


Обнаружение объектов с помощью Глубокого Обучения на Raspberry Pi

 Анна Гуляева

12 месяцев ago

В мире существует множество устройств с ограниченной памятью и небыстрыми процессорами, такие как, например, мобильные телефоны и Raspberry Pi, которые не могут запускать сложные модели глубокого обучения. Эта статья показывает, как вы тем не менее можете распознавать объекты при помощи Raspberry Pi.

Если вам не терпится, вы можете пролистать пост до ссылок на GitHub.

Почему Raspberry Pi?

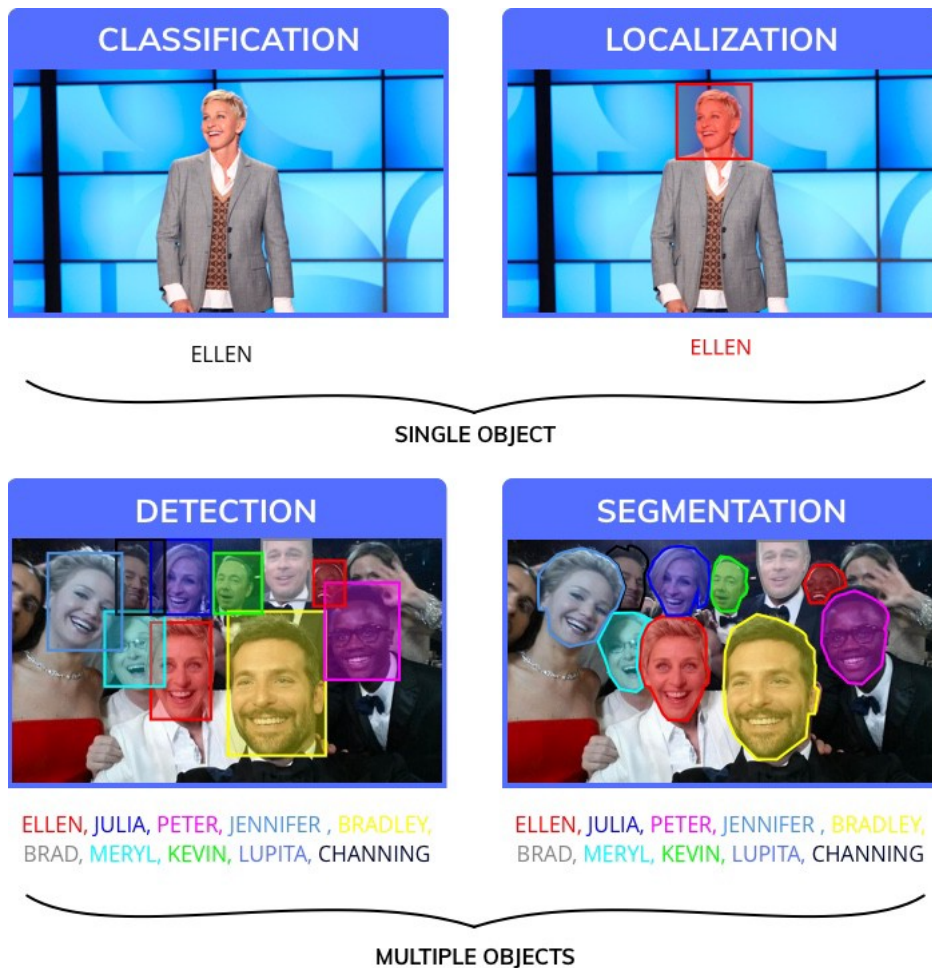
Raspberry Pi покорила сердца людей: в мире продано около 15 миллионов устройств, на которых создаются отличные проекты. Учитывая популярность Raspberry Pi и глубокого обучения, мы решили создать систему распознавания любого объекта при помощи Pi.



Что такое обнаружение объектов?

За 20 миллионов лет эволюции человеческое зрение сильно развилось. 30% нейронов мозга человека работают над обработкой визуальной информации, для осязания этот показатель составляет 8%, а для слуха — 3%.

По сравнению с машинами, у людей есть два больших преимущества: стереоскопическое зрение и бесконечное количество данных для обучения (за пять лет жизни ребенок обрабатывает примерно 2,7 миллиарда изображений со скоростью 30fps).



Чтобы имитировать производительность людей, учёные разбили задачу визуального восприятия на четыре категории:

1. Классификация — присваивание ярлыка целому изображению.
2. Локализация — определение рамки вокруг объекта и его описание.
3. Обнаружение объектов — создание нескольких рамок на изображении.
4. Сегментация изображения — создание точных сегментов, содержащих объекты, на изображении.

