# Smart Cat Bouncer: Cascading Machine Learning Models for Prey Detection

Mirela Dubravac
dubravac.mirela@hotmail.com

June 2024

**Abstract**

This project addresses the problem of domestic cats bringing prey into homes. We developed a smart cat bouncer system based on four cascading convolutional neural networks implemented on a Raspberry Pi, which is connected to a camera installed at the cat door with a locking mechanism. The first model, adopted from EfficientNet Lite, detects cats and was used to gather tens of thousands of cat images for training subsequent models. The second model, Cat Approach, identifies if a detected cat is approaching. If approaching, the third model, Cat Classification, identifies the cat (Hali, Rex, Simba). For Simba, the fourth model, Prey Detection, determines if prey is present. This final model is not yet implemented due to limited training data, but the other models are operational. Performance metrics from the test dataset indicated satisfactory performance. However, there remains room for improvement in accuracy and recall rates. To pinpoint areas for improvement, we analyzed the models' learning patterns and their focus within images using visual explanation techniques. Heatmaps revealed that the models accurately identified key features like cat ears and fur textures. However, they also overly focused on the background, confusing cloud structures with fur, which sometimes led to misclassifications. We propose several solutions to this problem. The most promising is to crop the images after the first Cat Detection model, ensuring subsequent models focus more on the cat, thereby reducing the impact of varying backgrounds on inference. Further potential improvements are discussed considering the limited computing power of the Raspberry Pi. Future work includes enhancing data quality, addressing dataset imbalances, and improving model architectures to ensure reliable prey detection. This project showcases the potential of artificial intelligence to address everyday challenges in pet management.

# 1 Introduction

Imagine coming home to find your beloved pet cat proudly presenting you with its latest catch—a half-dead mouse. While the cat sees this as a gift, most humans are less than thrilled with these offerings. There are several approaches to tackle this problem. One straightforward approach is to keep the cat door locked, preventing the cat from going in and out. However, this can be impractical as some cats are notably persistent and vocal in their demands for freedom to move around as they please. Another approach would be to have a smart system deciding when to unlock the cat door. Recently, there has been a surge of start-ups [1, 2, 3, 4] developing exactly this: a cat door with an integrated artificial intelligence system that detects whether the cat carries prey or not. This technology promises to offer a more elegant solution by using advanced image recognition and machine learning algorithms to analyze the cat's entrance behavior in real-time. Despite these promising developments, these start-ups are still in the early stages, and fully functional products are not yet easily available on the market. Thus, we decided to build our own smart cat bouncer.

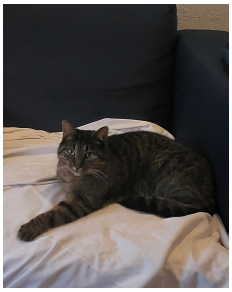
The cat bouncer set-up comprises several components:

- Cat flap door with a locking mechanism
- Camera with IR LED boards (OV5647-IR)
- Raspberry-Pi 3 model b (a small computer running Linux)
- 3D printer (Crealty ender-3 s1 pro)
- File server (Synology NAS)
- Telegram chat bot
- Computer for model training (192 GB DDR5 RAM, RTX4090 GPU, i9 CPU)
- Machine learning system for image classification

Figure 1 presents the primary suspects in our investigation. Hali, the cutest (a), is mostly inside. When he goes outside, he often sits in front of the camera. Simba, the adventurous explorer (b), spends most of his time in the forest hunting. We believe he brings home the most mice, making him our top suspect. Rex, the resilient survivor (c), is also mostly outside. He avoids using the flap door, so we know less about his behavior. However, we are cautious about underfeeding him because he might go into survival mode and catch mice and birds to eat. Initially, these were merely accusations, but now we have the footage saved by the camera on the file server as proof.

(a) Hali

(b) Simba



(c) Rex

Figure 1: The primary suspects

Figure 2 shows the hardware setup at the cat flap door. On the right you can see the camera with IR LED boards. Night vision is crucial for our task as cats typically hunt at night. Beside the camera is the Raspberry Pi, which runs machine learning models on each frame. The processed images are saved with a timestamp on the file server in different folders based on the classification outcome. Additionally, selected images are sent to a Telegram chat, allowing us to monitor what comes through the door and verify the model's accuracy. This proved especially useful when Hali once destroyed the setup (the footage proofed him guilty). Without the Telegram chat, we would have lost more time collecting data, since we check it more frequently than the file server. We quickly replaced the destroyed camera mount by 3D printing a new one. This setup ensures minimal downtime in data collection when issues arise.

Figure 2: Cat Door with Camera and Raspberry Pi

After collecting tens of thousands of images, we began labeling them and training the models. We opted to cascade different image classification models for three main practical reasons: First, the Raspberry Pi has limited computing power, requiring the model to be lightweight. A single model capable of detecting cats bringing in prey would be too large, whereas the Raspberry Pi can manage several smaller models more effectively. Second, the data is highly unbalanced, which complicates model training. While there are methods to address unbalanced datasets, they remain impractical in this context. Third, we have three cats with distinct behaviors and needs. Cascading several image classification models increases flexibility and allows us to tailor the system to each cat. This setup also enables us to easily replace models if necessary. Figure 3 illustrates the model cascade.

