МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Сыктывкарский государственный университет имени Питирима Сорокина»
(ФГБОУ ВО «СГУ им. Питирима Сорокина»)

Институт точных наук и информационных технологий
Кафедра прикладной математики и информационных технологий в образовании

Допустить к защите
Зав. кафедрой прикладной математики,
к. ф.-м. н.,
_____ А. В. Ермоленко
«____»_____ 2016 года

Курсовая работа

Алгоритмы обнаружения автомобильного номера и исправление искажений перспективы

Научный руководитель,
ст. преп. ПМиИТО
_____ Н. К. Попова
«____»_____ 2016 года

Исполнитель, студент 135 группы
_____ Е. А. Белых
«____»_____ 2016 года

Сыктывкар, 2016 г.

Содержание

1. Введение

Данная работа посвящена задаче распознавания номеров. Суть этой задачи состоит в следующем: имеется изображение — требуется найти на нем пластину с автомобильным номером и вывести символы, находящиеся на ней, в виде текста.

Несмотря на то, что компьютерные программы, решающие данную задачу уже существуют, некоторые проблемы по-прежнему имеются. Главная из них состоит в том, что качественных материалов, посвященных решению этой проблемы крайне мало. В большинстве статей авторы либо используют OpenCV (фреймворк, содержащий готовые реализации алгоритмов, необходимых для решения), либо утаивают важные детали реализации, без которых проблему решить невозможно, либо просто демонстрируют полное непонимание используемых ими методов.

Первая цель этой работы — написать программу, способную распознавать автомобильные номера, не используя каких-либо сторонних библиотек, кроме стандартных библиотек языка Си и библиотеки для открытия разных форматов изображений. Вторая цель — это попытаться описать используемые методы доступным языком, так как, как уже было сказано выше, при их описании часто упускаются из виду важные детали.

Задача распознавания автомобильных номеров состоит из нескольких этапов: первый — это нахождение пластины с номером на изображении, второй — приведение его к стандартному виду (исправление перспективных искажений, выравнивание освещения, и т. д.), а третий — распознавание символов, находящихся на пластине.

Для решения задачи было выбрано машинное обучение. Метод, с помощью которого, имея большое число эмпирических данных об исследуемой проблеме (в случае с автомобильными номерами — это фотографии, на которых пластины с номером находятся в естественных для них условях), можно найти способ её решения.

В первом разделе описывается, что такое машинное обучение, а также дается значение терминов, используемых в этой работе. Остальные разделы описывают применяемые методы, их реализацию, а также результаты их работы.

2. Необходимая терминология

2.1. Машинное обучение

Машинное обучение — это выявление каких-либо закономерностей по набору эмпирических данных. Т.е. имеется какое-либо явление или объект, а также множество данных, полученных непосредственно с него (это множество называют *обучающей выборкой*), требуется на основе этого множества выявить взаимосвязи, присутствующие в данном объекте или явлении.

Машинное обучение можно разделить на *обучение с учителем* и *обучение без учителя*. В первом случае к каждому примеру из обучающей выборки прилагается ответ, который должна дать обучаемая система, исследуя его, а во втором — системе дается только обучающая выборка.

Распространенным примером обучения с учителем является задача классификации, о которой пойдет речь в следующем разделе. А как пример обучения без учителя, можно привести задачу *обнаружения выбросов*: поиск в обучающей выборке небольшого числа значений, сильно отличащихся от остальных.

2.2. Классификация

Классификация — это задача, суть которой состоит в следующем: имеется множество объектов (обучающая выборка), разделенное на определенные группы (классы), по какому-либо признаку, требуется найти способ, которым можно определить принадлежность к одному из этих классов для произвольного объекта (классифицировать объект) того же типа, что и объекты из обучающей выборки. По сути, задача классификации является разновидностью машинного обучения.

Как пример можно привести распознавание текста, где в качестве классов выступают буквы, цифры, знаки препинания и все остальное (отдельный класс), а изображения с ними — в качестве обучающей выборки. Тогда, чтобы классифицировать изображение, надо определить, изображены ли на ней буква, цифра или знак препинания, если да, то какие.

Еще одним интересным примером является проверка на наличие определенного объекта на изображении, например, лица. В этом случае класса всего два: изображение лица, изображение не лица. Тогда обучающая выборка будет состоять из разных изображений, где на одних лица есть, а на других — нет. Классификация изображения будет состоять в том, чтобы определить, изображено ли на нем лицо.

3. Каскад Хаара

Каскад Хаара — способ обнаружения объектов на изображении, основанный на машинном обучении, идея которого была была предложена в статье за авторством *Пола Виолы (Paul Viola)* и *Майкла Джонса (Michael Jones)*.

Обученный каскад Хаара, принимая на вход изображение, определяет, есть ли на нем искомый объект, т.е. выполняет задачу классификации, разделяя входные данные на два класса (есть искомый объект, нет искомого объекта).

Правильно обученный каскад Хаара имеет хорошую скорость выполнения классификации, а также неплохую устойчивость к разного рода отклонениям. Изначально данный способ был предназначен для обнаружения лиц, однако его можно использовать и для обнаружения других объектов, например, пластины с автомобильным номером.

3.1. Признаки Хаара

Признак Хаара является набором прямоугольных областей изображения, примыкающих друг к другу и разделенных на две группы. Возможных признаков Хаара огромное множество (разнообразные комбинации областей разной ширины и высоты с разными позициями на изображении). Первоначальный набор признаков зависит от реализации и конкретной задачи (в разделе, посвященном алгоритму *AdaBoost*, описано, как из этого набора выбираются требуемые признаки).

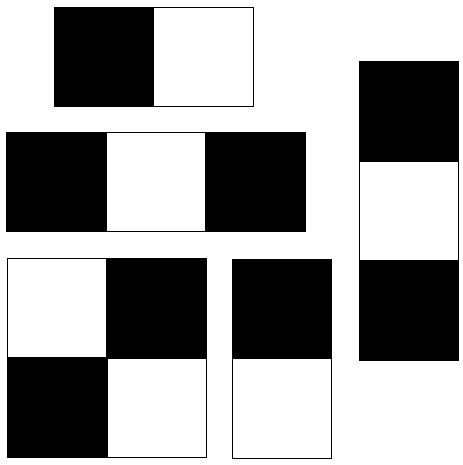

Рис. 1. Комбинации прямоугольных областей, которые использовались при написании этой статьи.

Чтобы вычислить значение конкретного признака Хаара для какого-либо изображения, надо сложить яркости пикселей изображения в первой и второй группах прямоугольных областей по отдельности, а затем вычесть из первой полученной суммы вторую. Полученная разность и есть значение конкретного признака Хаара для данного изображения.

Рис. 2. Признак Хаара на изображении.

Рис. 2 изображает пример признака Хаара на изображении: белые прямоугольники — первая группа группа областей, а черный — вторая. Значение признака Хаара — это разность сумм яркостей пикселей первой и второй группы. В математической форме это будет выглядеть так:

$$a_i = \sum_{i=y_{a_i}}^{y_{a_i}+h_{a_i}-1} \sum_{j=x_{a_i}}^{x_{a_i}+w_{a_i}-1} x_{ij} \qquad b_i = \sum_{i=y_{b_i}}^{y_{b_i}+h_{b_i}-1} \sum_{j=x_{b_i}}^{x_{b_i}+w_{b_i}-1} x_{ij}$$

$$h = \sum_{i=1}^{N_a} a_i - \sum_{i=1}^{N_b} b_i \,,$$

где x_{ij} — яркость пискеля с координатами $[i, j]$, a_i — сумма яркостей пикселей в i-й области первой группы, b_i — сумма яркостей пикселей в i-й области второй группы, h — значение признака Хаара для этого изображения, h_{a_i} — высота i-й области первой группы, w_{a_i} — ширина i-й области первой группы, h_{b_i} — высота i-й области второй группы, w_{b_i} — ширина i-й области второй группы, N_a — количество областей первой группы, N_b — количество областей второй группы, а h — значение признака Хаара для этого изображения.

3.2. Нормализация изображения

Прежде чем классифицировать изображение или использовать его как пример для обучения, это изображение следует нормализовать, т. е. привести к стандартному виду. При использовании каскадов Хаара это означает, что надо перевести изображение из текущей цветовой схемы в черно-белую, а также нормализовать (сделать равным единице) его среднеквадратичное отклонение.

