электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель Общероссийская общественная организация "Академия инженерных наук им. А.М. Прохорова"
ISSN 2307-0609

 # 02, июнь 2020

УДК 004.93.12

Алгоритм обучения каскада Хаара

Андреев Е.В., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана (НИУ),
кафедра «Подводные аппараты и роботы»
edirab@yandex.ru

Минеев А.Б., ст. преподаватель
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана (НИУ),
кафедра «Инженерная графика»
mineev30@yandex.ru

Аннотация: Кратко описаны теоретические основы работы каскадного детектора. Приведена последовательность действий по подготовке данных и обучению каскадного классификатора Хаара. Описаны решения типовых ошибок, возникающих при работе алгоритма. Проведено сравнение итоговых детекторов, натренированных на различных наборах изображений, а также приведены исходные коды некоторых полезных утилит, разработанных автором для целей администрирования.

Ключевые слова: каскадный классификатор Хаара, машинное обучение, большие данные, компьютерное зрение, распознавание образов.

Введение

Системы технического зрения уже давно плотно вошли в сферу современной робототехники. С их помощью решаются задачи распознавания и классификации, навигации и одновременного построения карты, обнаружения и слежения за различными объектами. В этой работе описывается процесс обучения каскадного классификатора Хаара для задачи распознавания и те проблемы, и их решения, с которыми столкнулся автор. И хотя подобная технология, разработанная в 80-х годах прошлого века, может казаться несколько устаревшей, она всё же ещё используется в мобильной робототехнике. Причин тому несколько: будучи узко направленным детектором, «заточенным» на обнаружение какого-то конкретного объекта, алгоритм потребляет намного меньше вычислительных ресурсов, чем, скажем, глубокая нейронная сеть и работает быстрее за счёт вычисления интегрального представления изображения [1].

Теоретические основы

Как уже было сказано во введении, детектор Хаара представляет собой каскадный классификатор. Его работа основывается на каскаде решающих деревьев [1], содержащих в листьях конкретные примитивы Хаара (рисунок 1) и пороговые значения, им соответствующие.

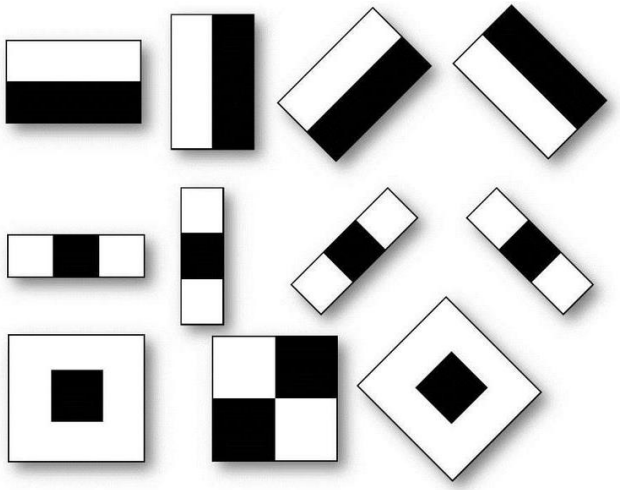

Рис. 1. Примитивы Хаара

Обучение каскада построено на последовательном переборе примитивов и расчёте значений признаков как разницы между суммарной яркостью пикселей, покрытых былой областью текущего примитива и суммарной яркости пикселей чёрной области.

Этап подготовки и разметки данных

Далее приведена последовательность действий для подготовки изображений и их разметки:

1. Для тренировки каскада потребуется два набора данных: с объектом и без него. Допускается наличие в кадре одновременно нескольких объектов. Удобнее всего не снимать последовательно несколько сотен кадров, а заснять два видео. Подобный подход позволит существенно сэкономить время, а также одновременно учесть все негативные эффекты, такие как размытия и расфокусировка, возникающие во время движения реального робота.

2. Полученные видеофайлы разбиваются на кадры. Например, это можно сделать с помощью бесплатной утилиты «Free Video to Jpeg Converter». Наибольшее количество уникальных кадров определяется по формуле:

$$F * T,$$

где:

F – частота, кадров/с;

T – длительность видеоролика, с.

