**Настройка среды + AWS**

То, что не развернуто в промышленной эксплуатации, -

не в счет. Не автоматизировано — значит не работает.

Ной Гифт. Прагматичный ИИ

Веб-сервисы Amazon (AWS) — безоговорочный лидер среди облачных сервисов. (<https://www.amazon.jobs/en/principles>[)](https://www.amazon.jobs/en/principles)/). За последние годы в связи с развитием машинного обучения AWS сделали колоссалиный скачок вперед.

Настройка основной среды разработки:

1. Создание сборочного файла

2. виртуальной среды

3. добавление в bash команды быстрого доступа для перехода в вирт среду

4. автоматическое выполнение профиля AWS в тек среде и т. д.

Сборочный файл Makefile — своеобразная точка отсчета для сборки проекта. (C его помощью сборка выполняется одной командой - make setup).

Создание псевдонима в .bashrc или .zshrc обычно бывает очень удобным.





Настройка аккаунта AWS: [https://docs.aws.amazon.com/iot/latest/developerguide/setting-up.html#create-iam-user](https://docs.aws.amazon.com/iot/latest/developerguide/setting-up.html" \l "create-iam-user)

Управление доступом AWS: <https://docs.aws.amazon.com/IAM/latest/UserGuide/access.html>

Установка SDK на Raspberry: <https://docs.aws.amazon.com/iot/latest/developerguide/connecting-to-existing-device.html>

**Обучение классификатора**

Для того чтобы обучить нейросеть классификации нам понадобится сгенерировать для нее размеченные данные. От качества и количества данных будет зависеть качество нейронной сети, а от идеи, положенной в их разметку будет зависеть то, основываясь на чем нейросеть будет принимать решения.

Сначала будем сохранять кадры, на которых будет находиться объект, с именем `object\_N.jpeg`, где N — номер кадра. Для этого создаем программу `save\_frame.py`. Программа запускается командой:

python3 save\_frame.py --label 'object'

«object» - первая часть имени сохраненного кадра. По умолчанию сохраняется каждый двадцатый кадр, но это можно изменить, указав `--delay i`, после чего будет сохраняться каждый i-й кадр. Кадры по умолчанию сохраняются в папку `images`.

Для получения информации о дополнительных опциях программы наберите:

python3 save\_frame.py --help

Cначала при запуске открывается окно, в котором просто воспроизводится поток, захватываемый видеокамерой. Это нужно для калибровки положения камеры. При нажатии «R» происходит переключение в режим сохранения кадров и обратно, при нажатии «Q» происходит выход из программы.

Мы можем сразу расширить базу данных (augmentation), поворачивая изображение. Дело в том, что планируется выполнить нормировку после выделения объектов на изображении, которая сделает модель нечувствительной к локальному расположению объекта и его размерам (ну это еще, конечно, предстоит), но это не компенсирует поворот объекта. Поэтому сохраняется каждый захваченный кадр, а затем его копии, повернутые на 10 и 15 градусов по часовой и на 15 градусов против часовой стрелки.

Для справки: для обнаружения ключевых точек человека предварительно обученная модель возвращает ключевые точки в следующем порядке:

COCO\_PERSON\_KEYPOINT\_NAMES = [ 'nose', 'left\_eye', 'right\_eye', 'left\_ear', 'right\_ear', 'left\_shoulder', 'right\_shoulder', 'left\_elbow', 'right\_elbow', 'left\_wrist', 'right\_wrist', 'left\_hip', 'right\_hip', 'left\_knee', 'right\_knee', 'left\_ankle', 'right\_ankle']

