PetSTAR

Authors: Rebecca Manley, Brandon Wei, Max Jantos
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of detecting movement and classifying animals using a Raspberry Pi and camera in order to report an animal's movement into a forbidden zone to a user through a web application. We are implementing a convolutional neural network for animal classification and identification, and are using a Raspberry Pi to minimize cost so that the system is competitive with other pet monitoring devices. Our primary goal is to provide a cheap hardware solution that accomplishes more than market pet monitors.**

***Index Terms*— Axios, Convolutional Neural Network, Computer Vision, Forbidden Zones, OpenCV, Raspberry Pi, Web Application**

1 INTRODUCTION

Pet cameras are becoming increasingly popular as the technology becomes more accessible and people find themselves spending more time away from their home and pets. At their core, traditional approaches to pet monitoring are all quite similar - a camera which watches 24/7, and transmits this feed to a pet owner's personal device. However, this means that the pet owner would need to be actively watching (or spend time reviewing footage later) in order to get any idea of what their pet has been up to. In this project, we are making the task of monitoring and deterring actions by the pet more convenient, which is not currently available with market pet monitors. This is accomplished by incorporating computer vision and machine learning to detect animal movement in order notify users when a pet has entered an area it should not, while simultaneously playing a deterring sound to stop the pets behavior. It is our belief that this system could prove valuable in any home with pets. We will create a system to monitor one room of the house through a camera that can track the location of an animal when it is in frame, differentiate between multiple animals, determine if an animal goes into a user-defined forbidden area, and report all of this data to the user in a neat and comprehensive web portal. As we make this system, we also aim to make the product competitive with the current market by maintaining a low cost of components.

2 USE-CASE REQUIREMENTS

The use case for our project is, broadly, any home that has one or more free-range pets. The first broad category of our use case requirements relates to the overall functionality that we want to achieve with our system. Since we will alert the user when a pet goes somewhere it shouldn't, we want to minimize unnecessary disruptions to the user by having a low false positive rate of $< 10\%$. We also want to make sure that these reports reach the user quickly so that they are aware of any potentially dangerous situations with their pet(s). Counting from the time that the animal enters the forbidden area, we aim to be able to detect and notify the user within 10 seconds. Another core aspect of our functionality is the summarizing activity logs that we will provide for each animal. Since these logs are only useful to the user if they are accurate, our goal is to have logs which accurately reflect the movement of each animal $> 90\%$ of the time.

The other broad category of our use case requirements relates to accessibility. Since there is an enormous diversity in homes that have pets, it is important to us that we keep our project as inclusive as possible. For our finished project, it is our goal that the average user would be able to set up with system in ≤ 5 minutes, with instructions. We also want the web application to be intuitive and user friendly, with $> 95\%$ of users able to complete the core tasks (select forbidden zone, upload pet images, request activity logs, etc.) with little or no additional instruction. Lastly, we want to keep our system as affordable as possible. We aim to keep the overall cost to $\leq \$100$ as our research has indicated that \$100 is roughly the starting price for other pet monitoring systems with features beyond just an app-connected camera.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

Figure 1 shows the overall configuration of our system.

The machine learning CNN will implement a pretrained InceptionV4 dog and cat classifier with additional transfer layers. These additional layers are for differentiating between the user's pets. Images of the user's pet(s) will then be used to train these additional layers.

The web application is where the user will be able do all the relevant tasks and receive notifications for monitoring his/her pet(s). Specifically, the user can upload pet images, which will then be used for pet classification. After that, the user will be given an image of the room that is partitioned by a grid system. The user will then click on the grid squares that are forbidden zones for a pet, and this can be done multiple times for each pet a user has. After finishing this process, the user will receive notifications on the web application whenever a pet has entered a forbidden zone. Another core feature given to users is that a user can request for pet activity logs through a heat map that shows

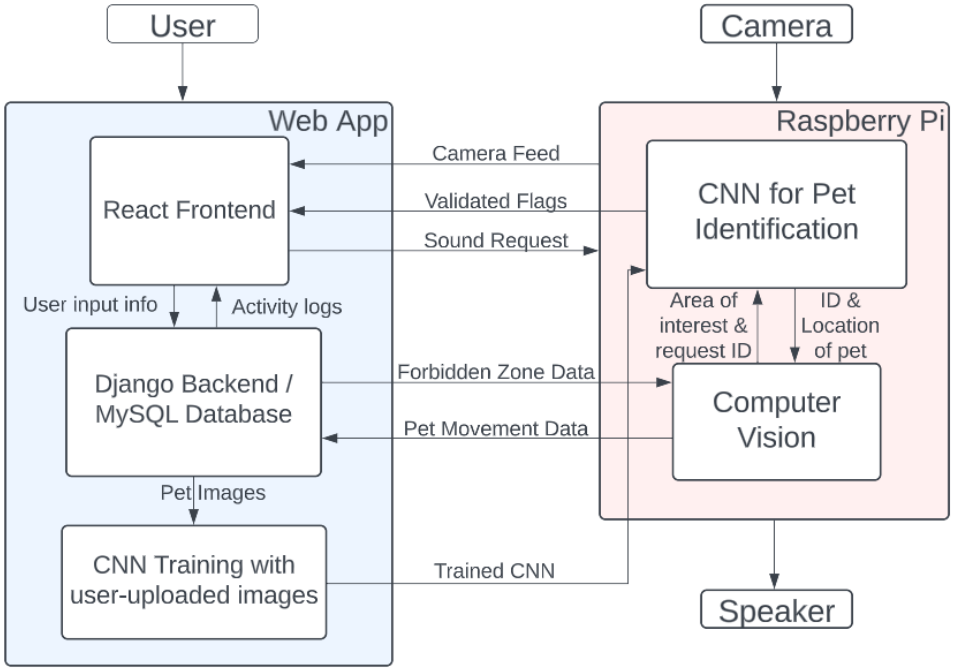

Figure 1: Overall PetSTAR System Architecture

the movement of the pet in the room for a long period of time. Not only that, a user can request for live video feed of the room, which is similar to what traditional cameras offer. Lastly, a user can request for a sound to be played so that the pet moves away from the forbidden zone. For any user input on the front end, the web application will interact with the back end to get or send any relevant data for a specific task on the front end. See Figure 3 at the end of the paper for the illustrated flow of how the user will interact with the front end app.

