Literature Review:

Smart Cat-Door System (SCDS)

(Via Convolutional Neural Networks)

Roderick L. Renwick
University of Michigan - Dearborn

April 2020

Abstraction

Cats are inherently territorial creatures and as such, it is unfair to expect these domesticated animals to tolerate each other, especially when bringing a new cat into a household of unsuspecting cats. This type of situation is likely to result in multiple forms of aggression between cats as they fight for dominance and territory. Furthermore, this may lead to an imperiled cat constantly laid victim to the stalking and pouncing of by a more dominant cat [3].

For an overwhelmed-homeowner, a dedicated safe room may prove to be the simple-solution they need. However, the question arises: how do we mediate access to and from this room, for a cat in distress? The answer proposed is a smart cat-door that uses computer vision to guarantee a specified cat access through the door, while blocking any other cat or household pet out.

Thus, the purpose of this literature review is to identify any publications relevant to the use of computer vision as the means to construct a smart cat-door which shall distinguish between different instances (breeds) of a given class of animals (cats).

Introduction

While the field of computer vision is well documented and has been studied since the 1950s [1], recent breakthroughs in deep learning via convolutional neural networks [2] have drastically increased the potential for new and relevant computer vision applications in today’s world. Through the usage of deep learning algorithms, convolutional neural networks (CNNs) can train on large image-datasets to learn key features that describe an object or pattern. In turn, this allows for a system to make predictions and classify objects in images with promisingly high levels of accuracy [4].

Although image classification systems have been structured and deployed for many purposes, the scope of this paper attempts to highlight specific research involving the recognition of cats as an object-class, and the differentiation of different instances or breeds within the animal-class of cats. This type of application is known as fine-grained object categorization [11]. Fine-grained image classification attempts to home in on exceedingly subtle variances between instances within a unique class [10].

The following segments provide a balance of basic insight into fine-grained image classification [12] research-projects, as well as various methods that may be used to implement the architecture required of a CNN which fits the smart cat-door system (SCDS).

Methodology

There are two levels of feature extractions that shall be implemented for this SCDS to make inferences from: the first being a high-level feature extractor to recognize cat objects within an image, and the second being a low-level feature extractor to distinguish between different cat instances (i.e., breeds). Isolating these two extractors provides multiple approaches to independent and distinct problems.

Fine-Grained Inferencing

Regarding the low-level feature extractor, we must look for an alternate approach as this task is much more unique and specialized. The basic idea behind fine-grained image classification is to distinguish between instances within a certain class, whose differences are subtle. This is the concept that describes a capacity to discriminate between bird species, plant families, vehicle models, and even animal breeds [7].

The most relevant studies into fine-grained object classification may be found from references [10, 11, and 12]. As these studies suggest, an exceptionally large dataset is required for fine-grained object classification. As such, this often makes transfer learning apropos for establishing a base-model, since there are already many pre-trained models well established for high-level image classification [10]. Fine-grained object categorization between breeds of cat and dog classes is investigated thoroughly in reference [12]. The research study uses the Oxford-IIIT-Pet data collection to gauge variance between breeds of different classifications and discriminate between instances. Section 4.2 of this study is specifically geared towards breed discrimination and found a performance accuracy of 63.48% for cats across 12 breeds.

Moving Forward

After considering the available publications relevant to a SCDS. It appears a multi-faceted approach will be most useful in this endeavor. While fine-grained image classification systems have been researched for the purpose of distinguishing breeds within an animal class, it remains unclear how these architectures will perform in a real-world setting.

There is room for experimentation into building a system that will efficiently distinguish between cat breeds in real-time via computer vision and constructing a smart cat-door prototype that will act accordingly. The next step in this feasibility study involves a closer inspection of CNN architectures.

Relevant Systems

The following chart below displays commercially available products that have any relationship with a smart cat door product. These products all depend on some external signal, whether it be RFID or ultrasonic communications, to function as intended. This means that the animals must have an accessory on them, either in the form of a collar, or implanted in them with a microchip.

Commercially Available "Smart" Cat-Door System Products					
Product	Description	Mechanisms	Options	Company	Cost (USD)
P1 - Electronic SmartDoor [20]	SmartKey RFID collar attachment to communicate with door.	RFID Key Fob	● Timer / Curfew Constraints	PetSafe	$100
P2 - Microchip Cat Flap Connect [21]	Unique RFID microchip implant is remembered by the door, and the internet hub links to a phone application for notifications.	RFID Microchip Implant; IoT Notifications	● Timer / Curfew Constraints ● Remote Locking ● Pet Specific Control	Sure Petcare	$243
P3 - Fully Automatic Pet Door [22]	Ultrasonic collar attachment to communicate with door.	Ultrasound Key Fob	● Timer / Curfew Constraints	High Tech Pet Products	$400
● *Note: While these products are labeled as "smart", none of these use machine learning or computer vision solutions*					

Figure-1: Commercially Available SCDSs

Although there are zero products on the market that use machine learning to implement a smart cat door, the next table illustrates two notable projects that have done so.

Standalone Smart Cat-Door System Projects via Machine Learning				
Project	Description	Hardware	Machine Learning	Sponsor
P4 - AI-Powered Cat Flap to Keep Out Dead Prey [23]	Using computer vision to identify a cat, and whether it is carrying prey in its mouth. If so, the cat door locks, stopping dead prey from entering the house.	● Webcam ● Arduino ● Servo Motors	● Computer Vision via Convolutional Neural Network	Ben Hamm
P5 - Microsoft Facial Recognition Cat Door [24]	A motion sensor triggers a webcam to capture a frame and run it through a Haar cascade classifier to search for a cat. The door then unlocks if a cat face is found.	● Webcam ● IR Sensor ● R-Pi 3 ● Servo Motors	● Computer Vision via Haar Cascade Classifiers	Microsoft

Figure-2: Standalone Projects SCDSs

Side by Side Review

The following table highlights the unique pros and cons of each product and project.

Compare & Contrast Key Features		
Products	Pros	Cons
P1	● Relatively cheap ○ Considered the standard smart cat door / first to market	● Collar and battery dependent ● Will unlock when collar is within 2’ radius ○ If can is inside and near door, any animal may enter from outside
P2	● Control door lock settings for individual cats ○ Keep other cats indoors, while only unlocking the door one-way for an outside cat to walk inside	● Requires microchip implant in cat ● Can only remember up to 5 unique IDs ● Battery powered
P3	● Automatically opens door ○ Good for smaller animals that cannot push through the door	● Expensive ● Requires cat collar
P4	● Prevents dead prey from entering home	● No mechanism for keeping other animals out ○ Door stays unlocked unless it sees cat with dead prey
P5	● Fast algorithm	● Requires clear face-shot of cat for detection ● No instance classification of individual cats

Figure-3: Comparison Table for SCDSs

After reviewing these key features, there is clearly room for more innovation in the area of smart cat doors. The most apparent problem for the commercial products is the need to have an external signal communicate with the door to signify whether a cat is nearby or not. Computer vision can provide a solution to this, as a system may be trained to recognize a cat using only a camera. However, the two computer vision projects mentioned above (P4, &P5), are very limited in scope and are not fitting for the purposes of a SCDS. They do not distinguish between different instances of cats, nor were they created for the purpose of a traditional cat door.

This helps narrow down the scope and unique functionality for a new SCDS project. The next section details the requirements for this new computer vision-based system.

Requirements

Functional

The functional requirements of this project are laid out in the following table.

Unsolved Problems
System shall be able to:
1. Detect whether a cat is present in each image (SW - model 1)
2. Identify a unique cat from a segmentation image (SW - model 2)
3. Grant access according to model predictions (HW - structure)
System shall not depend on:
4. External signal identifiers (i.e., microchip implants)
5. Animal accessories (i.e., collars)
6. Batteries
System shall not be constrained by:
7. The number of unique cats it may recognize
8. The need for a clear image of the cat’s face
9. High implementation costs

Figure-4: My SCDS Requirements

With the proposed requirements stated above (figure-4), we cannot include the new SCDS project in the comparison table shown below in figure-5.

Compare & Contrast Key Features		
Products	Pros	Cons
P1	● Relatively cheap	● Collar and battery dependent ● Will unlock when collar is within 2’ radius
P2	● Control lock settings for individual cats	● Requires microchip implant in cat ● Can only remember up to 5 unique IDs ● Battery powered
P3	● Automatically opens door	● Expensive ● Requires cat collar
P4	● Prevents dead prey from entering home	● No mechanism for keeping other animals out
P5	● Fast algorithm	● Requires clear face-shot of cat for detection ● No instance classification for individual cats
My SCDS	● Cat detect and identify individual cats ● Stores recent snapshots when granting access ● Does not require microchip or collar	● Does not do night vision

Figure-5: Comparison Table for SCDSs (including my SCDS)

Non-Functional

For the programming aspect of this project, Python will be used as it is easiest for integration of various modules and is well documented. The computer vision modules will primarily use TensorFlow as the backend and Keras as the front end for any machine learning applications. As for the hardware, a Nvidia Jetson-Nano GPU will be used, with an attached camera. Google Colaboratory will be the programming environment used before implementing the design onto the physical Jetson-Nano hardware.

