Table of Contents

Overview                                       3
• Parts
• Additional Required Parts
• Optional Parts

Train a Model with Lobe                       6
• Take Some Photos
• Collect your images in Lobe
• Test your model
• Export your Model

Raspberry Pi Setup                             9
• Hardware Setup
• Installing Lobe SDK
• Additional CircuitPython Libraries
• Copy the Model onto the Pi

3D Printing                                    10
• Parts List

Assembly                                       11
• Raspberry Pi Standoffs
• Install Standoffs
• Connect Flex Cable
• Secure Pi to the Stand
• Servo Installation
• Camera Housing Installation
• Camera Installation
• Attach the Keys
• Attach the Wheel
• Finish up the Wiring
• Attach the Rubber Feet

Python Code                                    19
• Run the Code
• Code Walkthrough
Overview

Do you have cats that love to jump up on counters and you don't want to have to constantly monitor to make sure your cats are behaving? With Microsoft Lobe and machine learning, you can easily set up a Raspberry Pi to monitor things for you and make sure your counters are free of kitty paw prints.

This project will train your cats to stay off the counter by detecting a cat on the counter using Machine Learning and activating a servo to jingle some keys and scare your cat away. This complete project includes a 3D printable stand that holds everything together to make it really convenient.

By using machine learning, you will no longer need to be on high alert every time you leave some meat on the counter or anything else enticing to the average cat.

Parts

Microsoft Machine Learning Kit for Lobe with Raspberry Pi 4 4GB
Machine learning is a transformative tool that's redefining how we build software—but up until now, it was only accessible to a small group of experts. At Adafruit, we...
https://www.adafruit.com/product/4963
Standard Size - High Torque - Metal Gear Servo

This high-torque standard servo now comes in a metal-gear flavor, for extra-high torque (10 kg\(\text{cm}\)) and reliability! It can rotate at least 120 degrees (60 in each direction) with a...

https://www.adafruit.com/product/1142

Continuous Rotation Servo Wheel

Plastic wheel with a cutout specially designed to allow attachment to our larger continuous rotation servo. Makes it easy to get your...

https://www.adafruit.com/product/167

STEMMA JST PH 2mm 3-Pin to Male Header Cable - 200mm

This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" male header plugs on the end. We’re carrying these to match up with our...

https://www.adafruit.com/product/3893

Additional Required Parts

You will also need a couple of other items. Around 8 old keys or something similar which rattles. At least 4 zip ties.

Optional Parts

To make use of the 3D printable stand, you will need these parts:
Black Nylon Machine Screw and Stand-off Set – M2.5 Thread
Totaling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You’ll have enough screws, nuts, and hex standoffs to fuel your maker...
https://www.adafruit.com/product/3299

1/4" Screw with D-Ring - for Cameras / Tripods / Photo / Video
Whaddya got a screw loose or something? This is an industry standard 1/4" Photo Screw with a handy D-ring for tightening purposes. This will fit...
https://www.adafruit.com/product/2629

Button Hex Machine Screw - M4 thread - 8mm long - pack of 50
Cute as a button, these button-head hex screws are what we suggest for putting together a project with our slotted extruded aluminum. Use a 2.5mm hex wrench to attach/detach. This...
https://www.adafruit.com/product/1160

Little Rubber Bumper Feet - Pack of 4
Keep your electronics from going barefoot, give them little rubber feet! These small sticky bumpers are our favorite accessory for any electronic kit or device. They are sticky, but...
https://www.adafruit.com/product/550
Train a Model with Lobe

The first thing you will do is train a Machine Learning model so that Lobe can recognize your cat or cats. If you only have a single cat, you will want to train it for Cat and No Cats and the code should work fine without any changes.

Take Some Photos

First, you will want to capture some photos of your cat in the place it shouldn't be. Cat treats work well for enticing the cat to hold still while taking photos. You will want to get as much variety as possible in the photo content, so that Lobe is able to more accurately detect if a cat is there or not in a variety of different conditions.

You can vary it by moving around different dishes and taking it from different angles. You will want to take photos from where you are planning on keeping the setup for better accuracy. You may also want to have your cat facing various angles.

Collect your images in Lobe

Once you are happy with your photo set, download and install Lobe from the link below:

Download Lobe
Open Lobe and create a new project.

From the top right, select Import and choose Images in the drop-down menu.
Cat treats work great for distracting cats while you take photos of them.