We have two GitHub repositories: Bouncer, which details the deployment process, including cascading models and shell scripts for the Raspberry Pi, and CatWatcher, which focuses on data labeling, model training and includes accompanying notebooks. The CatWatcher repository is thus central to this report, as it focuses on the machine learning aspects of the project.

In the following sections, I will describe the data collection and preparation process, followed by an overview of the architecture of our three models and their performance metrics. These metrics will be discussed in the context of their usefulness in our setup. I will conclude the report by outlining potential improvements and features we plan to implement in the future.

## 2 Data

Data was collected between November 2023 and June 2024. Figure 4 shows a variety of snapshots. The images saved on the file server are 480x640 pixels. Using the original size was not feasible, as the resulting models were too large for the Raspberry Pi, with over 100 million parameters and around half a gigabyte for the tflite files. Therefore, we reduced the image size to 180x180 pixels, which still provides an acceptable resolution. For an illustration of different image sizes, please refer to imagesizing.ipynb.

The first model in the cascade detects cats. For this task, we chose the TF EfficientNet Lite model [5], because it is lightweight, efficient and optimized for speed. It is used for general image classification tasks. Testing this model on Simba produced the classification shown in Figure 5. The model returned "hamster" with the highest probability (0.48), followed by "Siamese cat"

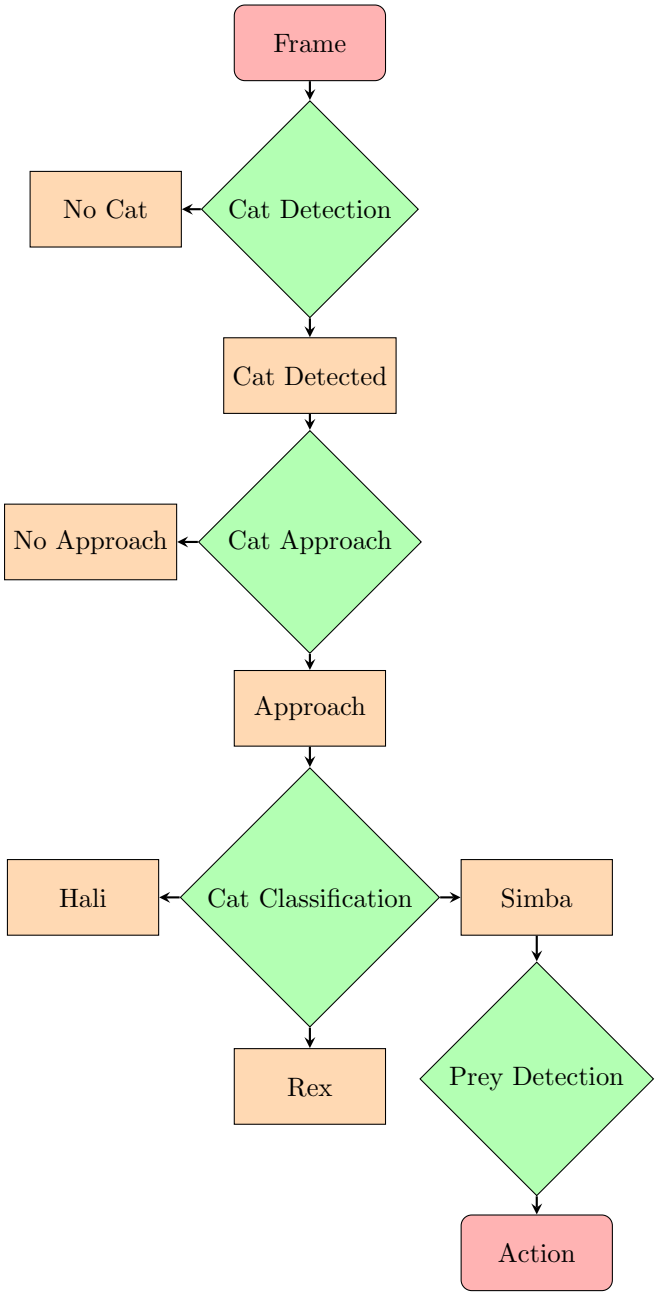



Figure 3: Flowchart of the cascading models

(0.12) and "Egyptian cat" (0.08). Needless to say, Simba was not amused when the model classified him as a hamster.

At this first stage, our aim was to get a near-perfect hit rate for detecting cats. Thus, we included many cat-like keywords [1] that triggered the Cat Detection model to save each frame on the file server. The saved frames served as training data for the next model in the cascade.

