**CCS-модель детекции изображений**

Всем доброго дня. Недавно закончила продвинутый курс от Deep Lerning и вашему вниманию предлагаю часть своего MVP-проекта с использованием модели детекции изображений. Курс объемный, много свежей информации. Мне, как закончившей прикладную математику и часто по работе соприкасающейся с искусственным интеллектом (нейросетки, генетика, fuzzy logics) было не сильно сложно, но мега-увлекательно за счет того, что ребята очень заинтересованные и рассказывали про свежие интересные модели, еще и на русском языке. Приятно видеть, что ИИ-сфера в нашей стране тоже не стоит на месте.

Это учебный проект, поэтому не стала его выгружать в облачные хранилища в открытый доступ. Я еще подумаю об этом ☺

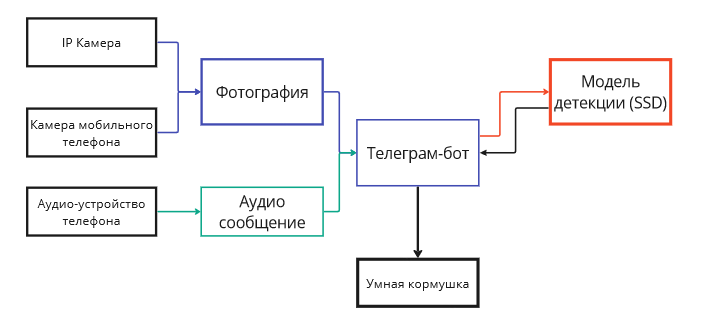
Сейчас большую популярность получили системы умных домов, которые позволяют автоматизировать некоторые регулярные повторяющиеся действия, наступающие при определенных условиях. В этом направлении есть целый кластер задач, связанных не только с управлением функциями бытовых устройств голосовыми командами (или удаленно), но и с программированием определенных действий при определенных условиях.

Так, например, для собственника дома, имеющего домашних животных, можно сформулировать следующую проблематику: как покормить своего кота (собаку), т.е. открывать кормушку только для него, чтобы корм, особенно в отсутствие хозяина, не растащили другие животные (другие коты, собаки, птицы, дикие животные).

В целях данного проекта в качестве задачи выбрана автоматизация кормления домашнего животного, кошки. В качестве решения данной проблемы предлагается использование моделей детекции и классификации, изученные на курсе.

**Цель проекта**: разработать компонент детекции домашнего животного для последующей интеграции в систему умного дома.

**Схема целевого проекта**



Реализуемый компонент детекции домашнего животного должен выполнять следующие функции, которые можно разделить на 3 блока:

1 блок: детекция по фотографии

1. Сервис получает фотографию
2. Отправляет изображение в модель детекции
3. Модель детекции идентифицирует животное
4. В случае, если это питомец пользователя, отправляется команда на открытие кормушки
5. Владельцу в чат-бот отправляется сообщение, что кормушка открыта.

2 блок: потоковая детекция по видео

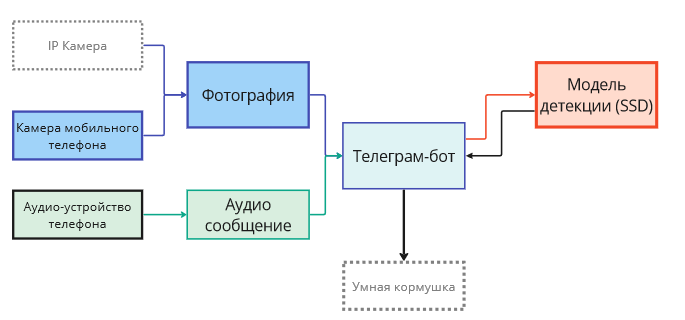
1. Сервис интегрирован с потоковым видео с IP-камеры
2. Модель детекции идентифицирует животное
3. В случае, если это питомец пользователя, отправляется команда на открытие кормушки
4. Владельцу в чат-бот отправляется сообщение, что кормушка открыта.

3 блок: голосовое управление Сервисом

1. Пользователь отправляет голосовую команду
2. Сервис переводит аудио в текст
3. Сервис классифицирует сообщение: команда или нет
4. В случае команды из заданного интегрированного списка сервис отправляет команду соответствующему сервису, например, кормушке, либо же любому другому устройству, планируемому к интеграции.