Следует начать с перевода изображения в черно-белую цветовую схему. Так как обычно изображения изначально загружаются в цветовой схеме *RGB*, будет приведена формула только для нее:

$$i[x; y] = \frac{I_r[x; y] + I_g[x; y] + I_b[x; y]}{3} \,,$$

где $i[x; y]$ — пиксель черно-белого изображения, а $I_r[x; y]$, $I_g[x; y]$ и $I_b[x; y]$ — соотвественно красная, синяя или зеленая компоненты изображения.

После этого надо нормализовать среднеквадратическое отклонение. Это необходимо для того, чтобы разница в освещенности на фотографиях оказывала минимум влияния на результат классификации. Для этого следует вычислить среднее арифметическое изображения по следующей формуле:

$$a = \frac{1}{wh} \sum_{i=1}^{w} \sum_{j=1}^{h} x_{ij} \,,$$

а потом найти среднеквадратическое отклонение по формуле:

$$\sigma = \left(\frac{1}{hw} \sum_{i=1}^{h} \sum_{j=1}^{w} (x_{ij} - a)^2 \right)^{\frac{1}{2}}$$

После того, как среднеквадратическое отклонение было найдено, можно его нормализовать, поделив на него значение каждого пикселя в черно-белом изображении:

$$x_{ij} = \frac{x_{ji}}{\sigma}$$

Теперь полученное изображение можно использовать для обучения каскада Хаара. Также эти операции следует выполнять с изображением перед его классификацией.

3.3. Интегральная форма представления изображения

Вычисление значения каждого признака Хаара для изображения требует много операций сложения. Однако, если использовать интегральную форму изображения, количество вычислений можно сильно уменьшить.

В интегральной форме изображения значение каждого пикселя является суммой яркостей этого пикселя и всех пикселей, что находятся выше и левее него (если пиксель с координатами [1;1] находится в верхнем левом углу изображения):

$$S_{yx} = \sum_{i=1}^{y} \sum_{j=1}^{x} x_{ij} ,$$

где S_{yx} — значение пикселя интегральной формы изображения с координатами $[y; x]$, а x_{ij} — значение пикселя исходного изображения с координатами $[i; j]$.

Переведя изображение в интегральную форму, можно вычислять значения признаков Хаара для него, не выполняя суммирование всех требуемых значений яркостей каждый раз по новой. Достаточно посчитать сумму яркостей для каждой прямоугольной области, используя свойства интегральной формы изображения:

$$\begin{cases} a_i = S[y_{a_i} + h_{a_i}; x_{a_i} + w_{a_i}] & , y = 1, x = 1 \\ a_i = S[y_{a_i} + h_{a_i}; x_{a_i} + w_{a_i}] - S[y_{a_i} + h_{a_i}; x_{a_i}] & , y > 1, x = 1 \\ a_i = S[y_{a_i} + h_{a_i}; x_{a_i} + w_{a_i}] - S[y_{a_i}; x_{a_i} + w_{a_i}] & , y = 1, x > 1 \\ a_i = S[y_{a_i} + h_{a_i}; x_{a_i} + w_{a_i}] - S[y_{a_i}; x_{a_i} + w_{a_i}] - S[y_{a_i} + h_{a_i}; x_{a_i}] + S[y_{a_i}; x_{a_i}], & y > 1, x > 1 \end{cases}$$

$$\begin{cases} b_i = S[y_{b_i} + h_{b_i}; x_{b_i} + w_{b_i}] & , y = 1, x = 1 \\ b_i = S[y_{b_i} + h_{b_i}; x_{b_i} + w_{b_i}] - S[y_{b_i} + h_{b_i}; x_{b_i}] & , y > 1, x = 1 \\ b_i = S[y_{b_i} + h_{b_i}; x_{b_i} + w_{b_i}] - S[y_{b_i}; x_{b_i} + w_{b_i}] & , y = 1, x > 1 \\ b_i = S[y_{b_i} + h_{b_i}; x_{b_i} + w_{b_i}] - S[y_{b_i}; x_{b_i} + w_{b_i}] - S[y_{b_i} + h_{b_i}; x_{b_i}] + S[y_{b_i}; x_{b_i}], & y > 1, x > 1 \end{cases}$$

где $S[y; x]$ — значение пикселя интегральной формы изображения. Далее можно вычислить значение признака Хаара как обычно:

$$h = \sum_{i=1}^{N_a} a_i - \sum_{i=1}^{N_b} b_i ,$$

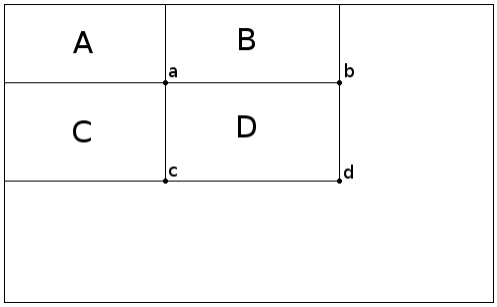

Рис. 3. Вычисление с помощью интегральной формы изображения.

Рис. 3 иллюстрирует, что для того, чтобы вычислить сумму значений пикселей в прямоугольнике D, можно использовать интегральную форму изображения: в ней точка a будет являться суммой значений пикселей в прямоугольнике A, точка b — суммой значений пикселей в прямоугольниках A и B, точка c — суммой значений пикселей в прямоугольниках A и C, а точка d — суммой значений пикселей в прямоугольниках A, B, C и D. Тогда сумма значений пикселей в прямоугольнике D будет равна $d - c - b + a$.

3.4. Алгоритм AdaBoost

Для выбора признаков, лучше всего классифицирующих изображения используется алгоритм *AdaBoost* (adaptive boosting). Этот алгоритм построен на идее, что из большого числа простых способов классификации (называемых *слабыми классификаторами*) можно составить новый способ, выполняющий эту задачу намного эффективнее.

В данном случае слабый классификатор — это функция, которая принимает на вход изображение, вычисляет значение соответствующего ей признака Хаара для этого изображения и сравнивет это значение с порогом, возвращая либо 0, либо 1.

$$h_i(x) = \begin{cases} 1, p_i f_i(x) < p_i \theta_i \\ 0, \text{иначе} \end{cases},$$

где θ_i — порог, x — входное изображение, $f_i(x)$ — значение соотвествующего признака Хаара для изображения x, p_i — направление знака неравенства[1], а $h_i(x)$ — слабый классификатор.

Данный алгоритм перебирает все возможные слабые классификаторы и выбирает те, которые допускают меньше всего ошибок. Важная особенность алгоритма в том, что каждому изображению из обучающей выборки соответсвует определенный вес, и после выбора очередного слабого классификатора веса перераспределяется так, что неверно классифицированные изображения начинают сильнее влиять на значение ошибки. Ниже приведен полный алгоритм:

Входные данные:

h_i — i-й признак Хаара (из всех возможных).
E_i — i-й обучающий пример.
y_i — 0, если i-й обучающий пример отрицательный, и 1, если i-й обучающий пример положительный.
n — количество обучающих примеров.

[1] Принимает одно из двух значений: 1 или −1. Если оно равно 1, ничего не меняется, если же оно равно −1, знак неравенства начинает работать в обратную сторону. Такая запись используется из-за своей краткости, а так же потому, что при реализации на некоторых архитектурах операция умножения работает намного быстрее, чем условные операторы.

Переменные:

w_i — вес, соответствующий i-му обучающему примеру.
θ_i — порог для i-го признака Хаара, дающий наименьшую ошибку.
p_i — направление знака неравенства для i-го признака Хаара, дающее наименьшую ошибку.
ε_i — ошибка i-го слабого классификатора.
$\dot{h}_i$ — слабый классификатор, выбранный на i-й итерации.
$\dot{\varepsilon}$ — минимальное значение ошибки слабого классификатора на текущей итерации.
β_i — минимальное значение ошибки слабого классификатора на i-й итерации, представленное в другой форме (для оптимизации вычислений и экономии места на бумаге).

1. Инициализировать веса обучающих примеров:

$$\begin{cases} w_i = \frac{1}{2m}, \text{если } y_i = 0 \\ w_i = \frac{1}{2l}, \text{если } y_i = 1 \end{cases},$$

где m — число отрицательных примеров, а l — число положительных примеров.

