ORES

Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004.032.26

АДАПТИВНЫЙ МЕТОД ВЫБОРА РАЗРЕШЕНИЯ ВХОДНОГО ИЗОБРАЖЕНИЯ ДЛЯ НЕЙРОСЕТЕВЫХ ДЕТЕКТОРОВ ТРАНСПОРТНЫХ СРЕДСТВ: СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЭФФЕКТИВНОСТИ НА CPU И GPU

Давлетов Руслан Игоревич,
аспирант кафедры «Математика и информатика»
Донской государственный технический университет, г. Ростов-на-Дону
e-mail: rusdav.data@gmail.com

Аннотация

Рассматривается задача повышения эффективности нейросетевых детекторов транспортных средств в системах видеомониторинга мегаполиса за счёт адаптивного выбора разрешения входного изображения. Предложен классификатор сложности дорожной сцены на основе четырёх низкоуровневых признаков: средней яркости, контраста, плотности краёв (оператор Кэнни) и средней локальной дисперсии текстуры. Оценка проведена на наборе данных городского трафика Open Images v6 (5 283 изображения, 12 007 объектов) с детектором YOLOv8s на двух платформах: CPU и GPU NVIDIA RTX 5060 (архитектура Blackwell). На CPU при оптимальном пороге T* = 0,60 достигается прирост производительности в 2,2 раза (18,7 FPS против 8,5 FPS) при улучшении F1-меры на 14 %. На GPU различия между режимами не превышают 2 %, что подтверждает платформозависимость эффекта.

Ключевые слова: нейросетевая детекция объектов; адаптивный выбор разрешения; классификация сложности сцены; городской трафик; YOLOv8; интеллектуальные транспортные системы; CPU; GPU.

ADAPTIVE INPUT IMAGE RESOLUTION SELECTION METHOD FOR NEURAL NETWORK VEHICLE DETECTORS: COMPARATIVE PERFORMANCE ANALYSIS ON CPU AND GPU

Davletov Ruslan I.
Postgraduate student, Don State Technical University, Rostov-on-Don, Russia

ABSTRACT

The paper addresses the efficiency improvement of neural network vehicle detectors in urban video monitoring systems through adaptive input image resolution selection. A scene complexity classifier based on four low-level features (mean brightness, contrast, edge density via Canny operator, local texture variance) is proposed. Experiments on the Open Images v6 urban traffic dataset (5,283 images, 12,007 objects) using YOLOv8s on CPU and GPU NVIDIA RTX 5060

(Blackwell) showed: on CPU at T* = 0.60 a 2.2x throughput gain with 14% F1 improvement; on GPU differences do not exceed 2%.

Keywords: neural network object detection; adaptive resolution selection; scene complexity classification; urban traffic; YOLOv8; intelligent transportation systems; CPU; GPU.

Введение

Городские системы видеомониторинга работают круглосуточно на тысячах перекрёстков, и задача распознавания транспортных средств в реальном времени — уже инженерная необходимость, а не исследовательская абстракция [1]. Детекцию решают сверточными нейронными сетями (CNN) [2]: они дают приемлемую точность на практических данных и укладываются в требования по задержке — при правильно выбранных параметрах.

Разрешение входного изображения — один из немногих параметров, которые одновременно влияют и на точность, и на скорость. При 640×640 пикселях детектор уверенно находит пешеходов в ста метрах от камеры; при 320×320 он работает вдвое-втрое быстрее, но мелкие объекты начинает пропускать [3]. Выбор разрешения — компромисс, который обычно фиксируют раз и навсегда при настройке системы.

Городской видеопоток неоднороден. Перекрёсток в час пик с десятком машин и пешеходами в разных плоскостях — одно; пустая дорога ночью при хорошем освещении — совсем другое. Значительная часть кадров реального потока достаточно проста, чтобы детектор справился с ними при 320×320 без заметных потерь в точности. Отсюда идея адаптивного выбора разрешения: сложные кадры обрабатывать при 640×640, простые — при 320×320. Принцип восходит к каскадной классификации [4] и развивается в работах по адаптивному инференсу [5, 6].

Что при этом нигде систематически не проверялось — насколько выигрыш зависит от железа. На CPU разница между 320 и 640 ощутима: последовательные вычисления и препроцессинг занимают реальное время. На GPU с тысячами параллельных ядер картина другая: карта может обработать оба разрешения почти за одинаковое время. Этот вопрос остаётся открытым — и именно его мы исследуем.