Design

Broad Overview

The following figure illustrates the system functionality from the highest abstraction. This system is broken up into two sub-modules, meeting requirements 1 and 2 respectively.

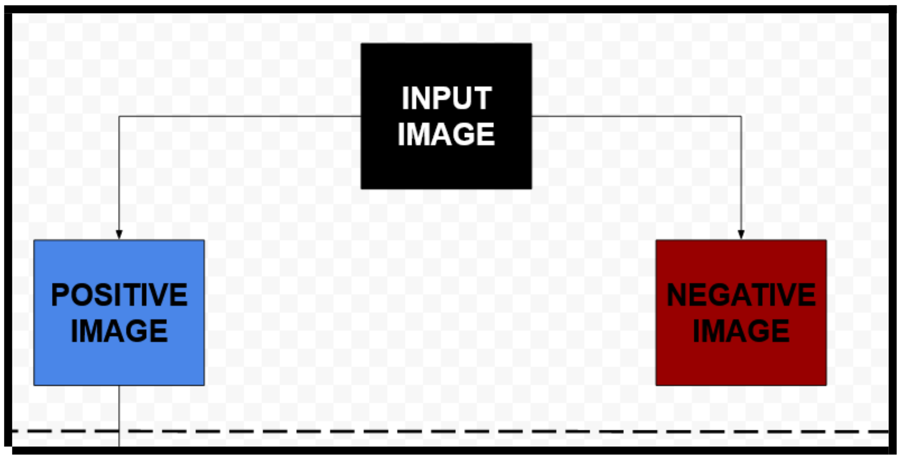


Figure-6.A: High-Level Design Overview for Module 1

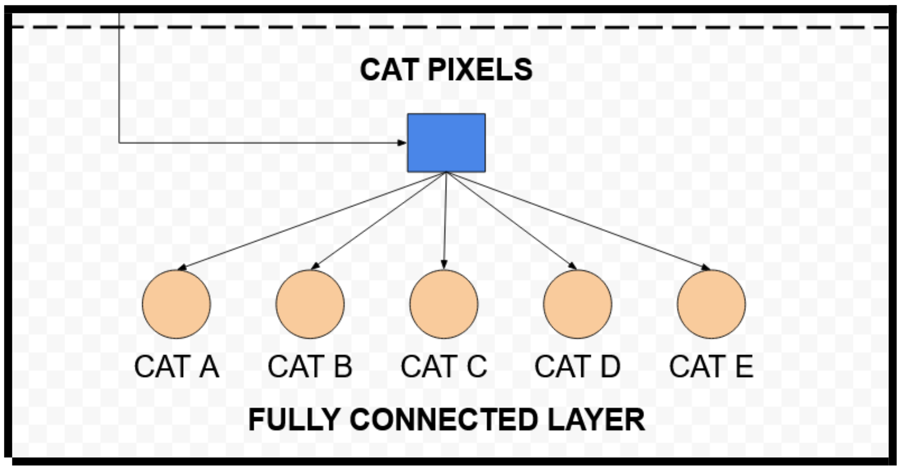


Figure-6.B: High-Level Design Overview for Module 2

Module 1 - Cat Detector

The following figure (figure-7) illustrates the inputs and outputs of the first CNN, module-1, from the highest level of abstraction. This module shall be responsible for taking an input image frame (from a live-camera feed) and determining whether this image frame contains a cat within it. If it is determined that there is a cat in the frame, this frame then gets flagged "positive", and gets passed onto the next module (module 2).

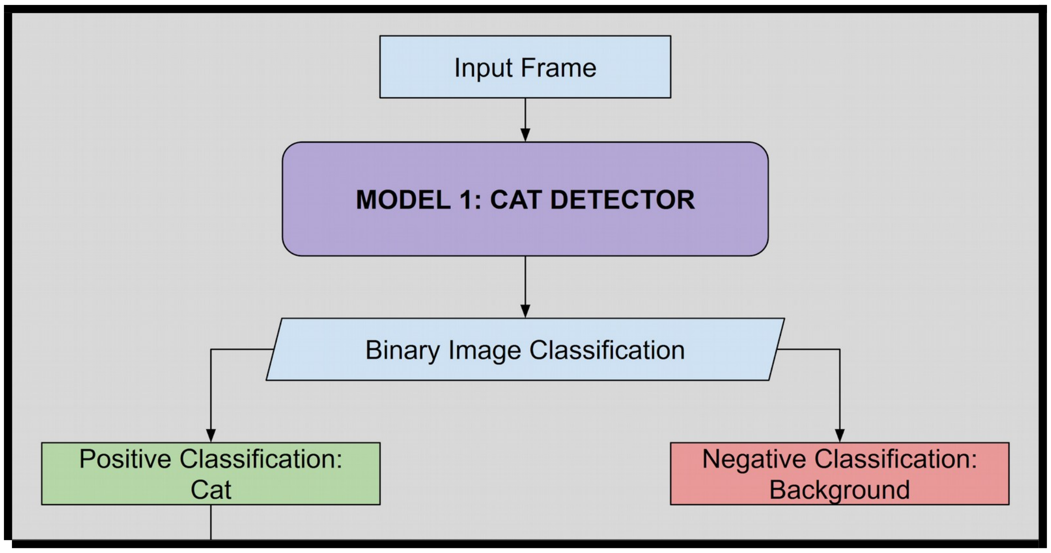

Input Frame
MODEL 1: CAT DETECTOR
Binary Image Classification
Positive Classification: Cat
Negative Classification: Background

Figure-7: Model-1 High-Level Design (Cat Detector)

Below is a conceptual illustration of the CNN architecture (figure 9) used for this module. As shown, an input tensor is provided as an input, having a width and height of 224 pixels (likely after having been resized and/or reshaped) and a depth of 3 (representing the 3 color-channels, RGB). There are 32 convolution filters (each a collection of 3 channels or "kernels") applied to the input image, followed by a max-pooling operation that down samples the image to a width and height of 112 pixels and a depth of 64 layers (or "filters" which would give us 192 feature-maps or "kernels").

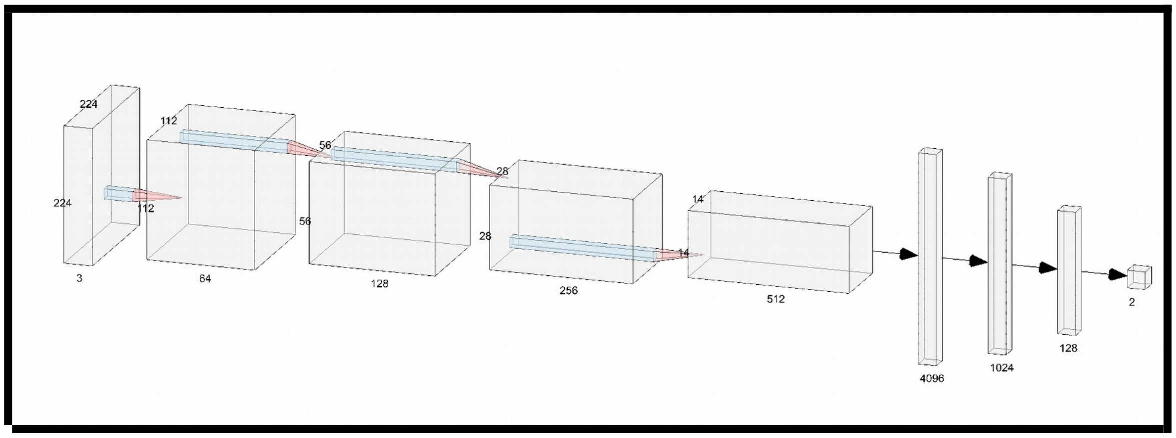


Figure-9: Model-1 CNN Architecture (Cat Detector)

The following flowchart illustrates the structuring of the convolutional neural network.