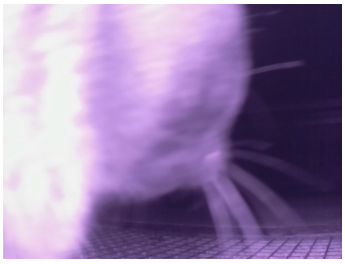
After collecting 3'101 images, we labeled the data as either containing a cat mouth or not and trained a first model.[2] After deploying this model on the Raspberry Pi, we realized that these

[1] e.g., "cat", "kitten", "hamster", "chihuahua", "dog", "puppy", "rabbit", "bunny", "ferret", "guinea pig", "small dog", "fluffy", "stuffed animal", "fur coat", "wool", "lynx", "leopard cub", "tiger cub", "lion cub", "wildcat", "domestic animal", "pet", "faux fur"

[2] We also tried the haarcascade frontalcatface model by OpenCV. Unfortunately, that did not work well in our setup.

(a) Hali approaching

(b) Hali leaving



(c) Hali's mouth

(d) Rex approaching

(e) Rex as Lama

(f) Simba approaching

(g) Simba's legs

(h) Simba sitting

(i) Fur

(j) Simba with prey

(k) Simba with prey at night

(l) Rex with prey

Figure 4: Variety of Snapshots

labels were not useful, as we obtained positive images that were irrelevant to our goal of preventing cats from bringing prey into the house. For example, consider Figure 4c. This image shows Hali peacefully sitting in front of the door, which is not a threat, yet it would still be considered a positive instance of the cat mouth. For our purposes, we needed a classification that more directly detects potential threats (i.e., cats aiming to enter the house). Thus, we continued collecting data with the aim of relabeling images as either approaching or not approaching cats [6]. In this context, 4c would not be considered a positive instance and would thus not be analyzed further.

For the labeling, we used the PsychoPy toolbox, usually used for programming psychological experiments. For this task, we created what is probably the world's simplest "experiment," where an image is displayed, and a key press saves the keystroke along with the image name in a CSV file. This script iterated through all 61'345 images we had collected at that time.

Sometimes the cats would sit very still in front of the camera for extended periods, resulting in many (nearly) identical images. Thus, we cleaned the dataset by removing duplicates and near


Figure 5: Classification by EfficientNet Lite

duplicates. This reduced the dataset to 21'306 images (14'450 of class "no approach" and 6'856 of class "approach") used for training the Approach Detection model. The images classified by this model as approaching cats were used for training the next model.

The initial plan was to implement prey detection next. However, after reviewing the collected data, we reconsidered this plan due to an insufficient number of training images. As suspected, we observed that prey images were almost exclusively of Simba (186 images). Rex was captured only on two occasions, resulting in 26 images of Rex with prey in his mouth. Hali was never captured bringing prey. This imbalance led us to implement an additional model before the final Prey Detection model; a Simba detection - or more generally, a Cat Classification model. For training this model, we used 1'220 images of Hali, 820 of Rex, and 1'500 of Simba.

We initially thought classifying our three cats would be easy due to their distinct appearances (cf. Fig.1). However, the model struggled to extract distinguishing features, necessitating a larger model with many parameters. This led to significant lags during inference, as the Raspberry Pi was unable to manage the model size. Consequently, we reduced the model size (by reducing image size from 480x480 to 180x180 pixels) without sacrificing too much performance.

These challenges and technical limitations delayed the final stage, which remains less tested than earlier stages and still has room for improvement. The fourth model assesses whether Simba is approaching with prey or not. For this, 212 prey images were matched with 121[3] no-prey images that were similar except for the presence of prey. This created a dataset of 333 images. Ongoing work will allow for retraining this model as more prey images are collected.

### 2.1 Interim Summary

Three of the four cascading models were trained from scratch using our own data. For the Approach Detection model, we used 21'306 images, with 14'450 labeled as "no approach" and 6'856 as "approach." The Cat Classification model was trained with 1'220 images of Hali, 820 of Rex, and 1'500 of Simba. For the Prey Detection model, data limitations were noted, 212 prey images and 121 matched no-prey images. Ongoing data collection will allow for retraining and continuous model improvements. Test-validation dataset splits were 80:20.

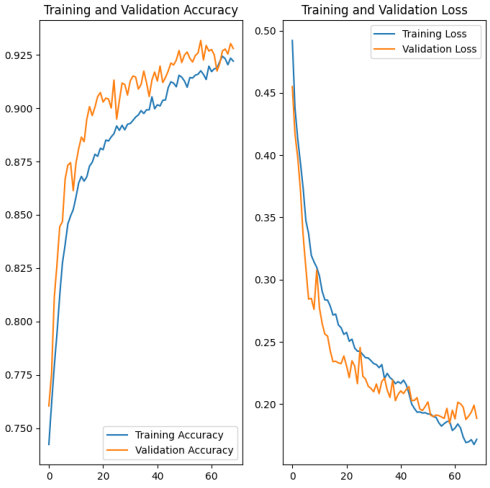
## 3 Model Architectures

Our current models are the result of iterative improvements based on continuous monitoring of deployed models. Specifically, we observed the file server where the Raspberry Pi saves images in different folders according to the classification outcomes of various models. Reviewing these folders provided insights into performance in the deployment environment. Whenever the categorization accuracy was underwhelming, we modified and retrained the model. This section describes the convolutional neural network (CNN) architecture that have yielded the best results to date (cf. Table 1 and Figure 6).