3. Для удобства предлагается переименовать изображения, оставив короткий псевдоним из латинских символов и порядковый номер (далее будет обоснована важность этого шага). Например, IMG_20200509_144931.jpg превращается в with0001.jpg. Используется бесплатное ПО FastStoneImageViewer, имеющее инструменты пакетной обработки фотографий.

4. Для тренировки алгоритма, как правило, используются изображения небольшого разрешения (1280 x 720 пикселей, 640 x 480 пикселей или меньше). В противном случае обучение займёт чрезвычайно много времени. В этом же ПО после переименования можно отмасштабировать изображения с уменьшением разрешения до указанных выше значений.

5. С помощью утилиты opencv_createsamples.exe (входит в пакет поставки фреймворка OpenCV) на позитивных изображениях необходимо вручную указать объект(-ы). Запуск утилиты производится командой:

```
opencv_annotation.exe --annotations=good_2.dat
--images=./extracted_images/with_original
```

Ключ «annotations» с параметром «good_2.dat» задаёт выходной файл, в который будет сохранена информация о разметке, ключ «images» определяет папку с изображениями, содержащими требуемый объект. На рисунке 2 показан процесс аннотации изображения, содержащего макет донной зарядной станции с нанесёнными на неё маркерами специального вида. Подлежащие детектированию объекты выделены рамками. Слева – подтверждённая аннотация (после нажатия клавиши «c»), справа – не подтверждённая. Предполагается осуществлять навигацию автономного необитаемого подводного аппарата по системе технического зрения в процессе его стыковки с зарядной станцией.

О том, как взаимодействовать с программой разметки [2], после запуска напоминает подсказка в командной строке:

```
* mark rectangles with the left mouse button,
* press 'c' to accept a selection,
* press 'd' to delete the latest selection,
* press 'n' to proceed with next image,
* press 'esc' to stop.
```

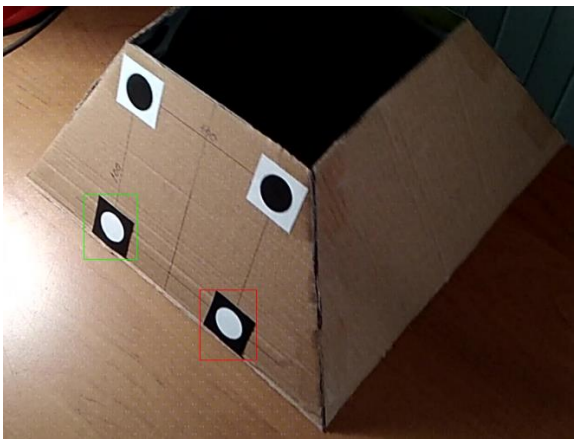

Рис. 2. Процесс разметки изображений

Программа не будет откликаться на действия пользователя, если выбрана отличная от английской раскладка клавиатуры. По нажатии клавиши «Escape» приложение завершит работу и по указанному пути появится файл с содержимым, показанным на рисунке 3.

```
1  ./extracted_images/with_original\pyramid0289.jpg 2 491 325 63 85 642 417 80 99
2  ./extracted_images/with_original\pyramid0290.jpg 2 456 319 70 87 603 413 80 102
3  ./extracted_images/with_original\pyramid0291.jpg 2 421 334 74 84 566 433 80 94
4  ./extracted_images/with_original\pyramid0292.jpg 2 408 335 77 85 555 434 72 103
5  ./extracted_images/with_original\pyramid0293.jpg 2 422 336 65 83 552 438 77 95
```

Рис. 3. Вид содержимого файла аннотаций изображений с объектом

Каждая строчка соответствует отдельной аннотации, сначала указан путь к файлу (относительный или абсолютный, с зависимости от указанных при запуске программы разметки аргументов); следующий параметр – количество объектов на изображении, в данном примере на каждом кадре по два маркера одного вида; оставшиеся цифры обозначают координаты прямоугольников: левый верхний угол, затем правый нижний. Порядок описания прямоугольников соответствует порядку разметки пользователем.