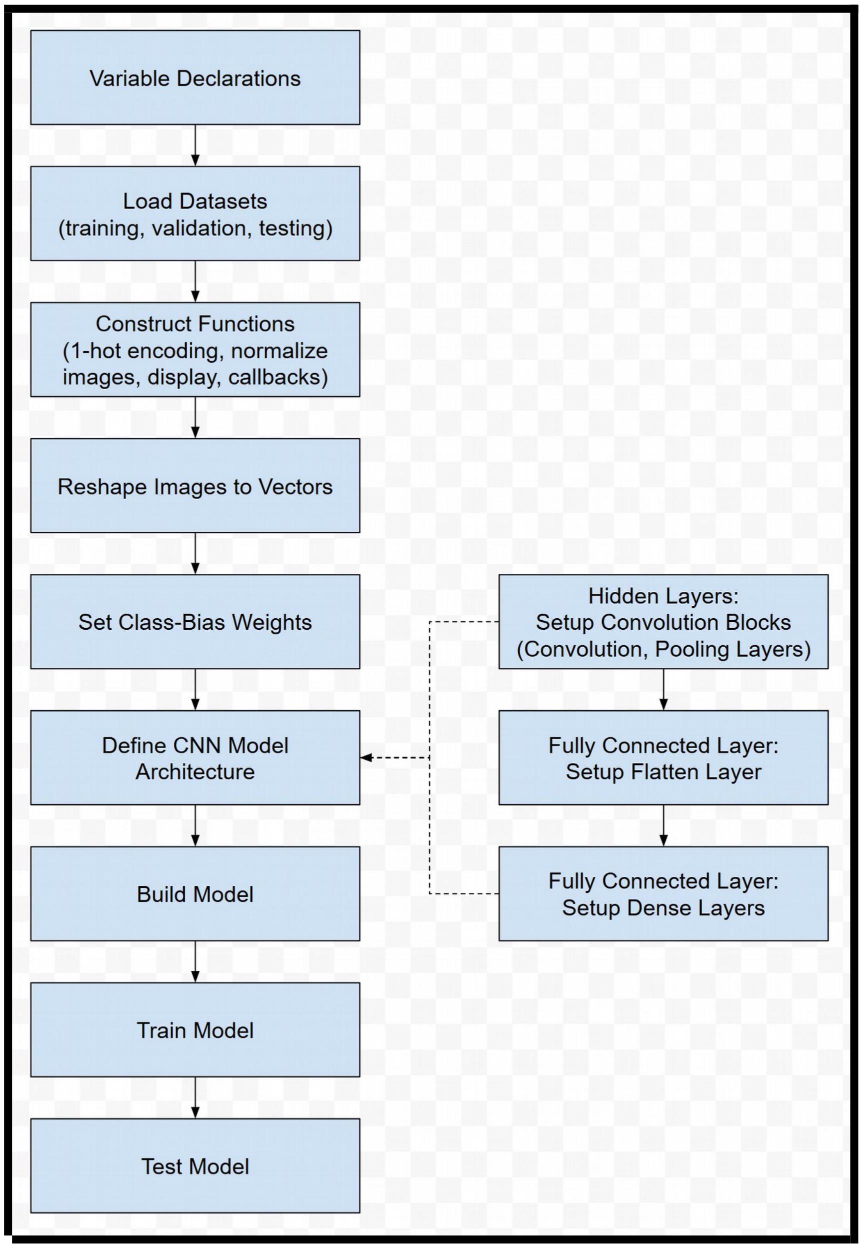

Variable Declarations
Load Datasets (training, validation, testing)
Construct Functions (1-hot encoding, normalize images, display, callbacks)
Reshape Images to Vectors
Set Class-Bias Weights
Define CNN Model Architecture
Build Model
Train Model
Test Model
Hidden Layers: Setup Convolution Blocks (Convolution, Pooling Layers)
Fully Connected Layer: Setup Flatten Layer
Fully Connected Layer: Setup Dense Layers

Figure-8: Model-1 Low-Level Design (Cat Detector)

For the first model, a rather extensive dataset is required to thoroughly train the network on background images, as well as the cats so that it does not miss-identify. Both the Oxford-IIIT and 2012 Visual Object Classes (VOC) Datasets are excellent and will be used here [15]. One of the biggest challenges will be correcting the bias of the weights while training from this dataset. Given that there are 20 classes, and we only care about distinguishing between cat and non-cat images, this means that the ratio of training images will be about 1:19. Depending on the exact total of images for the positive and negative image classifications, weight adjustments may be set in an attempt to balance the network for more stable training.

Module 2 - Cat Identifier

This last module is responsible for the identification of individual cats. Figure-10 below, illustrates the inputs and outputs of module-2, which distinguishes which cat the image likely belong to.

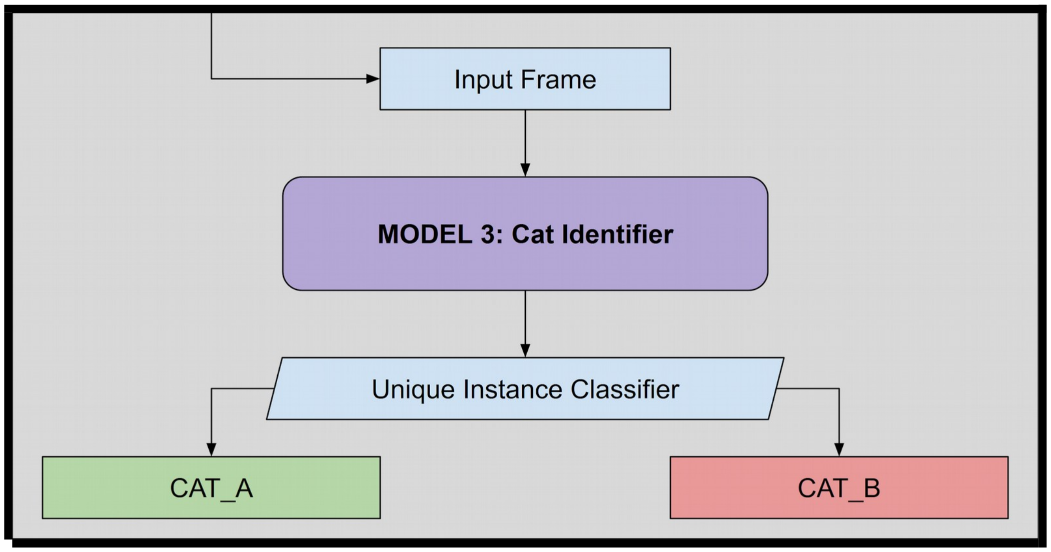


Figure-10: Model-2 High-Level Design (Cat Identifier)

Since this module uses a traditional CNN style architecture, the figure below (figure-11) is another illustration of the CNN, like that of figure-9. Here the input image has a height of 128 x 128, with a depth of 3.

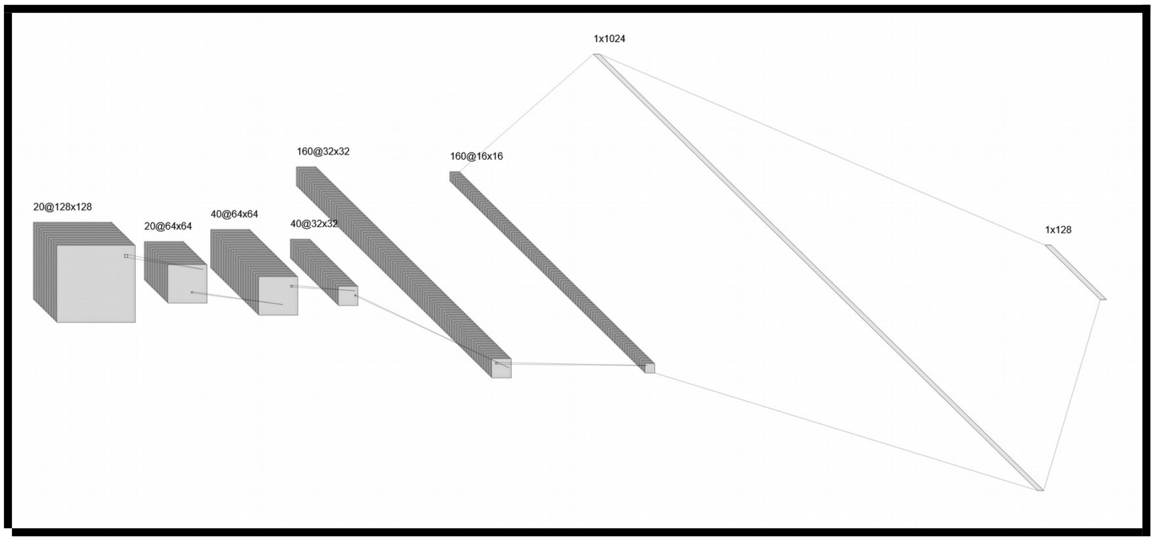


Figure-11: Model-2 CNN Architecture (Cat Identifier)

The development for Model 2 will follow the same structure as laid out in figure-10, except reduced in complexity as it will have much less layers since it will be a similar task.