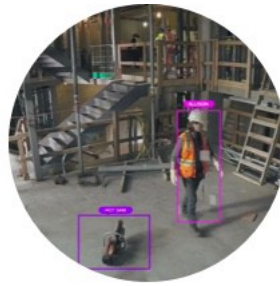
Обнаружение объектов применяется в нескольких случаях. Хотя сегментация изображения дает более точный результат, её проблемой является сложность создания данных для обучения. Человек в 12 раз дольше сегментирует изображение по сравнению с созданием рамок. Более того, после обнаружения объекта есть возможность сегментировать его из рамки.

Использование обнаружения объектов

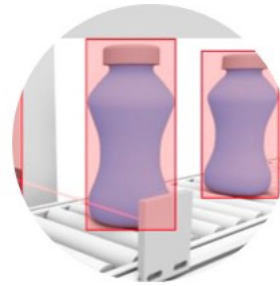
Обнаружение объектов — это важная функция, которая используется во многих индустриях. Несколько примеров показаны ниже:



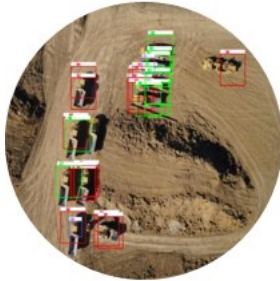
AUTONOMOUS VEHICLES



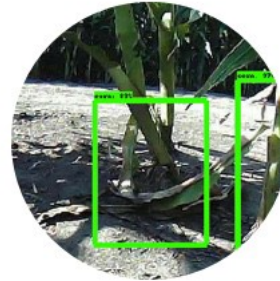
WORKPLACE AUTOMATION



MANUFACTURING



DRONE IMAGERY



AGRICULTURE



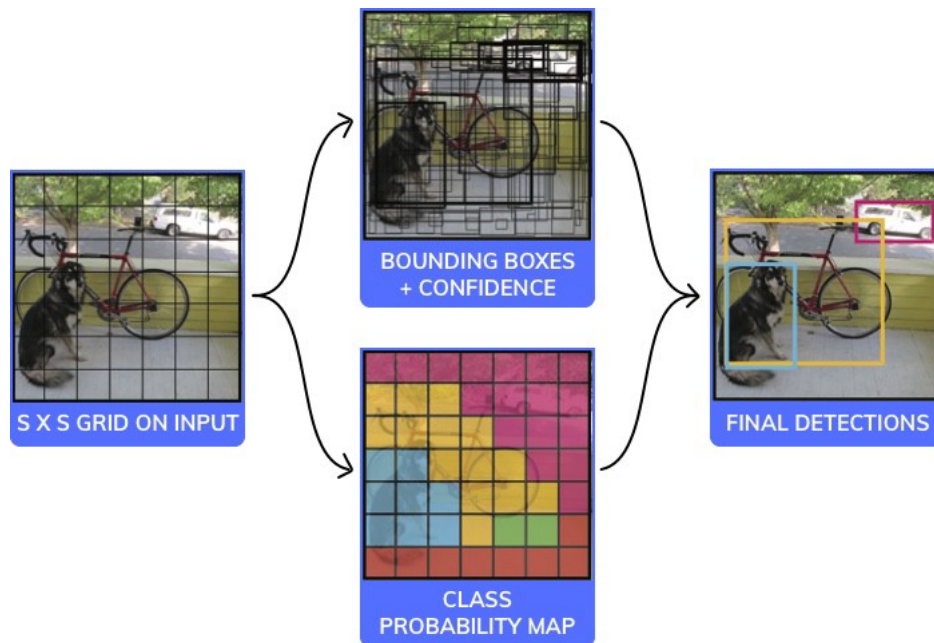
AUGMENTED REALITY

Как использовать обнаружение объектов для решения моих проблем?

Обнаружение объектов можно использовать для ответа на многие вопросы. Вот несколько категорий:

1. Присутствует ли объект на моем изображении? Пример: нет ли в моем доме грабителя?
2. Где находится объект на изображении? Пример: когда автомобиль перемещается, важно знать, где находятся разные объекты.
3. Сколько объектов на изображении? Пример: сколько коробок стоит на полке в складе?
4. Какие типы объектов есть на изображении? Пример: какое животное находится в конкретной части зоопарка?
5. Каков размер объекта? При помощи статичной камеры просто определить размер объекта. Пример: какого размера манго?
6. Как разные объекты взаимодействуют друг с другом? Как группировка на футбольном поле влияет на результат?
7. Где находится объект во времени? Пример: отслеживание движущегося объекта вроде поезда и вычисление скорости.