[3]Yes, this should have been 212. This error is an example of dysnumeria, a condition characterized by difficulties in recognizing and processing numbers accurately. It will be corrected in the next training iteration.

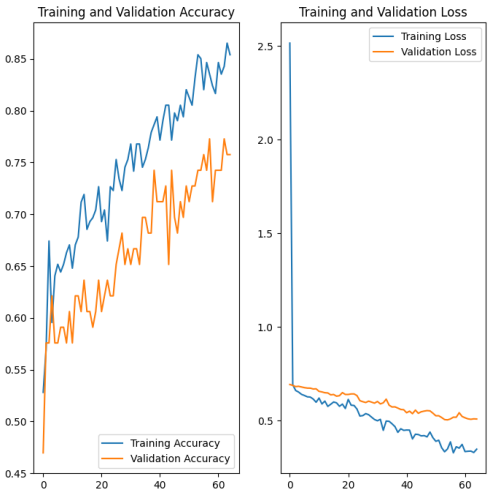
| Layer Type | Output Shape | Parameter # |
|---|---|---|
| **Cat Approach Model (3'988'769 Parameters)** | | |
| Sequential | (None, 180, 180, 3) | 0 |
| Rescaling | (None, 180, 180, 3) | 0 |
| Conv2D | (None, 180, 180, 16) | 448 |
| MaxPooling2D | (None, 90, 90, 16) | 0 |
| Dropout | (None, 90, 90, 16) | 0 |
| Conv2D | (None, 90, 90, 32) | 4640 |
| MaxPooling2D | (None, 45, 45, 32) | 0 |
| Dropout | (None, 45, 45, 32) | 0 |
| Conv2D | (None, 45, 45, 64) | 18496 |
| MaxPooling2D | (None, 22, 22, 64) | 0 |
| Dropout | (None, 22, 22, 64) | 0 |
| Flatten | (None, 30976) | 0 |
| Dense | (None, 128) | 3965056 |
| Dropout | (None, 128) | 0 |
| Dense | (None, 1) | 129 |
| **Cat Classification Model (16'250'179 Parameters)** | | |
| Sequential | (None, 180, 180, 3) | 0 |
| Rescaling | (None, 180, 180, 3) | 0 |
| Conv2D | (None, 180, 180, 32) | 896 |
| MaxPooling2D | (None, 90, 90, 32) | 0 |
| Dropout | (None, 90, 90, 32) | 0 |
| Conv2D | (None, 90, 90, 64) | 18496 |
| MaxPooling2D | (None, 45, 45, 64) | 0 |
| Dropout | (None, 45, 45, 64) | 0 |
| Conv2D | (None, 45, 45, 128) | 73856 |
| MaxPooling2D | (None, 22, 22, 128) | 0 |
| Dropout | (None, 22, 22, 128) | 0 |
| Conv2D | (None, 22, 22, 256) | 295168 |
| MaxPooling2D | (None, 11, 11, 256) | 0 |
| Dropout | (None, 11, 11, 256) | 0 |
| Flatten | (None, 30976) | 0 |
| Dense | (None, 512) | 15860224 |
| Dropout | (None, 512) | 0 |
| Dense | (None, 3) | 1539 |
| **Prey Detection Model (29'515'041 Parameters)** | | |
| Sequential | (None, 480, 480, 3) | 0 |
| Rescaling | (None, 480, 480, 3) | 0 |
| Conv2D | (None, 480, 480, 16) | 448 |
| MaxPooling2D | (None, 240, 240, 16) | 0 |
| Dropout | (None, 240, 240, 16) | 0 |
| Conv2D | (None, 240, 240, 32) | 4640 |
| MaxPooling2D | (None, 120, 120, 32) | 0 |
| Dropout | (None, 120, 120, 32) | 0 |
| Conv2D | (None, 120, 120, 64) | 18496 |
| MaxPooling2D | (None, 60, 60, 64) | 0 |
| Dropout | (None, 60, 60, 64) | 0 |
| Flatten | (None, 230400) | 0 |
| Dense | (None, 128) | 29491328 |
| Dropout | (None, 128) | 0 |
| Dense | (None, 1) | 129 |

Table 1: Model Summaries




(a) Approach Detection Model


(b) Cat Classificaton Model




(c) Prey Detection Model

Figure 6: Model Training and Validation Metrics

To avoid overfitting, we incorporated dropout layers and data augmentation layers. Additionally, we experimented with batch normalization layers but ultimately decided against using them

because our deployment environment processes images individually rather than in batches. This setup reduces the computational efficiency and benefits of batch normalization during inference. Callback functions monitored the validation loss across epochs, stopping training early when the validation loss stagnated and reducing the learning rate to a minimum of 0.00001. The models were trained with a batch size of 32, defined using `tf.keras.utils.image_dataset_from_directory`. We used ReLu activation functions and Adam optimizer.

In binary classifications (Cat Approach and Prey Detection), we used a sigmoid activation function in the last dense layer and `tf.keras.losses.BinaryCrossentropy(from_logits=False)` as the loss function. For Cat Classification with three classes, we used no activation function in the last dense layer and `tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)` as the loss function. For a comprehensive overview of different model configurations and their impact on training and validation metrics, please refer to the respective notebooks (Cat Approach Model, Cat Classification Model, and Prey Detection Model).