In the Bottom Left corner, add a label such as **Cat**, **No Cats**, or **Multiple Cats**. If you use different names, you can always change them in the code to reflect the labels you chose.

Go ahead and add at least 10 photos per label and try and keep it to 2-3 labels. The more photos that you use to train each label, the more accurate the predictions will be. Remember that you will need a diverse set of pictures without cats as well because the items on the counter tend to change over time, so a pot that it doesn’t recognize could be confused for a cat.

**Test your model**

In Lobe, switch to the Use tab in the left menu and select either Images to test new images or Camera to test what the camera is seeing.

Click the green button when your model predicts the correct label. This will add the image with the correct label to your dataset.

Click the red button when your model predicts the wrong label. You can then provide the correct label and the image will get added to your dataset.

If you find that one of the images is consistently confusing the model, try collecting more images for the corresponding label.
Export your Model

Next, export your Lobe model to use on the Raspberry Pi. We'll use TensorFlow Lite which is a format that is optimized for mobile and edge devices, like the Pi.

In Lobe, navigate to the Use tab and click Export.

Select TensorFlow Lite and select a location to save the model. We'll transfer the model to our Raspberry Pi later in the next step.

Raspberry Pi Setup

Hardware Setup

Before starting, set up your Pi and BrainCraft HAT. Follow this primary guide for the BrainCraft to configure these services:

- Blinka
- Fan Service
- Display Module

Installing Lobe SDK

Connect to your Pi using SSH and run the following script to install the Lobe SDK:

```
cd ~
sudo bash lobe-rpi-install.sh
```
Additional CircuitPython Libraries

You will need a few more CircuitPython libraries which can be installed with this command:

```
pip3 install adafruit-circuitpython-motor adafruit-circuitpython-dotstar
```

Copy the Model onto the Pi

You will want to establish an FTP or SCP connection to transfer the file onto your Pi. If you're not sure how to do that, you can check out this Microsoft Lobe guide. On the Pi, you'll want to make sure a folder called model exists in your home directory, otherwise, you'll want to create it:

```
cd ~
mkdir model
```

Copy saved_model.tflite and signature.json from your exported Lobe model to the model directory on the Pi.

We'll cover uploading the code to the Pi in the Code Walkthrough section.

3D Printing

Parts List

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below.

File names

Cat Detector.stl

Download STL file

Download CAD source files
Assembly

Raspberry Pi Standoffs
Use the following hardware to install standoffs to the Raspberry Pi.

- 4x M2.5 x 12mm FF standoffs
- 4x M2.5 x 4mm long machine screws

Install Standoffs
Insert machine screws through the four mounting holes on the Raspberry Pi. Fasten the standoffs onto the machine screws.

These standoffs will be used to secure the Raspberry Pi to the stand.
Connect Flex Cable
Insert and install the flex PCB cable on to the camera connector on the Raspberry Pi. Thread the flex PCB cable through the slot on the BrainCraft HAT – Pull the cable all the way through. Then, install the BrainCraft HAT on top of the Raspberry Pi. Firmly press the headers together.
Secure Pi to the Stand

Use 4x M2.5 x 10mm long machine screws to secure the Raspberry Pi to the stand. Place the Raspberry Pi with four standoffs that you installed earlier over the holes on the bottom of the stand. The BrainCraft HAT should be facing towards you.
Servo Installation

Install the Servo in the large rectangle opening on the front of the stand. The fit is a bit tight, but it can be worked in.

Use 4x M4 x 8mm long machine screws to secure the servo to the stand.
Camera Housing Installation
Using the 1/4" Screw with D-Ring, attach the backside of the housing onto the stand.
Camera Installation

Slip the camera Ribbon cable through the slot under the housing making sure the contacts are facing away from the camera.

Make sure the plastic cable holder is gently pulled out. Insert the ribbon cable into the camera and slide the cable holder back up into place.

Put the camera in the housing and attach the cover.
Attach the Keys
Attach about 2 keys per zip tie, attach the keys evenly spaced around the wheel. You will want to keep the Zip ties kind of loose so they can still jingle, but no so loose that the keys end up flopping behind the wheel.

Place the fine pitched screw that came with the servo into the wheel.