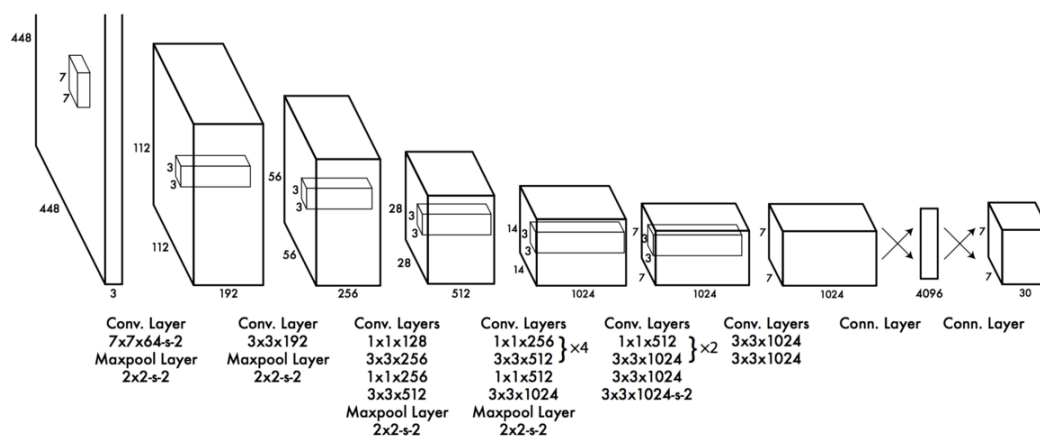
Обнаружение объектов менее чем за 20 строк кода



Существуют разные модели и архитектуры для обнаружения объектов. Каждая отличается по скорости, размеру и точности. Мы выбрали один из самых популярных алгоритмов — YOLO, и показали, как он работает в 20 строк кода (если игнорировать комментарии).

Примечание: это псевдокод, который не должен быть рабочим примером. В нем есть черный ящик, часть со сверточной нейронной сетью, которая довольно стандартна и показана на изображении ниже. Вы можете прочитать полную статью здесь:

https://pjreddie.com/media/files/papers/yolo_1.pdf.



Архитектура сверточной нейронной сети, которая используется в YOLO

```

1  #this is an Image of size 140x140. We will assume it to be black and white (ie only one channel, it would have been
2  image = readImage()
3
4  #We will break the Image into 7 columns and 7 rows and process each of the 49 different parts independently
5  NoOfCells = 7
6
7  #we will try and predict if an image is a dog, cat, cow or wolf. Therefore the number of classes is 4