# 4 Results and Discussion

We compared different models using images collected after model training. Since test datasets are currently available only for the Approach Detection and Cat Classification models, results are reported solely for these models.

Figure 7 displays performance metrics (accuracy, precision, recall, and F1-Score) for two Approach Detection models. Model 1 represents an older model architecture without a data augmentation layer and with smaller input images (64x64). Model 2 represents the current architecture (cf. Table 1). The performance increase from Model 1 to Model 2 indicates that higher image resolution and increased image count improved the model. These enhancements resulted in larger tflite files, increasing from 5'111 KB to 15'586 KB, which remains within the capacity limits of the Raspberry Pi.

Figure 8 displays performance metrics for two Cat Classification models. Model 1 represents an older model architecture without data augmentation layer and with larger input images (480x480). Model 2 represents the current model architecture (cf. Table 1). The performance increase from Model 1 to Model 2 indicates that adding data augmentation layer is more beneficial than increased image resolution.

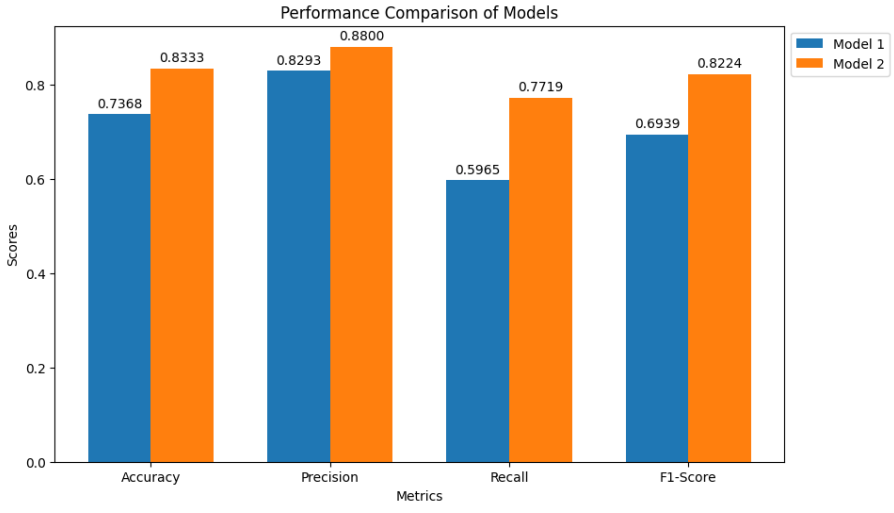



Figure 7: Cat Approach Model Comparison Results

Although the numbers do not appear too bad, the model's performance is unsatisfactory in practice. There are too many misses in the Approach model, as the cats may enter without any saved images in the 'approach' folder on the file server. This inconsistency, coupled with the low recall rate, indicates that approaching cats are not reliably identified. This issue may stem from

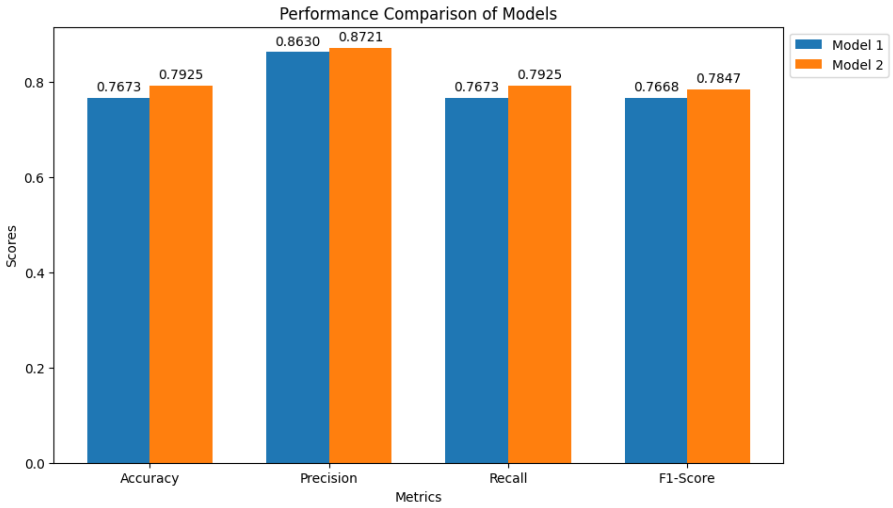



Figure 8: Cat Classification Model Comparison Results

the significantly lower number of training images in the "approach" class compared to the "no approach" class.

Similarly, the Cat Classification model is also unsatisfactory. Although the correct cat images predominate in the respective folder on the file server, there is considerable confusion among the cat images

The extent of misclassifications does not justify deploying the Prey Detection model at this stage. Before refining and deploying this model, we need to address issues with the previous models in the cascade. To better understand the classification errors, we decided to first explore what the models are learning.

