features through a forward pass of the FCN; 3) threshold the trimap so it is binary (i.e. take the ambiguous pixels as dog pixels); 4) replace the background pixels with their Gaussian blur equivalent. The video stream terminates as when the viewing window is closed.

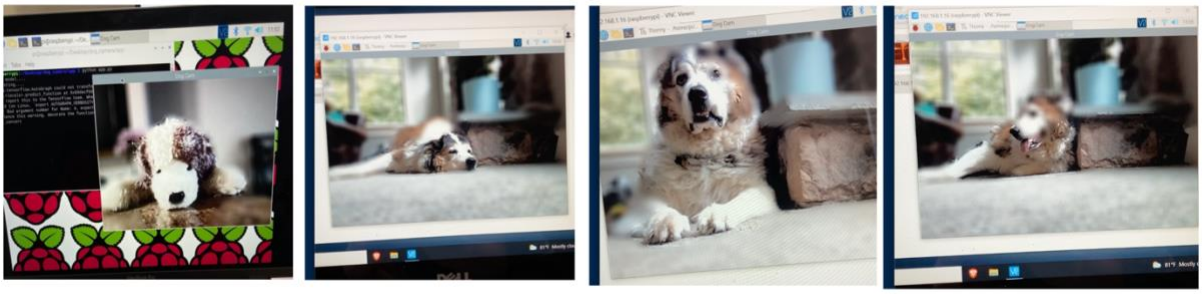


Figure 9. Various live camera results

Figure 9 shows three screenshots of live tests. First, I tested the processing out with a dog stuffed animal. I noticed quickly that despite it the tremendous lag (I attribute this to not utilizing my Raspberry Pi's GPU), it successfully blurred the background and enhanced majority of the area containing the dog. Then, I tested the system out on my two dogs, one who moves around quickly and the other who is older, and thus stationary. My tests with the first dog were unsuccessful; the combination of the lag and his quick movement resulted in a very distorted image. However, the camera was successfully able to blur the background surrounding the slower dog as he moved. I purposely situated the camera in an active environment to evaluate its full performance, and despite any humans entering the frame or animal dog toys in the

background, it sufficiently blurred the background. Regardless of the runtime issues and some misclassified boundary pixels, the selected combination of image processing and computer vision techniques achieved the desired effect.

V. CONCLUSION

Future Work

There is a lot of room for improvement for the current solution, particularly in the runtime and fine-grain performance of the semantic segmentation. When the camera is first turned on, it takes approximately two minutes for the broadcast to start. This is attributed to the time required to load and instantiate the FCN. Additionally, while the image processing processes take milliseconds, each segmentation map prediction takes approximately two seconds, causing additional lag between frames. To address this, I plan to switch from the Raspberry Pi built-in CPU to GPU, or deploy the prediction tool as an API to a cloud-computing resource.

In terms of the segmentation, I noticed that the FCN consistently identifies the face of the dog, but has issues along the boundary and especially with the lower half of the body. As suggested by the paper [3], I plan to retrain the model with data augmentation to see if this improves the classification, as well as adding more training data as it comes available. I also may explore other deep learning networks proposed by "Image Segmentation Using Deep Learning: A Survey" [6].

Overall, I plan to continue working on both the software and hardware components of this project. Inspired by Companion, I hope to incorporate more user and pet interaction elements, namely an automatic/manual treat throwing mechanism and push notifications.

Summary

In this paper, I detail the development of the backend to a privacy-conscious dog monitoring system, specifically the processing of the real-time video feed. Given a frame, I aimed to classify each pixel as belonging to a dog or background (i.e. not dog) classes, and then enhance or distort the pixels accordingly. I tested various image processing and computer vision techniques for the pixel classification task, called semantic segmentation. The unsupervised techniques (namely Global and Adaptive Thresholding, Clustering, Edge Detection and Linking) did not adequately segment the images, as expected. Contrarily, the supervised technique (U-Net [3], an adaption of the Fully Convolutional Network architecture) performed up to standard and was therefore incorporated into the final solution. Using the segmentation maps output by the FCN, the camera backend is able to determine which pixels (classified as dogs) to enhance using Histogram Equalization, and which pixels to blur (classified as not-dog) using Gaussian Blur. After deploying the camera scripts onto the Raspberry Pi and testing with live dogs, I observed successful frame processing but an expensive runtime. Therefore, in the future I plan to utilize a GPU resource for the semantic segmentation operation, in addition to retraining the model on more data. I also plan to continue developing new capabilities in both the hardware and software components, aligning with the goal to create a unique user, as well as pet, experience.