Attach the Wheel
Using the screw in the wheel, attach it to the servo. The weight may make it feel slightly unstable until you attach the feet to the bottom.
Finish up the Wiring
Connect the pins from the JST connector to the servo as follows:

Red on the JST connector to Red on the Servo
White on the JST connector to Yellow or Orange on the Servo
Black on the JST connector to Black or Brown on the Servo
Once connected, tuck the wiring out of the way. Underneath the Pi works very well.
Attach the Rubber Feet
For the Final touch, attach some rubber feet to the underside. Try and place the ones in the front as close to the edge as possible. This helps with the balance of the entire stand.

Congrats! The project is assembled and ready for use.

Python Code
To download the code to your computer, you can click on the Download Project Bundle link below, and uncompress the .zip file. Inside the zip file are a couple of CircuitPython folders. You can use either one. Since the necessary files are already on your Pi, you can ignore the lib folder.

```python
LABEL_CAT = "Cat"
LABEL_MULTI_CAT = "Multiple Cats"
```
LABEL NOTHING = "No Cats"
SERVO_PIN = board.D12
WARNING_COUNT = 3

pwm = pwmio.PWMOut(SERVO_PIN, duty_cycle=0, frequency=50)
servo = servo.Servo(pwm, min_pulse=400, max_pulse=2400)

# Boiler Plate code for buttons and joystick on the braincraft
BUTTON_PIN = board.D17
JOYDOWN_PIN = board.D27
JOYLES_PIN = board.D22
JOYUP_PIN = board.D23
JOYRIGHT_PIN = board.D24
JOYSELECT_PIN = board.D16

buttons = [BUTTON_PIN, JOYUP_PIN, JOYDOWN_PIN,
JOYLES_PIN, JOYRIGHT_PIN, JOYSELECT_PIN]

for i, pin in enumerate(buttons):
    buttons[i] = DigitalInOut(pin)
    buttons[i].direction = Direction.INPUT
    buttons[i].pull = Pull.UP

button, joyup, joydown, joyleft, joyright, joyselect = buttons

class Input(Enum):
    BUTTON = auto()
    UP = auto()
    DOWN = auto()
    LEFT = auto()
    RIGHT = auto()
    SELECT = auto()

def get_inputs():
    inputs = []
    if not button.value:
        inputs.append(Input.BUTTON)
    if not joyup.value:
        inputs.append(Input.UP)
    if not joydown.value:
        inputs.append(Input.DOWN)
    if not joyleft.value:
        inputs.append(Input.LEFT)
    if not joyright.value:
        inputs.append(Input.RIGHT)
    if not joyselect.value:
        inputs.append(Input.SELECT)
    return inputs

DOTSTAR_DATA = board.D5
DOTSTAR_CLOCK = board.D6
RED = (0, 0, 255)
GREEN = (255, 0, 0)
OFF = (0, 0, 0)

dots = adafruit_dotstar.DotStar(DOTSTAR_CLOCK, DOTSTAR_DATA, 3, brightness=0.1)

jingle_count = 0

def color_fill(color, wait):
    dots.fill(color)
    dots.show()
    time.sleep(wait)

def jingle_keys(jingle_hard=False):
    global jingle_count
    jingle_count += 1
if jingle_count > WARNING_COUNT:
    jingle_hard = True
delay = 0.5 if jingle_hard else 2
loop = 5 if jingle_hard else 1
travel = 180 if jingle_hard else 135
for _ in range(0, loop):
    for angle in (0, travel):
        servo.angle = angle
time.sleep(delay)
servo.angle = None

def main():
    global jingle_count
    model = ImageModel.load('~/model')

    # Check if there is a folder to keep the retraining data, if it there isn't
    if (not os.path.exists('./retraining_data')):
        os.mkdir('./retraining_data')

    with picamera.PiCamera(resolution=(224, 224), framerate=30) as camera:
        stream = io.BytesIO()
camera.start_preview()
        # Camera warm-up time
time.sleep(2)
        label = ''
        while True:
            stream.seek(0)
camera.annotate_text = None
camera.capture(stream, format='jpeg')
camera.annotate_text = label
img = Image.open(stream)
result = model.predict(img)
label = result.prediction
confidence = result.labels[0][1]
camera.annotate_text = label
print(f'\nLabel: {label} | Confidence: {confidence*100: .2f}%', end='', flush=True)