Implementation

Module 1 - Cat Detector

The figure below shows the input layer for the sequential mode. Note that the input tensor shape must be the consistant size of the original image. In this case, the input tensor is 256, by 256, by 3 (height, width, and color channels respectivley).

```
# INPUT_LAYER :
# Start w/ kernel size of 7 for lrg feature detection, stride of 2 to reduce output volume
# NOTE: MaxPool( ReLU( Conv( M ) ) ) == ReLU( MaxPool( Conv( M ) ) )
# Order does not matter since it will yield same result
#-----------------------------------------------------------------------------------------------------

def add_input_layer( model, shape = INPUT_SHAPE, filters = FILTER_COUNT ):

    model.add( Conv2D( filters = filters, kernel_size = 3, padding = 'same', input_shape = shape ) )

    model.add( MaxPooling2D( pool_size = 2 ) )
    model.add( LeakyReLU( alpha = 0.1 ) )
    model.add( Dropout( DROPOUT ) )
```

Figure-12: Input Layer for Model-1

The next figure shows the convolution blocks of the CNN model. Each convolution block has two layers, where each layer goes through a 2D convolution filter, batch normalization, and a relu activation filter. In between sets of convolution blocks, max pooling is applied to down sample the image while retaining needed information for classification.

```
# CONVOLUTION_BLOCKS : Layers of convolution, activation, batch normalization, dropout
#-----------------------------------------------------------------------------------------------------
def add_convolution_block( model, filters = FILTER_COUNT, kernel_size = 3 ):

    model.add( Conv2D( filters = filters, kernel_size = kernel_size, padding = 'same' ) )
    model.add( LeakyReLU( alpha = 0.1 ) )

    model.add( Conv2D( filters = filters, kernel_size = kernel_size, padding = 'same' ) )
    model.add( LeakyReLU( alpha = 0.1 ) )

    model.add( BatchNormalization() )
    model.add( MaxPooling2D( pool_size = 2 ) )
    model.add( Dropout( DROPOUT ) )
```

Figure-13: Layers of a Convolution Block for Model-1

The next figure shows the fully connected layer. After the last pooling operation is applied, the tensor is then flattened into a vector. Every element within this vector is connected to the following fully connected vectors until reaching a two-unit vector whose elements represent different classification categories. Thus, these fully connected layers are also known as densely connected layers.

```
# FULLY_CONNECTED_LAYER ( FCL ) : Takes in all neurons or nodes for the layer (aka dense layer )
# Input Layer   : Expects arrays of shape: (*, k ), where k is number of input features
# Hidden Layers : Num of nodes in HL = output dim of dense layer, relu activation
# Output layer  : 1 node, softmax activation fn
#----------------------------------------------------------------------------------------------------

def add_fully_connected_layer( model ):

    model.add( Flatten() )

def add_dense_block( model, units ):

    model.add( Dense( units = units ) )
    model.add( LeakyReLU( alpha = 0.1 ) )

    model.add( Dense( units = units ) )
    model.add( LeakyReLU( alpha = 0.1 ) )

    model.add( BatchNormalization() )
    model.add( Dropout( DROPOUT * 2 ) )
```

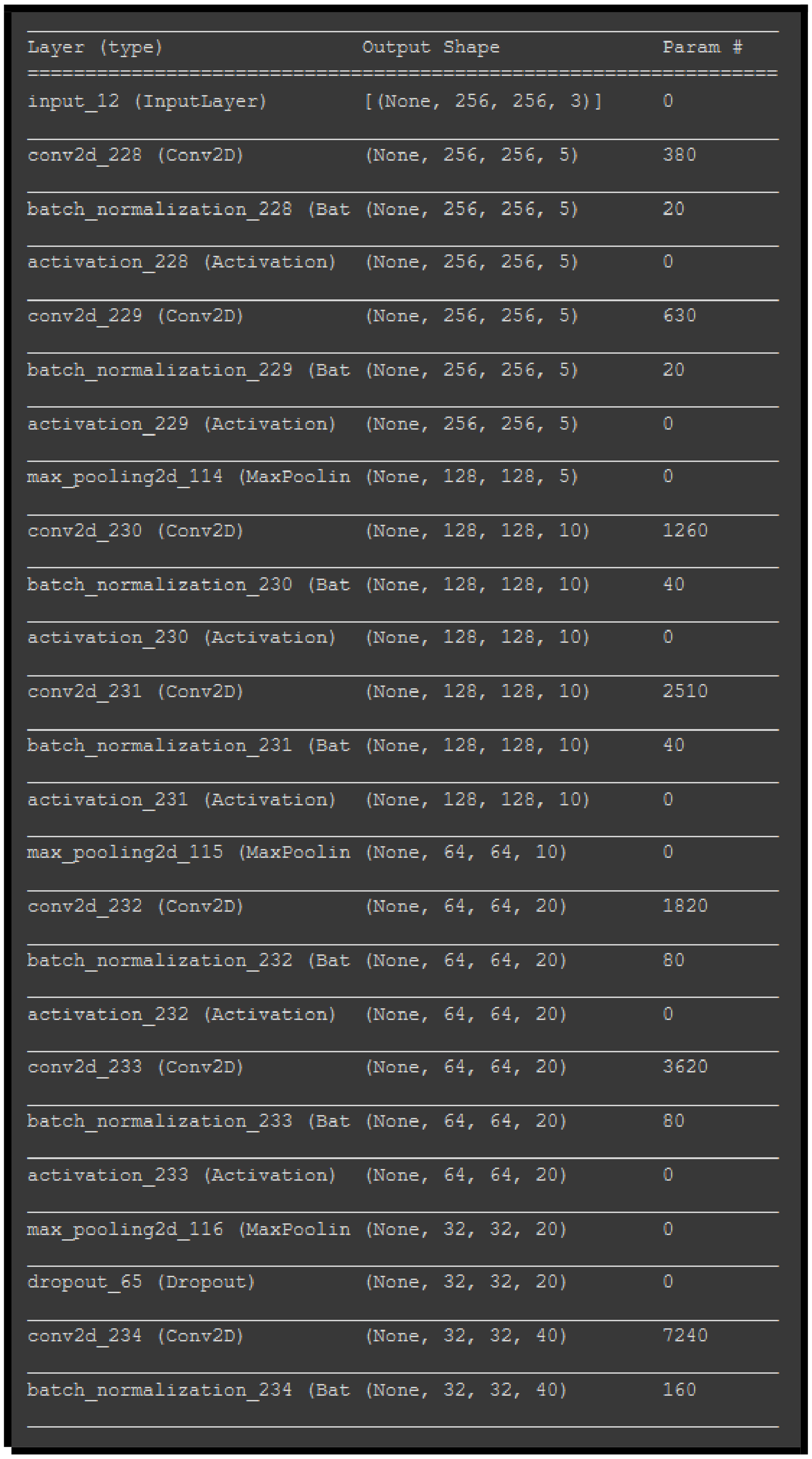
Figure-14: Fully Connected Layer for Model-1

The last figure below shows a simple function to build the model structure using the encoder and fully connected layer sections implemented above.

```
def build_model( shape = INPUT_SHAPE, filters = FILTER_COUNT ):

    classifier = Sequential()

    add_input_layer( model = classifier, shape = shape, filters = filters )     # Input layer

    # Convolution Blocks
    add_convolution_block( model = classifier, filters = ( filters *  2 ), kernel_size = 3 )
    add_convolution_block( model = classifier, filters = ( filters *  4 ), kernel_size = 3 )
    add_convolution_block( model = classifier, filters = ( filters *  8 ), kernel_size = 3 )
    add_convolution_block( model = classifier, filters = ( filters * 16 ), kernel_size = 3 )
    add_convolution_block( model = classifier, filters = ( filters * 32 ), kernel_size = 3 )

    add_fully_connected_layer( classifier )                            # FCL
    add_dense_block( model = classifier, units = 128 )                 # Dense Blocks
    classifier.add( Dense( units = CLASSES_COUNT, activation = 'softmax' ) )    # Softmax Activation
    compile_model( model = classifier )                                 # Optimizer

    return( classifier )
```

Figure-15: CNN Model Build for Model-1

The last figure shows the resulting summary of the model after it has been constructed.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_12 (InputLayer)        [(None, 256, 256, 3)]     0
_________________________________________________________________
conv2d_228 (Conv2D)          (None, 256, 256, 5)       380
_________________________________________________________________
batch_normalization_228 (Bat (None, 256, 256, 5)       20
_________________________________________________________________
activation_228 (Activation)  (None, 256, 256, 5)       0
_________________________________________________________________
conv2d_229 (Conv2D)          (None, 256, 256, 5)       630
_________________________________________________________________
batch_normalization_229 (Bat (None, 256, 256, 5)       20
_________________________________________________________________
activation_229 (Activation)  (None, 256, 256, 5)       0
_________________________________________________________________
max_pooling2d_114 (MaxPoolin (None, 128, 128, 5)       0
_________________________________________________________________
conv2d_230 (Conv2D)          (None, 128, 128, 10)      1260
_________________________________________________________________
batch_normalization_230 (Bat (None, 128, 128, 10)      40
_________________________________________________________________
activation_230 (Activation)  (None, 128, 128, 10)      0
_________________________________________________________________
conv2d_231 (Conv2D)          (None, 128, 128, 10)      2510
_________________________________________________________________
batch_normalization_231 (Bat (None, 128, 128, 10)      40
_________________________________________________________________
activation_231 (Activation)  (None, 128, 128, 10)      0
_________________________________________________________________
max_pooling2d_115 (MaxPoolin (None, 64, 64, 10)        0
_________________________________________________________________
conv2d_232 (Conv2D)          (None, 64, 64, 20)        1820
_________________________________________________________________
batch_normalization_232 (Bat (None, 64, 64, 20)        80
_________________________________________________________________
activation_232 (Activation)  (None, 64, 64, 20)        0
_________________________________________________________________
conv2d_233 (Conv2D)          (None, 64, 64, 20)        3620
_________________________________________________________________
batch_normalization_233 (Bat (None, 64, 64, 20)        80
_________________________________________________________________
activation_233 (Activation)  (None, 64, 64, 20)        0
_________________________________________________________________
max_pooling2d_116 (MaxPoolin (None, 32, 32, 20)        0
_________________________________________________________________
dropout_65 (Dropout)         (None, 32, 32, 20)        0
_________________________________________________________________
conv2d_234 (Conv2D)          (None, 32, 32, 40)        7240
_________________________________________________________________
batch_normalization_234 (Bat (None, 32, 32, 40)        160
_________________________________________________________________
```

Figure-16: Architecture Summary for Model-1

Module 2 - Cat Identifier

The last network is a very simple CNN model which will only take in input pixels relating to a given cat. Since the number of pixels is minimized, and the variance between cats needed to be identified is relatively large, this network only requires a couple simple layers to effectively and efficiently distinguish between the different cat classes. The following figure below provides the implementation for building the model. Three convolution blocks are followed by a flattening layer, and then two dense layers to arrive at the final classification categorization. This completes the main bulk of the software implementation for this SCDS.