```

```

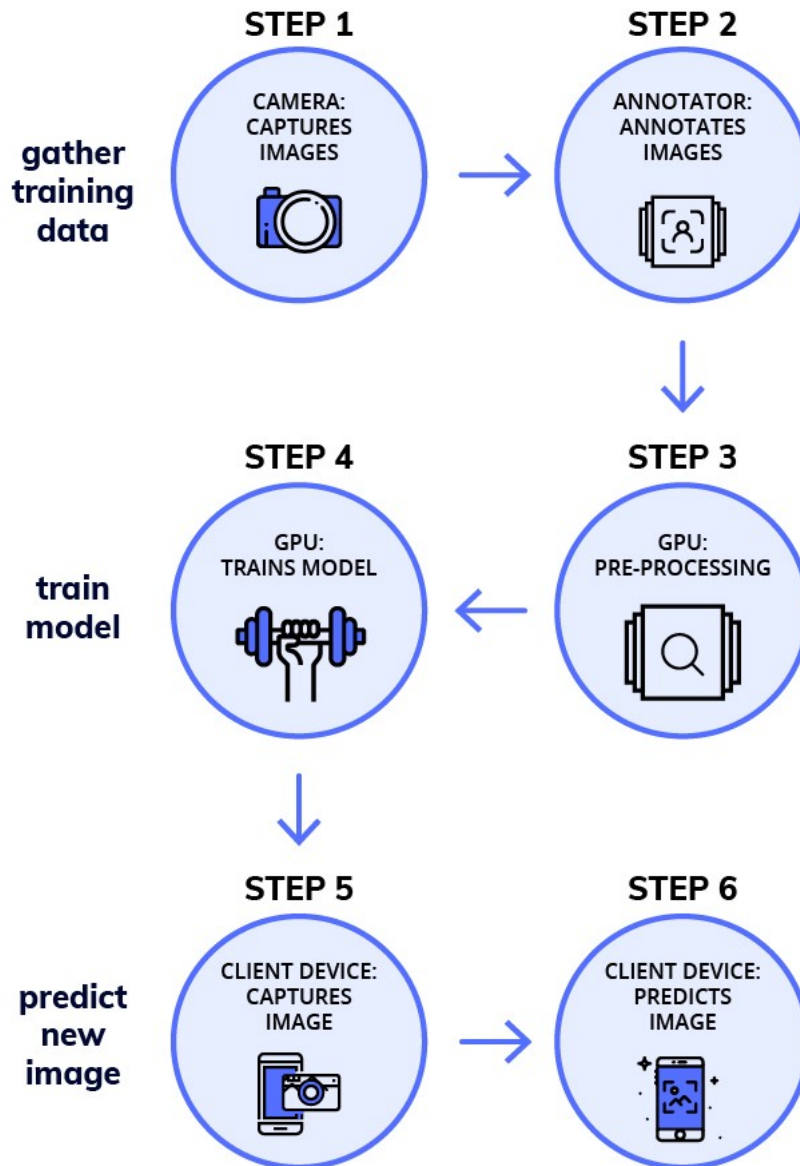
8  NoOfClasses = 4
9  threshold = 0.7
10
11 #step will be the size of step to take when moving across the image. Since the image has 7 cells step will be 140/7
12 step = height(image)/NoOfCells
13
14 #stores the class for each of the 49 cells, each cell will have 4 values which correspond to the probability of a ce
15 #prediction_class_array[i,j] is a vector of size 4 which would look like [0.5 #cat, 0.3 #dog, 0.1 #wolf, 0.2 #cow]
16 prediction_class_array = new_array(size(NoOfCells,NoOfCells,NoOfClasses))
17
18 #stores 2 bounding box suggestions for each of the 49 cells, each cell will have 2 bounding boxes, with each boundir
19 predictions_bounding_box_array = new_array(size(NoOfCells,NoOfCells,NoOfCells,NoOfCells))
20
21 #it's a blank array in which we will add the final list of predictions
22 final_predictions = []
23
24 #minimum confidence level we require to make a prediction
25 threshold = 0.7
26
27 for (i<0; i<NoOfCells; i=i+1):
28     for (j<0; j<NoOfCells;j=j+1):
29         #we will get each "cell" of size 20x20, 140(image height)/7(no of rows)=20 (step) (size of each cell
30         #each cell will be of size (step, step)
31         cell = image(i:i+step,j:j+step)
32
33         #we will first make a prediction on each cell as to what is the probability of it being one of cat,
34         #prediction_class_array[i,j] is a vector of size 4 which would look like [0.5 #cat, 0.3 #dog, 0.1 #w
35         #sum(prediction_class_array[i,j]) = 1
36         #this gives us our preidction as to what each of the different 49 cells are
37         #class predictor is a neural network that has 9 convolutional layers that make a final prediction
38         prediction_class_array[i,j] = class_predictor(cell)
39
40         #predictions_bounding_box_array is an array of 2 bounding boxes made for each cell
41         #size(predictions_bounding_box_array[i,j]) is [2,5]
42         #predictions_bounding_box_array[i,j,1] is bounding box1, predictions_bounding_box_array[i,j,2] is bo
43         #predictions_bounding_box_array[i,j,1] has 5 values for the bounding box [x,y,w,h,c]
44         #the values are x, y (coordinates of the center of the bounding box) which are within the bounding
45         #the values are h, w (height and width of the bounding box) they extend outside the cell and are in
46         #the value is c a confidence of overlap with an acutal bounding box that should be predicted
47         predictions_bounding_box_array[i,j] = bounding_box_predictor(cell)
48
49         #predictions_bounding_box_array[i,j,0, 4] is the confidence value for the first bounding box predict
50         best_bounding_box = [0 if predictions_bounding_box_array[i,j,0, 4] > predictions_bounding_box_array
51
52         # we will get the class which has the highest probability, for [0.5 #cat, 0.3 #dog, 0.1 #wolf, 0.2 #
53         predicted_class = index_of_max_value(prediction_class_array[i,j])
54
55         #we will check if the prediction is above a certain threshold (could be something like 0.7)
56         if predictions_bounding_box_array[i,j,best_bounding_box, 4] * max_value(prediction_class_array[i,j])
57
58         #the prediction is an array which has the x,y coordinate of the box, the height and the width
59         prediction = [predictions_bounding_box_array[i,j,best_bounding_box, 0:4], predicted_class]
60
61         final_predictions.append(prediction)
62
63
64 print final_predictions

```

Как создать модель глубокого обучения для обнаружения объектов?

Процесс глубокого обучения состоит из шести шагов, которые разбиты на три части:

1. Сбор данных для обучения
2. Обучение модели
3. Прогнозы на новых изображениях



Фаза 1 — сбор данных для обучения

Шаг 1. Соберите изображения (минимум 100 на один объект)

Для этой задачи вам нужно будет сто изображений для одного объекта. Попробуйте собрать данные, максимально близкие к тем, для которых вы потом будете делать прогнозы.



Шаг 2. Аннотации (нарисуйте рамки на изображениях вручную)

Нарисуйте рамки на изображениях. Вы можете использовать инструмент вроде labelImg. Вам понадобятся люди, которые будут работать над аннотациями. Это занятие может занять много времени.

