Санкт-Петербургский государственный университет

Кафедра системного программирования
Программная инженерия

Лунев Артем Евгеньевич

Определение полос движения на заснеженной дороге по видео

Отчет по производственной практике

Научный руководитель:
доц. кафедры СП, к. т. н., Ю. В. Литвинов

Консультант:
М. С. Осечкина, инженер-программист ООО "Тиквижн"

Санкт-Петербург
2020

Оглавление

Введение 3

1. Цели и задачи 4

1.1 Цель работы 4

1.2 Поставленные задачи 4

2. Обзор 5

2.1 Подходы с использованием нейронных сетей 5

2.2 Watershed 6

2.3 Змеи, модель активных контуров 8

2.4 Техника разделения и объединения 9

2.5 Eroding and Dilating 9

2.5 Вывод из обзора 10

3. Описание алгоритма 11

3.1 Поиск капота автомобиля 11

3.2 Предобработка 11

3.3 Обработка изображения 13

4. Замеры времени 15

5. Оценка качества 16

6. Результаты 21

Список литературы 22

Введение

В последнее время активно развивается направление ADAS — продвинутые системы помощи водителю. В данные системы входят различные алгоритмы и устройства, которые помогают водителю в управлении автомобилем. Технология ADAS позволяет повысить безопасность и комфорт водителей и пассажиров. ADAS передает водителю транспортного средства информацию о сложных дорожных ситуациях и ДТП. Система оповещает водителя с помощью вибрации, сообщений на сенсорном экране или звуком.

Одна из задач ADAS систем — определение области возможного движения автомобиля. В этом направлении проведено довольно много успешных исследований. Но гораздо меньше исследований сделано для этой задачи в плохих погодных условиях, например при заснеженной дороге. Одной из проблем этой задачи является вычислительная сложность. В онлайн системах обработка входного видео потока не должна занимать слишком много времени. Также значительная часть решений основана на подходе с использованием нейронных сетей, который имеет некоторые недостатки – сложность построения датасета (на заснеженной дороге не всегда видно полосы, следовательно не всегда разметка человеком будет одинаково верной), возможное снижение качества при переходе на камеру, отличную от той, которая использовалась во время обучения, а также сложность вычислений. Поэтому для решения данной задачи планируется использовать аналитические методы.

Прототип системы, позволяющей определить полосы движения на дороге при заснеженных условиях по видео с открытым исходным кодом может быть использован как основа для решений, встраиваемых в системы помощи водителям, а также для сравнения эффективности с другими аналогичными системами и их возможной оптимизации.

1. Цели и задачи

1.1. Цель работы

Целью данной работы является разработка прототипа системы, позволяющей определить полосы движения на дороге при заснеженных условиях по видео.

1.2. Поставленные задачи

Для выполнения данной цели были выделены следующие задачи:

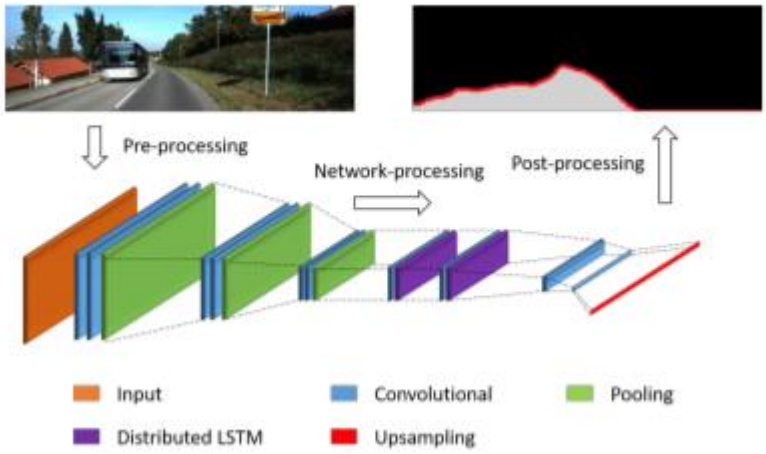
1. Изучить существующие алгоритмы сегментации (нахождения группы пикселей, относящихся к объекту);
2. Сделать обзор существующих решений;
3. Разработать алгоритм определения полос движения;
4. Реализовать прототип;
5. Провести апробацию решения на размеченных кадрах.

2. Обзор

Целью данного обзора является поиск техник и алгоритмов сегментации, которые пригодятся в решении данной задачи.