```
def build_model( input_shape = INPUT_SHAPE, filters = FILTER_COUNT ):

    classifier = Sequential()
    classifier.add( Conv2D( filters = filters * 1, kernel_size = 3,
                    strides = 2, padding = 'same', input_shape = input_shape ) )

    classifier.add( Dropout( DROPOUT ) )

    classifier.add( Conv2D( filters = filters * 2, kernel_size = 3, padding = 'same' ) )
    classifier.add( LeakyReLU( alpha = 0.1 ) )

    classifier.add( Conv2D( filters = filters * 2, kernel_size = 3, padding = 'same' ) )
    classifier.add( LeakyReLU( alpha = 0.1 ) )

    classifier.add( BatchNormalization() )
    classifier.add( MaxPooling2D( pool_size = 2 ) )
    classifier.add( Dropout( DROPOUT ) )

    classifier.add( Conv2D( filters = filters * 4, kernel_size = 3, padding = 'same' ) )
    classifier.add( LeakyReLU( alpha = 0.1 ) )

    classifier.add( Conv2D( filters = filters * 4, kernel_size = 3, padding = 'same' ) )
    classifier.add( LeakyReLU( alpha = 0.1 ) )

    classifier.add( BatchNormalization() )
    classifier.add( MaxPooling2D( pool_size = 2 ) )
    classifier.add( Dropout( DROPOUT ) )

    # FCL
    classifier.add( Flatten() )

    classifier.add( Dense( units = 64 ) )
    classifier.add( LeakyReLU( alpha = 0.1 ) )

    classifier.add( Dense( units = 64 ) )
    classifier.add( LeakyReLU( alpha = 0.1 ) )

    classifier.add( BatchNormalization() )
    classifier.add( Dropout( DROPOUT * 2 ) )

    # Softmax Classifier
    classifier.add( Dense( units = 2 ) ) # Number of classes = 2
    classifier.add( Activation( 'softmax' ) )