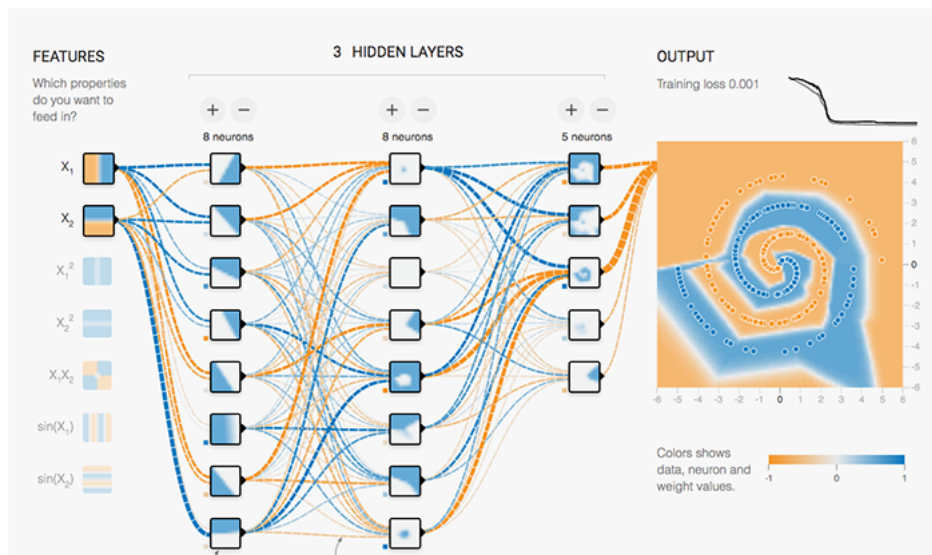


Фаза 2 — обучение модели на графическом процессоре

Шаг 3 — поиск уже обученной модели для обучения на основе переноса

Вы можете прочитать об этом больше [здесь](#). Вам нужна будет обученная модель, чтобы сократить необходимое для обучения количество данных. Без этого шага вам понадобится минимум 100 тысяч изображений для обучения модели. Вы можете найти обученные модели в этом [репозитории](#).

Шаг 4 — обучение на графическом процессоре (облачные сервисы вроде AWS/GCP или вашем собственном процессоре)



Для упрощения процесса обучения мы создали docker image, которое сделает обучение проще.

Чтобы начать обучение модели, вы можете запустить:

```
sudo nvidia-docker run -p 8000:8000 -v `pwd`:data docker.nanonets.com/pi_training -m train -a ssd_mobilenet_v1_coco -e ssd_mobilenet
```

По этой [ссылке](#) вы можете узнать больше о деталях процесса.

В docker image содержится скрипт run.sh, который можно вызвать при помощи следующих параметров.

```
run.sh [-m mode] [-a architecture] [-h help] [-e experiment_id] [-c checkpoint] [-p hyperparameters]
```

```
-h      display this help and exit
-m      mode: should be either `train` or `export`
-p      key value pairs of hyperparameters as json string
-e      experiment id. Used as path inside data folder to run current experiment
-c      applicable when mode is export, used to specify checkpoint to use for export
```

Больше деталей можно найти [здесь](#).

Чтобы обучить модель, вам нужно подобрать правильные гиперпараметры.

Подбор гиперпараметров

Искусство глубокого обучения требует поиска лучших параметров для повышения точности модели. Этот процесс — это немного магии и немного теории. Вот отличный [ресурс](#), который поможет вам найти правильные параметры.

Квантование модели (чтобы она уместилась на маленьких устройствах)

На маленьких устройствах вроде телефонов или Raspberry Pi очень мало памяти и вычислительной мощности.

Обучение нейронной сети происходит при помощи внесения крохотных изменений в веса, и этим изменениям для работы нужна точность плавающей запятой (хотя в некоторых исследованиях здесь пытаются применить и квантованные репрезентации).

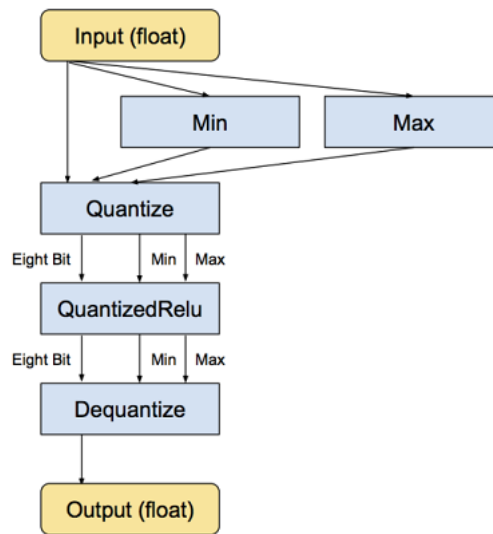
Использование обученной модели и получение вывода очень отличаются. Одно из магических свойств сетей глубокого обучения — они хорошо справляются с высоким уровнем шума в данных.

Зачем производить квантование?

Модели на основе нейронных сетей могут занимать много места на диске. Почти все пространство занимают веса нейронных связей, так как в одной модели много миллионов

таких весов.