The computer vision aspect of the project will be based mostly in the OpenCV library. It aims to detect motion to see when an animal enters the frame and report this to the CNN. Once we receive ID from the CNN it will then track all animals in frame as they move through the environment. Using the forbidden zone data communicated from the web app, it will check for significant overlap between the position of the animals and any forbidden squares, raising a notification flag if needed. This will also request ID from the CNN to double check the identity of the animal.

Communications between the RPi and web app will be done via an internet connection. In terms of the data that will need to be sent, the web app must communicate user-input data. This includes the specified forbidden zones which will be used for collision detection in the computer vision, sounds requests, and updated CNN calculated based on the user-uploaded pet images which will be used for all on-hardware identifications. The RPi must communicate back with activity data and a live camera feed that can be displayed to the user as well as pushing a notification whenever an animal enters a forbidden area. The camera will connect to the RPi via a physical port on the board. The speaker will connect via bluetooth, most likely.

4 DESIGN REQUIREMENTS

Our primary use case concerns are having a user notification speed of less than 10 seconds, limiting the number of false positive user notifications, maintaining an pet activity log that accurately displays pet movement, and maintaining a low system cost. User's must be notified quickly when an animal has entered a forbidden zone assigned to it. In order to achieve this, we want our computer vision to be able to flag when an animal has entered a forbidden zone within 1 second of it having entered, and for our CNN to be able to validate these zone flags within 5 seconds of receiving a flag. This means that our CNN should perform its computation and report its result to the computer vision (every time) and web application (only on the case of a validated zone flag) within 5 seconds of being pinged by the computer vision. This has our computer vision and CNN speed to be combined less than 6 seconds, but we leave additional time overall for data communication between the Raspberry Pi and web application.

In order to prevent false positive notifications to the user, we want our computer vision to track the animals within 1 foot of their actual position to avoid false zone flags from being passed to the CNN, and we want our CNN to have a classification accuracy of greater than 90% across our testing trials. These will help prevent false positives by ensuring we do not mistakenly send zone flags to the CNN, which could correctly identify an animal and send a trigger a notification to the user by communicating with

the web app, even though the animal has not actually entered the forbidden zone. Ensuring an accurate CNN also prevents falsely identified animals from triggering a user notification, which could be especially annoying in the case of a multi-pet household where the animals could have their own unique forbidden zones, for example a house with a cat and a dog. If the cat enters the dog's forbidden zone, but is identified as a dog and triggers a user notification, this notifies the user for no reason.

Our ability to maintain an accurate pet activity log also depend on the same classification and pet tracking accuracy requirements as above, but now also include the ability to identify a new animal quickly when it enters the frame. If an animal leaves the room observed by the camera, we want to identify when the animal reenters the room as soon as possible to maintain an accurate activity log of the animal. This means that in the case of new motion detected on the edges of the camera frame, we want this motion detection and message to the CNN for classification to occur within 5 seconds.

Lastly, in order to maintain a low user cost, we want to use low cost hardware options. We want an overall system cost of $100 or less to make our option competitive with the current pet monitor market. We are achieving this using a Raspberry Pi (with a baseline price of $35 pre-pandemic, $45 post-pandemic) as our primary hardware in order to maintain a low hardware cost.

5 DESIGN TRADE STUDIES

5.1 User choosing forbidden zones

We had to think about how to implement a user choosing forbidden zones for an image of a room that the pet will be in. Specifically, we looked at the type of image displayed (2D or 3D image) and how we wanted to partition the forbidden zones (free forming zones or grid system).

Option 1: User choosing forbidden zones on a 3D Image

A benefit to allowing users choose forbidden zones on a 3D image is that users are able to be very specific in how they define these zones. For example, they are able to choose a forbidden zones in front of a box or behind a box, which is not possible with a 2D image. But, we believe they are many issues with implementing this option. Specifically, it is significantly harder to get a 3D model of a room just from the camera feed rather than retrieving a 2D image of the room. There would be more data needed to be stored in the database for a 3D image compared to a 2D image. In terms of user experience, it may be harder for users to navigate a 3D model of the room and choose forbidden zones compared to clicking and/or dragging on a 2D image to create forbidden zones.

Option 2: User choosing forbidden zones on a 2D Image using free forming zones

The advantage to free forming zones compared to a grid system is that the user is able to create finer forbidden zones as they can click and drag to create multiple zones which can fully encapsulate everything in the room that is forbidden without including parts of the room that is not forbidden for the pet to be in. As well, we believe a user will be less confused when trying to create these free foaming forbidden zones on a 2D image compared to navigating a 3D image to create forbidden zones. The main issues with this option is that storing this data into the database and giving this data for the computer vision on the Raspberry Pi to use is complex compared to the more simplistic data structure offered by using a grid system. As well, there is no sense of depth in the 2D image, so choosing in front of a box versus behind a box is not possible.

Option 3: User choosing forbidden zones on a 2D Image using a grid system

We believe this is the best option to implement as there is a simple data structure, specifically an array like structure, that can be used to store data into the database and also passed along to the Raspberry Pi. As well, this method should be not confusing for users as users will click on the grid squares to choose forbidden zones on the 2D image. But, they are still consequences such as this is the worst option out of the three in terms of choosing specific forbidden zones on an image. For example, the grid square may mostly include the forbidden zone that the user intended to create, but it will more than likely includes a few parts that the user did not want to include into the forbidden zone. To address this, we will make sure to get user feedback on grid square sizes so that users are satisfied with the forbidden zones they are creating on the image. Overall, we believe this option makes the back end of the project much more simpler while not having significant impact on the front end of the project.