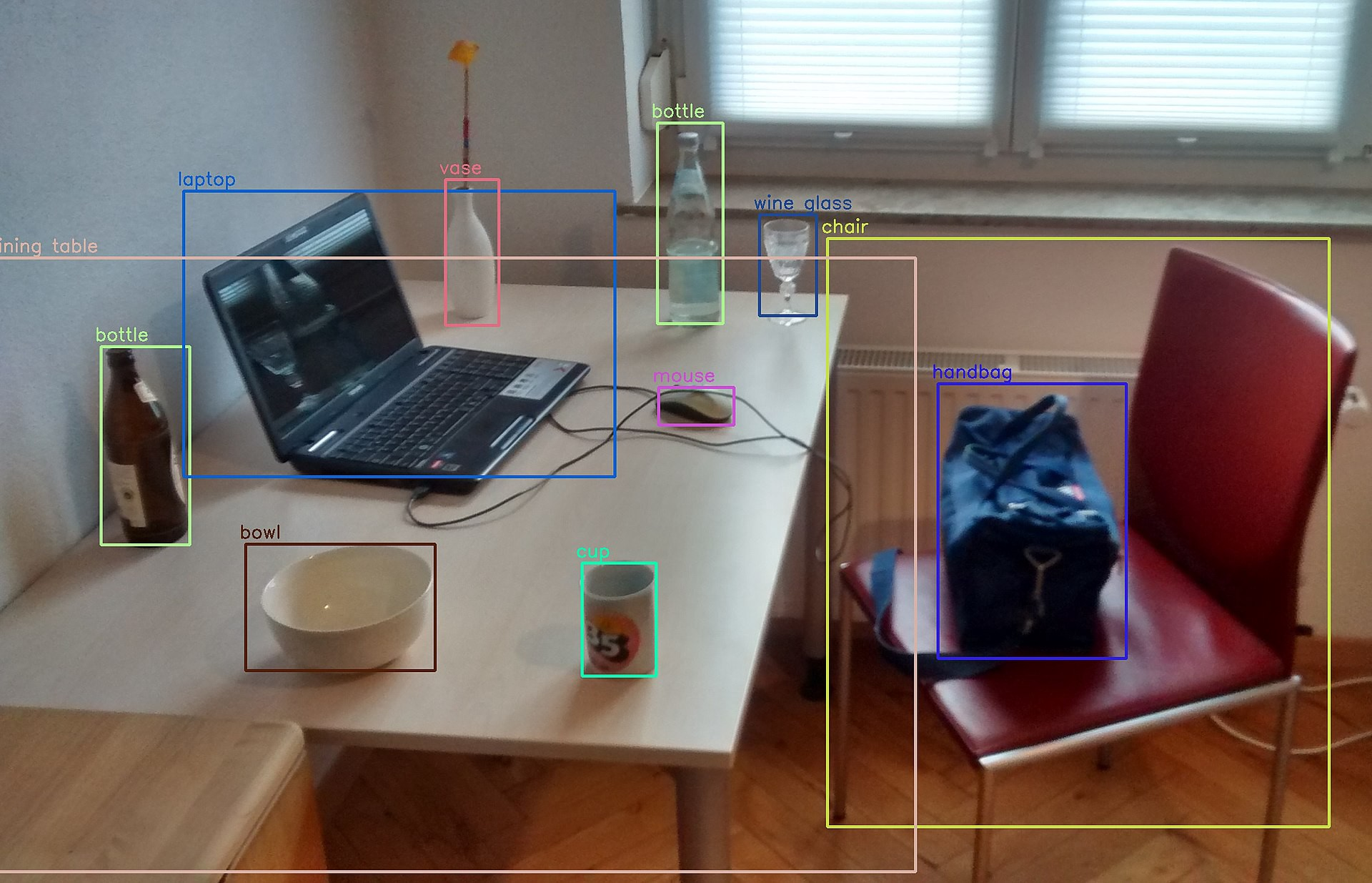
**Обучение нейросети для детектирования РЛС**

****

Для того чтобы обучить нейросеть для детектирования понадобится набор данных, размеченный специальным образом. Посмотрим, как могут быть размечены данные на примере MSCOCO (https://cocodataset.org/#home):

1) Данные для обнаружения и одновременной сегментации

2) Данные для обнаружения с ограничивающими прямоугольниками (эта задача проще, как минимум можно обвести сегментированный объект прямоугольником и получить тот же результат)



Нам потребуется самим сгенерировать подобную базу данных.

Для оценки работы нейросети используются специальные показатели. Следующие 12 показателей используются для характеристики работы детектора объектов на COCO и предложены создателями COCO+код для оценки (https://cocodataset.org/#detection-eval):

**Средняя точность (AP):**

AP % AP при IoU = 0,50: 0,05: 0,95 **(основной показатель сложности)**

AP IoU = 0,50 % AP при IoU = 0,50 (метрика PASCAL VOC)

AP IoU = 0,75 % AP при IoU = 0,75 (строгая метрика)

**AP в масштабах:**

AP малый % AP для небольших объектов: площадь <32 2

AP средний % AP для средних объектов: 32 2 <площадь <96 2

AP большой % AP для больших объектов: площадь> 96 2

**Средний отзыв (AR):**

AR макс = 1 % AR дано 1 обнаружение на изображение

AR макс = 10 % AR дает 10 обнаружений на изображение

AR макс = 100 % AR с учетом 100 обнаружений на изображение

**AR в разных масштабах:**

AR маленький % AR для небольших объектов: площадь <32 2

Среда AR % AR для средних объектов: 32 2 <площадь <96 2

AR большой % AR для больших объектов: площадь> 96 2

Теперь определим архитектуру модели. Мы можем написать и обучить модель с нуля самостоятельно, но почему бы не воспользоваться готовыми обученными моделями (<https://pytorch.org/vision/stable/models.html>) и дообучить их под конкретную задачу. Так реализуется принцип переноса обучения (Transfer learning). Согласно документации, все изображения должны включать значения в диапазоне [0,1] и нормализованы с помощью **mean = [0.43216,0.394666, 0.37645], std = [0.22803, 0.22145, 0.216989]**