2. Для $t = 1, \ldots, T$:
   a. Нормализовать веса обучающих примеров:

$$w_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

   b. Найти для каждого признака Хаара, порог и направление знака неравенства такие, чтобы слабый классификатор, составленный из них дал наименьшую ошибку:

$$\dot{h}(x) = \begin{cases} 1, p_j h_j(x) < p_j \theta_j \\ 0, \text{иначе} \end{cases}, \quad \varepsilon_j = \sum_{k}^{n} w_k |\dot{h}(E_k) - y_k| \text{ — минимальный}$$

   c. Найти классификатор с наименьшей ошибкой:

$$\dot{\varepsilon} = \min \varepsilon_j$$

$$\dot{h}_t = \begin{cases} 1, p_n h_n(x) < p_n \theta_n \\ 0, \text{иначе} \end{cases},$$

   где n — номер слабого классификатора, дающего наименьшую ошибку.

   d. Обновить веса обучающих примеров:

$$\beta_t = \frac{\dot{\varepsilon}}{(1 - \dot{\varepsilon})}$$

$$w_i = w_i \beta_t^{1 - |\dot{h}_t(E_i) - y_i|}$$

3. Итоговый сильный классификатор:

$$H(x) = \begin{cases} 1, \sum_{t=1}^{T} \log(\frac{1}{\beta_t}) \dot{h}_t(E_i) \geq \frac{1}{2} \sum_{t=1}^{T} \log(\frac{1}{\beta_t}) \\ 0, \text{иначе} \end{cases}$$

Стоит отметить, что под "всеми возможными признаками Хаара" подразумеваются признаки Хаара, имеющие заранее определенное взаимное расположение прямоугольников, а также их версии, маштабированные по ширине и высоте так, чтобы те не выходили за границы окна с заданными шириной и высотой. Все обучающие примеры должны иметь такую же ширину и высоту, как и окно.

Полученный сильный классификатор уже способен выполнять задачу классификации, хоть и очень медленно. Значение $a_i = \log \frac{1}{\beta_i}$ далее будет рассматриваться как "коэффициент i-го слабого классификатора".

3.5. Быстрое вычисление наилучшего порога для слабого классификатора

Перебор всех возможных признаков Хаара выполняется за приемлимое время. Однако нахождение наилучшего порога и направления знака неравенства таким же способом требует такого количества времени, что применение на практике каскадов Хаара становится затруднительным (так как порог — число вещественое, количество его возможных значений ограничено лишь особенностями представления вещественных чисел на конкретной машине).

Однако, если обратить больше внимания на то, какую задачу должен выполнять порог в слабом классификаторе, можно заметить, что перебирать все возможные его значения нет нужды.

Если предположить, что веса всех обучающих примеров равны, то порог должен разделять плохие и хорошие примеры по значению признака Хаара таким образом, чтобы с одной стороны было как можно больше хороших, а с другой — как можно больше плохих (с какой стороны должны быть какие примеры, определяет направление знака неравенства). Но так как веса обычно разные, то задачу надо изменить так: сумма весов примеров, находящихся не по ту сторону порога, по которую им следует находиться, должна быть минимальной.

Так как значение признака Хаара для изображения есть обычное вещественное число, то это можно изобразить так:

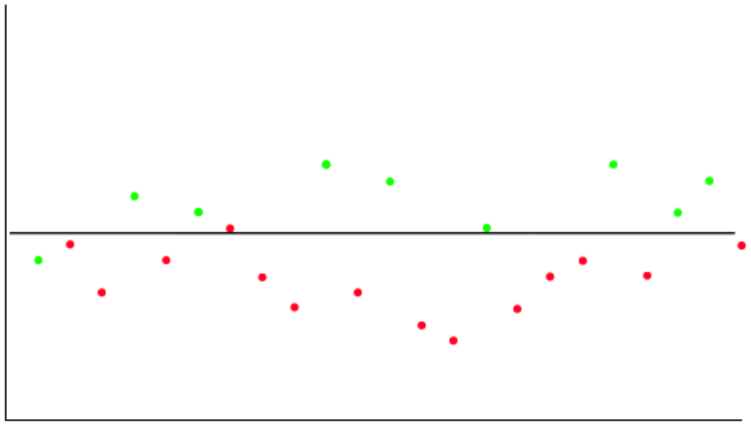

Рис. 4

На рис. 4: по горизонтальной оси расположены номера обучающих примеров, по вертикальной — значения признаков Хаара для соответствующих обучающих примеров. Зеленые точки — значение признака Хаара для хороших примеров, красные точки — для плохих. Горизонтальная линия на графике — один из возможных порогов.

Отсортировав обучающие примеры по их значению признака Хаара, можно получить такую картинку:

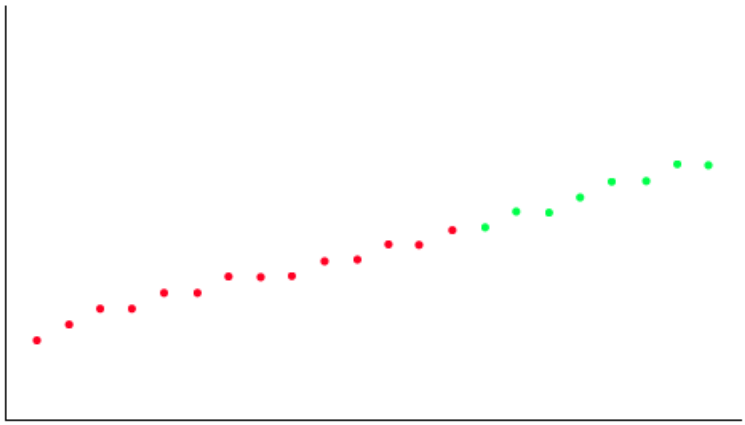

Рис. 5

На рис. 5 по горизонтальной оси расположены номера обучающих примеров, по вертикальной — значения признаков Хаара для соответствующих обучающих примеров. Зеленые точки — значение признака Хаара для хороших примеров, красные точки — для плохих.

Теперь можно заметить, что все пороги, находящиеся между двумя соседними точками на отсортированном графике, дают одинаковые результаты. Следовательно, чтобы получить наименьшую возможную ошибку, достаточно проверить только $n-1$ порогов, где n — количество обучающих примеров, беря как значение порога, например, среднее между двумя соседними отсортированными значениями признака Хаара.

Однако, даже после этого, перебор признаков Хаара занимает очень много времени. Одной из причин является вычисление ошибки слабого классификатора, требующее большое количество операций суммирования и выполняющееся огромное количество раз.

К счастью, есть способ вычислять ошибку слабого классификатора, не выполняя столько операций суммирования. Для этого можно использовать трюк, очень похожий на интегральную форму представления изображения:

h — признак Хаара, для которого ищется порог.
E_i — i-й обучающий пример.
y_i — 0, если i-й обучающий пример отрицательный, и 1, если i-й обучающий пример положительный.
n — количество обучающих примеров.
w_i — вес, соответствующий i-му обучающему примеру.
w^+, w^- — дополнительные массивы.
$\dot{\theta}$ — порог проверяемый на текущей итерации.
$\dot{\varepsilon}$ — ошибка на текущей итерации.
ε — наименьшая полученная ошибка, изначально равна 1.
θ — значение порога, при котором была получена наименьшая ошибка.
p — направление знака неравенства, при котором была получена наименьшая ошибка.

1. Отсортировать обучающие примеры. Веса и элементы массива y расположить в таком же порядке, как и отсортированные примеры (т.е. чтобы каждому примеру после сортировки соответствовал тот же вес и элемент из y, что и до сортировки).
2. Сделать два дополнительных массива, в первом массиве i-й элемент содержит сумму весов хороших примеров до i-го значения, а во втором — плохих:

$$w^+_i = \sum_{j=1}^{i} \begin{cases} w_j, \text{ если } y_j = 1 \\ 0, \text{ если } y_j = 0 \end{cases}$$

$$w^-_i = \sum_{j=1}^{i} \begin{cases} w_j, \text{ если } y_j = 0 \\ 0, \text{ если } y_j = 1 \end{cases}$$

3. Для всех i от 2 до n:
   a. Вычислить значение проверяемого порога:

$$\dot{\theta} = \frac{h(E_{i-1}) + h(E_i)}{2}$$

   b. Посчитать ошибку для $p = 1$:

$$\dot{\varepsilon} = w^-_{i-1} + w^+_n - w^+_{i-1}$$

   c. Если ошибка проверяемого порога меньше текущей минимальной ошибки, запомнить этот порог, ошибку и направление знака неравенства:

$$\text{если } \dot{\varepsilon} < \varepsilon, \text{тогда } \varepsilon = \dot{\varepsilon}, \theta = \dot{\theta}, p = 1$$

   d. Посчитать ошибку для $p = -1$:

$$\dot{\varepsilon} = w^+_{i-1} + w^-_n - w^-_{i-1}$$

   e. Если ошибка проверяемого порога меньше текущей минимальной ошибки, запомнить этот порог, ошибку и направление знака неравенства:

$$\text{если } \dot{\varepsilon} < \varepsilon, \text{тогда } \varepsilon = \dot{\varepsilon}, \theta = \dot{\theta}, p = -1$$

3.6. Каскад классификаторов

Если уменьшать значение порога *сильного* классификатора, уменьшается количество ложных негативных срабатываний (неверно классифицированных хороших примеров) и увеличивается количество ложных позитивных (неверно классифицированных плохих примеров). Таким образом, даже если сильный классификатор состоит из малого количества слабых классификаторов, понижая порог, ценой большего числа ложных позитивных срабатываний можно свести к минимуму количество ложных негативных.