    # Optimizer
    classifier.compile( optimizer = OPTIMIZER, loss = LOSSES, metrics = METRICS, run_eagerly = False )

    return( classifier )
```

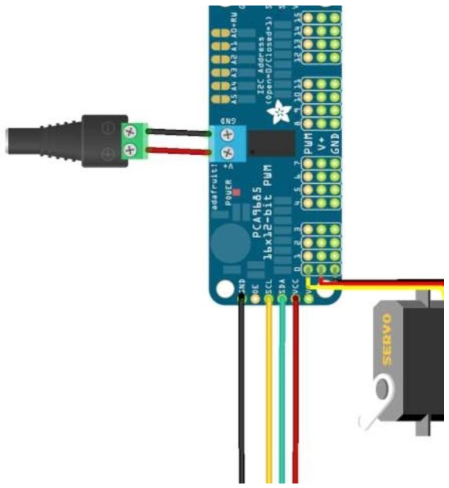
Figure-17: CNN Model Build for Model-2

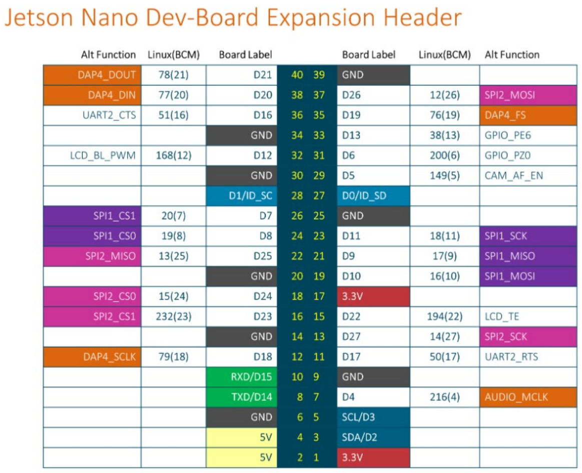
Physical Structue

Here is an image of the working enviornment, where the rug is the region of interest for the camera. This is the cat door which the test subjects will be attempting to pass through. Notice the Jetson Nano is stationed at the bottom left corner near the door.

Figure-18: Working Prototype Enviornment

The next peripheral to setup was a pwm driver to control the servo latch and the blinking LEDs. The first figure below on the rigth ([13]) is the adafruit pwm driver that we used to power the servo motor and the LEDs. The figure to the right of that is the Jetson Nano pinout table that we used to find the I2C bus pins (power, ground, data, and clock) [14, 15, and 16]. After the pwm driver was connected to the Jetson Nano.




Jetson Nano Dev-Board Expansion Header

Alt Function	Linux(BCM)	Board Label			Board Label	Linux(BCM)	Alt Function
DAP4_DOUT	78(21)	D21	40	39	GND		
DAP4_DIN	77(20)	D20	38	37	D26	12(26)	SPI2_MOSI
UART2_CTS	51(16)	D16	36	35	D19	76(19)	DAP4_FS
		GND	34	33	D13	38(13)	GPIO_PE6
LCD_BL_PWM	168(12)	D12	32	31	D6	200(6)	GPIO_PZ0
		GND	30	29	D5	149(5)	CAM_AF_EN
		D1/ID_SC	28	27	D0/ID_SD		
SPI1_CS1	20(7)	D7	26	25	GND		
SPI1_CS0	19(8)	D8	24	23	D11	18(11)	SPI1_SCK
SPI2_MISO	13(25)	D25	22	21	D9	17(9)	SPI1_MISO
		GND	20	19	D10	16(10)	SPI1_MOSI
SPI2_CS0	15(24)	D24	18	17	3.3V		
SPI2_CS1	232(23)	D23	16	15	D22	194(22)	LCD_TE
		GND	14	13	D27	14(27)	SPI2_SCK
DAP4_SCLK	79(18)	D18	12	11	D17	50(17)	UART2_RTS
		RXD/D15	10	9	GND		
		TXD/D14	8	7	D4	216(4)	AUDIO_MCLK
		GND	6	5	SCL/D3		
		5V	4	3	SDA/D2		
		5V	2	1	3.3V		

Figure-19: PWD Driver & Jetson Nano Pinouts

The following figures below show the physical implementation of the pwm driver.

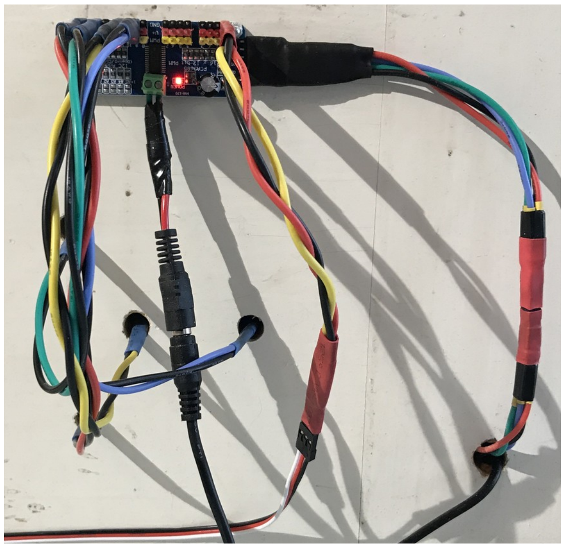


Figure-20: PWM Driver Impmentation

The next peripheral to setup was the servo latch and the blinking LEDs, operated by the PWM driver. The implmentations are dislayed below, where the left image shows the front of the cat door, and the status-LEDs, and the right side shows the servo latch visable before mounting the cat door.

For the LEDs, the blue left-most light indicates that the system is on, and running. The yellow LED turns on if the system has found a cat within the field of view, and then the red LED will turn on if that cat is determined to be a cat not allowed through, or else the red LED will turn on to indicated that the cat is allowed through. If the cat is allowed through, the servo latch will rotate clockwise, where the arm is in a horizontal position, allowing for the cat door to swing open. Otherwise, the servo latch will rotate counter-clockwise, aligning the arm vertically as to block the cat door from swinging open.

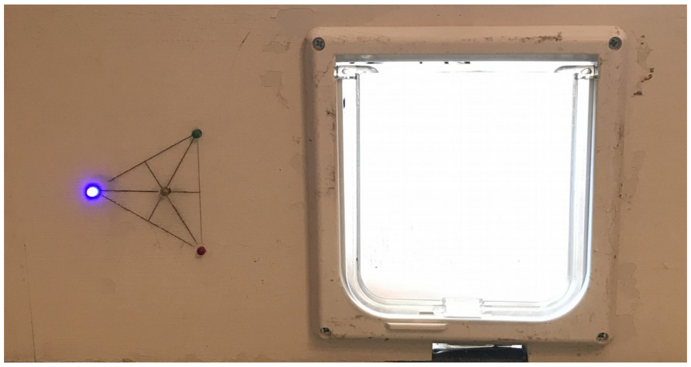
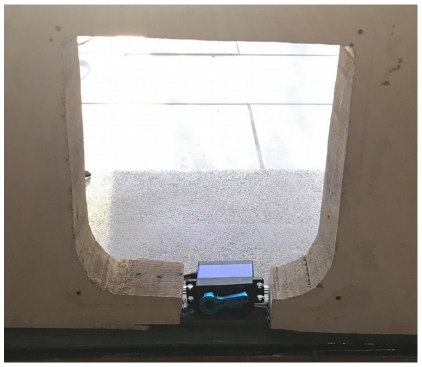



Figure-21: LED Implmentation (RHS) & Servo Latch Impmentation (RHS)

Test Subjects

Here are the two cats in question for this project. The cat on the left is named Remi (Cat A), and the cat on the right is named Lightning (Cat B). Remi is the cat to be prevented from going through this cat door, while Lightning is the cat to be accepted.

Figure-22: Test Subjects (Cats A & B)

Testing

Module 1 - Cat Detector

The following figures show the model accuracy and cross-entropy loss during training for model-1 on both the validation and training datasets.

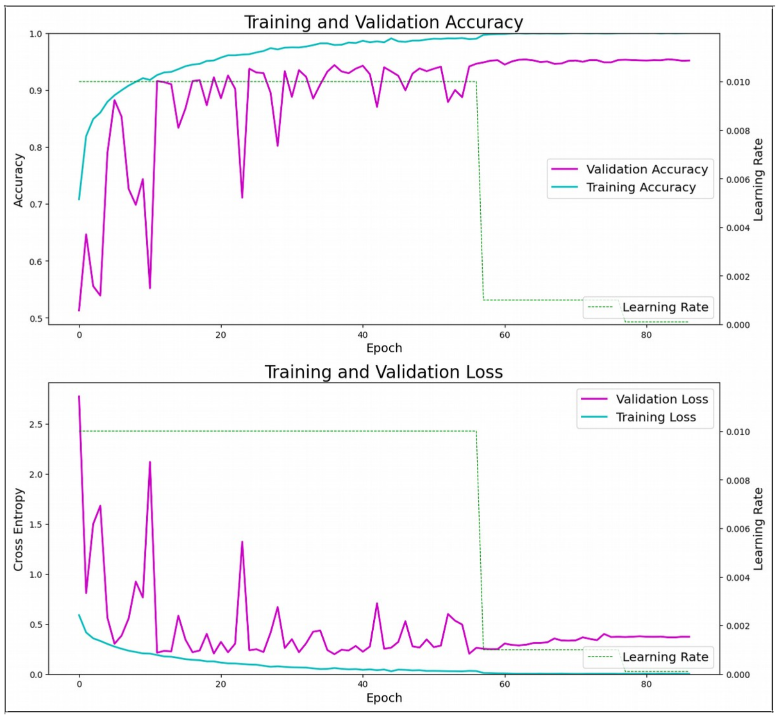


Figure-23 Training Accuracy and Cross Entropy for Model-1

Image classification results are provided below in from both the testing data.

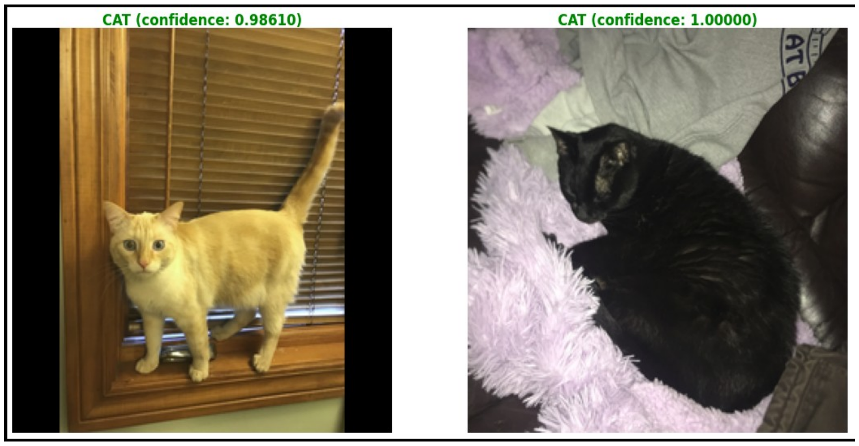