Подходы с использованием нейронных сетей

В качестве примера, рассмотрим статью Road Segmentation Using CNN and Distributed LSTM [1]. Архитектура сети схематично представлена ниже



Она состоит из трех частей:

1) Local feature encoder. Извлекает локальные признаки из входного изображения. Представлен несколькими слоями свертки и пулинга, из-за чего в начале значительно снижается размер карты признаков. Из-за этого данный алгоритм показывает хороший результат по времени.

2) Feature Processor. Извлекает контекстные признаки (более общие). Представлен несколькими слоями свертки и LSTM-слоями. LSTM это разновидность архитектуры рекуррентных нейронных сетей, для которой характерна не только возможность обрабатывать серии событий во времени или последовательные пространственные цепочки,

но и относительная невосприимчивость к длительности временного разрыва. Данная архитектура довольно часто применяется в подобных задачах.

3) Output decoder. Переводит входной тензор в результирующий вектор.

Конкретные цифры сжатия изображения поэтапно приведены в таблице 1.

Layer	Kernel ($w_1 \times h_1 \times d_1$)	Input ($w \times h \times d$)	Output ($w_2 \times h_2 \times d_1$)
Input	–	600×160×5	–
Conv1 s=2	3 × 3 × 64	600×160×5	600×160×64
Conv2	3 × 3 × 64	600 × 160 × 64	600×160×64
Conv3 s=2	3 × 3 × 64	300×80×64	300 × 80 × 64
Conv4	3 × 3 × 64	300×80×64	300 × 80 × 64
Conv5 s=2	3 × 3 × 64	150×40×64	150 × 40 × 64
Conv6	3 × 3 × 64	150×40×64	150 × 40 × 64
Conv7	3 × 3 × 64	75 × 20 × 64	75 × 20 × 64
D-LSTM 1	64	75 × 20 × 64	75 × 20 × 64
Conv8	3 × 3 × 64	75 × 20 × 64	75 × 20 × 64
D-LSTM 2	64	75 × 20 × 64	75 × 20 × 64
Conv9	1 × 5 × 64	75 × 20 × 64	75 × 4 × 64
Conv10	1 × 4 × 1	75 × 4 × 64	75 × 1×1
Up-sample	8 × 1 × 1	75 × 1 × 1	600 × 1 × 1
Output	–	–	600 × 1 × 1

Таблица 1. Размеры изображений и ядра на каждом этапе сжатия

Хоть такие подходы вероятнее всего будут страдать от недостатков, описанных во введении, данное решение значительно выигрывает во времени относительно других сетей и показывает сравнимый результат по разным метрикам качества. Возможно, похожий подход удастся применить в случаях, когда аналитические методы будут сильно сбоить, например при поворотах или препятствиях на дороге.

Watershed

Одно из классических преобразований изображений с целью сегментации — watershed, водоразделы. Любое серое изображение

можно представить как топографическую карту, где яркость каждой точки обозначает ее высоту. Интуитивно идею классического варианта трансформации можно понимать так: начинаем заполнять локальные минимумы водой. В местах встреч источников воды строим барьеры, которые их разделяют. Получившееся множество барьеров называют водоразделами, а разделенные ими участки водосборными бассейнами.

В статье Road segmentation by watersheds algorithms [2] описан вариант применения данного алгоритма к проблеме распознавания дороги. В ней мы рассматриваем изображение как функцию *f* (чаще всего понимается функция действующая из R^2 на вещественную прямую). Оказывается, что объекты на изображении в идеальных условиях соответствуют минимумам морфологического градиента функции, а контуры – водоразделам. Морфологический градиент *g* функции *f* определяется как

$$g(f) = [(f \oplus b) - (f \ominus b)]$$

где $(f \oplus b)(x) = \sup_{y \in b_x}(f(y))$,

и $(f \ominus b)(x) = \inf_{y \in b_x}(f(y))$,

и $b(x) = 0$, *при* $|x| \leqslant 1$, и $-\infty$ иначе

Но на практике из-за шума и неровностей получается очень много этих самых мелких контуров. Предлагается сначала находить маркеры, некую грубую оценку искомых объектов, в данном случае дороги, и далее вычислять водоразделы модифицированного градиента.

Первый способ нахождения маркеров – вычислить регуляризованный градиент, в результате чего уменьшится избыточность сегментации изображения. Далее берется максимальный близкий к камере водосборный бассейн и считаем, что это маркер дороги.

Второй способ нахождения маркеров – упрощение изображения и привидение его к так называемому mosaic-image (мозаичное изображение).

Далее эти маркеры используются, чтобы вычислить модифицированный градиент и вычислить по нему водоразделы, которые теперь будут показывать контур дороги.