**MVP проекта или минимально работающая версия**

В виду ограниченности по времени в целях данной работы разрабатываем минимально работающий продукт (демо), оставив за рамками работы интеграцию с самим умными устройствами (IP Камерой и Умной кормушкой)



Cо следующим содержанием:

1. Сервис получает изображение посредством интерфейса телеграмм-бота через загрузку фотографии или же через камеру
2. Отправляет полученное изображение в модель детекции. К модели детекции предъявляется требование определения класса "кота/кошки", дообучение на фотосете конкретной заданной кошки оставим за скоупом данной работы.
3. В случае детекции домашнего питомца отправляет ответное сообщение "Пришел ваш кот. Кормушка открыта". Непосредственно интеграция с умной кормушкой и, соответственно, отправка команды на открытие дверцы выходит за рамки данного проекта. Т.е. В целях данной работы это функция - заглушка.
4. Владелец в любой момент времени может отдать голосовую команду, например, на закрытие кормушки.
5. Команда из голосовой преобразуется в текст. В целях данной работы это функция - заглушка, где текстовое сообщение только отправится пользователю

**Стек**

В рамках данной работы была выбрана **Single Shot MultiBox Detector model for object detection** (далее SSD-модель), разработанная компанией NVIDIA. Данная модель была выбрана как одна из наиболее перспективных моделей детекции изображений. Особенности модели:

* Исходный датасет, на котором обучена модель: COCO с Torch Hub.
* Входной размер картинки: 300х300 пикселей.
* Основа модели: ResNet-50, что позволяет ей извлекать более высокоуровневые признаки из изображений и повышает ее общую производительность.
* Улучшенная точность и скорость.
* Одновременное обнаружение объектов.
* Множество детекций на разных уровнях.
* Возможность обрабатывать изображения в реальном времени

Эта модель имеет сложную архитектуру.

**Вставить разбор картинки и код**

SSD300(

(feature\_extractor): ResNet(

(feature\_extractor): Sequential(

(0): Conv2d(3, 64, kernel\_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)

(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): MaxPool2d(kernel\_size=3, stride=2, padding=1, dilation=1, ceil\_mode=False)

(4): Sequential(

(0): Bottleneck(

(conv1): Conv2d(64, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(64, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(downsample): Sequential(

(0): Conv2d(64, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): Bottleneck(

(conv1): Conv2d(256, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(64, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

(2): Bottleneck(

(conv1): Conv2d(256, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(64, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

)

(5): Sequential(

(0): Bottleneck(

(conv1): Conv2d(256, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(128, 512, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(downsample): Sequential(

(0): Conv2d(256, 512, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): Bottleneck(

(conv1): Conv2d(512, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(128, 512, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

(2): Bottleneck(

(conv1): Conv2d(512, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(128, 512, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

(3): Bottleneck(

(conv1): Conv2d(512, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(128, 512, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

)

(6): Sequential(

(0): Bottleneck(

(conv1): Conv2d(512, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(256, 1024, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(downsample): Sequential(

(0): Conv2d(512, 1024, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): Bottleneck(

(conv1): Conv2d(1024, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(256, 1024, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

(2): Bottleneck(

(conv1): Conv2d(1024, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(256, 1024, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

(3): Bottleneck(

(conv1): Conv2d(1024, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(256, 1024, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

(4): Bottleneck(

(conv1): Conv2d(1024, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(256, 1024, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

(5): Bottleneck(

(conv1): Conv2d(1024, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(conv3): Conv2d(256, 1024, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

)

)

)

)

(additional\_blocks): ModuleList(

(0): Sequential(

(0): Conv2d(1024, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): Conv2d(256, 512, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU(inplace=True)

)

(1): Sequential(

(0): Conv2d(512, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): Conv2d(256, 512, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU(inplace=True)

)

(2): Sequential(

(0): Conv2d(512, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): Conv2d(128, 256, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU(inplace=True)

)

(3-4): 2 x Sequential(

(0): Conv2d(256, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(2): ReLU(inplace=True)

(3): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), bias=False)

(4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(5): ReLU(inplace=True)

)

)

(loc): ModuleList(

(0): Conv2d(1024, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1-2): 2 x Conv2d(512, 24, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): Conv2d(256, 24, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(4-5): 2 x Conv2d(256, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

)

(conf): ModuleList(

(0): Conv2d(1024, 324, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1-2): 2 x Conv2d(512, 486, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): Conv2d(256, 486, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(4-5): 2 x Conv2d(256, 324, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

)

)

В качестве сервиса был выбран сервис телеграм-ботов. Обоснование:

* широкая доступность на всех устройствах пользователя,
* понятный и привычный интерфейс телеграмма,
* удобство использования реализуемого функционала.