5.2 Displaying of Pet Activity Logs

We thought about how we wanted to display pet activity to logs for users when they request them, specifically if we wanted to display a short vs long time frame and specific vs generalized pet positions.

Option 1: Time-sensitive graph

A time-sensitive graph displays a short time period of the pet's movement, but a user will be able to scroll back and forth in time to see the exact position of the pet in the room at a specific time. The benefits to this option is that if a user has in mind what time they want to see what his/her pet is doing or the user will only be gone for a short amount of time, then they can check this graph to get precise locations of where the pet has been in a specific time frame compared to a heat map. A consequence is that the user cannot get the overall picture of where the pet has been due to the short time frame that is offered by the graph, and the user will have to be more interactive with the graph compared to a heat map to understand what the pet has been doing.

Option 2: Heat map

A heat map will display locations of where the pet has been over a long time period using different colors that rep-

resent the spectrum between low amount of activity in a place of the room to a high amount of activity. The benefits to a heat map is that a user will receive information quickly about the pet's activity as the heat map is not interactive, and the user can get overall statistics on a pet's activity over a longer time frame such as a whole day. The consequences is that the user will not know where a pet is located at a specific time, which may be more of an interest to the user. Though both options seems to benefit the user in different ways, we believe this is the best option as it has a simpler implementation, and it is faster and easier for the user to understand the pet activity information.

5.3 Machine Learning Algorithm

We looked at different classification options, specifically varying neural networks, in order to find what best suited our needs.

Option 1: CNN

CNN is the best option for processing data like images or videos due to the addition of convolutional layers and pooling layers, which take advantage of inherent properties of the input images. Convolutional layers perform convolution, an operation commonly used in digital signal processing that examines the effect two functions have on each other. For images, this involves adding each pixel to the value of its neighbors, and weighting these values by a given kernel. Weighting each input (or pixel) by the values of its adjacent pixels emphasizes spatial coherence and allows for pooling layers, which downscale the image, as the emphasis on spatial coherence means that downscaling will not cause key feature loss. These two layers make CNN's much more efficient in terms of the number of operations, computational time, and amount of data maintained throughout computation than other NN's. This allows us to perform faster classification times than on other NN's due to less layers, while simultaneously maintaining a high accuracy. This enables us to use a lower cost machine due to less computational and data intensity, while still having a accurate and quick classification in order for users to receive fast and accurate notifications.

Option 2: NN

Normal NN's treat the elements of their input vectors with equal weight. This impacts their ability to downsize the given data. The lack of convolutional layers prevents downscaling the image through pooling layers, which are not possible because downscaling the input to a normal NN would mean the loss of key pixels which could cause severe feature loss. This makes a non-convolutional NN less desirable, as it lacks all of the helpful features inherent to a CNN. A normal NN would require more computational power in order to meet our current user requirements, such as fast classification (less than 5 seconds) than our current hardware would support, which would cause issues with our user requirement for cost (See the Hardware trade off section below).

5.4 Tensorflow vs Pytorch

In determining the framework we wanted to use to implement our CNN, we looked primarily at Tensorflow and Pytorch. We want a platform that will be well suited to computer vision problems, with easy ability to debug and the potential ability to link to other existing API's. Pytorch is a relatively new framework that is best suited for nature language processing (NLP). It is known to be easy to use and has effective memory usage, but has fewer API options due to its youth (developed 2017) and is currently mostly used for NLP. Tensorflow, on the other hand, has a few more years (developed 2015) and is commonly used across many machine learning applications. Tensorflow has access the Keras, which is an existing API for high level neural networks, such as the InceptionV4 architecture, which are extremely well suited for computer vision problems such as our own. This includes pre-trained computer vision models for object classification, and extensive documentation on the Keras API, Inception architectures, and Tensorflow as a whole. This framework also has great flexibility in terms of resource management, as well as a lot of helpful tools for visualization, which will be extremely helpful during debugging. Though the two are very similar in terms of resource management, and therefore will both be able to meet our given use-case and design requirements, the greater documentation, access to Keras and the InceptionV4 architecture, and extra tools for visualization and debugging will make for an easier path to achieving our goal due to easier ways to address issues as they come up and providing us the tools and information that will be crucial to overcoming technical and programming challenges that arise.

5.5 Tracking vs. Full Image Detection

When considering the algorithms we wanted to use to keep track of the animal within the room, there seemed to be two possible solutions. Many examples of ML identification on a Raspberry Pi used fed every single frame of the video feed to an ML model in order to identify and pinpoint features of interest. The benefit to this approach is that it is quite accurate, but the downside is that it runs a good amount slower, since so much more processing has to be done on every single frame. The other possibility we saw was to use the ML model once to identify features of interest then follow their movement with tracking algorithms, avoiding the need to use the ML model so frequently. It would still need to be called every so often to correct for tracking failures, drift, and new objects entering the frame. As this approach runs the full ML model significantly less often, it thus runs a good bit faster - tracking algorithms require much less image processing as they mostly only look at neighboring pixels. However, they tend to lose accuracy over time, meaning that even if we re-correct every so often with the ML model, our position estimate for each animal will still be overall less accurate. We have decided to go with the latter approach, as we feel we can tune it to be

accurate enough (vary how much the ML model needs to be called) while saving a fair amount on computing resources. In this way, we hope to make our software run acceptably well on our chosen hardware, discussed below.