Этап администрирования большого объёма данных

Хотя работающий детектор можно получить и при небольшом количестве позитивных (т.е. содержащих объект детектирования) и негативных (т.е. фоновых) изображений – примерно 100-200 штук, - для стабильного обнаружения требуется больше 1000 позитивных и около 4000 негативных примеров. По этой причине процесс создания аннотаций, как правило, занимает очень много времени и проходит в несколько подходов. Также возможен вариант добавления позитивных изображений с последующим переобучением каскада с целью улучшения его характеристик.

Важно иметь в виду, что программа создания аннотаций перезапишет файл, если в качестве пути будет указан уже существующий. При повторном запуске утилита заново

сканирует папку с исходными изображениями, поэтому возможно попадание дубликатов, что не желательно. Для наилучшей работы требуется обеспечить как можно большее разнообразие как фоновых, так и изображений объекта.

Для целей администрирования сотен и тысяч фотографий были разработаны несколько программ-сценариев, написанных на языке Python [3].

```
import shutil

abs_path = 'Your_absolute_path_to_base_folder/'
which_file = 2

if which_file == 1:
  f = open(abs_path+'good_1.dat')
  f_fixed = open(abs_path+'good_1_fixed.dat', 'a')
else:
  f = open(abs_path + 'good_2.dat')
  f_fixed = open(abs_path + 'good_2_fixed.dat', 'a')

data = f.readlines()
print(type(data))

for line in data:

  first_part = line.strip().split()[0]
  #print(first_part)
  file_name = first_part.split("\\")[-1]
  print(file_name)

  if which_file == 1:
    try:
      shutil.move(abs_path+"from_folder/"+file_name,
      abs_path+" /to_folder/"+file_name)
      line = line.strip().split('\\')
      new_line = line[0] + "/to_folder/" + line[1] + "\n"
      print(new_line, end='')
      f_fixed.write(new_line)
    except FileNotFoundError:
      print(file_name, " does not exist")
      input()
  else:
    try:
      shutil.move(abs_path+"from_folder_2/"+file_name,
      abs_path+"to_folder_2/"+file_name)
        line = line.strip().split('\\')
        new_line = line[0] + "/to_folder_2/" + line[1] + "\n"
        print(new_line, end='')
        f_fixed.write(new_line)

    except FileNotFoundError:
        print(file_name, " does not exist")
        input()
```

Сценарий открывает файл «good_1.dat» на чтение и файл «good_1_fixed» на дозапись. Всё содержимое первого файла считывается в переменную data, которая имеет тип списка. Каждый элемент списка – это строка. В цикле осуществляется итерация по списку: сначала извлекается непосредственно имя файла вместе с расширением и сохраняется в переменной file_name, затем осуществляется перенос самого изображения из директории, условно названной «from_folder/» в директорию «to_folder/». В конце необходимо модифицировать путь к кадру с учётом изменения его местоположения и добавить сформированную строку в конец файла «good_1_fixed».

После чего можно продолжить процесс создания аннотаций: предшествующие записи надёжно сохранены в другом файле, а обработанные изображения не будут учтены повторно.

Этап создания файла фоновых изображений

Вообще говоря, файл негативных примеров может содержать любые изображения, включая и те, которые роботу не доведётся встретить в процессе выполнения задачи. Однако детектор будет функционировать намного лучше, если в качестве негативных примеров использовать изображения обстановки, в которой предстоит вести обнаружение. На рисунке 4 показан фрагмент такого файла. Он не содержит никакой лишней информации, кроме относительного пути к изображениям.

```
4614  ../preparing data/extracted_images/without_advanced/advanced1658.jpg
4615  ../preparing data/extracted_images/without_advanced/advanced1659.jpg
4616  ../preparing data/extracted_images/without_advanced/advanced1660.jpg
4617  ../preparing data/extracted_images/without_advanced/advanced1661.jpg
```

Рис. 4. Вид содержимого файла с описанием фоновых изображений

Для генерации файла был разработан следующий сценарий:

```
import os

f = open('bad.dat', 'w')

for i in range(1, 2956):
    s = './extracted_images/without_800/advanced' +
'{:04d}'.format(i) + ".jpg\n"
    f.write(s)
# print(s, end = '')
```

Считаем, что код выше сложности не представляет и в комментариях не нуждается. Однако, отметим, насколько на данном этапе становится очевидна важность действий, описанных в пункте 3 раздела по подготовке данных о пакетном переименовании в удобный вид.