    # Check if the current label is package and that the label has changed
    since last time the code ran
    if label == LABEL_CAT:
        # Make Servo Jingle Keys
        jingle_keys()
    elif label == LABEL_MULTI_CAT:
        jingle_keys(True)
    elif label == LABEL_NOTHING:
        jingle_count = 0
time.sleep(0.5)

    inputs = get_inputs()
    # Check if the joystick is pushed up
    if (Input.UP in inputs):
        color_fill(GREEN, 0)
        # Check if there is a folder to keep the retraining data, if it
        there isn't make it
        if (not os.path.exists(f'./retraining_data/{label}')):
            os.mkdir(f'./retraining_data/{label}')
        # Remove the text annotation
        camera.annotate_text = None
        # File name
        name = datetime.now()
        # Save the current frame
        camera.capture(
            os.path.join(  
                f'./retraining_data/{label}',
                f'{datetime.now().strftime("%Y-%m-%d_%H:%M:%S")}.jpg'
            )  
        )
# Check if the joystick is pushed down
eelif (Input.DOWN in inputs or Input.BUTTON in inputs):
color_fill(RED, 0)
# Remove the text annotation
camera.annotate_text = None
# Save the current frame to the top level retraining directory
camera.capture(
    os.path.join(
        f'./retraining_data',
        f'{datetime.now().strftime("%Y-%m-%d_%H:%M:%S")}.jpg'
    )
)
color_fill(OFF, 0)

if __name__ == '__main__':
    try:
        print(f"Predictions starting, to stop press "CTRL+C"")
        main()
    except KeyboardInterrupt:
        print("")
        print(f"Caught interrupt, exiting...")

You can now make changes to the code such as adjusting your label names or just upload directly to your Pi in the same way you did the model file, except place it in your home directory.

Alternatively, you can use `wget` to directly download the code to your Pi:

```
```

**Run the Code**

To run the code, just run it using Python:

```
python3 lobe-cat-detector.py
```

It should display what the camera sees along with a label of what is being detected. If it detects a cat, the servo should start turning. If it still detects a cat after a few seconds, the servo will start moving a bit more. Also, if multiple cats are detected, the servo will move more as soon as it detects them.
When you point it at your cat, it should display "Cat" at the top and start jingling the keys.

Code Walkthrough

Now let's go through the code by each section. Since there's so many imports, we'll skip past that for now and get into what the code is actually doing. First we have the settings you can change. Since changing labels in the model isn't as easy as in the code, you can adjust them here.
**SERVO_PIN** corresponds to the side of the board that you plugged the servo into. If you plugged it in the other side, you can change this to `board.D13`.

**WARNING_COUNT** is the number of attempts the script will make before jingling the keys even harder.

```python
LABEL_CAT = "Cat"
LABEL_MULTI_CAT = "Multiple Cats"
LABEL NOTHING = "No Cats"
SERVO_PIN = board.D12
WARNING_COUNT = 3
```

This next section initializes the servo. You can adjust the `min_pulse` and `max_pulse` if you used a different servo and want to get more range.

```python
pwm = pwmio.PWMOut(SERVO_PIN, duty_cycle=0, frequency=50)
servo = servo.Servo(pwm, min_pulse=400, max_pulse=2400)
```

This next section initialize the buttons on the BrainCraft HAT.

```python
# Boiler Plate code for buttons and joystick on the braincraft
BUTTON_PIN = board.D17
JOYDOWN PIN = board.D27
JOYLEFT PIN = board.D22
JOYUP PIN = board.D23
JOYRIGHT PIN = board.D24
JOYSELECT_PIN = board.D16

buttons = [BUTTON PIN, JOYUP PIN, JOYDOWN PIN, JOYLEFT PIN, JOYRIGHT PIN, JOYSELECT PIN]

for i, pin in enumerate(buttons):
    buttons[i] = DigitalInOut(pin)
    buttons[i].direction = Direction.INPUT
    buttons[i].pull = Pull.UP

button, joyup, joydown, joyleft, joyright, joyselect = buttons
```

These functions are used to get the current state of button pushes. They support pushing multiple buttons at the same time.

```python
class Input(Enum):
    BUTTON = auto()
    UP = auto()
    DOWN = auto()
    LEFT = auto()
    RIGHT = auto()
    SELECT = auto()

def get inputs():
    inputs = []
    if not button.value:
        inputs.append(Input.BUTTON)
    if not joyup.value:
        inputs.append(Input.UP)
    if not joydown.value:
        inputs.append(Input.DOWN)
    if not joyleft.value:
        inputs.append(Input.LEFT)
    if not joyright.value:
        inputs.append(Input.RIGHT)
    if not joyselect.value:
        inputs.append(Input.SELECT)
```
inputs.append(Input.DOWN)
if not joyleft.value:
    inputs.append(Input.LEFT)
if not joyright.value:
    inputs.append(Input.RIGHT)
if not joyselect.value:
    inputs.append(Input.SELECT)
return inputs

This next section initializes the on-board DotStar LEDs and provides an easy function for filling the DotStars for a set amount of time.