Используя это свойство, из найденных слабых классификаторов можно составить *каскад*, являющийся набором сильных классификаторов (называемых стадиями), через которые последовательно проходит проверяемое изображение. Каждая стадия содержит больше слабых классификаторов, чем предыдущие. После применения каждой последующей стадии, вероятность ложного позитивного срабатывания снижается, а вероятность ложного

негативного остается маленькой.

Проверяемое изображение классифицируется положительно, только если оно дало положительный результат на каждой стадии. Если изображение дало отрицательный результат хотя бы на одной стадии, оно классифицируется отрицательно. Так как на практике проверяется большое количество изображений, лишь малая часть которых содержит искомый объект, большинство отсеивается на ранних стадиях (тех, которые состоят из меньшего количества слабых классификаторов).

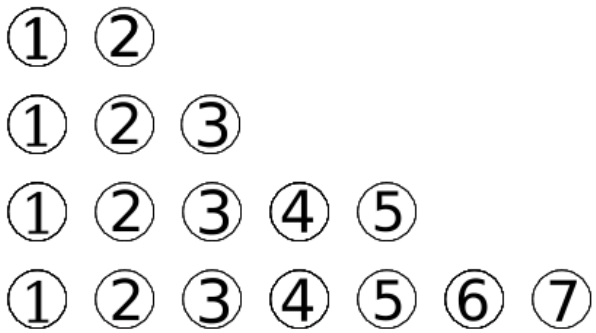

Рис. 6. Иллюстрация структуры каскада.

Рис. 6 иллюстрирует структуру каскада: круги — это слабые классификаторы (цифры в кругах показывают их порядковый номер), каждый ряд кругов — это стадия каскада.

Построить каскад классификаторов можно добавляя к текущей стадии слабые классификаторы (и задавая нужный порог), пока доля ложных позитивных срабатываний (отношение количества неверно классифицированных отрицательных примеров к общему их количеству) на этой стадии больше заданного значения. Как только доля ложных позитивных срабатываний стала меньше этого значения, надо перейти к следующей стадии и точно также добавлять к ней слабые классификаторы. Полный алгоритм будет выглядеть так:

N_i — число слабых классификаторов на i-й стадии.
θ_i — порог на i-й стадии.
n — номер текущей стадии.
ϕ — заданное наперед значение.
p_i — доля ложных позитивных срабатываний i-й стадии.
n_i — доля ложных негативных срабатываний i-й стадии.
a_i — коэффициент i-го слабого классификатора.

1. Установить текущей стадией первую, сделав число слабых классификаторов на ней равным 1:

$$n = 1$$

$$N_n = 1$$

2. Установить для текущей стадии такой порог, чтобы все положительные примеры классифицировались верно:

$$\text{установить } \theta_n, \text{ такой, что } n_n = 0$$

3. Если доля ложных позитивных срабатываний текущей стадии больше ϕ, увеличить число слабых классификаторов на ней, в противном случае сделать текущей следующую стадию, установив в ней число слабых классификаторов равным числу слабых классификаторов предыдущей, увеличенному на 1:

если $p_n > \phi$, тогда $N_n = N_n + 1$
иначе $n = n + 1, N_n = N_{n-1} + 1$

4. Если число слабых классификаторов на текущей стадии меньше числа доступных слабых классификаторов или доля ложных положительных срабатываний равна 0, перейти к пункту 2.
5. Вернуть стандартный порог последней стадии, если выход из цикла произошел из-за того, что закончились доступные слабые классификаторы:

$$\text{если } p_n > 0, \text{тогда } \theta_n = \frac{1}{2}\sum_{i=1}^{N_n} a_i$$

Стоит заметить, что не следует использовать для построения каскада те же хорошие примеры, что и для нахождения слабых классификаторов. Так как слабые классификаторы слишком хорошо обучены для этих примеров, требуемая доля ложных позитивных срабатываний будет достигнута слишком быстро. В этом случае полученный каскад будет содержать мало стадий и начнет допускать большое количество ошибок.

3.7. Оптимизация алгоритма построения каскада классификаторов

Во втором пункте указанного выше алгоритма подразумевается, что требуемый порог ищется перебором, что на практике не очень эффективно (как и в случае с порогами слабых классификаторов, причиной этому являются большие затраты времени и потери точности).

Однако, если использовать упрощенный вариант оптимизации, использовавшейся для нахождения порогов слабых классификаторов, можно свести затраты времени на построение каскада к минимуму.

Для того, чтобы применить этот метод, надо изменить условия, при которых порог считается наилучшим. Также следует учесть, что у порога сильного классификатора фиксированное направление знака неравенства (>).

Для слабого классификатора порог считался наилучшим, если сумма весов неверно классифицированных примеров была минимальной. В сильных же классификаторах наилучший порог тот, при котором верно классифицируются все хорошие примеры, а также максимальное число отрицательных.

Значение, сравниваемое с порогом сильного классификатора, является суммой коэффициентов тех классификаторов, которые верно классифицировали входное изображение:

$$\sum_{i=1}^{N} a_i \dot{h}_i(E)\,,$$

где N — число слабых классификаторов, a_i — коэффициент i-го слабого классификатора, $\dot{h}_i$ — i-й слабый классификатор, а E — входное изображение.

Следовательно, чтобы все хорошие примеры были классифицированы верно, достаточно, чтобы порог сильного классификатора был меньше такой суммы для каждого из них. Или, если выражаться математически, порог должен быть меньше наименьшей среди всех таких сумм для хороших примеров.

Учитывая, что чем порог ниже, тем больше плохих примеров будет классифицировано неверно, условие стоит изменить так: порог должен быть меньше наименьшей среди всех таких сумм для хороших примеров на наименьшее возможное значение (в теории — бесконечно малое, а на практике — зависит от реализации чисел с плавающей точкой на конкретной машине).

При таком условии, в отличие от порогов слабых классификаторов, не требуется ни сортировки, ни проверки выбранного порога. Теперь можно оптимизировать вычислениие сумм для хороших примеров. Для этого следует обратить внимание на тот факт, что при каждой новой итерации число слабых классификаторов на текущей стадии на 1 больше, чем на предыдущей. Тогда вместо того, чтобы каждый раз вычислять сумму заново, можно сделать дополнительный массив (для каждого хорошего примера — соответствующий элемент массива) и при каждой итерации добавлять туда необходимые слагаемые. Ниже приведен алгоритм с учетом всех описанных оптимизаций:

$\dot{h}_i$ — i-й слабый классификатор.
N_i — число слабых классификаторов на i-й стадии.
θ_i — порог на i-й стадии.
n — номер текущей стадии.
ϕ — заданное наперед значение.
p_i — доля ложных позитивных срабатываний i-й стадии.
n_i — доля ложных негативных срабатываний i-й стадии.
g_i — i-й хороший пример.
a_i — коэффициент i-го слабого классификатора.
s_i — i-я ячейка дополнительного массива, соответствующая i-му хорошему примеру.
δ — миинимальное возможное число, которое можно вычесть из уменьшаемого.

1. Задать каждому элементу дополнительного массива значение 0:

$$s_i = 0$$

2. Установить текущей стадией первую, сделав число слабых классификаторов на ней равным 1:

$$n = 1$$
$$N_n = 1$$

3. Для всех хороших примеров: если последний классификатор текущей стадии классифицировал пример верно, прибавить коэффициент этого классификатора к значению в соответствующей примеру ячейке дополнительного массива:

$$s_i = \dot{h}_{N_n}(g_i)$$

4. Задать порогу текущей стадии такое значение, меньшее наименьшего элемента дополнительного массива на минимально возможное число:

$$s_i = \min(s_i) - \delta$$

5. Если доля ложных позитивных срабатываний текущей стадии больше ϕ, увеличить число слабых классификаторов на ней, в противном случае сделать текущей следующую стадию, установив в ней число слабых классификаторов равным числу слабых классификаторов предыдущей, увеличенному на 1:

$$\text{если } p_n > \phi, \text{ тогда } N_n = N_n + 1$$
$$\text{иначе } n = n + 1, N_n = N_{n-1} + 1$$

6. Если число слабых классификаторов на текущей стадии меньше числа доступных слабых классификаторов или доля ложных положительных срабатываний равна 0, перейти к пункту 2.