Этап обучения

Для обучения каскада [2] требуется передать последнему файл-вектор, представляющий собой бинарный набор сжатых изображений объекта. Рассмотрим подробнее команду создания вектора:

```
opencv_createsamples.exe -info good_1_fixed.dat -vec samples.vec
-num 1274 -w 24 -h 24 -show
```

Данное действие выполняет программа opencv_createsamples.exe, также входящая в пакет поставки фреймворка компьютерного зрения. С ключом «info» указывается путь к файлу с аннотациями, полученному ранее. Ключ «vec» обозначает имя выходного вектор-файла, «num» — это количество позитивных изображений, а параметры «w» и «h» специфицируют ширину и высоту в пикселях. Чем больше размеры объекта, тем дольше будет проходить обучение, но тем точнее будет детектирование. Важно, чтобы отношение указанных величин соответствовало пропорциям реального объекта: к примеру, для поиска на изображении конфеты или болта стоит указать -w 40 -h 10, а для обнаружения маркера как на рисунке 2 лучше оставить квадратные пропорции.

Запуск обучения каскада выполняется командой:

```
opencv_traincascade.exe -data haar_output -vec samples.vec -bg
bad.dat -numStages 16 -numThreads 12 -w 24 -h 24 -numPos 1100 -numNeg
2955 -mode ALL,
```

где:

data – путь к каталогу куда будут сохранены результаты;

vec – входной вектор-файл;

bg – файл фоновых изображений (background);

numStages – количество итераций (каскадов);

numThreads – количество задействованных потоков процессора;

w, *h* – ширина и высота изображений вектора в пикселях;

numPos – количество позитивных примеров, необходимых для обучения каскада. Это число обязательно должно быть меньше числа изображений в vec-файле. Причина будет объяснена ниже;

numNeg – количество фоновых изображений, чем больше, тем лучше;

mode – режим на стандартном наборе примитивов Хаара или же на расширенном (включая повёрнутые на 90º).

Программа отобразит следующий вывод:

```
PARAMETERS:
cascadeDirName: haar_output
vecFileName: samples.vec
```

```
bgFileName: bad_navigation.dat
numPos: 1100
numNeg: 2955
numStages: 16
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
acceptanceRatioBreakValue : -1
stageType: BOOST
featureType: HAAR
sampleWidth: 24
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: ALL
Number of unique features given windowSize [24,24] : 261600
```

Здесь важными параметрами являются minHitRate и maxFalseAlarmRate. Первый обозначает точность определения, которую должен достичь каскад на каждой стадии. Значение 0,995 означает, что из 1000 изображений 5 будут ложноположительными срабатываниями. Второй – какое количество изображений, не содержащих объект, будет отсеяно на каждой стадии. К примеру, для обученного каскада из 8 стадий будет отсеиваться 1 – 0,58 = 0,996 = 99,6% всех негативных изображений.

По завершении обучения каждой стадии программа будет выдавать подобные сообщения:

```
===== TRAINING 12-stage =====
<BEGIN
POS count : consumed   1100 : 1195
NEG count : acceptanceRatio    2955 : 2.89746e-05
Precalculation time: 4.876
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|        1|
+----+---------+---------+
|   4|        1|        1|
+----+---------+---------+
|   5| 0.999333| 0.836887|
+----+---------+---------+
|   6| 0.999333| 0.843316|
+----+---------+---------+
|   7| 0.998667| 0.649746|
+----+---------+---------+
|   8| 0.996667| 0.581726|
+----+---------+---------+
```

```
|   9| 0.996667| 0.461591|
+----+---------+---------+
END>
Training until now has taken 0 days 1 hours 23 minutes 3 seconds.
```

Третий столбец – значение FalseAlarmRate, переход к следующей стадии происходит, когда оно становится менее 0,5. Фраза «Required leaf false alarm rate achieved. Branch training terminated.» сообщит об успешном завершении обучения классификатора [4]. Работа обученного детектора по распознаванию маркеров специального вида показана на рисунке 5. Небольшие красные круги обозначают не центр круга маркера, а геометрический центр детектированного объекта.