Figure-24: Test Case Image

Module 2 - Cat Identifier

The following figures show the model accuracy and cross-entropy loss during training for model-2 on both the validation and training datasets.

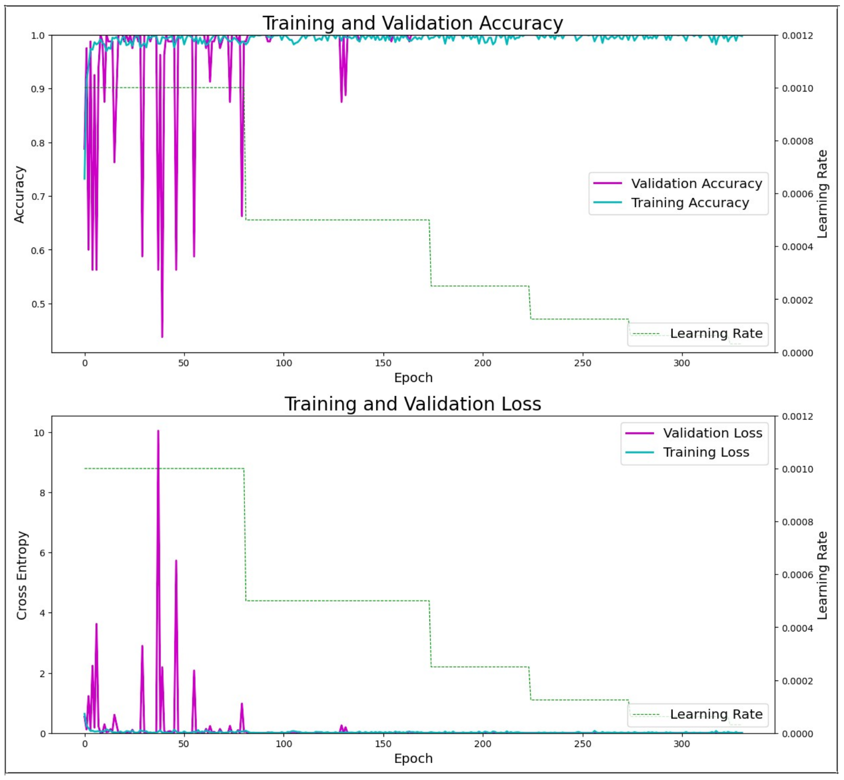


Figure-25: Training Accuracy and Cross Entropy for Model-2

Again, image classification results are provided below in from both the testing data.

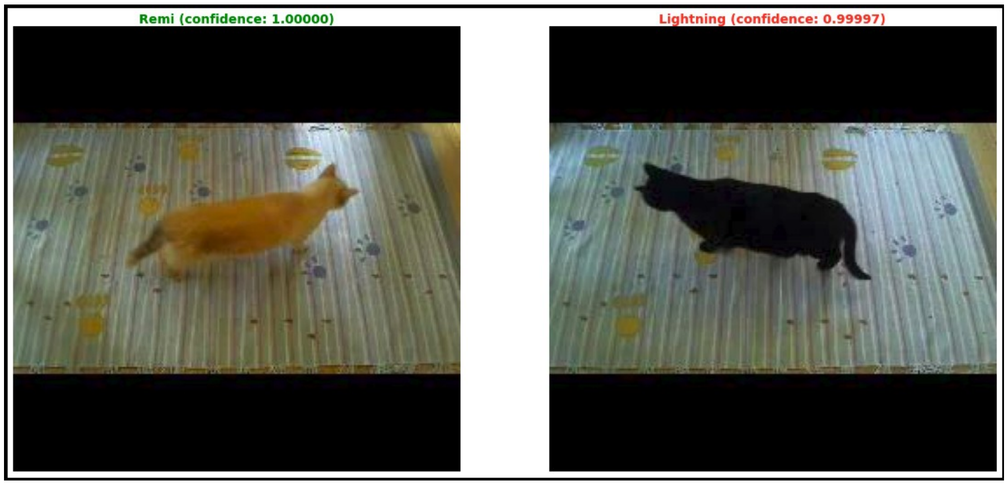


Figure-26: Cat Identifier Test Case

Video 1 Pipeline

The first test video for the pipeline test is of Rem (Cat A) running through the field of view. As illustrated in the image below, the resulting frame classifications from model 1 are very percise. Above each frame is it's classification from model 1 (cat or no cat), follwed by the frame index, and then the prediction confidence level.

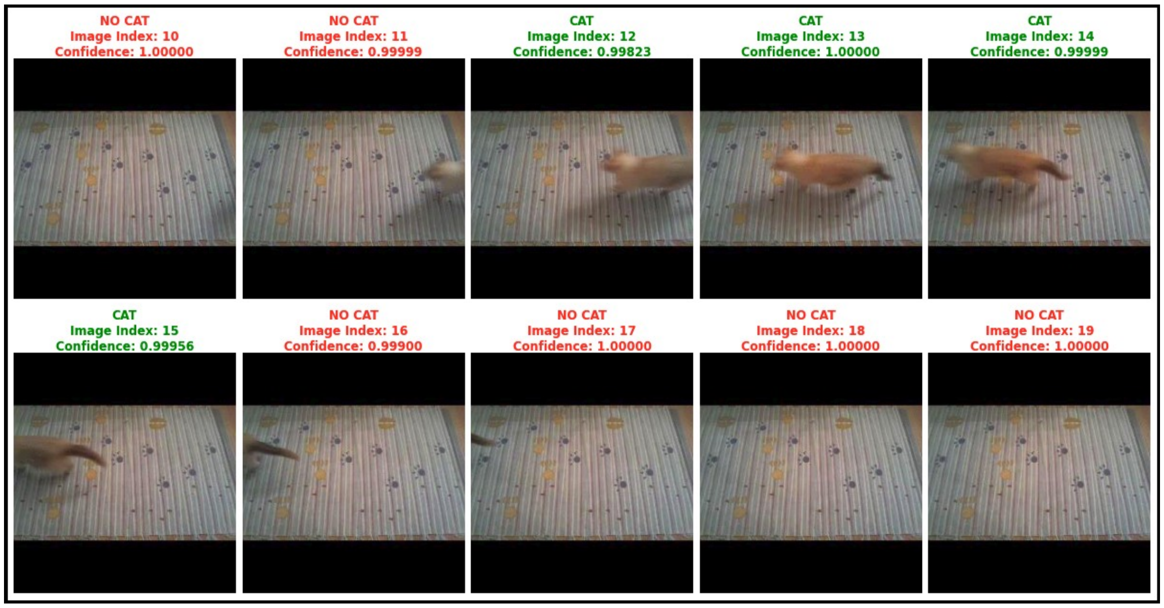


Figure-27: Video 1 Test Case for Model 1 (Cat Detector)

The next image illustrates the predictions made from model 2, after the frames which have been determined as cat images have been passed down the pipeline from model 1. This time the frame classification is either 'Remi' or 'Lightning'. The cat is correctly identified in each frame.

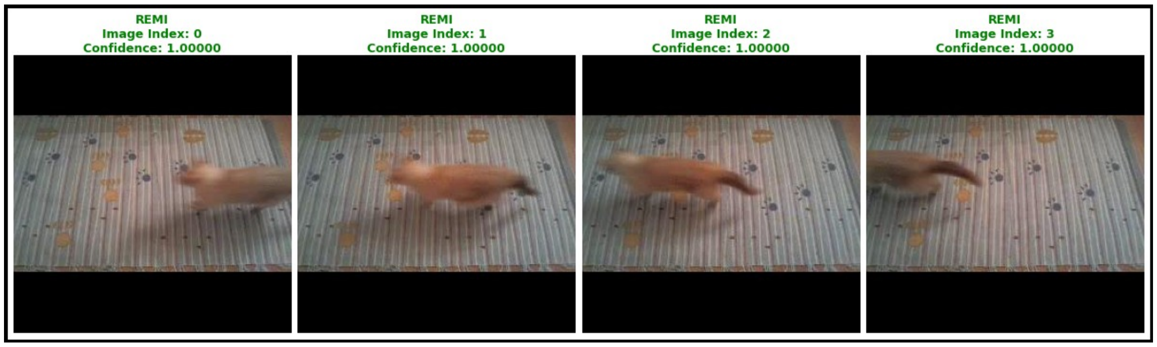


Figure-28: Video 1 Test Case for Model 2 (Cat Identifier)

Video 2 Pipeline

The second test video for the pipeline test is of Lightning (Cat B) running through the field of view. The image below illustrates the resulting frame classifications from model 1 are again very percise, correctly identifying almost all frames where Lightning enters the view. Once the whole body of the cat is in the image, the predictions are 100% accurate.

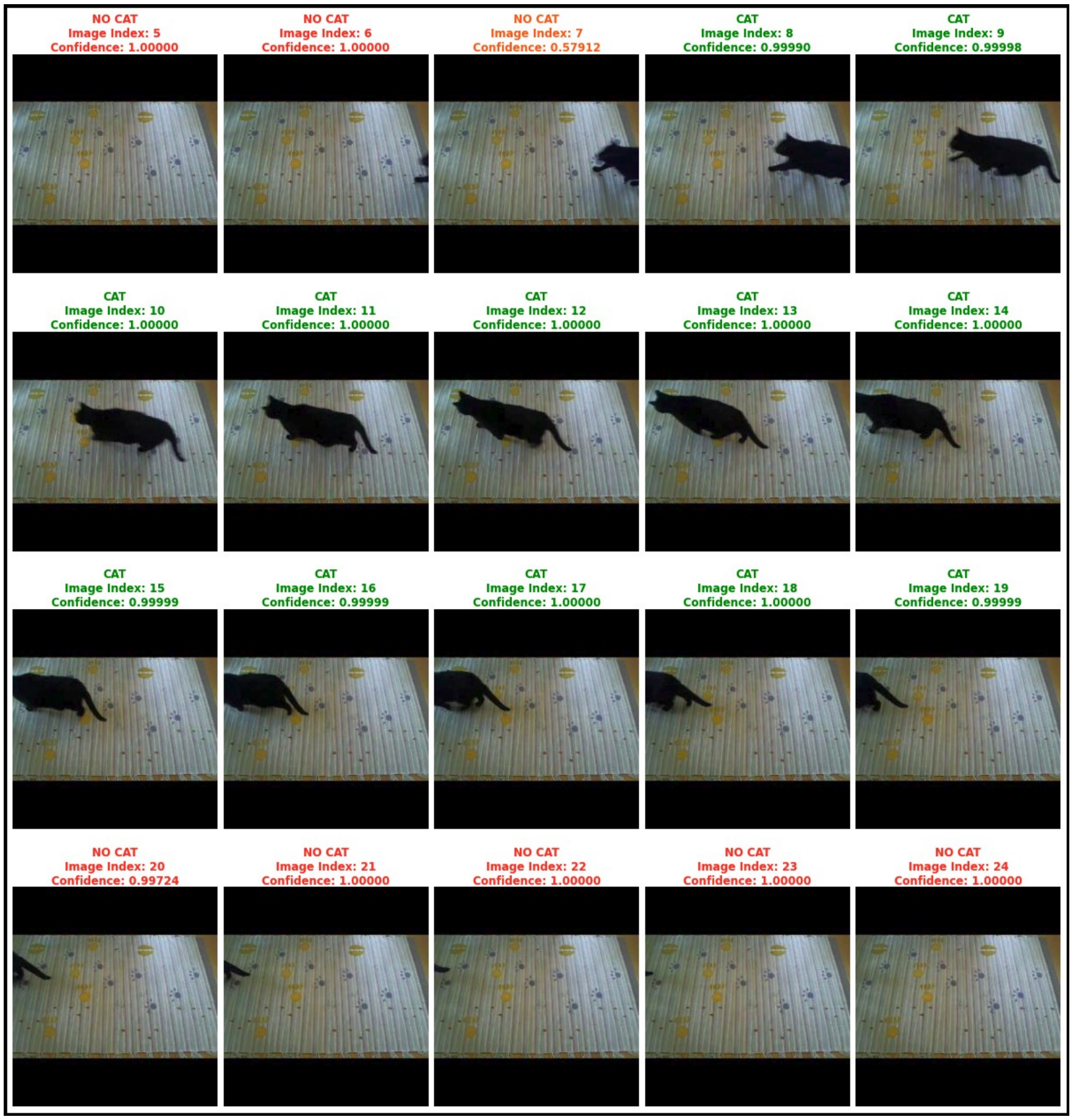


Figure-29: Video 2 Test Case for Model 1 (Cat Detector)

The next image illustrates the predictions made from model 2 for the second video, again with 100% accuracey.

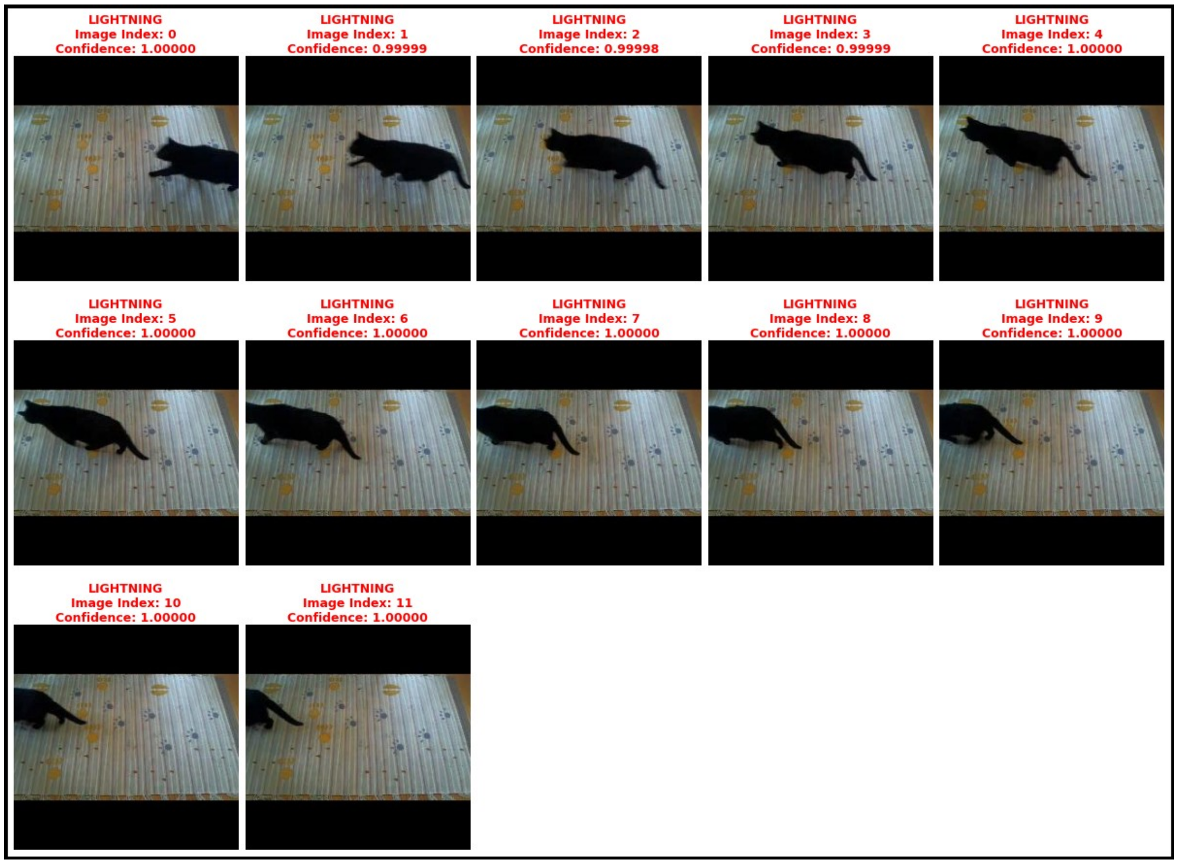


Figure-30: Video 2 Test Case for Model 2 (Cat Identifier)

Final Full System

After assembling the peripheral devices, training the models, and developing the camera pipeline, the final test is that of the full system from start to finish. Note that these test result images were taken from a camera other than the one used by the system, so that the images could display the cat as well as the status LEDs on the cat door in order to best represent the Smart Cat-Door System system. The first image shows the result of the default state, where the system is running (indicated by the blue LED being on) and there is no cat detected (indicated by the yellow LED being off). Although not visible, the latch is in the closed state, followed down being the door, leaving the cat door unlocked and able to swing open.

Figure-31: Full System Test Case for No Cat

The next set of images are the testing results for Remi (Cat-A). Here you can see that both a cat has been detected (indicated by the yellow LED being on), and that it has been correctly identified as Remi (indicated by the red LED being on). At the point, the latch is in an upward position to lock the cat door and not allow Remi through.

Figure-32: Full System Test Case for Remi (Cat A)

The last set of images are the testing results for Lightning. Again, the system has indicated that a cat has been detected (yellow LED is on), and now it has correctly identified the cat as Lightning (indicated by the green LED being on).

Figure-33: Full System Test Case for Lightning (Cat B)

References

Background Research, CNN Implmentations

1. Mihajlovic, Ilija. “Everything You Ever Wanted to Know About Computer Vision. Here's A Look Why It's So Awesome.” *Medium*, Towards Data Science, 28 Nov. 2019, towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e.