5.6 Hardware

One primary trade study that we've spent a long time considering is which hardware platform we want to base our project around. The two options we've been considering are a Raspberry Pi 4 (RPi) or and NVIDIA Jetson Nano. In short, we feel it is a tradeoff between performance (with the Jetson) and cost plus ease of use (with the RPi). Jetsons are specialized to deal with graphical processing, which we will be incorporating a lot of between the vision and ML aspects of our project, whereas an RPi is not. Implementing our system using a Jetson would certainly give better performance in terms of the frame rate that we're able to process. However, a Jetson is more expensive than an RPi, especially in the current market, as summarized in Table 1 (note that this assumes the cost for a model with 2GB memory).

Although we realize that a Jetson is certainly more suited to the nature of our project, we are not concerned with our system being the most optimal that it could possibly be. We believe that we can achieve sufficient functionality even with the less computational RPi, so currently this is the platform we intend to use. This allows us more room to include the camera and speaker we need without risk of exceeding the $100 price point we have set for our use-case. The RPi also has the built-in capability of being able to communicate over the Internet, which a Jetson does not, and saves us from needing additional components for our wireless communication needs.

6 SYSTEM IMPLEMENTATION

6.1 Web Application

The front end will be developed using React, which will allow us to create interactive tasks that should enhance user experience. Before interacting with these tasks, the user will login through Google OAuth 2.0. The back end will be developed using Django, which will store important data from user input on the front end. Each user will be represented as a User object, which contains basic information about the user from login and Pet objects, which is based off the Pet model. The Pet model will store all of the relevant data for each pet, which will include fields such as pet images, forbidden zone data, and pet activity logs. As Django does not supply array fields, other models will be created so that we can use the ManyToOne field supplied by Django to represent an array. One model that will be created is a Box model, which represents a grid square of the grid system for the forbidden zone, that contains a boolean field that specifies if that grid square is forbidden or not. Therefore, we can use the ManyToOne field to specify one pet to many grid squares, or essentially one pet to one whole grid of user created forbidden zones. Another model that will be created is the Movement model, which stores the position of the pet and the time that the pet was in this position. By using a ManyToOne field, we can store many multiple movement data points for one pet, which can be later compiled into a heat map on the front end. Lastly, we will add a Image model, which stores one pet image, so that we can store multiple images of the pet for pet classification. To connect the front end and the back end, a library called Axios will be used to call GET and POST requests to send data between the front end and the back end. As well, a toolkit called Django REST Framework will be used in between the Axios calls and the back end that will make storing data into the MySQL database and sending data from the MySQL database much easier. This web application will be deployed using Apache and Amazon EC2.

6.2 Pet Classification

The CNN for classification is a pretrained InceptionV4 dog and cat breed classifier with additional layers at the end for classifying the user's animal(s). This takes advantage of transfer learning, a machine learning technique in which a pretrained neural network designed for another problem either sends its outputs to another neural network or has additional computational layers added to solve a similar problem. In this case, since we are working with the classification of specific pets, a dog and cat breed classifier is a good starting point to then solve the issue of identifying different pets, which may be different dogs or cats, that can be distinguished by breed first then differentiated further with more minute features. This secondary level requires additional training, which is why we require the user to upload images of their pet(s). Training of the CNN will occur on an external Amazon server, communicating with our SQL database through our Django backend to retrieve the user uploaded images alongside a predetermined training and validation set. This to remove the computationally intensive act of training the remaining layers of the CNN from the Raspberry Pi, as these computations may take exceedingly long on the hardware. After the CNN is fully trained it will be sent to the Raspberry Pi as a file for the Raspberry Pi to then utilize through OpenCV's built-in neural network modules. Once the trained CNN is on the Raspberry Pi, it will be called by the computer vision algorithm in order to validate the presence of an animal in a given frame. The CNN might be sent a validation request on a given image box if there is motion detected in, near, or entering a forbidden zone, or simply to re-validate detected motion as an animal. In either case, the CNN will receive an image box and run it's classification on the image to determine if there is an animal in the frame. This will involve running through the full CNN, with the final outputs of the additional transfer learning layers determining whether or not an animal is present in the given image box. If the CNN was sent this image as a validation request

Table 1: Hardware Cost Comparison

Device	Pre-Pandemic	Current
Raspberry Pi 4	$35	$45
Jetson Nano	$59	$150

for a user notification (as in motion detection found something inside, near, or moving into a forbidden zone) then the notification's validity is verified by the CNN's output, either triggering a user notification in the case of a correct classification of an animal in its forbidden zone, or the notification is cancelled due to no animal being detected or the incorrect animal being detected.

6.3 Computer Vision

The overall flow of information between the CV and ML components is illustrated in Figure 2. In order to detect when an animal first comes into view, the computer vision will rely on pixel differences between frames as a form of motion detection. If enough pixels differ to meet a threshold size that we set, then we will assume that something has entered the frame and get a bounding box that encompasses all the differing pixels. We will add a a fairly high margin to this box to ensure that we have as much of the animal as possible in the image. This subset of the image will be sent to the CNN along with a request to identify it. The CNN will produce an identification along with a tighter bounding box, which will be sent to the second portion of the CV algorithm which will use a built-in OpenCV tracking algorithm to track the animal. The ID will also be associated with the position of the tracked object in one data structure.

As the animal moves, the tracking algorithm will continually update its position. We expect that over time the tracker will decrease in accuracy (or we will lose sight of the animal), so the tracker's position will be cross referenced against the location(s) reported by pixel differences. If a significant mismatch is detected, we will request reidentification from the CNN to reset the bounding box we're tracking. The bounding box we're tracking will also continually be checked for collision with any of the designated forbidden areas. If a significant overlap is detected, we will again send a request to the CNN to double check the identity and verify that we should send a notification to the user.

6.4 Overall System

Figure 4 at the end of this document shows our overall system block diagram, as well as where each part is coming from. To integrate all the components of the system, we will be using Axios to send and receive data as both the web application and the Raspberry Pi will be ran on individual Apache servers, which means that we can use Axios to create GET and POST requests to send data between the two servers and be able to send data between the web application, computer vision, and machine learning.