Змеи, модель активных контуров

Другими популярными алгоритмами сегментации являются змеи. Данный вид алгоритмов работает только при наличии грубой оценки контура. Змея это сплайн, который задается множеством n точек v_i, i=0,…,n-1, которые изначально находятся рядом с предполагаемым контуром. В процессе работы алгоритма сплайн стягивается к контуру за счет минимизации суммы внутренней энергии сплайна, которая отвечает за деформацию сплайна, и внешней энергии, которая контролирует подгонку к контуру.

$$E_{snake} = \int_0^1 (E_{internal}(v(s)) + E_{external}(v(s)))ds$$

Внутренняя энергия определяется как сумма энергии непрерывности контура и энергии гладкости контура.

$$E_{internal} = ½ (w_1|v_s|^2) + ½ (w_2|v_{ss}(s)|^2)$$

Где w_1 и w_2 веса, контролирующие значимость энергии.

Для изображения $I(x,y)$ внешняя энергия определяется как

$$E_{external} = w_3 I(x,y) – w_4|\nabla I(x,y)|^2$$

Где w_3 и w_4 также являются весами.

Для минимизации внешней и внутренней энергий можно использовать стандартные алгоритмы оптимизации, например градиентный спуск.

Техника разделения и объединения

Идея техники разделения и объединения довольна проста. Сначала рассматриваем изображение как единое целое. Если все пиксели в данной области удовлетворяют некоторому ограничению на сходство, то не меняем область. Иначе делим ее на несколько частей, обычно 4. Таким образом, продолжаем разбивать все части, пока не останется частей для разбиения. После этапа разбиения необходимо объединять соседние регионы, если они удовлетворяют критерию сходства.

Данный алгоритм вероятнее всего не подойдет для точного определения контура, но может быть использован для грубой оценки, что является полезным для некоторых алгоритмов.

Eroding and Dilating

Два данных алгоритма работают по принципу свертки – задается ядро (матрица) с выделенной точкой, обычно являющейся центром. Далее эта матрица наслаивается на каждый участок изображения и в результате перемножения наслоившихся матриц изменяется значение пикселя исходного изображения, соответствующего выделенной точке ядра. В случае алгоритма Eroding (разрушение) результатом является минимальное значение из всех произведений точки ядра и соответствующей ему точки исходного изображения, а в случае Dilating (расширение) – максимальное значение. В результате разрушения визуально более яркие области изображения сужаются, а в случае расширения – расширяются. Эти алгоритмы часто используются вместе, например для снижения шума.

Вывод из обзора

Планируется решать задачу обнаружения полос движения с помощью алгоритма watershed, так как другие алгоритмы могут показывать нестабильный результат или не удовлетворять критериям качества. Если получится определять ситуации на дороге, например такие как поездка по прямой, поворот, значительное количество препятствий впереди, то в разных ситуациях возможно будут использованы разные алгоритмы, в зависимости от того как они себя покажут при той или иной ситуации.

3. Описание алгоритма

3.1 Поиск капота автомобиля

Для начала определим границы капота, который находится в кадре для того, чтобы маркировать его отдельно для алгоритма watershed. Заметим, что пиксели относящиеся к капоту, не меняют свое положение от кадра к кадру, так как движутся вместе с камерой. Возьмем несколько кадров в серых оттенках (одноканальных) и применим к ним функцию из библиотеки opencv bitwise_and, которая вычисляет побитовую конъюнкцию изображений. В результирующем изображении мало меняющиеся точки становятся темными. В данной работе использовано 40 последовательных кадров. Применим простейший алгоритм «порог», в результате которого все пиксели темнее порога станут 0, а выше – 255, чтобы легче было работать с изображением. Далее применим алгоритм расширения, чтобы лучше отделить капот от остального пространства. Пример получившейся в результате нижней части изображения, на котором в центре видна граница разделения, изображен на рис. 1. Черные области по краям возникли из-за загрязнений на окне.

Рис. 1. Часть изображения после обработки, на которой видна граница капота

3.2 Предобработка

Цель данного этапа – выделить маркеры для разных частей изображения – неба, дороги, части, находящейся слева от дороги и части, находящейся справа от дороги. Если левая и правая часть не

очень контрастны по отношению к небу, можно выделить два маркера – дороги и неба. Также применим алгоритм порог и будем работать с бинарным изображением. Так как на дороге лежит снег (в иных случаях прибегать к таким алгоритмам нет смысла и проще ориентироваться на разметку или более четкие линии по краям дороги) и из-за особенностей камер на дороге будет больше светлых пикселей чем по краям. В случае темного неба, оно тоже будет иметь меньше светлых точек.