7. Вернуть стандартный порог последней стадии, если выход из цикла произошел из-за того, что закончились доступные слабые классификаторы:

$$\text{если } p_n > 0, \text{ тогда } \theta_n = \frac{1}{2}\sum_{i=1}^{N_n} a_i$$

3.8. Поиск объектов на большом изображении

Обученный каскад Хаара может классифицировать только изображения того же размера, что и обучающие примеры. Для того, чтобы искать объекты на большом изображении, надо по отдельности классифицировать части этого изображения. Для этого используется окно, двигаемое по изображению, размеры которого соотвествуют рабочим размерам каскада. Если какая-либо часть изображения была классифицирована положительно, значит в ней изображен искомый объект.

Также следует построить несколько маштабированных вариантов изображения (это называется пирамидой изображений) и тоже пройтись по ним окном. Это необходимо потому, что изображение искомого объекта совсем необязательно будет того же размера, что и примеры из обучающей выборки.

Рис. 7. Пример пирамиды изображений: размер сторон каждой следующей версии изображения равняется 0.75 размера сторон предыдущего.

В данной работе сначала с помощью окна проверяется исходное изображение. После этого изображение уменьшается в заранее заданное количество раз и точно также проверяется, пока длина одной из его сторон не станет меньше длины соответствующей стороны окна. Если изображение в окне было классифицировано положительно, кординаты углов этого окна переводятся в систему координат исходного изображения и сохраняются в специальном массиве.

3.9. Объединение пересекающихся прямоугольников

Так как каскад Хаара имеет достаточно неплохую устойчивость к маштабированию и смещению, после полного прохода по изображению и его маштабированным вариантам искомый объект будет выделен окном несколько в соседних позициях и маштабах.

Рис. 8. Пример выделения объекта после полного прохода окном по изображению: зелеными прямоугольниками выделены области, изображение в которых было классифицировано положительно.

Чтобы исправить эту проблему, был выбран следующий подход: все прямоугольники разбиваются на группы. Прямоугольник, принадлежащий какой-либо группе пересекается хотябы с одним из других прямоугольников этой же группы. После этого каждая группа заменяется одним прямоугольником, являющимся усереднением всех прямоугольников этой группы.

Задачу объединения прямоугольников в группы можно хорошо описать и решить с помощью понятий из теории графов. Прямоугольники можно считать вершинами графа. Если они пересекаются, можно считать что они соединены ребром. Тогда если разбить граф на компоненты связности, вершины принадлежащие одной и той же компоненте и будут представлять из себя искомые группы.

Сначала надо представить набор прямоугольников в виде графа. Для этого прямоугольники нумерются. Также каждому прямоугольнику в соотвествие ставится список, содержащий номера других прямоугольников, пресекающих его. Полученные списки и будут преставлять из себя граф. Построить эти списки можно по следующему алгоритму:

r_i — i —й прямоугольник.
R — количество имеющихся прямоугольников.
e_{ij} — номер (в r_i) j-го прямоугольника из тех, которые пересекаются с i-м прямоугольником.
E_i — количество прямоугольников, пресекающихся с i-м прямоугольником. Изначально равно нулю.

Для всех i и j от 0 до N:
Если прямоугольники r_i и r_j пересекаются, добавить в список, соответствующий вершине r_i значение j:

$$\text{если } r_i, r_j \text{ пересекаются, тогда } e_{iN_i} = j$$

$$N_i = N_i + 1$$

Проверка пересечения двух прямоугольников выполняется с помощью проекций на оси. То есть, проверяются на пересечение их проекций на ось X и на ось Y по отдельности. Если пересечение имеется при проекций на обе оси, то прямоугольники пересекаются. Проверка пересечений проекций выполняется следующим образом:

A_{x0}, A_{y0} — наименьшие координаты угла первого прямоугольника.
A_{x1}, A_{y1} — наибольшие координаты угла первого прямоугольника.
B_{x0}, B_{y0} — наименьшие координаты угла второго прямоугольника.
B_{x1}, B_{y1} — наибольшие координаты угла второго прямоугольника.

если $A_{x0} > B_{x1}$ или $A_{x1} < B_{x0,}$ то проекции на ось X пересекаются

если $A_{y0} > B_{y1}$ или $A_{y1} < B_{y0,}$ то проекции на ось Y пересекаются

После этого требуется разбить полученный граф на компоненты связности. Если точнее, требуется узнать, к какой компоненте относится каждая вершина. Информация о ребрах внутри этих компонент, не требуется. Исходя из этого можно использовать следующий алгоритм:

r_i — соответствует i-й вершине. Если равен 1, то эта вершина была достигнута из выбранной начальной вершины.
R_i — соответствует i-й вершине. Если равен 1, то эта вершина уже была достигнута из другой вершины. Изначально равен 0.
V_i — количество вершин, в i-й компоненте связности. Изначально равно нулю.
c_{ij} — номер j-й вершины в i-й компоненте связности.
C — количество компонент связности. Изначально равно нулю.
e_{ij} — номер j-й вершины из тех, в которые можно попасть из i-й вершины.
E_i — количество вершин, в которые можно попасть из i-й вершины.
n — номер вершины, с которой начинается выделение компоненты связности. Изначально равен нулю.
v — i-я вершина.
N — общее количество вершин.

Пока $n < N$:

1. Обнулить r_i:

   для всех i от 0 до $N-1$: $r_i = 0$

2. Отметить все вершины, которые можно достигнуть из текущей вершины:

   если из вершины v_n можно достигнуть вершину v_i, то $r_i = 1$, $R_i = 1$

3. Добавить все отмеченные на текущем шаге вершины в новую компоненту связности:

   для всех i от 0 до $N-1$: если $r_i = 1$, тогда $c_{CV_C} = i$, $V_C = V_C + 1$

   $C = C + 1$

4. Найти вершину с которой начнется выделение следующей компоненты связности:

   пока $R_n = 1$ и $n < N$: $n = n + 1$,

Нахождение всех вершин, которые можно достигнуть из текущей выполняется простым проходом по графу и отметкой уже пройденных вершин. В развернутом виде пункт 2. будет выглядеть так:

1. Отметить вершину с номером n:

   $r_n = 1$, $R_n = 1$

2. Перейти по всем ребрам вершины с номером n:

   для всех j от 0 до $E_i - 1$: если $r_{e_{ij}} \neq 1$, перейти к пункту 1. сделав $n = e_{ij}$

После того, как было определенно, к какой компоненте связности относится каждая вершина, можно заменить каждую группу пересекающихся прямоугольников (каждой из которых соответствует компонента связности графа) одним усередненным прямоугольником.

В качестве усередненния используется прямоугольник, чьи наименьшие и наибольшие координаты угла находятся как среднее арифметическое наименьших и наибольших координат угла всех прямоугольников группы:

c_{ij} — номер j-й вершины в i-й компоненте связности.
C — количество компонент связности.

4. Исправление искажений перспективы

После того, как область, в которой находится пластина с автомобильным номером, была выделена, ее следует привести к виду, пригодному для распознаванию отдельных символов на этой пластине. Одна из основных проблем, мешающих корректному распознаванию символом — это искажения перспективы, возникающие когда номер снимается не под прямым углом.

Для решения этой проблемы используется комбинация нескольких методов. Сначала вычисляется градиент изображения, на котором лучше видны резкие перепады яркости. С его помощью на этом изображении обнаружаются контуры объектов. Среди найденных контуров выделяются только те, которые являются прямыми линиями. Далее из них выбирается четыре линии, с наибольшей вероятностью являющиеся контурами пластины с номером. После этого, с помощью этих четырех линий, находятся углы пластины с номером и вычисляется, каким образом нужно преобразовать изображение, что бы исправить искажения перспективы.

Рис. 9. Пример пластины с номером, обнаруженной на фотографии. Зелеными точками обозначены углы номерной пластины.

Рис. 10. Пластина с номером после исправления искажений перспективы. В таком виде номер пригоден для дальнейшего распознавания.