7 TEST & VALIDATION

In general, all off the vision and ML related testing will be done in one (or more) of three ways: with still images, with stuffed animals, and with a real live animal. All tests described will start with still images unless otherwise specified. Some of the vision related tests may also move onto stuffed animals to test how the tracking handles an animal turning around. Regarding live animals, we hope to get to this as the very last stage of our integration testing. We would aim to set our project up in Rebecca's Apartment by the end of the semester to measure its performance in her one-cat household.

7.1 Classification Speed and Accuracy

Testing of the CNN will be performed on both a laptop and Raspberry Pi. Classification accuracy will be done in stages, starting with a CNN trained on one animal, then two distinct animals, then two animals of similar appearance. Training of each phase will be done in trials with varying numbers of additionally uploaded images, those being 3 images, 5 images, and 10 images for each of the given animals. CNN will then be run against a validation set composed of 10 images of each trained animal and 10 images of other animals (distinct and non-distinct), as well as 5 non-animal images. In the case of correct classifications we want a confidence value of 90% or higher outputted by the CNN that the given image matches the desired animal, with other animals outputting confidence values less than this. When run against the validation data, we want the CNN to have an overall accuracy of 90% or higher in classifying all the given images. For speed, we want this classification to occur in less than 5 seconds.

7.2 Computer Vision Speed and Accuracy

In order to test the speed at which an animal can be detected moving into the frame, we will gradually move an object (image or stuffed animal) into the frame and time how long it takes for the pixel difference threshold to trigger - averaged across several trials. We will do this at various speeds (although it will likely be done by hand, so not completely precise). The threshold for detecting motion will be fine-tuned to hopefully not pick up on small glitches, but be able to detect a reasonably slow moving animal entering the frame within our allotted 5 seconds.

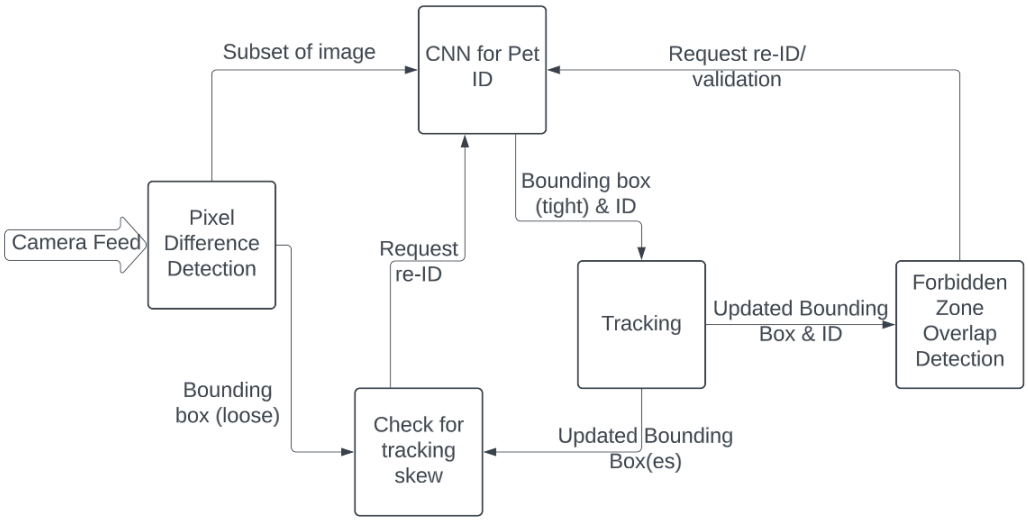

Figure 2: Computer vision flow of information

Testing the speed at which the forbidden zone overlap is detected will work similarly. We will manually move the image or stuffed animal to overlap an arbitrarily designated forbidden zone, and time how long it takes for the program to flag this. For ease of measurement, we will film the event (possibly in slow motion, if needed), and we will define "overlapping the forbidden zone" as $> 50\%$ overlap. This test will also be performed at a variety of movement speeds to ensure that it is robust to a pet moving particularly fast or slow, within reason.

To test the vision accuracy, as we move an image or later stuffed animal around the space we will display on the video footage where we believe its position is centered, as well as a circle around that point with a radius tuned to be roughly 1ft under those camera conditions (i.e. how far away the camera is). This will be monitored (and again, likely recorded to allow for closer review), to confirm that the object we're tracking remains within that radius for the full duration of the test. Tests will be performed at a variety of different speeds (within reason), and at least one test will be performed for a prolonged duration of at least 5 minutes to make sure that error doesn't appear to accumulate. We will also make sure that the tracking doesn't falter under a condition where the animal has stopped moving.

7.3 Validating User Experience on Web Application

We plan to bring 10 participants, which we hope to be a diverse group of participants in terms of age and experience with web applications, and let them use the web application to see if they are able to complete the core tasks, such as creating forbidden zones or request for activity logs, of the website. Though we are aiming for 95% of user to complete these tasks, we will be looking for 9 out of 10 participants to fully complete the tasks as achieving our goal as the only way to achieve the use-case requirement is with a 100% completion rate, which is unrealistic when generalizing to a bigger population size.

7.4 Validating System Use-Case Requirements

To understand the speed at which users can setup the whole system, we will use the same 10 participants from the user experience test and see on average if the 10 users can do the setup within less than 5 minutes. To test that the whole system will cost under $100, we will add the costs of significant components such as the Raspberry Pi and the camera used and calculate if the sum is under $100.

8 PROJECT MANAGEMENT

8.1 Schedule

Our schedule is below as shown in Fig. 5. In general, we will be trying to finish our individual components of the project and start working on integrating the components by the interim demo. By the final demo, we hope to have fully integrated the components and done thorough testing to see if the system meets the design and use-case requirements.