Последовательно применим алгоритмы разрушения и расширения, чтобы снизить шум и избавиться от мелких частей изображения (Рис 2.2). Далее найдем связные компоненты на изображении с помощью функции из библиотеки opencv connectedComponentsWithStats, которая отчасти работает по технике разделения и объединения (Рис 2.3). Заметим, что самыми большими связными компонентами будут являться дорога, небо и боковые участки. Также боковые участки могут быть слиты с небом, если они мало контрастные и небо может быть слито с дорогой, если оно довольно светлое. Заметим, что в последнем случае линией разделения будет самая узкая по ширине часть компоненты, так как небо и дорога сужаются при приближении к горизонту. Также это используем и в случае раздельных компонент неба и дороги, так как одно может залазить на другое. Выделяем маркеры, ориентируясь на расположение центров и размеров компонент.

Рис. 2.1. Оригинальное изображение

Рис. 2.2. После порога и отчистки шумов

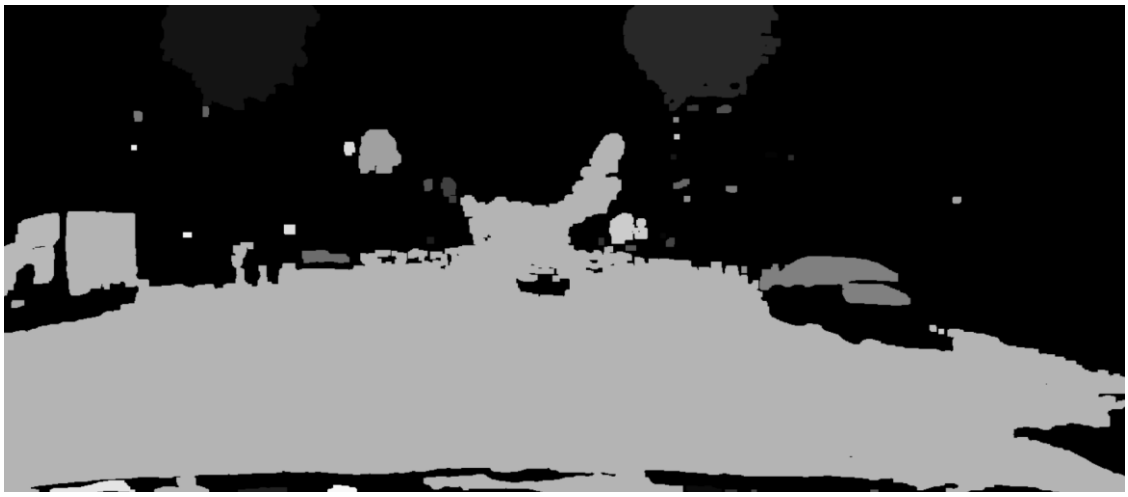


Рис. 2.3. После группировки по компонентам

3.3 Обработка изображения

Применим алгоритм watershed по полученным на предыдущем этапе маркерам и маркеру капота (Рис. 3.1). Для относящейся к дороге области, выделим точки, которые относятся к левой и правой границам контура. Построим по ним прямые так, чтобы сумма разностей значений прямой в точке и значения точки для каждой точки была минимальна. Воспользуемся градиентным спуском для минимизации значения. Получим прямые, относящиеся к границам дороги (Рис. 3.2). Алгоритм представляет границы как прямые линии и не работает с поворотами.

Рис 3.1. Разбитие на области после алгоритма watershed

Рис 3.2. Результат

4. Замеры времени

Для замеров был использован ноутбук с параметрами:

- Процессор Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz, 1992 МГц, ядер: 4;
- Оперативная память 16 Гб.

Было взято 3 кадра размерами 680x1280. На всех кадрах алгоритм был запущен 10 раз и затем для каждого этапа алгоритма взято среднее. Получившееся время действия основных этапов (в скобках указано среднеквадратичное отклонение):

- Подготовка кадра для предобработки («порог», удаление шума): 5 мс (1.2 мс);
- Нахождение связных компонент: 3 мс (0.4 мс);
- Нахождение горизонта между небом и дорогой: 2 мс (0.2 мс);
- Выделение нужных компонент и их обработка: 6 мс (1.3 мс);
- Watershed: 15 мс (2 мс);
- Выделение левой и правой части контура: 3 мс (0.6 мс);
- Нахождение границ с помощью минимизации функции: 40 мс (9 мс);
- Среднее время работы всего алгоритма: 70 мс (16 мс).