Узлы и веса нейронной сети обычно хранятся как 32-битные числа с плавающей запятой. Цель квантования — сжать размер файлов при помощи хранения минимума и максимума каждого слоя, а затем привести каждое плавающее значение к восьмибитному целому числу. Размер файлов сокращается на 75%.



Код для квантования:

```

curl -L "https://storage.googleapis.com/download.tensorflow.org/models/inception_v3_2016_08_28_frozen.pb.tar.gz" |
  tar -C tensorflow/examples/label_image/data -x

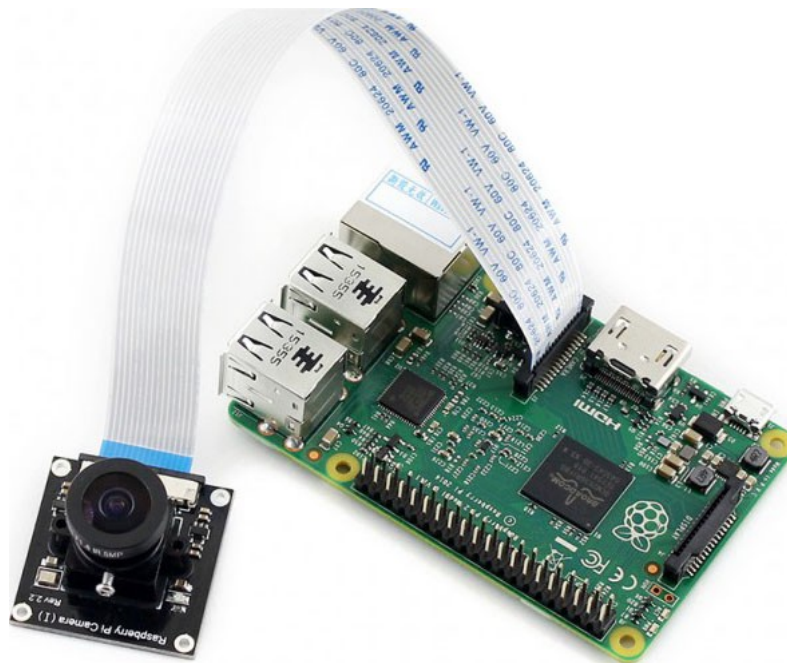
bazel build tensorflow/tools/graph_transforms:transform_graph
bazel-bin/tensorflow/tools/graph_transforms/transform_graph \
--in_graph=tensorflow/examples/label_image/data/inception_v3_2016_08_28_frozen.pb \
--out_graph=/tmp/quantized_graph.pb \
--inputs=input \
--outputs=InceptionV3/Predictions/Reshape_1 \
--transforms='add_default_attributes strip_unused_nodes(type=float, shape="1,299,299,3")
remove_nodes(op=Identity, op=CheckNumerics) fold_constants(ignore_errors=true)
fold_batch_norms fold_old_batch_norms quantize_weights quantize_nodes
strip_unused_nodes sort_by_execution_order'
  
```

Примечание: в наш docker image уже встроено квантование.

Фаза 3 — предсказание на новых изображениях

Шаг 5 — захват нового изображения посредством камеры

Вам нужна будет работающая камера Raspberry Pi. Затем захватите новое изображение.



Для инструкции по установке посмотрите эту [ссылку](#).

Шаг 6 — прогноз на основе нового изображения

Скачайте модель

Как только вы закончите обучать модель, вы можете скачать её на Pi. Чтобы экспортировать модель, запустите:

```
sudo nvidia-docker run -v `pwd`:data docker.nanonets.com/pi_training -m export -a ssd_mobilenet_v1_coco -e ssd_mobilenet_v1_coco_0 -
```

Затем загрузите модель на Raspberry Pi.

Установка TensorFlow на Raspberry Pi

В зависимости от устройства вам, возможно, нужно будет изменить установку.

```
sudo apt-get install libblas-dev liblapack-dev python-dev libatlas-base-dev gfortran python-setuptools libjpeg-dev
```

```
sudo pip install Pillow
```

```
sudo pip install http://ci.tensorflow.org/view/Nightly/job/nightly-pi-zero/lastSuccessfulBuild/artifact/output-artifacts/tensorflow-
```

```
git clone https://github.com/tensorflow/models.git
```

```
sudo apt-get install -y protobuf-compiler
```

```
cd models/research/
```

```
protoc object_detection/protos/*.proto --python_out=.
```

```
export PYTHONPATH=$PYTHONPATH:/home/pi/models/research:/home/pi/models/research/slim
```

Запустите модель для прогноза

```
python ObjectDetectionPredict.py --model data/0/quantized_graph.pb --labels data/label_map.pbtxt --images /data/image1.jpg /data/ima
```