2. Demush, Rostyslav. “A Brief History of Computer Vision (and Convolutional Neural Networks).” *By Rostyslav Demush*, A Brief History of Computer Vision (and Convolutional Neural Networks), 26 Feb. 2019, hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3.

3. “Aggression between Cats.” *The Humane Society of the United States*, www.humanesociety.org/resources/aggression-between-cats.

4. Le, James. “How to Do Semantic Segmentation Using Deep Learning.” *AI & Machine Learning Blog*, AI & Machine Learning Blog, 23 Oct. 2019, nanonets.com/blog/how-to-do-semantic-segmentation-using-deep-learning/.

5. Gupta, Divam. "A Beginner's Guide to Deep Learning Based Semantic Segmentation Using Keras." *Divam Gupta*, 6 June 2019, divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html.

6. "CS231n Convolutional Neural Networks for Visual Recognition." *CS231n Convolutional Neural Networks for Visual Recognition*, cs231n.github.io/transfer-learning/.

7. Zheng, Heliang; Fu, Jianlong; Zha, Zheng-Jun; Luo, Jiebo. “Looking for the Devil in the Details: Learning Trilinear Attention SamplingNetwork for Fine-grained Image Recognition.” *ArXiv.org E-Print Archive*, 11 June 2019, arxiv.org/pdf/1903.06150v2.pdf.

8. "Distinguishing Classification Tasks with Convolutional Neural Networks." *Dummies*, 16 July 2019, www.dummies.com/programming/big-data/data-science/distinguishing-classification-tasks-with-convolutional-neural-networks/.

9. Boyer, Marnie. "Dog Breed Classification Using a Pre-trained CNN Model." *Medium*, 20 Mar. 2019, medium.com/@marnieboyer/dog-breed-classification-using-a-pre-trained-cnn-model-84d0af72b4fc.

10. Xiao, Tianjun; Xu, Yichong; Yang, Kuiyuan; Zhang, Jiaxing; Peng, Yuxin; Zhang, Zheng. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015"CVPR 2015 Open Access Repository." *Http://openaccess.thecvf.com/*, openaccess.thecvf.com/content_cvpr_2015/html/Xiao_The_Application_of_2015_CVPR_paper.html.

11. Parkhi, Omkar; Vedaldi, Andrea; Zisserman, Andrew; Jawahar, C.V. “Cats and Dogs.” *Information Engineering Main/Home Page*, www.robots.ox.ac.uk/~vgg/publications/2012/parkhi12a/parkhi12a.pdf.

PWM & I2C Setup:

13. Earl, B. (n.d.). Adafruit PCA9685 16-Channel Servo Driver. Retrieved from https://learn.adafruit.com/16-channel-pwm-servo-driver/hooking-it-up.
14. Warren, S., & Tan, A. (2019, October 30). NVIDIA/jetson-gpio. Retrieved from https://github.com/NVIDIA/jetson-gpio.
15. Jetson Nano - Using I2C. (2019, September 29). Retrieved from https://www.jetsonhacks.com/2019/07/22/jetson-nano-using-i2c/.
16. Element14: NVIDIA Jetson Nano Developer Kit Pinout and Diagrams. (2019, May 21). Retrieved from https://www.element14.com/community/community/designcenter/single-board-computers/blog/2019/05/21/nvidia-jetson-nano-developer-kit-pinout-and-diagrams.

Visual Object Classes (VOC) Dataset:

17. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

Oxford-IIIT Per Dataset:

18. https://www.robots.ox.ac.uk/~vgg/data/pets/

CNN Visualization Tool:

19. http://alexlenail.me/NN-SVG/AlexNet.html

Commercial Products

20. "Electronic SmartDoor™." *PetSafe*, store.petsafe.net/electronic-smartdoor.
21. "Microchip Cat Flap Connect." *SureFlap | Award Winning Microchip Cat Doors & Feeders*, www.surepetcare.com/en-us/pet-doors/microchip-cat-flap-connect.
22. "Electric Pet Door Keeps Out Raccoons & Stray Animals." *Electronic Pet Doors, Dog Fences, Bark & Training Collars*, www.hitecpet.com/autpetdoorfo1.html.

Public Projects

23. Vincent, James. "An Amazon Employee Made an AI-powered Cat Flap to Stop His Cat from Bringing Home Dead Animals." *The Verge*, 30 June 2019, www.theverge.com/tldr/2019/6/30/19102430/amazon-engineer-ai-powered-catflap-prey-ben-hamm.
24. "Cat Door with Pet Recognition." *Hackster.io*, Microsoft, 8 June 2017, www.hackster.io/windowsiot/cat-door-with-pet-recognition-514dac.