8.2 Team Member Responsibilities

In general, Max will be working on the machine learning component of the project, specifically using CNN for pet identification. Rebecca will be responsible for setting up the Raspberry Pi and working on the Computer Vision component, such as being able to detect movement from each pet in a room. Brandon will be in charge of setting up the front end and back end of the web application and

Table 2: Bill of Materials and Budget

Description	Model	Manufacturer	Project Cost	User Cost
Raspberry Pi 4	4GB	Raspberry Pi Foundantion	$0	$55
RPi Camera	3	Raspberry Pi Foundantion	$25	$25
Domain Name	N/A	GoDaddy	$15	$0
Speaker	BT107	LEZII	$10	$10
			$50.00	$90.00

will help set up running a server on the Raspberry Pi. All members will help with integrating and testing the components.

8.3 Bill of Materials and Budget

Shown in Table 2 is our bill of materials and budget. We have labelled each cost as a project cost (which would come out of our $600 budget) and/or a user cost, which would contribute towards our hypothetical retail price. The RPi will be from the ECE500 inventory, and so has a project cost of $0. As shown, we anticipate that our overall use of budget will be $50, and the overall user-facing cost of our system will be roughly $90.

8.4 Risk Mitigation Plans

The main risk of the project is that no one in the team has worked with a Raspberry Pi, which is hosting many of the operations done in this project, so we are unsure if all our ideas will work out on the Raspberry Pi. But, we have done extensive research on how computer vision and machine learning will work on the Raspberry Pi, and we have ordered components related to the Raspberry Pi early so that we can test as soon as possible. As we have learned that initial phrase of the machine learning for pet classification could potentially be slow on the Raspberry Pi, We have moved this training phase to the web application from the Raspberry Pi, which should reduce the workload on the Raspberry Pi and increase the speed of this training phrase. If the Raspberry Pi does not work out for our project, we plan to use a Jetson, which has a GPU and can handle the workload of the tasks we would want to do on the hardware. But, this would force us to increase the user cost in our use case requirement as the Jetson's cost is significantly higher than the Raspberry Pi's cost.

9 RELATED WORK

There are currently many pet monitors on the market. One most similar to ours is this Nest Cam made by Google. It has a cost of $99.99 and advertises a similar alert system, object identification, and data security. Specifically, it's identification can differentiate between a human, animal, or vehicle.

A cheaper existing option is the Wyze Cam v3 Pet Camera which simply sends users notifications whenever motion or sound is detected and allows users to talk to animals through a microphone or play a sound through a speaker. The detection options on this one are much less than those of the Google option, but it is also only $35.98, which is almost a third of the price of the Google option.

In terms of classification projects, there are too many to count online. One helpful example we found was object and animal classification using a Raspberry Pi and OpenCV here. This outlines a project for implementing a CNN on a Raspberry Pi, how to setup the Raspberry Pi, and what the OpenCV modules they are using can do. A more specific classification example is this dog and cat breed classifier that uses a pre-trained InceptionV4 architecture with additional transfer learning layers to create the classifier. They also include that they used the Cat and Dog Breed Classification Oxford dataset, which we will also be using as a starting point for our training and validation sets of our classifier.

10 SUMMARY

The intention of PetSTAR is to benefit pet owners with tasks that make taking care of their pets easier without negating any of the current features that a traditional camera, which is the current way pet owners watch their pets, offers. The system contains three major components, specifically a web application, machine learning, and computer vision, that interact together to support the purpose of helping pet owners take care of their pets. The biggest challenge is ensuring that a Raspberry Pi does everything it needs to do in terms of running the computer vision and machine learning that we would like it to do. As well, the time frame to finish the project is short, so we hope to be able to implement and test everything before the final demo.

Glossary of Acronyms

- CV - Computer Vision
- ML - Machine Learning
- RPi – Raspberry Pi

References

[1] Avicenna. "Cats and Dogs Breeds Classification Oxford Dataset." Kaggle, 12 Mar. 2019, https://www.kaggle.com/datasets/zippyz/cats-and-dogs-breeds-classification-oxford-dataset.

[2] Bhatia, Richa. "Why Convolutional Neural Networks Are the Go-to Models in Deep Learning." Analytics India Magazine, 15 Feb. 2021, https://analyticsindiamag.com/why-convolutional-neural-networks-are-the-go-to-models-in-deep-learning/.

[3] "Getting Started." Getting Started — Axios Docs, https://axios-http.com/docs/intro.

[4] Irabor, Jordan. "How to Build A to-Do Application Using Django and React." DigitalOcean, DigitalOcean, 17 Feb. 2021, https://www.digitalocean.com/community/tutorials/build-a-to-do-application-using-django-and-react.

[5] Mallick, Satya. "Object Tracking Using Opencv (c++/Python)." LearnOpenCV, 11 Nov. 2022, https://learnopencv.com/object-tracking-using-opencv-cpp-python/.

[6] Mwiti, Derrick. "Transfer Learning Guide: A Practical Tutorial with Examples for Images and Text in Keras." Neptune.ai, 20 Feb. 2023, https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras.

[7] O'Connor, Ryan. "Pytorch vs Tensorflow in 2023." News, Tutorials, AI Research, News, Tutorials, AI Research, 12 Feb. 2023, https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/.

[8] "OpenCV Modules." OpenCV, https://docs.opencv.org4.xindex.html.

[9] "Papers with Code - Inception-V4 Explained." Explained — Papers With Code, https://paperswithcode.com/method/inception-v4

[10] "Pytorch vs. Tensorflow: Which Framework Is Best for Your Deep Learning Project?" Built In, https://builtin.comdata-sciencepytorch-vs-tensorflow.

[11] Team, Keras. "Keras Documentation: Transfer Learning & Fine-Tuning." Keras, https://keras.ioguidestransfer_learning.