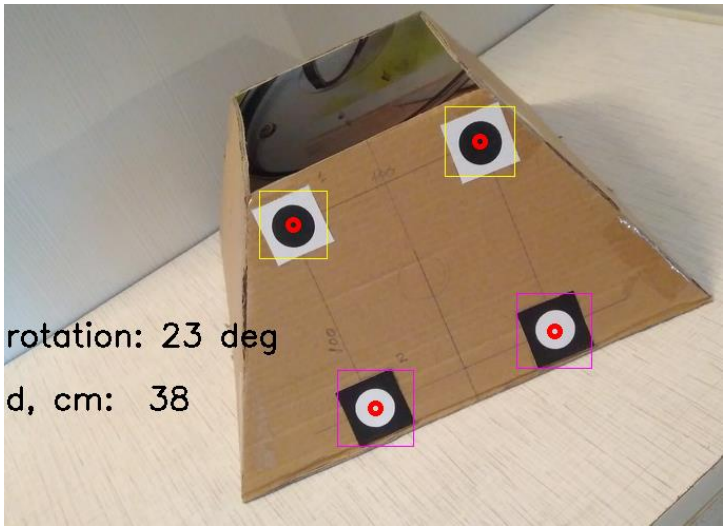

Рис. 5. Работа каскадного классификатора Хаара

На каждом этапе каскад совершает ошибки, принимая некоторые фоновые изображения за изображения с объектом. Для тренировки каждой последующей стадии берутся количество изображений, соответствующих параметру numPos плюс все ошибочные с предыдущей стадии. В примере вывода выше для 12-й стадии обучения присутствует строчка POS count : consumed 1100 : 1195, сообщающая о том, что для тренировки нынешней стадии было взято 1195 изображений, 1100 из которых изначально переданы в vec-файле. Следовательно, на предыдущем этапе ошибочно были распознаны 95 изображений.

Приблизительно количество позитивных изображений, необходимых для создания vec-файла, можно оценить по формуле:

$$V \geq numPos + (numStages - 1) * (1 - minHitRate) * numPos + S,$$

где:

V – число изображений, использованных для создания вектора;

S – суммарное количество нераспознанных изображений на всех стадиях.

По приведённой выше формуле также можно оценить необходимое число объектов, передаваемых opencv_traincascade.exe в параметре -numPos 1100. При меньшем числе пользователь получит ошибку: «Bad argument (Can not get new positive sample. The most possible reason is insufficient count of samples in given vec-file)».

Заключение

Описанный выше каскадный классификатор представляет собой специфичный детектор границ. В отличие от глубокой нейронной сети он не универсален, однако работает быстрее и намного проще устроен, а небольшую потерю в точности можно компенсировать с помощью продвинутых алгоритмов фильтрации. Однако скорость работы сопряжена с трудоёмкостью подготовки данных и обучения классификатора Хаара. В дальнейшем планируется использовать детектор для построения полноценной системы навигации подводного аппарата на основе системы технического зрения.

Список литературы

[1]. Метод Виолы-Джонса (Viola-Jones) как основа для распознавания лиц. Режим доступа: https://habr.com/ru/post/133826/ (дата обращения: 10.05.2020).

[2]. Документация фреймворка OpenCV. Режим доступа: https://docs.opencv.org/4.1.1/dc/d88/tutorial_traincascade.html (дата обращения: 12.05.2020).

[3]. Официальная страница разработчиков языка Python. Режим доступа: https://www.python.org/ (дата обращения: 14.05.2020).

[4]. Страница форума stackoverflow.com. Режим доступа: https://stackoverflow.com/questions/50186866/opencv-cascade-training-fast-fails-required-leaf-false-alarm-rate-achieved-br (дата обращения: 16.05.2020).