To identify the features the models rely on for image classification, we investigated the local linear approximation of each model's behavior [7]. For the Approach model, we selected four images (Hali approach, Hali no approach, Rex approach, Simba approach) to illustrate the model's strengths and weaknesses. We plotted a heatmap for each image to show the areas from which the model gathers evidence for classification.

Figure 9 shows example images (left column) with the predicted approach scores (0 = no approach, 1 = approach) and the respective heatmaps (right column). Positive values (blue colors) indicate areas that the model considers as evidence for the "approach" class. Negative values (red colors) indicate areas that the model considers as evidence for the "no approach" class.

For the first image, the model did not classify it as an approach image (score = 0.35), although it clearly is. As shown on the right side, the model relied on the top right corner for its classification. The clouds in this image resemble the many fur images in the training dataset (see Figure 4i for an example). This similarity strongly indicated the "no approach" class, leading the model to misclassify the image despite correctly identifying the approaching cat, as indicated by the blue color.

The second image was classified as approaching, while Hali is clearly not approaching. It appears that the training dataset lacks sufficient examples of such images, leading the model to miss relevant features (e.g., the tail) necessary for accurate classification.

The third image shows Rex approaching, which was correctly classified. However, the clouds seem to confuse the model. The score for the "approach" class could have been much higher if the cloud structure had not been considered by the model.

The fourth image shows Simba approaching, which was correctly classified with extremely high confidence. Interestingly, the image has a mostly black background, likely contributing to the high confidence since the model was not confused by irrelevant background structures. As shown in the heatmap, the model focused primarily on the ears as strong indicators and the tail as a weaker indicator for the "approach" class.

This evaluation clearly shows that the background matters. One hypothesis is that the model

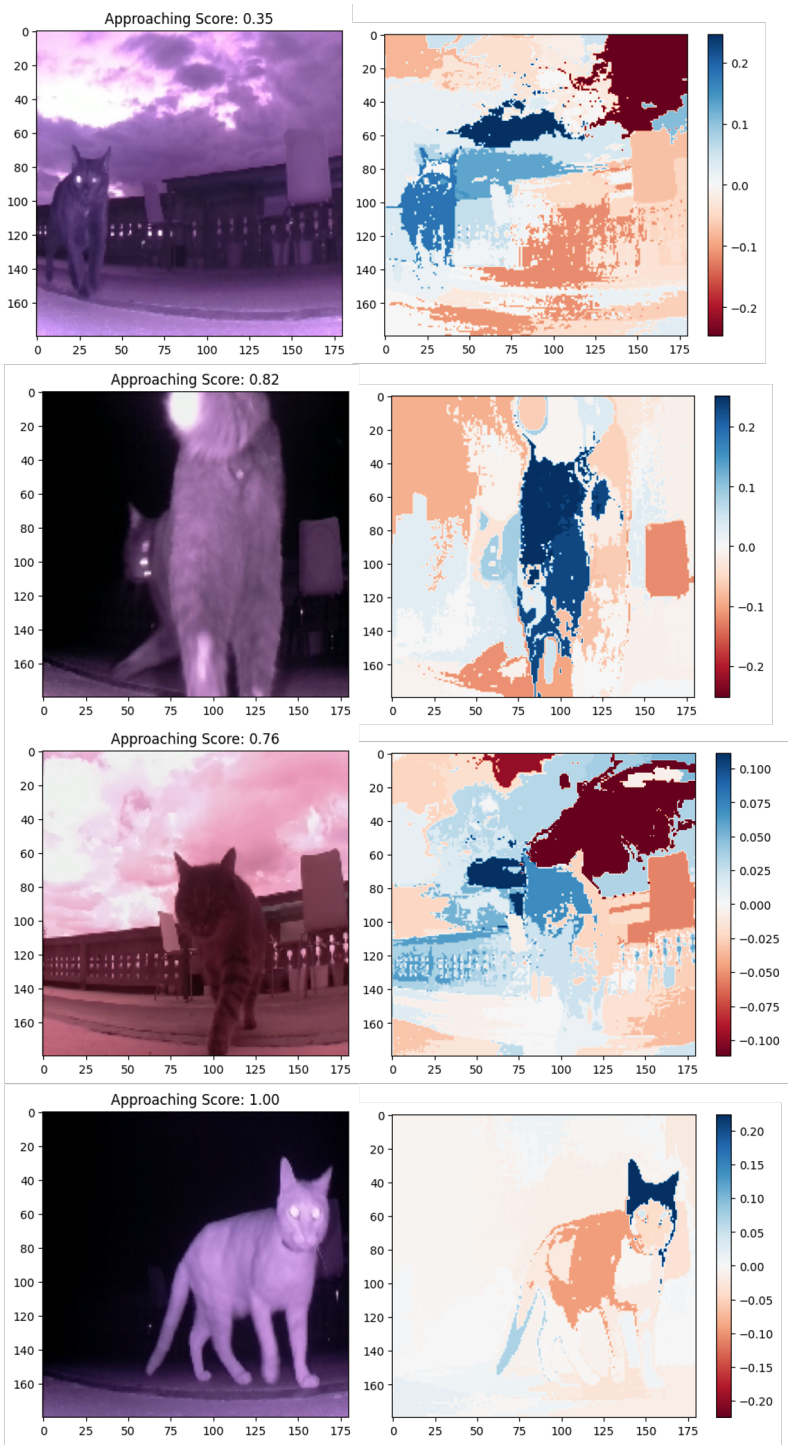



Figure 9: Explaining Cat Approach Model

performs better at night when the background is less prominent. This hypothesis has yet to be tested by comparing model performance on nighttime versus daytime images.