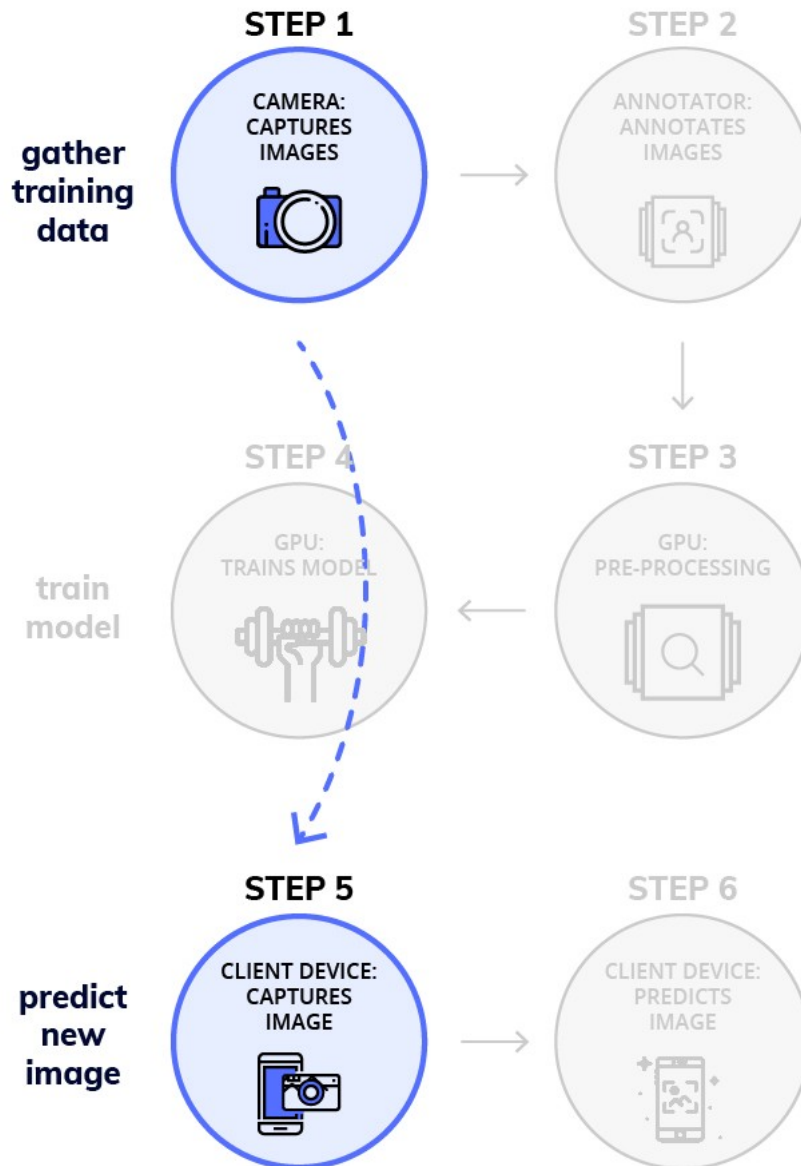
Измерение производительности Raspberry Pi

У Raspberry Pi есть ограничения по памяти и вычислительной мощности (совместимая с GPU Raspberry Pi версия TensorFlow пока недоступна). Таким образом, важно измерить, сколько времени у модели займет создание прогноза на новом изображении.

Model	Prediction Time in seconds	Graph Load Time in seconds	mAP	Training Steps
ssd_mobilenet_v1	0.72	44.06	0.76	9200
ssd_inception_v2	1.76	50.18	0.78	12000
faster_rcnn_inception_v2	9.89	78.9	0.8	20000
faster_rcnn_resnet50_lowproposals	15.9	80.83	0.82	20000
mask_rcnn_inception_v2	19.4	85.96	None	None
faster_rcnn_resnet50	34.52	87.55	0.82	20000
faster_rcnn_resnet101_coco	None	None	0.83	15000

Рабочий процесс с NanoNets

Наша цель в NanoNets — сделать работу с глубоким обучением очень простой. Обнаружение объектов — это важная область для нас, и мы создали процесс, который решает много проблем с внедрением моделей глубокого обучения.



Как NanoNets делает процесс проще:

1. Не требуется аннотация

Мы убрали потребность в аннотации изображений, у нас есть аннотаторы, которые сделают это за вас.

2. Автоматизация лучшей модели и выбор гиперпараметров

Мы автоматически обучаем лучшую модель для вас, запуская ряд моделей с разными параметрами, чтобы выбрать лучшую для ваших данных.