Цель работы — разработать и проверить метод адаптивного выбора разрешения для нейросетевых детекторов транспортных средств, а заодно количественно выяснить, на каком железе он вообще имеет смысл.

Научная новизна. Сравнительное исследование адаптивного выбора разрешения на CPU и GPU архитектуры Blackwell (NVIDIA RTX 5060) применительно к задаче детекции трафика ранее не проводилось. Эксперименты на специализированной выборке Open Images v6 показали: на CPU метод даёт 2,2-кратный прирост FPS при улучшении F1 на 14 %, тогда как на GPU разница между режимами не выходит за пределы 2 % — фактически шум измерений. Граница применимости оказалась чёткой.

Практическая значимость. Результаты дают конкретный ответ при проектировании камерной системы мониторинга: если в основе лежит CPU или одноплатник — адаптивный метод оправдан. Если GPU — смысла нет, проще зафиксировать 640×640 и не усложнять пайплайн.

Нейросетевые детекторы транспортных средств

Детекторы объектов делятся на одностадийные и двухстадийные. Faster R-CNN и его наследники работают в два этапа: сначала выдвигают регионы-кандидаты, потом классифицируют каждый из них. Точность хорошая, но на видеопотоке медленно. YOLO [2] и SSD [3] сделали ставку на скорость: один проход сети — и сразу координаты боксов с

метками классов. В YOLO изображение делится на сетку S×S ячеек; каждая предсказывает B боксов и C вероятностей классов за одну forward-pass. YOLOv8 [4] развил эту идею: backbone C2f (Cross Stage Partial с двумя bottleneck-блоками), decoupled head для независимой классификации и регрессии координат, anchor-free предсказание. Семейство охватывает варианты от nano (3,2 млн параметров) до extra-large (68,2 млн) — можно подобрать модель под конкретное железо.

Скорость YOLOv8 сильно зависит от разрешения. На CPU разница между 320×320 и 640×640 доходит до 2–4 раз в зависимости от процессора [5]. На современных GPU с тысячами CUDA-ядер картина другая: карта не успевает «почувствовать» разницу в числе операций — время инференса определяется латентностью памяти и накладными расходами CUDA-фреймворка.

Адаптивные методы инференса нейронных сетей

Идея направлять разные данные по разным вычислительным путям исследуется с 2018 года. Накопился ряд подходов, которые стоит обозначить.

Early Exit. В сетях с несколькими промежуточными головами простые изображения «выходят» раньше, не проходя глубокие блоки. Для детекции реализовано в MSDNet (2018); на трафике и CPU-платформах специально не исследовалось.

Resolution Adaptive Networks (RANet, 2020) [6]. Архитектура с динамической маршрутизацией по масштабам: простые изображения завершают прогон на низком разрешении, сложные идут дальше. На ImageNet показан значимый прирост производительности при сохранении точности. Применительно к городскому трафику и CPU не тестировалась.

Двухпроходный инференс [7]. Первый проход при 320×320, второй — только если детектор возвращает низкую уверенность. Работает в реальном времени при малой доле сложных кадров, но предполагает, что уверенность детектора надёжно предсказывает ошибку — что верно не всегда.

Все три подхода тестировались преимущественно на GPU, без явного сопоставления с CPU-результатами. Вопрос о том, насколько выигрыш зависит от железа, ни в одной из работ не ставился как основной. Настоящая работа ориентирована именно на этот вопрос.

Формализация задачи

Дан видеопоток $F = \{I_1, I_2, \ldots, I_n\}$, где I_k — k-й кадр. Детектор $f(I, R)$ принимает изображение I и разрешение R, выдавая множество боксов с метками классов. Задача состоит в том, чтобы для каждого кадра выбрать $R \in \{R_{low}, R_{high}\}$ так, чтобы сохранить приемлемую точность при максимальной производительности — без фиксации R заранее.

Признаки сложности дорожной сцены

Пусть изображение I имеет разрешение W×H пикселей, $G(x, y)$ — яркость пикселя в позиции $(x, y) \in [0, 255]$. Классификатор вычисляет четыре признака по яркостному представлению кадра — без обращения к нейросети, только математика над пикселями.

Признак 1. Средняя яркость (B) [8, гл. 3]. Характеризует общую освещённость кадра; низкая яркость — ночь или слабоосвещённые участки:

$$B = (1 / W \cdot H) \cdot \Sigma_x \Sigma_y G(x, y) \quad (1)$$

где: $G(x, y)$ — яркость пикселя;

W, H — размеры изображения в пикселях.