5. Оценка качества

Для того чтобы оценить качество работы алгоритма было размечено 2948 кадров. Для них были использованы стандартные метрики качества в таких задачах – accuracy, precision, recall и F-мера с параметром β = 1.

Введем понятия:

- TP (True Positive): часть изображения, которую мы разметили и определили в результате работы алгоритма как дорогу;
- TN (True Negative): часть изображения, которую мы разметили и определили в результате работы алгоритма как не относящуюся к дороге;
- FP (False Positive): часть изображения, которую мы разметили как не относящуюся к дороге, но в результате работы алгоритма отнесли ее к дороге;
- FN (False Negative): часть изображения, которую мы разметили как относящуюся к дороге, но в результате работы алгоритма отнесли ее не к дороге.

Таким образом:

- *accuracy* = (TP + TN) / (TP + TN + FP + FN);
- *precision* = TP / (TP + FP);
- *recall* = TP / (TP + FN);
- *F-мера* = 2 * *precision* * *recall* / (*precision* + *recall*).

Средние значения метрик, полученные в результате обработки кадров accuracy 0.751, precision 0.753, recall 0.88, F-мера 0.808. Подробные результаты представлены на графиках 1 – 4.

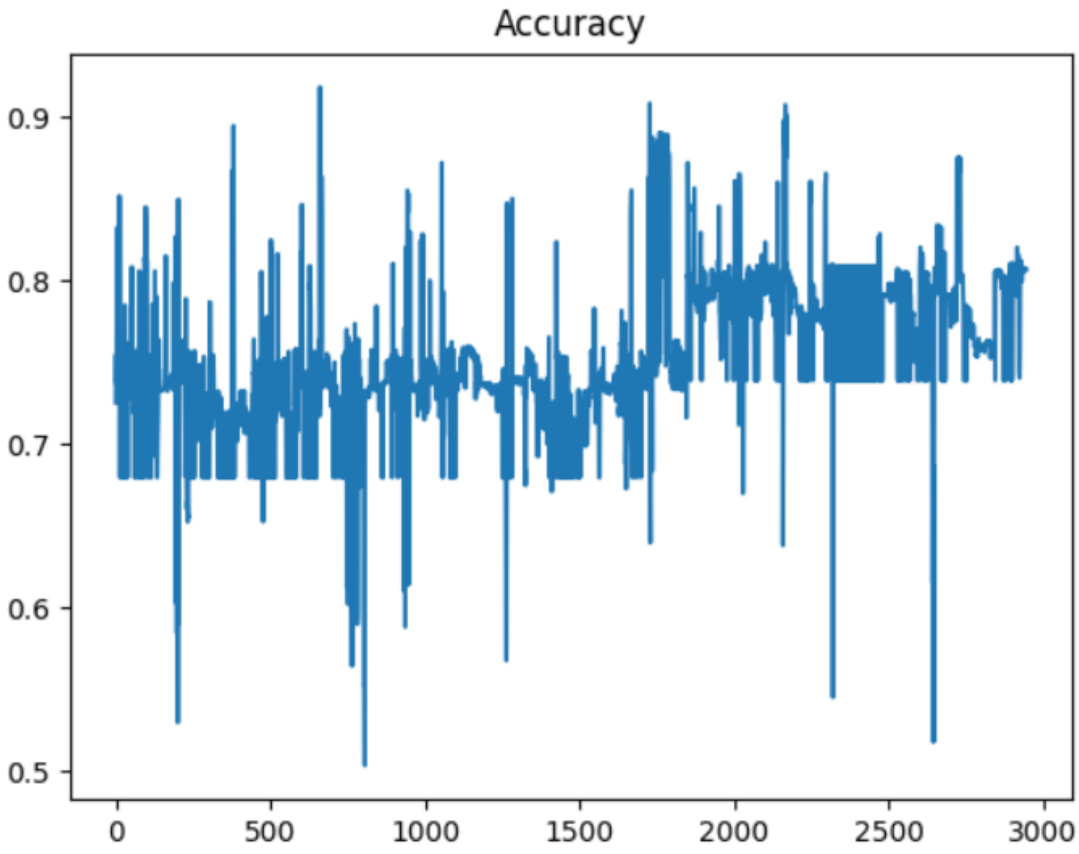


График 1. Accuracy

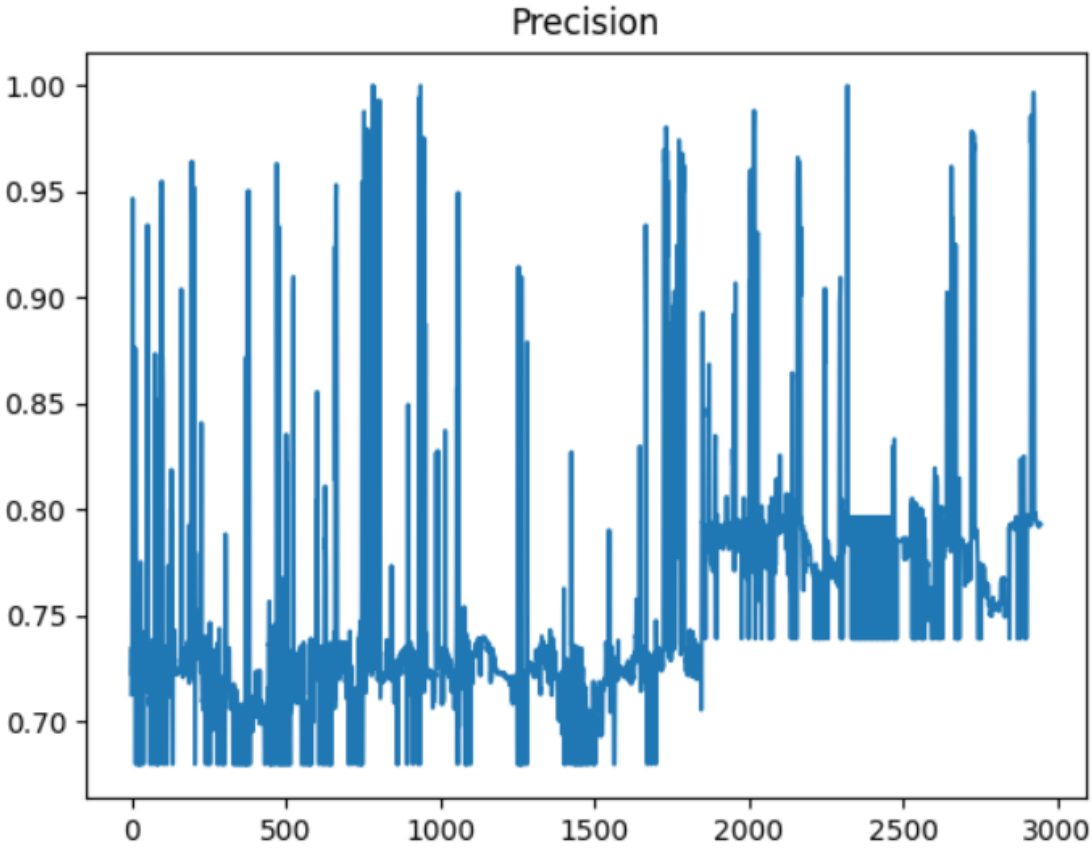


График 2. Precision