We conducted the same analysis with the Cat Classification model. Figure 10 shows example images of the three cats, along with the predicted class and a confidence indication. The heatmaps illustrate which areas the model considers as evidence for and against the classification. Although the model correctly classified these images, the heatmaps reveal that it focused on irrelevant aspects of the images. The model's satisfactory performance during validation suggests that the

data may be biased, possibly because the cats follow different schedules when they come home (e.g., daytime vs. nighttime).

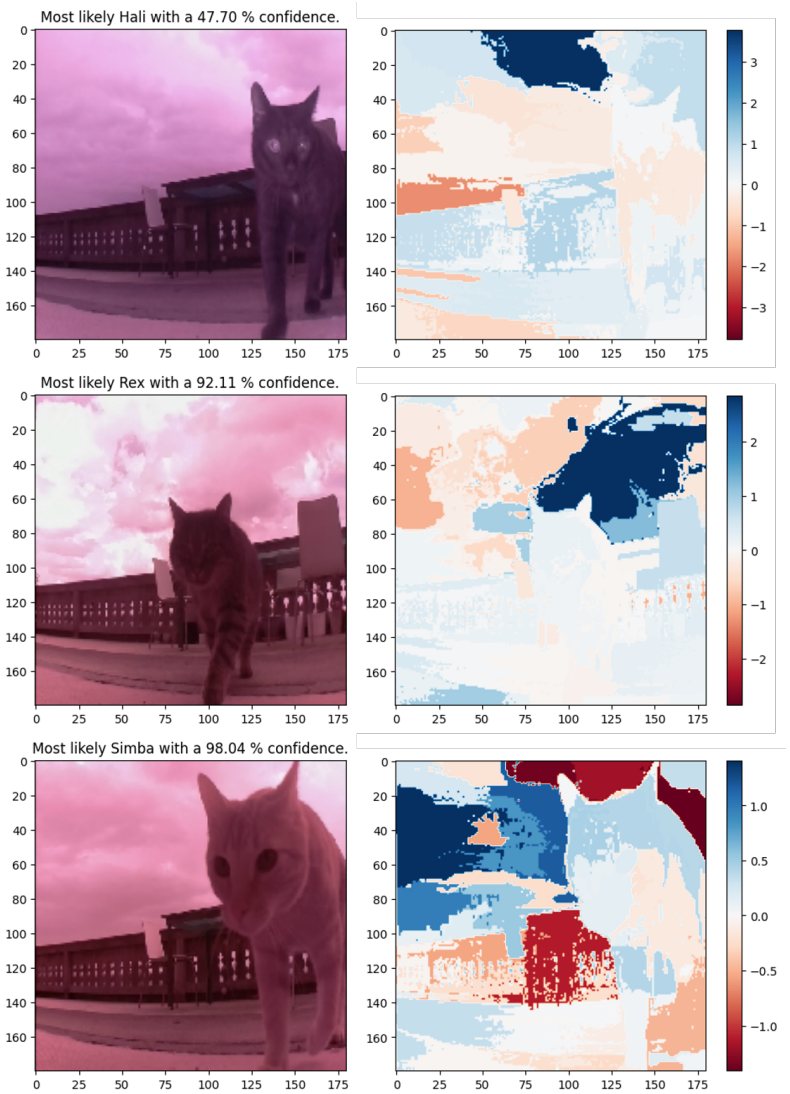



Figure 10: Explaining Cat Classification Model

## 4.1 Interim Summary

Evaluations of Approach Detection and Cat Classification models on a test dataset suggested satisfactory performance. However, significant improvements are anticipated with further fine-tuning of hyperparameters and enhancements to the datasets.

Heatmaps from explanatory analyses indicated that the models detect edges, contours, and textures. However, these features are not always relevant for correct classification, as the models often focused too much on the background.

Initially, we believed that with enough training data, the different backgrounds would cancel each other out, as we did not assume a correlation between the backgrounds and the classes. While a larger training dataset might mitigate this problem, it would require larger CNN models, which is challenging given the limited computing power of the Raspberry Pi. Therefore, we are considering other options to address this background issue.

One option is to draw bounding boxes around the cats using the first model (EfficientNet Lite's Cat Detection model) and crop the image before sending it further down the cascade. Another

option is to modify image depth, contrast, and similar features to make the background less prominent. Alternatively, we could install a box through which the cats must enter, ensuring that the background is always the same.