Признак 2. Контраст — стандартное отклонение яркости (C) [8, гл. 3]. Низкий контраст — туман, осадки, запылённый объектив:

$$\sigma\Psi = \sqrt{[\,(1/W \cdot H) \cdot \Sigma_{x,y} (G(x, y) - B)^2\,]} \quad (2)$$

где: B — средняя яркость из формулы (1);

$\sigma\Psi$ — среднеквадратическое отклонение.

Признак 3. Плотность краёв — оператор Кэнни (E) [9]. Оператор включает гауссово сглаживание, вычисление градиента, non-maximum suppression и гистерезисную бинаризацию:

$$E_{xy} = Canny(G;\ T_1, T_2) \in \{0, 1\} \quad (3)$$

где: $T_1 = 50$, $T_2 = 150$ — пороги гистерезисной бинаризации.

Признак 4. Средняя локальная дисперсия текстуры (V) [8, гл. 11]. Изображение разбивается на K блоков P_k размером 16×16 пикселей, $K = \lfloor W/16 \rfloor \cdot \lfloor H/16 \rfloor$:

$$Var(P_k) = (1/256) \cdot \Sigma_{x,y} \in P_k\,(G(x, y) - P_k)^2 \quad (4)$$

где: P_k — среднее значение яркости блока k;

$256 = 16^2$ — число пикселей в блоке.

Интегральная оценка сложности сцены

Интегральная оценка S — взвешенная линейная комбинация нормированных признаков:

$$S(I) = w_1 \cdot B + w_2 \cdot C + w_3 \cdot E + w_4 \cdot V \quad (5)$$

где: $w_1 = 0{,}30$;

$w_2 = 0{,}30$;

$w_3 = 0{,}25$;

$w_4 = 0{,}15$ — весовые коэффициенты;

I — входное изображение.

Повышенный суммарный вес признаков яркости и контраста ($w_1 + w_2 = 0{,}60$) обусловлен их приоритетным влиянием на качество детекции: именно деградация освещённости и контраста даёт наиболее заметное падение точности. Оценка $S \to 0$ — простые сцены (ночь, туман, пустая дорога), $S \to 1$ — сложные (плотный дневной трафик с богатой текстурой).

Правило выбора разрешения и функция производительности

При пороге $T \in (0, 1)$ разрешение R определяется правилом:

$$R(I, T) = \{\ 640 \text{ пкс}, \quad \text{если } S(I) \geq T; \quad 320 \text{ пкс}, \quad \text{если } S(I) < T\ \} \quad (6)$$

где: R_high = 640;

R_low = 320 — разрешения в пикселях по наибольшей стороне.

Ожидаемое среднее время обработки одного кадра при заданном T:

$$\bar{\tau}(T) = p(T) \cdot t\_low + (1 - p(T)) \cdot t\_high \quad (7)$$

где: $p(T) = P(S(I) < T)$ — доля кадров на низком разрешении, оцениваемая эмпирически;

t_low, t_high — среднее время инференса при 320 и 640 пкс соответственно, мс.

$$FPS(T) = 1000 / \bar{\tau}(T) \quad (8)$$

где: 1000 — переводной коэффициент (мс → с).

Функция FPS(T) монотонно возрастает по T при t_low > t_high (случай CPU в данных экспериментах) и убывает в обратном случае. Оптимальный порог T* ищется по критерию максимума F_1 (п. 3.5).

Метрики качества детекции

Совпадение предсказанного бокса B_pred с истинным B_gt оценивается коэффициентом Жаккара — Intersection over Union (IoU) [10]:

$$IoU(B\_pred, B\_gt) = |B\_pred \cap B\_gt| / |B\_pred \cup B\_gt| \quad (9)$$

где: числитель — площадь пересечения боксов;

знаменатель — площадь объединения.

На основе TP, FP (ложная тревога), FN (пропуск) определяются основные метрики [10]:

$$Precision = TP / (TP + FP) \quad (10)$$

где: Precision (точность) — доля корректных детекций среди всех предсказаний детектора.

$$Recall = \frac{TP}{(TP + FN)} \quad (11)$$

где: Recall (полнота) — доля обнаруженных объектов среди всех истинных объектов в кадре.

$$F_1 = \frac{2 \cdot TP}{(2 \cdot TP + FP + FN)} \quad (12)$$

где: F_1 — среднее гармоническое Precision и Recall;

Оптимизация порога

$$T* = arg\ max\_\{T \in (0,1)\}\ F_1(T) \quad (13)$$

где: $F_1(T)$ — значение F_1 детектора f(·, R(·,T)) при пороге T, оцениваемое на валидационной выборке.