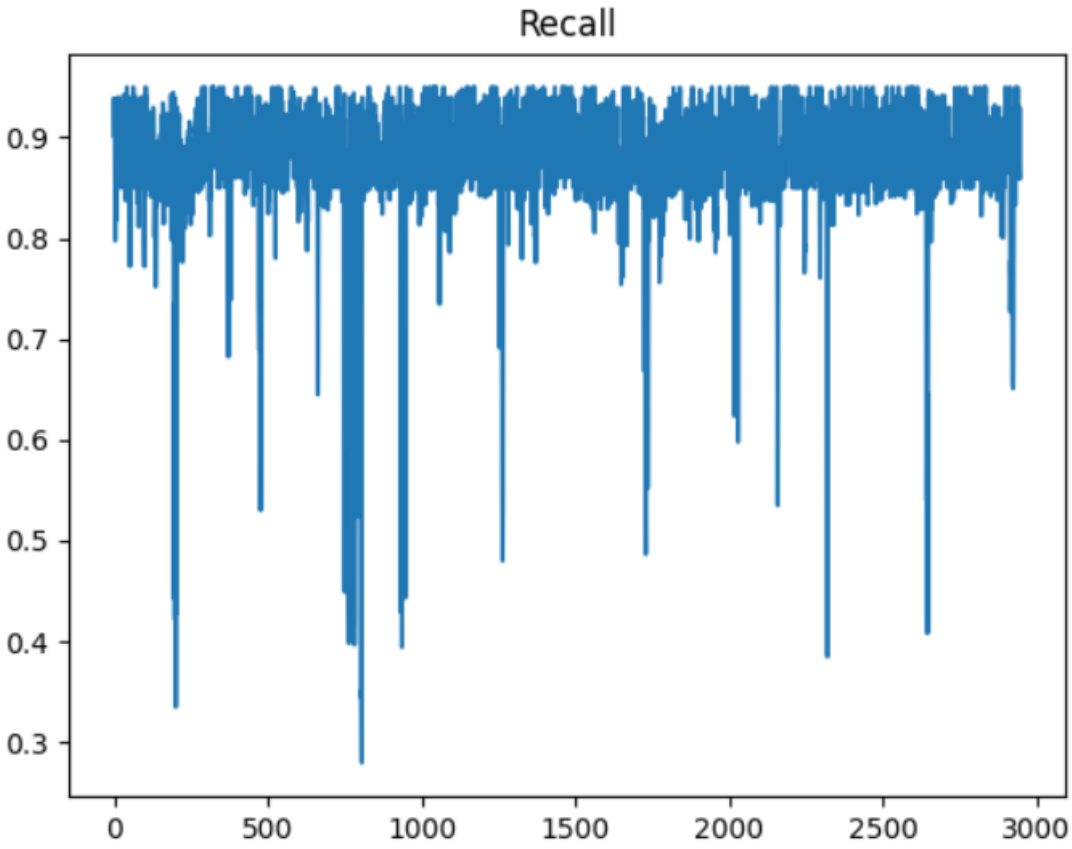


График 3. Recall

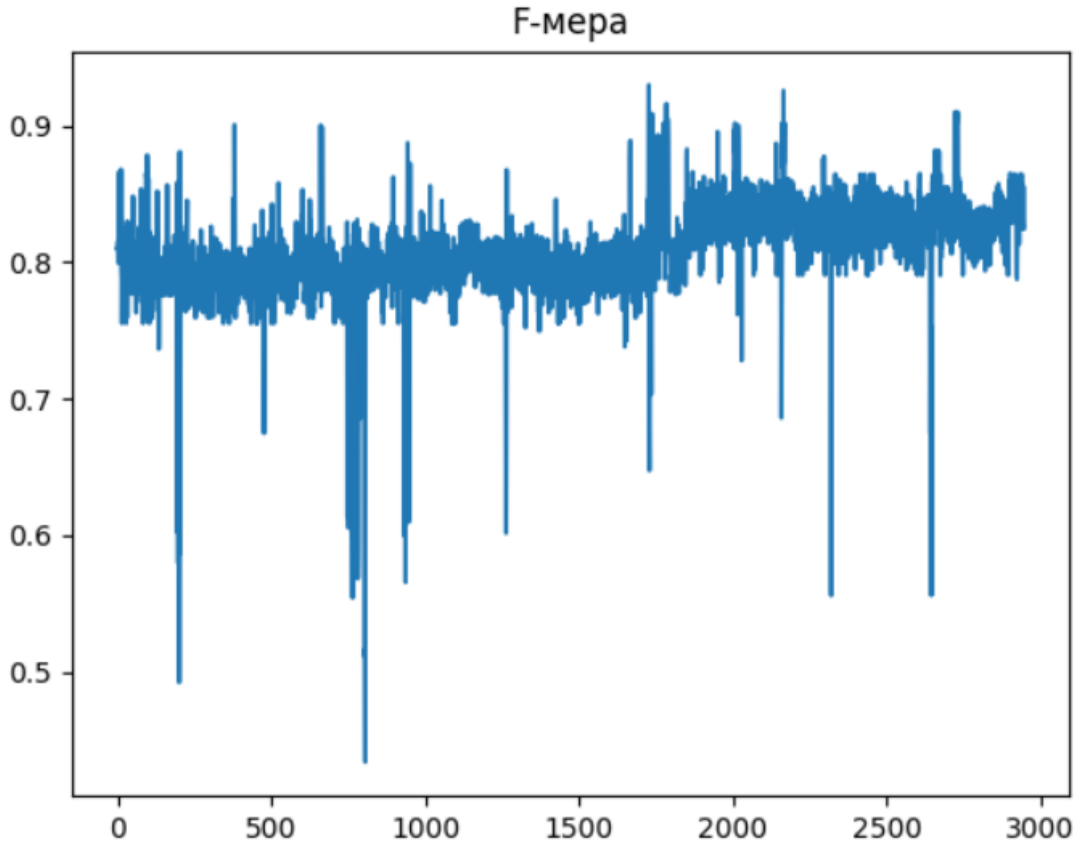


График 4. F-мера

Из довольно хорошего значения по метрике recall, которая показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм и более низкого значения метрики precision можно сделать вывод, что представленный алгоритм скорее получит в результате более широкую часть дороги, чем она есть на самом деле. Это происходит потому, что обычно реальные границы дороги не слишком выделяются и алгоритм watershed скорее находит границы забора, ограждающего дорогу или машин, стоящих на обочине, так как их границы выделяются сильнее.