# 5 Conclusion and Outlook

We developed cascading CNN models to detect prey on a continuous camera stream processed on a Raspberry Pi. First, the EfficientNet Lite model evaluates each frame to determine if it contains a cat (broadly defined). This model performs well in practice and created a dataset for training the subsequent model. If a cat is detected, the Cat Approach model assesses whether the cat is approaching. This model performs also well and created the training dataset for the subsequent model. If an approaching cat is detected, the Cat Classification model identifies which cat it is (Hali, Rex, Simba). Inspecting the resulting dataset indicated that Simba brings most of the prey. Therefore, whenever the models detect Simba approaching, we plan to run the final Prey Detection model to determine if Simba is carrying prey. The Prey Detection model has not yet completed training due to a limited training dataset and, consequently, has not been implemented on the Raspberry Pi.

The next steps are to collect more training data, address imbalanced datasets (e.g., by implementing class weights to give more importance to the minority class), and enhance data quality by cropping images according to the bounding boxes indicated by the first Cat Detection model. This approach may help subsequent models in the cascade focus on relevant parts of the image. Another technique is to convert the images to grayscale to reduce input dimensions and model size, enabling the use of broader and deeper neural networks, which may further improve model accuracy. Additionally, we can further fine-tune hyperparameters such as the number of filters in the convolution layers, dropout rates, batch size, and learning rate.

Continuous model evaluations in the deployment environment is crucial to balance effectiveness and efficiency. By continuously monitoring and comparing models and adding new data, we can iteratively improve our prey detection application. This iterative process is fundamental to the successful implementation of machine learning in our cat bouncer system. However, despite our best efforts, some level of error will persist either due to model uncertainty or factors related to the deployment environment (e.g., the cats finding new ways to enter without triggering the model cascade). These inevitable errors highlight the importance of setting clear thresholds for acceptable performance metrics. We must carefully consider the trade-offs between sensitivity and specificity to ensure the system operates optimally within its constraints. This decision is strongly related to the implementation of the locking mechanism of the cat door.

Regarding the locking mechanism, there are two options for converting the result of the model cascade into action. One option is to lock the door only when *prey is* detected. This option requires high sensitivity. Given the current recall rate (cf. Figures 7 and 8), we do not consider this a practical option, as the false negative rate of 20-23% is too high for our purpose.

The other, more practical option is to keep the door locked and only open it when *no prey* is detected or when Hali is approaching.[4] This option places the cats under general suspicion. The door unlocks only if a certain threshold of accumulated evidence against the presence of prey is met. In this scenario, the cat must show its face to the camera for a certain period (e.g., 10 seconds) while the model cascade evaluates multiple consecutive frames and calculates a final score reflecting the proportion of positive and negative frames. Accumulating evidence in this manner can mitigate the impact of uncertainty from each individual model on overall performance of the cascade. This method has proven successful in similar projects [8] and other contexts [9].

Further questions to investigate include the following: Is it advantageous to increase the dataset size, even when the additional examples are highly similar? Is cross-validation an effective technique to prevent overfitting to specific training/validation set splits? Would fine-tuning pre-existing classification models to our specific dataset yield performance improvements? Given the hardware constraints of the Raspberry Pi, are ensemble methods feasible? Additionally, how can we integrate the use of the cat's RFID chips to enhance the performance of our model cascade [10]?

[4] Due to Hali's medical conditions, he should have a free pass to enter at all times.

A long-term objective is to develop and publish a curated dataset enriched with additional features, such as date and time stamps. This comprehensive dataset could facilitate in-depth analysis of cat behavior, including patterns of entry and exit, seasonal variations in prey capture, and the types of prey brought home. Furthermore, it would enable the analysis of other animals interacting with the cat bouncer system (see Figure 11).

(a) Visitor cat "Pirat" (b) Marten (c) Fox

Figure 11: Nightly Visitors

## 6 Acknowledgements

Thanks to Yannick Dittrich for the hardware set-up and help with deployment on the Raspberry Pi, data labeling, and debugging. Thanks to Michèle Brönnimann for help with data labeling. Thanks to Alena Schild for sharing her photo of Hali (Figure 1a).

## References

[1] "Flappie," https://www.flappie.ch/, accessed: June 14, 2024.

[2] "Your pawly," https://www.yourpawly.com/, accessed: June 14, 2024.

[3] "Kitty flap," https://kittyflap.com/, accessed: June 14, 2024.

[4] "Zero mouse," https://www.zeromouse.de/, accessed: June 14, 2024.

[5] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[6] J. Vincent, "An amazon employee made an ai-powered cat flap to stop his cat from bringing home dead animals." [Online]. Available: https://www.theverge.com/tldr/2019/6/30/19102430/amazon-engineer-ai-powered-catflap-prey-ben-hamm

[7] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.

[8] N. Bume, "Cat prey analyzer," https://github.com/niciBume/Cat_Prey_Analyzer, 2019, accessed: June 14, 2024.

[9] A. Agarwal, A. K. Venkataramanan, K. Dunovan, E. Peterson, T. D. Verstynen, and K. P. Sycara, "Better safe than sorry: Evidence accumulation allows for safe reinforcement learning," *CoRR*, vol. abs/1809.09147, 2018. [Online]. Available: http://arxiv.org/abs/1809.09147

[10] J. Soderberg, "Catcierge," https://github.com/JoakimSoderberg/catcierge, 2024, accessed: June 14, 2024.