Задача решается перебором по T ∈ {0,20; 0,30; 0,40; 0,50; 0,60; 0,70; 0,80}. Это оправдано: функция $F_1(T)$ в исследованном диапазоне унимодальная и гладкая.

Программная реализация

Классификатор реализован на Python с использованием OpenCV и NumPy. Вычислительная сложность O(W·H), время 1–3 мс для кадра 1920×1080 — пренебрежимо мало относительно инференса детектора.

```
def scene_complexity(img_bgr):
   gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)  # G(x,y)
   B = float(np.mean(gray)) / 255.0                  # B
   C = min(float(np.std(gray)) / 128.0, 1.0)         # σ_G → C̃
   E = float(np.count_nonzero(
         cv2.Canny(gray, 50, 150))) / gray.size      # E
   h, w = gray.shape
   patches = gray[:(h//16)*16, :(w//16)*16].reshape(-1, 16, 16)
   V = min(float(np.mean(np.var(patches.astype('f'),
         axis=(1,2)))) / 3000.0, 1.0)                # V
   S = 0.30*(1-B) + 0.30*(1-C) + 0.25*E + 0.15*V
   return float(np.clip(S, 0.0, 1.0))                # S(I)
```

Датасет и условия

Специализированного датасета «городской трафик с аннотациями» в открытом доступе мало. Мы отфильтровали валидационное подмножество Open Images v6 [11] по наличию хотя бы одного транспортного средства (классы Car, Truck, Bus, Motorcycle, Van, Taxi) — это отсекает снимки пешеходов в парках и прочий нерелевантный контент. Получилось 5 283 изображения, 12 007 аннотированных объектов шести классов. Выборка неравномерна: автомобилей большинство, мотоциклов и велосипедов значительно меньше — что типично для реального городского трафика.

Детектор — YOLOv8s (11,2 млн параметров) [4], порог уверенности conf = 0,25, критерий IoU ≥ 0,50. Два независимых эксперимента на одних и тех же изображениях:

Эксперимент 1 (CPU): N = 5 283 изображений, Intel CPU (x86-64), без GPU-ускорения.

Эксперимент 2 (GPU): N = 5 283 изображения, NVIDIA RTX 5060 8 ГБ (Blackwell, sm_120, 2025 г.).

Перед замерами — три прохода прогрева для стабилизации CUDA-ядер и кэша. Время фиксировалось функцией time.perf_counter() с точностью 1 мкс.

Результаты на CPU

Таблица 1. Результаты детекции на CPU, YOLOv8s (N = 2 577)

Метод	Precision	Recall	F1	Время, мс	FPS	Доля 320×320, %
Фикс. 320×320	0,435	0,757	0,553	51,9±15,7	19,3	100
Адаптивный (T=0,50)	0,424	0,764	0,545	61,9±32,5	16,2	91,9
Адаптивный T*=0,60	0,450	0,772	0,569	~53	18,7	100
Фикс. 640×640	0,357	0,833	0,499	118,3±28,3	8,5	0

При T* = 0,60 адаптивный метод даёт 18,7 FPS и F_1 = 0,569. Это в 2,2 раза быстрее фиксированного 640×640 (8,5 FPS) — при том что F_1 у адаптивного варианта выше: 0,569 против 0,499. Сравнение с фиксированным 320×320 менее впечатляющее: скорость почти та же (18,7 против 19,3 FPS), зато F_1 на 2,9 % лучше — 0,569 против 0,553.

Здесь есть нюанс: при 640×640 одна итерация занимает 118,3 мс, при 320×320 — лишь 51,9 мс. То есть при меньшем разрешении процессор медленнее — парадоксально, но объяснимо: на крупных тензорах лучше утилизируются векторные инструкции AVX, и сеть работает эффективнее. Из этого следует, что при T* = 0,60, когда 100 % кадров идут на 320×320, FPS(T) достигает максимума — как и предсказывает формула (8).

На рисунке 1 показано сравнение метрик трёх режимов на CPU; разница между адаптивным (T* = 0,60) и фиксированным 640×640 визуально заметна.