4.1. Градиент изображения. Оператор Собеля

Градиент изображения — это набор векторов, в котором каждый элемент (вектор) соответствует своему пикселю изображения. Направление вектора указывает из соответствующего ему пикселя в ту сторону, двигаясь в которую, значения пикселей будут возрастать быстрее всего, а модуль его показывает скорость возрастания этих значених.

Если рассматривать непрерывную функцию, ее градиент является суммой произведений производных по x, y и векторов $i = (1, 0)$, $\mathrm{j} = (0, 1)$ (такие вектора называются базисными), соответственно:

$$\nabla_f = \frac{\delta f}{\delta x} i + \frac{\delta f}{\delta y} j,$$

где f — непрерывная функция, для которой ищется градиент, а ∇_f — искомый градиент функции f.

Так как изображение, по сути, является дискретной функцией, т.е. функцией, изменяющейся скачками, вычисление вектора градиента в каждой точке требует прохода по всему изображению. Поэтому, градиент изоображения обычно вычисляется приближенно. Для этого используется *Оператор Собеля*, представляющий из себя две матрицы 3×3, использующихся для нахождения приближенных производных по x и по y по отдельности.

Данные матрицы выглядят следующим образом:

$$S_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad S_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$

Для приближенного вычисления проиводных используется операция свертки:

$$G_x = S_x * I = \sum_{i=1}^{3} \sum_{j=1}^{3} I[x-i, y-j]\ S_x[i, j]$$

$$G_y = S_y * I = \sum_{i=1}^{3} \sum_{j=1}^{3} I[x-i, y-j]\ S_y[i, j]\ ,$$

где $I[x, y]$ — пиксель с координатами [x, y], а $S_y[i, j]$ и $S_x[i, j]$ — элемент матрицы S_x или S_y, соответственно, находящийся в i-й столбце и j-й колонке. После этого, с помощью найденных производных по x и y находятся направление и модуль градиента:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

$$\theta = \arctan \frac{G_y}{G_x}\ ,$$

где G — модуль градиента в точке, θ — направление градиента в точке[1], а G_x и G_y — значения частных производных по x и y в этой точке.

4.2. Детектор границ Канни

Обнаружениие границ выполняется с помощью *детектора границ Канни*. Детектор границ Канни является алгоритмом *бинаризации*, т.е. он разделяет пиксели на две группы: в первой группе находятся пиксели контура, а во второй — все остальные.

Данный алгоритм использует вычисленный с помощью оператора Собеля (или любым другим способом) градиент и состоит из нескольких этапов. Сначала выполняется *подавление немаксимумов* среди значений модуля градиента, потом выполняется *двойная пороговая фильтрация*. После этого обрабатываются пиксели, принадлежность контуру которых не была опредена на предыдущем этапе.

Результат возвращается в виде изображения (маски), где каждый пиксель соответствует пикселю исходного изображения с такими же координатами. Пиксели маски принимают одно из двух значений: 0 — если соответствующая точка исходного изображения не принадлежит границе, и любого другое значение, если принадлежит.

Подавление немаксимумов требуется для того, чтобы выделенная граница была мининальной толщины, иначе говоря, чтобы она была четкой настолько, наколько это возможно. Для этого выполняется проход по всем точкам градиента, и, если значение градиента в точке меньше, чем значение градиента в двух соседних точках (находящихся слева и справа от

[1] Важно учитывать, что если и G_x, и G_y в точке отрицательны, то минусы "уничтожают" друг друга. Этот случай следует обрабатывать отдельно, так как иначе будет полученое неверное значение.

вектора направления градиента), то ему присваивается значение 0.

Для упрощения реализации угол направления градиента дискретизируется, т.е., вместо вещественных чисел представляется фиксированным количеством значений. На основе дискретизированного значения угла уже определяется, с какими соседними значениями градиента сравнивать значение градиента в текущей точке. Ниже приведен полный алгоритм подавления немаксимумов:

$G[x, y]$ — значение градиента в точке с координатами [x, y].
$\theta[x, y]$ — угол градиента с осью X в точке с координатами [x, y].
w — ширина исходного изображения.
h — высота исходного изображения.
T — дискретизированное значение угла.
L_x — координата x точки слева от вектора направления градиента.
L_y — координата y точки слева от вектора направления градиента.
R_x — координата x точки справа от вектора направления градиента.
R_y — координата y точки справа от вектора направления градиента.

Для всех x от 1 до w:
Для всех y от 1 до h:

1. Дискретизировать[1] угол направления градиента в точке $[x, y]$:

$$T = \frac{8\left(\theta[x, y] + \frac{\pi}{8}\right)}{2\pi} \bmod 8 = \left[\frac{4\theta[x, y]}{\pi} + \frac{1}{2}\right] \bmod 8$$

2. Определить, с какими из соседних точек следует сравнивать значение градиента:

если $T = 0$ или $T = 4$: тогда $L_x = x,\ L_y = y+1,\ R_x = x,\ R_y = y-1$
если $T = 2$ или $T = 6$: тогда $L_x = x-1,\ L_y = y,\ R_x = x+1,\ R_y = y$
если $T = 1$ или $T = 5$: тогда $L_x = x-1,\ L_y = y-1,\ R_x = x+1,\ R_y = y+1$
если $T = 3$ или $T = 7$: тогда $L_x = x-1,\ L_y = y+1,\ R_x = x+1,\ R_y = y-1$

3. Сравнить значение градиента со значениями градиента в выбранных соседних точках. Если оно меньше любого из них, заменить его значение нулем:

если $G[x, y] < G[L_{x,}L_y]$ или $G[x, y] < G[R_{x,}R_y]$: тогда $G[x, y] \bmod = 0$

После подавления немаксимумов выполняется двойная пороговая фильтрация. Выбирается два значения (порога). Они задаются либо вручную, либо определяются динамически (например, *методом Оцу*). Точки, в которых значение градиента больше наибольшего из двух порогов, отмечаются как точки границы. Те точки, значения которых лежат между двумя порогами, отмечаются как неопределенные. Остальные точки отмечаются как фоновые (не являются точками границы):

$G[x, y]$ — значение градиента в точке с координатами [x, y].
θ_1 — наименьший из двух порогов
θ_2 — наибольший из двух порогов
w — ширина исходного изображения.
h — высота исходного изображения.
$m[x, y]$ — точка $[x, y]$ выходной маски. Если ее значение равно 0, то точка $[x, y]$ изображения — фоновая, если 2 — то это точка контура, если 1 — ее принадлежность контуру пока не определена.

[1] Здесь предполагается, что $\theta[x, y]$ лежит в интервале $[0; 2\pi]$. Если это не так, пред этим его следует привести к этому интервалу.

Для всех x от 1 до w:
Для всех y от 1 до h:
Сравнить значение градиента в точке $[x, y]$ с порогами θ_1 и θ_2:

если $G[x, y] < \theta_1$: тогда $m[x, y] = 0$
если $G[x, y] > \theta_2$: тогда $m[x, y] = 2$
если $G[x, y] > \theta_1$ и $G[x, y] < \theta_2$: тогда $m[x, y] = 1$

Далее обрабатываются точки, отмеченые как неопределенные. Это делается следующим образом: если принадлежность точки контуру не была определена на предыдущем этапе, но при этом хотябы одна из 8 соседних для нее точек принадлежит контуру, то она тоже помечается как точка контура:

$G[x, y]$ — значение градиента в точке с координатами [x, y].
w — ширина исходного изображения.
h — высота исходного изображения.
$m[x, y]$ — точка $[x, y]$ выходной маски. Если ее значение равно 0, то точка $[x, y]$ изображения — фоновая, если 2 — то это точка контура, если 1 — ее принадлежность контуру пока не определена.

Для всех x от 1 до w:
Для всех y от 1 до h:
Сравнить значение градиента в точке $[x, y]$ с порогами θ_1 и θ_2:

если $m[x, y] = 1$:
если $m[x-1, y-1] = 2$ или если $m[x-1, y] = 2$ или если $m[x-1, y+1] = 2$
или если $m[x, y-1] = 2$ или если $m[x, y] = 2$ или если $m[x, y+1] = 2$
или если $m[x+1, y-1] = 2$ или если $m[x+1, y] = 2$
или если $m[x+1, y+1] = 2$:
тогда $m[x, y] = 2$,
иначе $m[x+1, y+1] = 0$

Рис. 11. Пример работы детектора границ Канни: сверху находится исходное изображение, а снизу выходное изображение детектора.