[12] Terra, John. "Pytorch vs Tensorflow vs Keras: Here Are the Difference You Should Know." Simplilearn.com, Simplilearn, 3 Feb. 2023, https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article

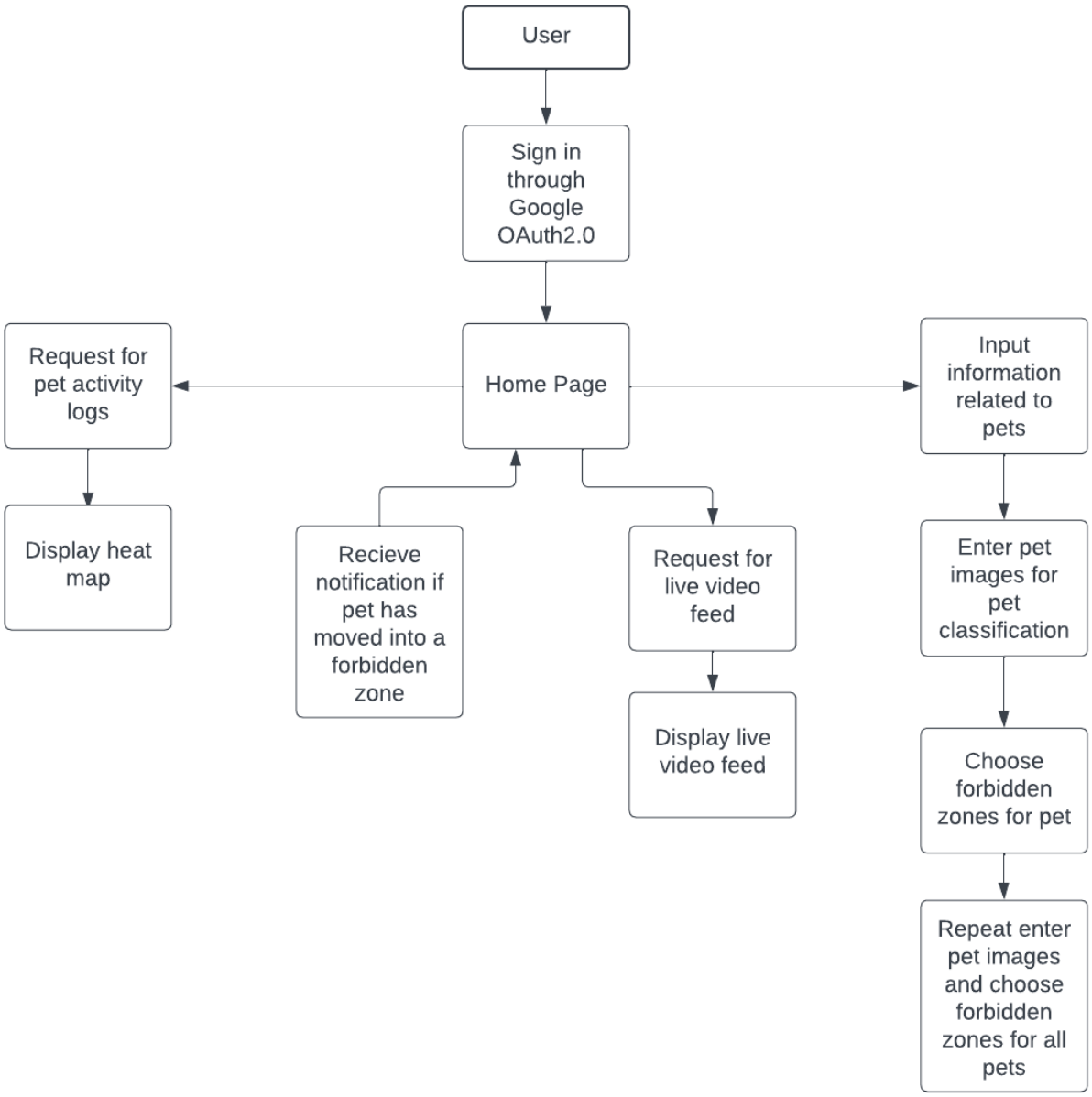