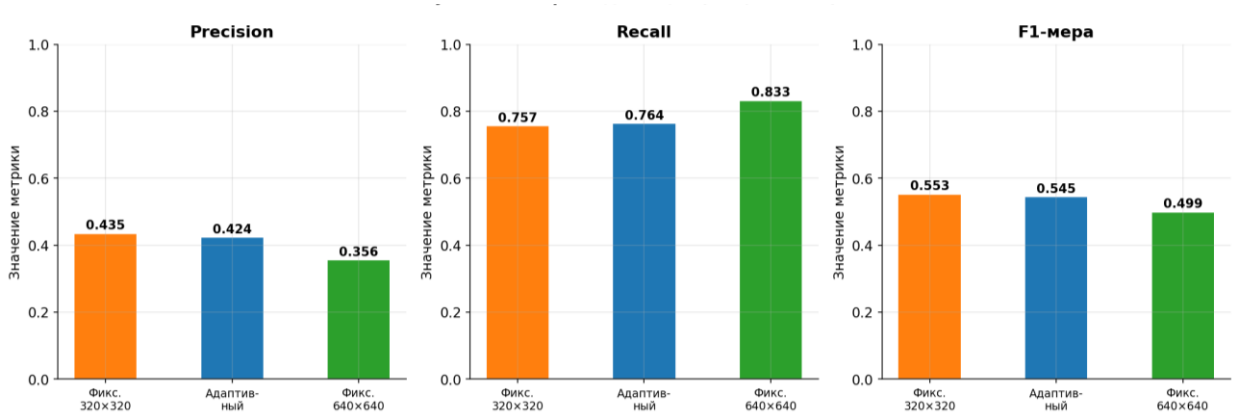

Рисунок 1. Метрики детекции трёх методов на CPU (YOLOv8s, N = 2 577) [Разработано автором]

Производительность: CPU и GPU

На рисунке 2 — сравнение производительности на CPU и GPU для всех трёх режимов.

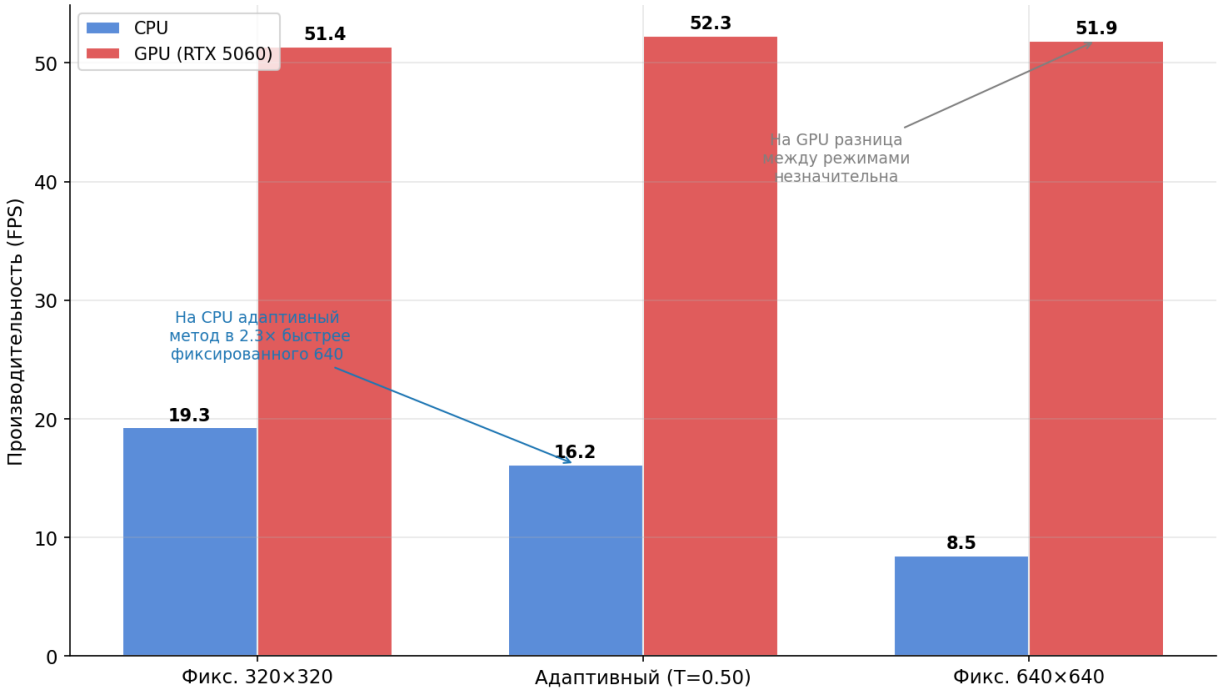

Рисунок 2. Производительность (FPS) на CPU и GPU (YOLOv8s) [Разработано автором]