4.3. Преобразование Хафа

После того, как контуры были выделены детектором границ Канни, среди них ищутся прямые линии. Для этого используется преобразование Хафа. Оно, как и детектор границ Канни, состоит из нескольких этапов. Сначала выполняется поиск прямых. Далее выполняется подавление немаксимумов. А после этого линии, оставшиеся после предыщего этапа сортируются по количеству голосов, полученных на первом этапе.

Для поиска прямых в используется *аккумулирующий массив*. Каждый элемент массива соотвествуют одной из всех возможных прямых. Он является целым числом, позывающим сколько точек, принадлежащих границе, лежит на этой прямой.

Изначально всем элементам аккумулирующего массива присваивается значение 0, далее выполняется проход по всем точкам, принадлежащим контуру. Для каждой такой точки определяются все прямые, которые могут через нее проходить и значения элементов аккумулирующего массива, соответствующие этим прямым, увеличиваются на 1.

Для поиска всех прямых, на которых может лежать точка (а также для определения количества элементов аккумулирующего массива) используется уравнение прямой, записанное в специальной форме:

$$r = x\cos\theta + y\sin\theta\,,$$

где θ — угол нормали прямой с осью X, а r — минимальное расстояние от точки $[0,0]$ до прямой.

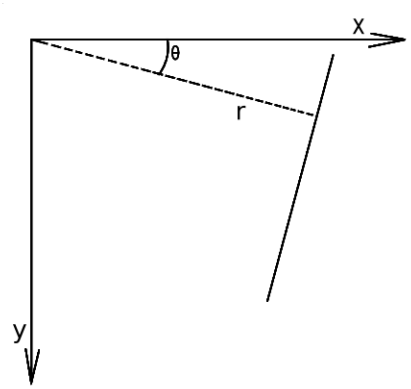

Рис. 12. Представление прямой через угол нормали с осью X и длину перпендиикуляра.

Так как параметры θ и r — вещественные числа, возможных прямых бесконечное множество. Следовательно, полный их перебор невозможен. Поэтому θ и r дискретизируются с заранее заданным шагом. Т.е. расстояние между двумя соседними значениями параметра в дискретизированным виде равно фиксированному значению (шагу). В данной работе параметр θ дискретизируется с шагом $\frac{\pi}{180}$ (вместо радиан используются целые углы), а параметр r с шагом 1 (просто округляется до ближайщего целого).

Все прямые, проходящие через точку находятся с помощью приведенного выше уравнения прямой. Выполняется проход для всех возможных значений θ, на каждом шаге текущее значения θ и координат точки подставляются в уравнение прямой, и вычисляется значение параметра r. Так как при фиксированном значении параметра θ, через конкретную точку может проходить только одна прямая, таким образом можно найти все искомые прямые.

Для того чтобы быстро определять, какой элемент аккумулирующего массива соответствует найденной прямой, аккумулирующий массив следует представить в виде изображения. Для этого надо расположить его элементы в несколько строк так, чтобы все элементы одной строки соответствовали прямым с одинаковым значением r (равным *номеру* строки) и разными значениями θ (равными *номеру* элемента в строке). Такое представление иногда называют пространством Хафа. Теперь, для того, чтобы определить, какой элемент аккумулирующего массива соответствует прямой со значениями параметров θ и r, достаточно найти точку в пространстве Хафа с координатами $[\theta, r]$.

Важно отметить, что при представлении изображения в компьютере значения координат точки (пикселя) обычно являются положительными целыми числами, а значения параметров прямой не обязательно являются целыми и могут быть отрицательными. Эта проблема решается в помощью смещения и маштабирования, т.е. прямой со значениями параметров θ и r соотвествует элемент аккумуллирующего массива с координатами $[a_\theta\theta + b_\theta, a_r r + b_r]$, где a_θ и a_r — маштабирующие коэффициенты параметров прямой, а b_θ и b_r — их смещения.

Ниже приведен полный алгоритм нахождения прямых:

$A[a, d]$ — элемент аккумулирующего массива с координатами $[a, d]$.
w — ширина входного изображения.
h — высота входного изображения.
T — ширина аккумулирующего массива.
R — высота аккумулирующего массива.
$d\theta$ — шаг дискретизации параметра θ.
dr — шаг дискретизации параметра r.
θ — значение параметра θ текущей прямой.
r — значение параметра r текущей прямой.

Для всех x от 1 до w:
Для всех y от 1 до h:
Если значение пикселя входного изображения с координатами $[x, y]$ не равно 0:
Для всех a от нуля до T:

1. Вычислить значение θ для текущей прямой:

$$\theta = a \cdot d\theta$$

2. Вычислить значение r для текущей прямой:

$$r = x \cos\theta + y \sin\theta$$

3. Увеличить значение элемента аккумулирующего массива, соответствующего текущей прямой:

$$A[a, d + \frac{R}{2}] = A[a, d + \frac{R}{2}] + 1$$

После заполнения аккумулирующего массива из него следует выбрать элементы, являющееся максимумами и посторить из них список. При этом используется метод, очень похожий на подавление немаксимумов в детекторе границ Канни. Только сравнениие выполняется с четырьмя соседними элементами вместо двух:

$A[a, d]$ — элемент аккумулирующего массива с координатами $[a, d]$.
T — ширина аккумулирующего массива.
R — высота аккумулирующего массива.
$l_\theta[i]$ — параметр θ i-й прямой в заполняемом списке.
$l_r[i]$ — параметр r i-й прямой в заполняемом списке.
$l_v[i]$ — количество голосов, отданных за i-ю прямую в заполняемом списке.
L — количество прямых в заполняемом списке. Изначально равно нулю.
$d\theta$ — шаг дискретизации параметра θ.
dr — шаг дискретизации параметра r.

Для всех a от 1 до T:
Для всех d от 1 до R:
Если значение элемента аккумулирующего массива с координатами $[a, d]$ не равно нулю:
Если значение элемента аккумулирующего массива с координатами $[a, d]$ больше всех четырех соседних элементов, добавить значение параметров θ и r прямой, которой он соответствует, в список:

если $A[a, d] > A[a, d-1]$ и $A[a, d] > A[a, d+1]$
и $A[a, d] > A[a-1, d]$ и $A[a, d] > A[a+1, d]$:
тогда $l_\theta[L] = a \cdot d\theta$, $l_r[L] = d - \frac{R}{2}$, $l_v[L] = A[a, d]$, $L = L + 1$

После этого прямые в полученном списке сортируются по количеству голосов (по убыванию) любым способом, например, *быстрой сортировкой*. На выход подается отсортированный список, содержащий значения параметров θ и r найденых прямых.

4.4. Поиск границ пластины с автомобильным номером

Данный раздел раздел посвящен тому, каким образом комбинируются описанные выше методы для того, чтобы находить границы номера.

Основой поиска границ пластины с номером в данной работе является выбор самых ярких прямых (получивших больше всего голосов), найденных преобразованием Хафа. Однако, границы пластины с номером не всегда являются самыми яркими прямыми.

Так как заранее примерно известно, в какой части изображения можно найти каждую из четырех границ, то первое, что можно сделать, чтобы это исправить — это разбить изображение на части и в каждой части искать отдельную границу.

Для поиска горизонтальных границ изображение разбивается горизонтальным сечением на две части: верхнюю и нижнюю половины. В верхней половине ищется верхняя граница номера, а в нижней — нижняя. Для поиска вертикальных границ изображение разбивается вертикальным сечением на несколько равных частей (в данной работе — на пять). В самой левой части ищется левая граница номера, а в самой прямой — правая.

Рис. 13. Исходное изображение пластины с номером.

Рис. 14. Изображение пластины с номером, разбитое на верхнюю и нижнюю половины.

Рис. 15. Изображение пластины с номером, разбитое на пять частей вертикальным сечением.

Для того, чтобы вертикальные границы пластины с номером (так как они менее яркие, чем горизонтальные) лучше обнаруживались с помощью преобразования Хафа, части изображения, в которых они ищутся, растягиваются по высоте (в данной работе они растягиваются так, чтобы их высота была равна ширине всего изображения с пластиной). Таким образом все вертикальные прямые на них при преобразовании Хафа получат больше голосов.

Рис. 16. Растянутые изображения, в которых ищутся вертикальные границы.

Также, можно использовать разные дополнительные условия, чтобы отсеять некоторые яркие прямые (найденные с помощью преобразования Хафа), не являющиеся границами пластины с номером. В данной работе используется два дополнительных условия.