Figure 3: User interaction with the front-end web application

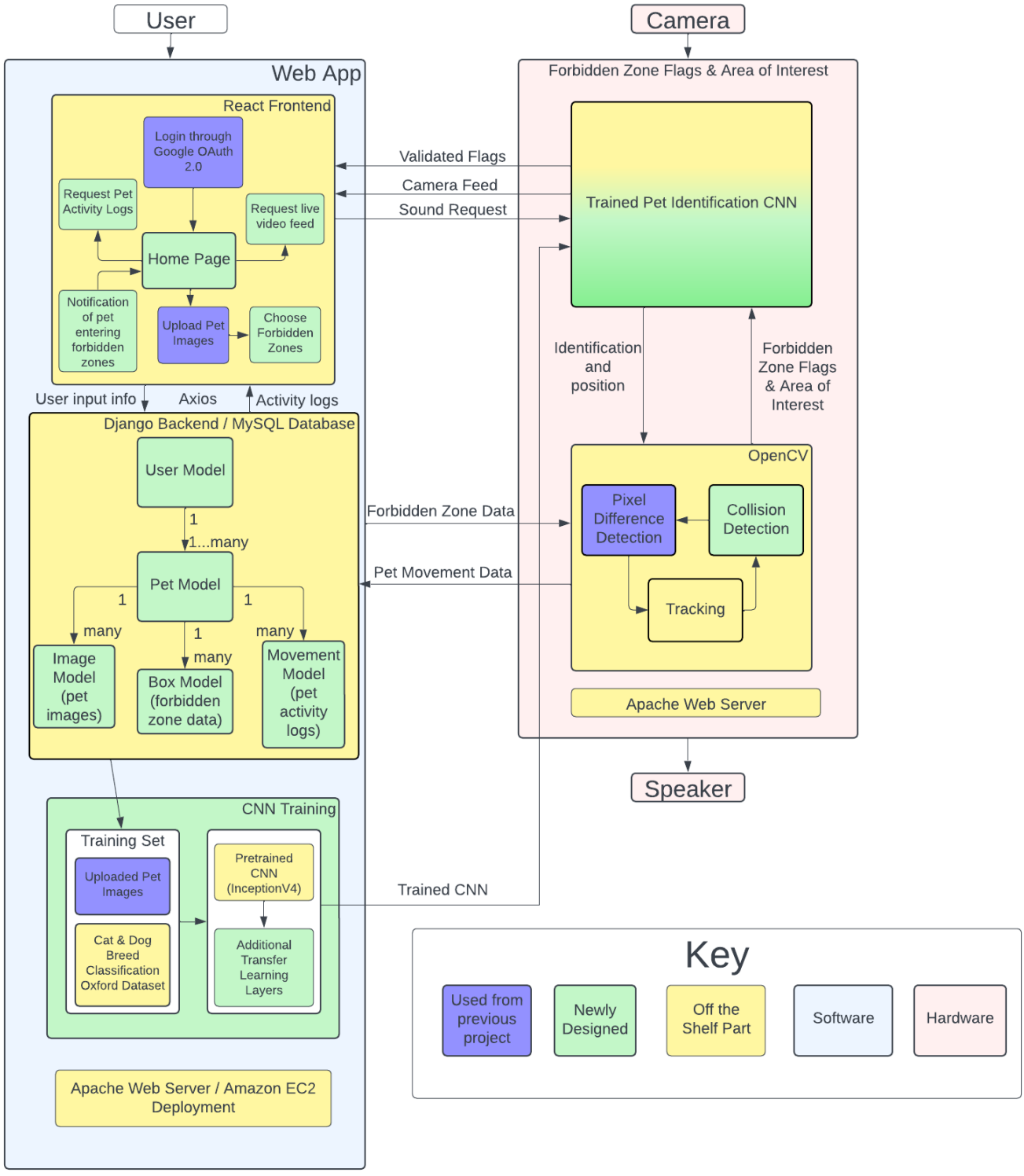

Figure 4: Full Block Diagram of PetSTAR

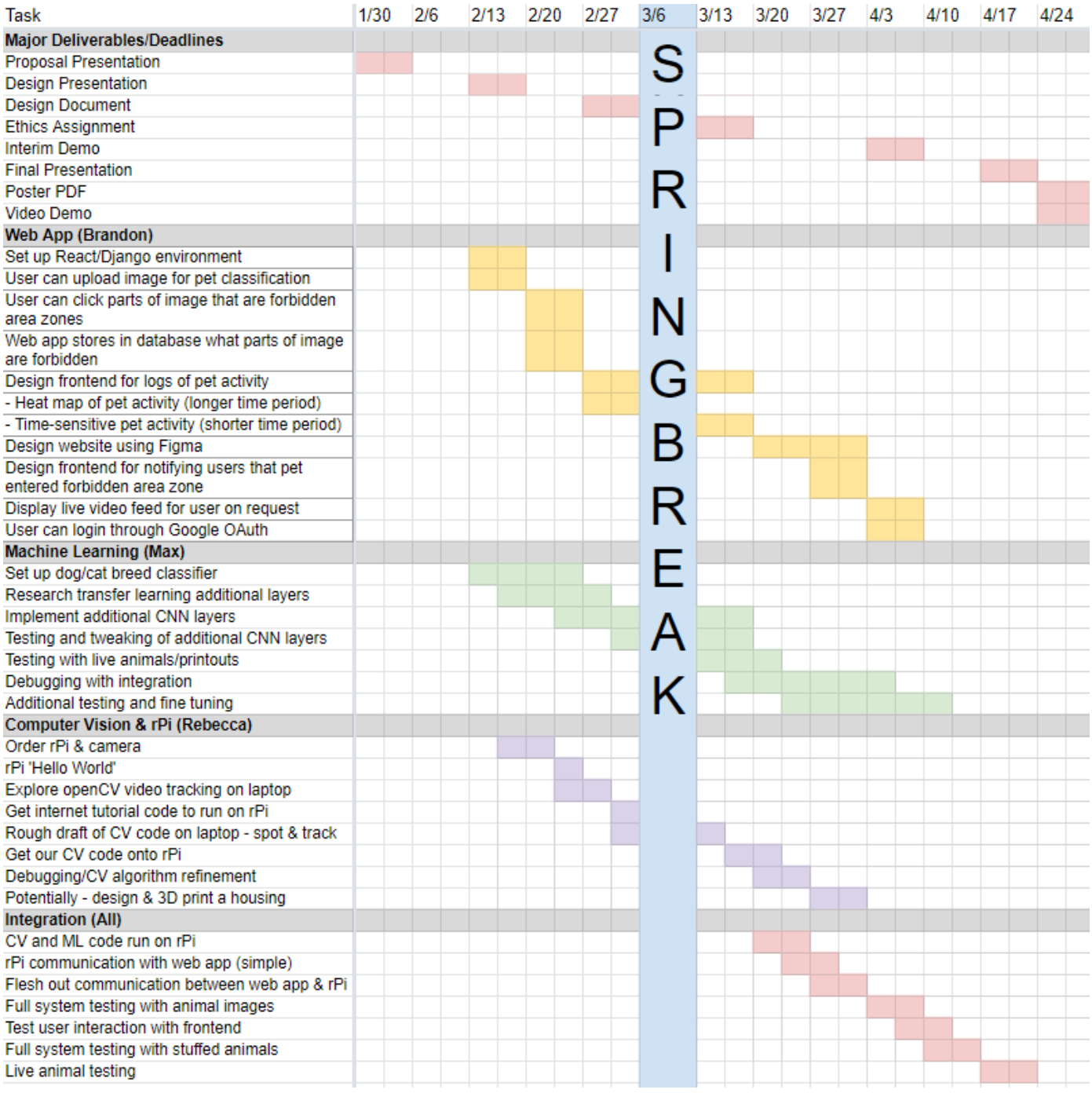

Figure 5: Gantt Chart