Оценка влияния различных факторов на результат алгоритма

Алгоритм получился хорошо устойчивым к шумам и различным мелким предметам в области дороги и за ее пределами, таким как летающие снежинки (которые при сильном выпадении снега закрывают значительную часть кадра), проталинам и неровностям на дороге. Дело в том, что при выделении связных компонент в части предобработки такие мелкие части оказываются внутри большой компоненты дороги и не влияют на результат. Но также это может наносить и вред, например, при появлении красных предупредительных конусов на дороге, хоть их и при желании можно легко находить отдельно из-за их яркости. Также по тем же соображениям незначительные заснеженности на лобовом стекле не влияют на результат. Полосы на дороге от шин могут разделить общую компоненту дороги, что негативно скажется на результате, но при заметных полосах на дороге задача становится легче, так как при выборе границ можно ориентироваться на эти полосы.

Машины на обочине в целом помогают лучше определить границы дороги. Между участком дороги и машиной обычно идет большой цветовой переход, из-за чего алгоритм watershed определяет низ машины как границу, что зачастую является верным решением. Но иногда граница определяется по верхней части машины, ее крыше. В

этих случаях эта часть оказывается выше реальной границы, и может влиять негативно.

Также на результат позитивно влияют объекты, находящиеся по краям от дороги, такие как здания или деревья. Они помогают находить горизонт, между дорогой и небом, который определяется как самая узкая часть компоненты дороги (см. рис. 2.2). Объекты по краям делают эту узкую часть более заметной. Также эти объекты разграничивают дорогу и части, находящиеся слева и справа от нее. Если бы слева и справа, например в поле, ничего бы не было, и лежал только снег, при предобработке кадра компонента дороги слилась бы с боковыми частями и алгоритм не показал бы хоть сколь либо хороший результат.

Алгоритм работает лучше в ночное время, так как это повышает контраст между дорогой, краями и небом.

6. Результаты

Были получены следующие результаты:

1. Изучены алгоритмы сегментации;
2. Сделан обзор алгоритмов сегментации;
3. Разработан прототип определения полос движения по видео, ссылка – https://github.com/artemlunev2000/winter-road-detection;
4. Проведена апробация решения на размеченных кадрах.

Из замеров времени можно понять, что алгоритм сможет обрабатывать в среднем 14 кадров в секунду, что является приемлемым результатом для прототипа. Большую часть времени занимает минимизация функций, расчет которых происходит на Python, а не за счет библиотек, написанных на C++. Так как при минимизации функции вычисляются несколько раз, тратится много времени. Вероятнее всего на C++ эта часть окажется быстрее.

Метрики также показали вполне неплохой результат по качеству. Но следует понимать, что для применения в ADAS системах алгоритм должен быть универсален и уметь обрабатывать все случаи на дороге, такие как слишком яркое солнце, препятствия на дороге, большую заснеженность стекла, сильно разная заснеженность дороги, наклон камеры, большие перекрестки и так далее. Работ в этом направлении еще много, но как база, алгоритм может использоваться в ADAS системах.

Список литературы

[1] Carlos Yecheng Lyu, Lin Bai and Xinming Huang.Road Segmentation Using CNN and Distributed LSTM. — 2019. — URL: https://arxiv.org/pdf/1808.04450.pdf (дата обращения 14.10.2020)

[2] S.BEUCHER, M.BILODEAU, X. YU. Road Segmentation by watersheds algorithms. — 1990. — URL: http://www.cmm.mines-paristech.fr/~beucher/publi/promet_1.pdf (дата обращения 20.10.2020)

[3] Richard Szeliski. Computer Vision: Algorithms and Applications. Springer, 2010. — P. 267-309. (дата обращения 5.11.2020)

[4] MICHAEL KASS, ANDREW WITKIN, and DEMETRI TERZOPOULOS. Active contour models. — 1988. — URL: http://imft.ftn.uns.ac.rs/math/cms/uploads/DigitalImageProcessing/Kass_1998_Snakes.pdf (дата обращения 20.11.2020)

[5] Sepp Hochreiter. LSTM networks. — 1997. — URL: https://www.bioinf.jku.at/publications/older/2604.pdf (дата обращения 10.10.2020)

[6] Opencv docs: https://docs.opencv.org/master/index.html (дата обращения 11.10.2020)