DOTSTAR_DATA = board.D5
DOTSTAR_CLOCK = board.D6
RED = (0, 0, 255)
GREEN = (255, 0, 0)
OFF = (0, 0, 0)
dots = adafruit_dotstar.DotStar(DOTSTAR_CLOCK, DOTSTAR_DATA, 3, brightness=0.1)
def color_fill(color, wait):
    dots.fill(color)
    dots.show()
    time.sleep(wait)

This next section contains the `jingle_keys` function. The `jingle_count` is used to keep track of the number of warnings that have been given to the cat. If `jingle_hard` is `True`, then it will jingle the keys harder.

jingle_count = 0
def jingle_keys(jingle_hard=False):
    global jingle_count
    jingle_count += 1
    if jingle_count &gt; WARNING_COUNT:
        jingle_hard = True
    delay = 0.5 if jingle_hard else 2
    loop = 5 if jingle_hard else 1
    travel = 180 if jingle_hard else 135
    for in range(0, loop):
        for angle in (0, travel):
            servo.angle = angle
            time.sleep(delay)
        servo.angle = None

Finally, we get to the main section of the code. The first bit here will load the model you created earlier. It will then capture the data from the camera and feed it into the model, which will return a label and a confidence value.

global jingle_count
model = ImageModel.load('~/model')

# Check if there is a folder to keep the retraining data, if it there isn't make it
if (not os.path.exists('./retraining_data')):
    os.mkdir('./retraining_data')

with picamera.PiCamera(resolution=(224, 224), framerate=30) as camera:
    stream = io.BytesIO()
camera.start_preview()
# Camera warm-up time
time.sleep(2)
label = ''
while True:
    stream.seek(0)
    camera.annotate_text = None
    camera.capture(stream, format='jpeg')
    camera.annotate_text = label
    img = Image.open(stream)
    result = model.predict(img)
    label = result.prediction
    confidence = result.labels[0][1]
    camera.annotate_text = label
    print(f'\rLabel: {label} | Confidence: {confidence*100: .2f}\%', end=' ', flush=True)

This section will compare the detected label and either jingle the keys or reset the jingle count.

# Check if the current label is package and that the label has changed since last time the code ran
if label == LABEL_CAT:
    # Make Servo Jingle Keys
    jingle_keys()
elif label == LABEL_MULTI_CAT:
    jingle_keys(True)
elif label == LABEL_NOTHING:
    jingle_count = 0
    time.sleep(0.5)

This last section will check the inputs and store the currently detected image in a retraining_data folder for improving the accuracy of the detection.

Press up to save the image in a subfolder with the name of the label that it was detected with.

Press down to save the image inside the root of the retraining_data folder.

inputs = get_inputs()
# Check if the joystick is pushed up
if (Input.UP in inputs):
    color_fill(GREEN, 0)
    # Check if there is a folder to keep the retraining data, if it there isn't make it
    if (not os.path.exists(f'./retraining_data/{label}')): os.mkdir(f'./retraining_data/{label}')
    # Remove the text annotation
    camera.annotate_text = None
    # File name
    name = datetime.now()
    # Save the current frame
    camera.capture(os.path.join(f'./retraining_data/{label}', f'{datetime.now().strftime("%Y-%m-%d_%H:%M:%S")}.jpg'))
}
# Check if the joystick is pushed down
elif (Input.DOWN in inputs or Input.BUTTON in inputs):
    color_fill(RED, 0)
    # Remove the text annotation
    camera.annotate_text = None
    # Save the current frame to the top level retraining directory
    camera.capture(
        os.path.join(
            f'./retraining_data',
            f'{datetime.now().strftime("%Y-%m-%d_%H:%M:%S")}.jpg'
        )
    )
    color_fill(OFF, 0)

This final bit is just the first thing that is run when the Python script is started. It displays messages and calls the `main()` function.

```python
if __name__ == '__main__':
    try:
        print(f"Predictions starting, to stop press \"CTRL+C\"")
        main()
    except KeyboardInterrupt:
        print("")
        print(f"Caught interrupt, exiting...")
```

Have fun using the project!