3. Отсутствие необходимости в дорогом оборудовании

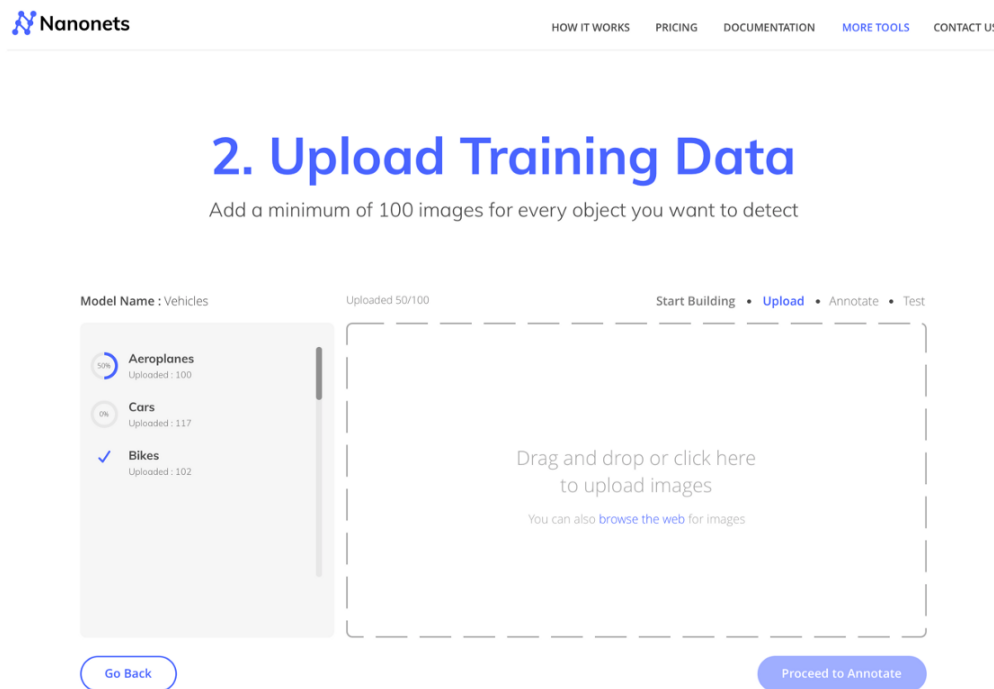
NanoNets находится в облаке и работает. не используя ваше оборудование.

4. Совместимость с мобильными устройствами вроде Raspberry Pi

Так как устройства вроде Raspberry Pi и смартфоны не созданы для сложных вычислительных задач, вы можете передать рабочий процесс в облако, которое делает все вычисления за вас.

Вот небольшой отрывок для создания прогноза на основе изображения при помощи NanoNets API

Создание своей NanoNet



Вы можете создать свою модель при помощи:

1. Используя [GUI и автоаннотацию модели](#)
2. Используя наш [API](#)

Шаг 1: клонируйте репозиторий

```
git clone https://github.com/NanoNets/object-detection-sample-python.git
cd object-detection-sample-python
sudo pip install requests
```

Шаг 2: получите бесплатный ключ API

Получите бесплатный ключ API [здесь](#).

Шаг 3: установите ключ API как Environment Variable

```
export NANONETS_API_KEY=YOUR_API_KEY_GOES_HERE
```

Шаг 4: создайте новую модель

```
python ./code/create-model.py
```

Примечание: здесь генерируется MODEL_ID, который вам нужен будет для следующего шага.

Шаг 5: установите Model ID как Environment Variable

```
export NANONETS_MODEL_ID=YOUR_MODEL_ID
```

Шаг 6: загрузите данные для обучения

Соберите изображения объекта, который вы хотите обнаруживать. Вы можете оставлять аннотации при помощи нашего веб-UI (https://app.nanonets.com/ObjectAnnotation/?appId=YOUR_MODEL_ID) или использовать инструменты с открытым исходным кодом вроде labelImg. Как только в вашем наборе данных будет готовы папки, аннотации и изображения, начните загружать набор данных.

```
python ./code/upload-training.py
```

Шаг 7 — обучение модели

Как только изображения загружены, начните обучать модель.

```
python ./code/train-model.py
```

Шаг 8 — получите состояние модели

Обучение модели занимает два часа. Вы получите письмо, как только обучение закончится. В процессе вы можете проверять состояние модели.

```
watch -n 100 python ./code/model-state.py
```

Шаг 9 — сделайте прогноз

Как только модель будет обучена, вы сможете делать прогнозы.

```
python ./code/prediction.py PATH_TO_YOUR_IMAGE.jpg
```

Код на GitHub

Обучение модели

[Код TensorFlow для обучения и квантования модели](#)

[Код NanoNets для обучения модели](#)

Прогнозы на Raspberry Pi

[Код TensorFlow для прогнозов на Raspberry Pi](#)

[Код NanoNets для прогнозов на Raspberry Pi](#)

Наборы данных с аннотациями

[Автомобили на индийских дорогах](#)

[Набор данных Coco](#)

Комментарии

Категории: [Разработка](#), [Статьи](#)

Тэги: [featured](#), [Raspberry Pi](#), [глубокое обучение](#), [для разработчиков](#), [нейронная сеть](#), [распознавание](#), [статья](#)

[Оставить комментарий](#)

AppTractor

[Работает на WordPress](#)

[Наверх](#)