VI. CODE INSTRUCTIONS

There are four directories in the **Aprile_CPE645_Final Project** directory. 'code' contains five iPython notebooks, one for each of the techniques I tried and the final camera output. 'pi_dog_camera_app' is the code deployed to the Raspberry Pi; this is just a consolidated version of the ANN and Final Camera Output code.

'live_results' contains three videos and one image of the live camera testing, and 'hardware_reference_images' contains images of the hardware components of the project.

REFERENCES

[1] B.M. Nagarajan, G. Prem Paul, and P. Senthil Babu, "Image Segmentation Using Unsupervised Techniques", International Journal of Innovative Research in Science, Engineering and Technology, vol. 3, no. 3, pp. 1167-1174, March 2014.

[2] Dilpreet Kaur and Yadwinder Kaur, "Various Image Segmentation Techniques: A Review", International Journal of Computer Science and Mobile Computing, vol. 3, no. 5, pp. 809-814, May 2014.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", CoRR, vol. 1505.04597, no. 1, May 2015.

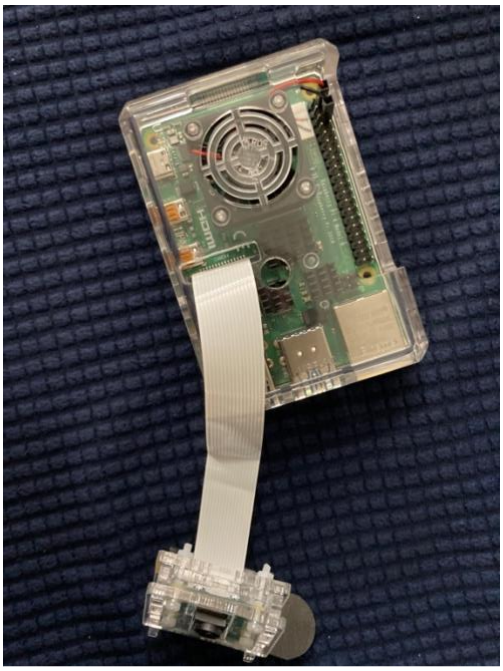
[4] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C.V. Jawahar, "Cats and Dogs", IEEE Conference on Computer Vision and Pattern Recognition, pp. 3498-3505, 2012.

[5] Rafael C. Gonzalez and Richard E. Woods, "Digital Image Processing", 3rd ed., Prentice Hall, 2008.

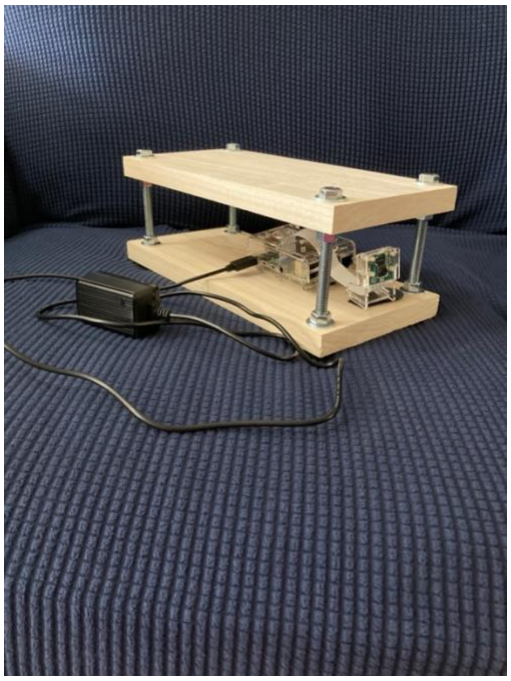
[6] Servin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos, "Image Segmentation Using Deep Learning: A Survey", IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1-22, 2021.

APPENDIX

A1. Arducam camera



A2. Raspberry Pi and camera setup



A3. Full dog monitoring system prototype