Первое из них заключается в проверке угла наклона прямых. Если ищется верхняя или нижняя граница номера, то прямые, параметр θ которых меньше $\frac{\pi}{4}$ или больше $\frac{3\pi}{4}$ (т.е. прямые, не являющиеся горизонтальными) не рассматриваются. Аналогично, если ищется левая или правая граница: не рассматриваются прямые, чей параметр θ больше $\frac{\pi}{4}$ и меньше $\frac{3\pi}{4}$ (прямые, не являющиеся вертикальными).

Второе дополнительное условие основанно на том факте, пластина с автомобильным номером обычно белого цвета и имеет черные края. Поэтому второе условие заключается в сравненнии сумм яркостей в двух полосах (область изображения, ограниченная двумя прямыми). Первая полоса, лежит непосредственно на обнаруженной прямой. Вторая полоса лежит вплотную к первой по ту сторому от нее, по которую должна находиться пластина с номером. Если сумма яркостей пикселей, находящихся в первой полосе меньше, чем такая же сумма во второй полосе, то обнаруженная прямая считается границей пластины с номером, иначе прямая больше не рассматривается.

Второе условие применяется только при поиске вертикальных границ пластины с номером (горизонтальные прямые, для которых выполняется первое условие, сразу считаются границами пластины с номером), так как на практике результат этой проверки для горизонтальных границ был неудовлетворительным. Это следствие того, что горизонтальные границы пластины итак обычно являются самыми яркими, и, следовательно добавление этой проверки при их поиске несет лишь дополнительные ошибки распознавания.

Ширина полосы выбирается как доля от ширины или высоты. Т.е. ширина горизонтальных полос равняется высоте изображения, умноженной на заранее определенную константу (которая больше нуля и меньше единицы), а ширина вертикальных полос — равняется ширине изображения, умноженной на другую заранее определенную константу (которая также больше нуля и меньше единицы).

Каждая полоса представляется в виде двух прямых (границ полосы), чьи параметры θ равны, а параметры r отличаются на ширину полосы (т.е. модуль их разности равен ширине полосы). Для того, чтобы проверить, лежит ли точка в полосе, ищется прямая, проходящая через эту точку, с таким же параметром θ, как и границы полосы. Если значение параметра r этой прямой лежит между значениями параметров r границ полосы, то точка лежит в полосе.

Нахождение параметра r такой прямой, по сути, является нахождением проекции точки $[x, y]$ на перпендикуляр к границам полосы. Эту проекцию можно найти как длину катета, прилежащего к углу между прямой, проведенной из точки $[0, 0]$ в точку $[x, y]$ и перпендикуляром к границам полосы.

Алгоритм суммирования яркостей всех пикселей, лежащих в полосе выглядит так:

$I[x, y]$ — пиксель изображения (содержащего яркости), на котором ищется прямая.
w — ширина изображения, на котором ищется прямая.
h — высота изображения, на котором ищется прямая.
θ — параметр θ обоих прямых, составляющих полосу.
r_0, r_1 — параметр r первой и второй прямых составляющих полосу. r_0 должен быть меньше, чем r_1.
ϕ — угол прямой, проведенной из точки $[0, 0]$ в точку $[x, y]$ с осью X.
d — длина прямой, проведенной из точки $[0, 0]$ в точку $[x, y]$.
ρ — проекция точки $[x, y]$ на перпендикуляр к границам полосы.
S — сумма пикселей в полосе. Изначально равна нулю.

Для всех a от 1 до w:
Для всех d от 1 до h:

1. Вычислить длину и угол с осью X прямой, проходящей из точки $[0, 0]$ в точку $[x, y]$:

$$\phi = \arctan \frac{y}{x}$$
$$d = \sqrt{x^2 + y^2}$$

2. Вычислить значение проекции точки $[x, y]$ на перпендикуляр к границам полосы:

$$\rho = d \cos(\phi - \theta)$$

3. Если проекция точки $[x, y]$ лежит между параметрами r границ полосы, то она лежит в полосе и, следовательно, значение яркости пикселя с этими координатами надо прибавить к сумме:

$$\text{если } \rho > r_0 \text{ и } \rho < r_1\text{: тогда } s = s + I[x, y]$$

После того, как граница пластины с номером была найдена, она переводится из представления через перпендикуляр и его угол с осью X в представление через две точки, лежащие на ней:

$$\begin{array}{ll} \text{если } \theta > \varepsilon: & p_{0x} = 0,\ p_{0y} = \dfrac{r}{\sin\theta},\ p_{1x} = w,\ p_{1y} = r - w\dfrac{\cos\theta}{\sin\theta}, \\ \text{иначе} & p_{0x} = r,\ p_{0y} = 0,\ p_{1x} = r,\ p_{1y} = h \end{array},$$

где θ и r — параметры прямой в представлении через перпендикуляр и его угол с осью X, w и h — ширина и высота изображения с пластиной, p_{0x} и p_{0y} — координаты первой точки в новом представлении линии, а p_{1x} и p_{1y} — координаты второй точки в новом представлении линии.

После того, как были найдены все четыре границы пластины, составляющие контур номера, ищутся координаты её углов. Для этого находятся точки пересечения левой и верхней (левый верхний угол пластины), левой и нижней (левый нижний угол пластины), правой и верхней (правый верхний угол пластины), правой и нижней (правый нижний угол пластины) границ.

Далее, с помощью найденных координат углов выполняется перспективное преобразование изображения с пластиной, и после этого выполняется распознавание отдельных символов на пластине.

5. Заключение

В результате проделанной работы был написан набор утилит для Unix-подобных систем, предназначенных для распознавания номеров. Их список включает в себя: утилиту для поиска слабых классификаторов каскада Хаара, утилиту для построения каскада Хаара из найденных примитивов, утилиту для поиска объектов на изображении с помощью обученного каскада Хаара (имеется уже обученный под поиск пластины с номером каскад), а также утилиту для исправления перспективных искажений в найденном изображении с пластиной.

Структура каскадов Хаара и алгоритм обучения были реализованы на основе оригинальной статьи. Также была предпринята попытка объяснить их суть доступным языком. Был добавлен ряд оптимизаций, не описаных в оригинальной статье, значительно увеличивающих скорость обучения.

Алгоритмы и фильтры, используемые при исправлении искажений перспективы (оператор Собеля, метод Оцу, детектор границ Канни, преобразование Хафа) были реализованы на основе их описаний в книге "Цифровая обработка изображений" Рафаеля Гонзалеза и Ричарда Вудса. При изучении особенностей фотографий с изображением автомобильных номеров был определен способ совмещения этих алгоритмов и фильтров, дающий описаные ниже результаты, а также были добавлены дополнительные алгоритмы для улучшения качества обнаружения границ.

Данная работа имеет ряд недостатков. Каскад Хаара не может обнаружать объекты, сильно наклоненные относительно того объекта, под который он обучался. Поэтому, автомобильные номера, сфотографированные под нестандартными углами, обычно не обнаружаются. Это не является серьезным недостатком, так как написанные утилиты в основном предназначены для специальных камер, снимающих машины под требуемым углом. Также, эту проблему можно решить, проверяя помимо исходного изображения его копии, повернутые на фиксированный угол.

Детектор границ Канни периодически может давать сбой. Это обычно связано с плохим освещением, грязными номерами, большим уровнем шума, яркими границами у пластины с номером или её физизескими искажениями (например, если пластина с номером загнута). В итоге, для дальнейшего распознавания пригодно только около 80% обнаруженных номеров.

Дальнешую работу планируется направить на исправление описанных выше недостатков. После этого планируется искать способы распознавания отдельных символов на изображении, полученном после исправления перспективных искажений.

6. Список литературы

1. *An Introduction to Practical Neural Networks and Genetic Algorithms For Engineers and Scientists. Christopher MacLeod.* — небольшая книга, посвященная нейронным сетям и генетическим алгоритмам. Также, в ней описаны разные методы, применяемые при машинном обучении.
2. *Rapid Object Detection using a Boosted Cascade of Simple Features. Paul Viola, Michael Jones.* — оригинальная статья, в которой были представлены каскады Хаара, содержит часть информации, необходимой для их реализации.
3. *Digital Image Processing. Rafael C. Gonzalez, Richard E. Woods.* — книга, посвященная цифровой обработке изображений. В ней описывается большая часть методов, применяемых в этой работе при исправлении искажений перспективы.
4. *Без паники! Цифровая обработка сигналов. Юкио Сато.* — введение в цифровую обработку сигналов.
5. *The Scientist and Engineer's Guide to Digital Signal Processing. Steven W. Smith.* — книга, посвященная цифровой обработке сигналов. В ней описано множетсво методов, применяемых улучшении сигналов и их анализе.