Следует обратить внимание, что обязательно перевести расчеты на GPU или на CUDA, т.к. CPU будет считать очень долго и медленно. Перевести модель в режим работы "evaluation" (предсказательном режиме без изменения весов и состояния модели). Я работала в Google Colab.

**Алгоритм создания MVP-проекта**

1 часть: подготовка к работе

* 1. Установим необходимые библиотеки

pip install numpy scipy scikit-image matplotlib

pip install pyTelegramBotAPI

pip install gtts

pip install SpeechRecognition

pip install pydub

pip install pydub[ffmpeg]

pip install pydub opencv-python-headless opuslib

* 1. Скачиваем выбранную CCS-модель
  2. Монтируем google-disk, на котором будем хранить вспомогательные материалы. Надо будет дать разрешение на работу с диском.
  3. Подгружаем необходимые библиотеки

import telebot

import cv2

import torch

import torch.nn as nn

import torch.nn.functional as F

import numpy as np

from matplotlib import pyplot as plt

import matplotlib.patches as patches

import telebot

import os

from gtts import gTTS

import speech\_recognition as sr

from pydub import AudioSegment

import subprocess

import time

2 часть: реализация логики, которая будет обрабатывать данные, полученные от модели

Напишем функцию, которая производит процедуру детекции и классификации объектов на изображении. В рамках проекта нам необходимо опознать на картинке кота, поэтому в качестве возвращаемого параметра выдадим булевую переменную is\_detected\_cat, которая в случае обнаружения кота на изображении принимает значение True

3 часть: создание телеграмм-бота через @BotFather

3.1. Создаем телеграмм-бота с помощью @BotFather и в целях безопасности сохраняем выданный токен в отдельный файл “token.txt”

3.2. Пишем логику бота. Наш бот будет уметь:

- здороваться

- так как в рамках MVP-проекта у нас нет полноценной интеграции с камерой, то модель детекции будет работать на примере загруженных в него фотографий. На них будет находить кота и в этом случае открывать кормушку. Справедливости ради стоит заметить, что реагировать данная модель будет на любого кота. Чтобы модель реагировала только на какого-то одного определенного кота, надо собрать дата сет и дообучить скачанную и сохраненную модель на поиск заданного кота.

- открывать кормушку

- обрабатывать команды пользователя в виде голосовых сообщений, у нас будет обрабатывать только одну команду «закрой кормушку»

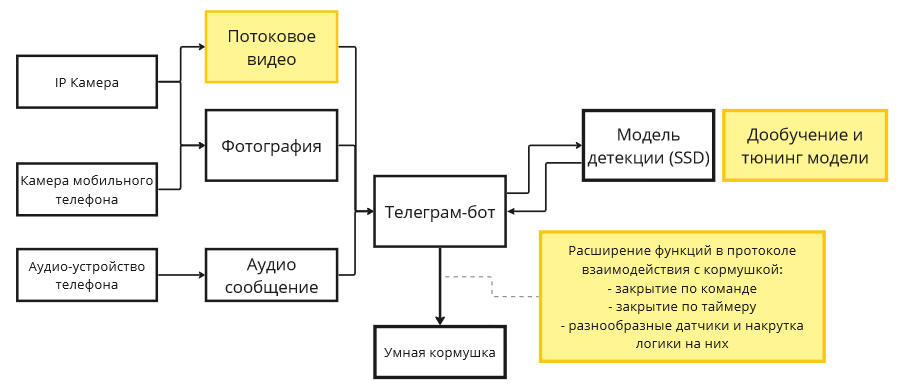
**3 часть: запуск бота-детектора**

bot.polling()

или можно загрузить бота на облако, например, Яндекс, тогда от будет работать без необходимости держать открытым google colab

**А что дальше…**

Вот такое интересное и практичное применение возможно моделям искусственного интелекта, а не вот эти вот все захваты мира)))



Главное желание, а уж идеи найдутся. Например:

1. Дообучение и тюнинг модели:

* Можно собрать собственный датасет пользователя с его питомцем и дообучить модель в целях повышения качества детекции именно домашнего питомца.
* Поработать с параметрами модели.

1. Расширение функций в протоколе взаимодействия с кормушкой:

* закрытие по команде
* закрытие по таймеру
* разнообразные датчики и накрутка логики на них

1. Добавление возможности детекции онлайн изображения питомца по потоковому видео.

Источники

Deep Lerning

<https://habr.com/ru/articles/550456/>