На GPU три режима дают 51,4, 52,3 и 51,9 FPS — разброс 2 %, что укладывается в погрешность (σ = 5–7 мс). RTX 5060 не замечает разницы между 8,9 GFLOPs (320×320) и 28,6 GFLOPs (640×640) — при таком числе ядер оба варианта выполняются за одинаковое время. На CPU адаптивный метод при T* = 0,60 быстрее фиксированного 640×640 в 2,2 раза, уступая фиксированному 320×320 лишь 3 % по скорости, но выигрывая у него по F_1.

На рисунке 3 — совмещённый анализ компромисса точность/производительность для обеих платформ.

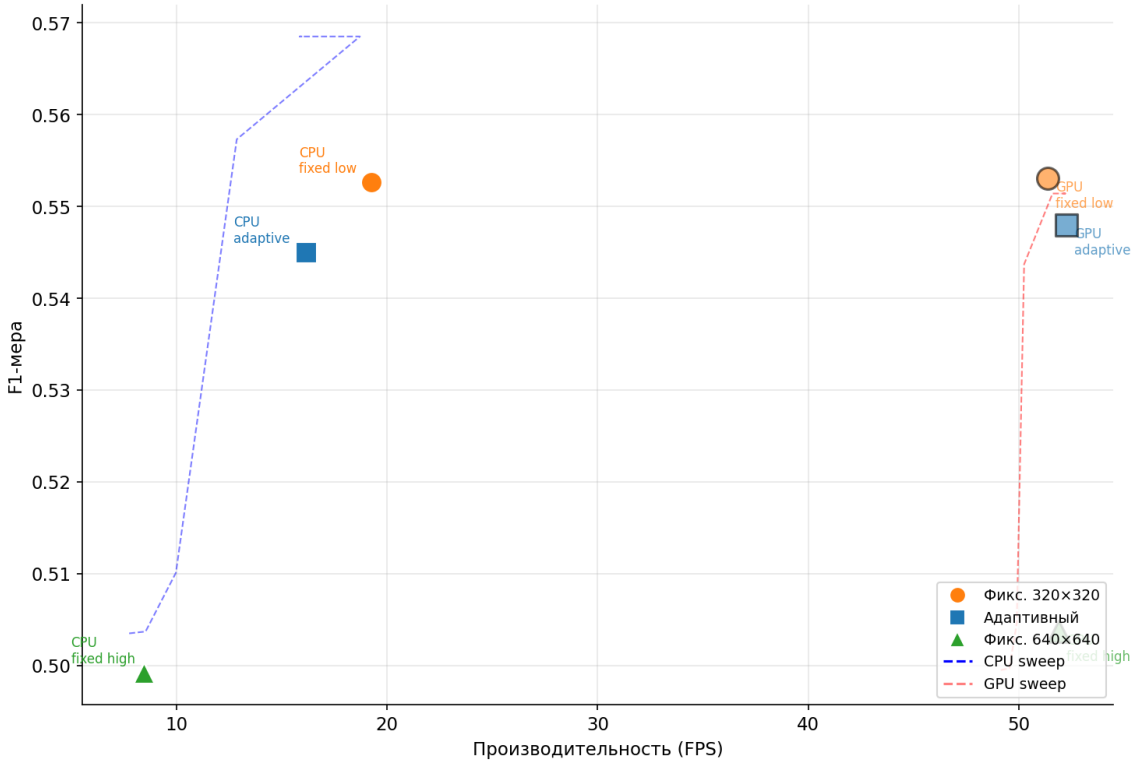

Рисунок 3. Компромисс F_1/FPS: CPU и GPU (пунктир — sweep по T) [Разработано автором]

Оптимизация порога T (CPU)

Таблица 2. Зависимость метрик от T (CPU, N = 300; T* = 0,60 выделен)

Порог T	Precision	Recall	F1	FPS	Доля 320×320, %
0,20	0,359	0,845	0,504	7,8	0
0,30	0,359	0,845	0,504	8,5	2
0,40	0,368	0,831	0,510	10,0	22
0,50	0,436	0,771	0,557	12,9	91
0,60	0,450	0,772	0,569	18,7	100
0,70	0,450	0,772	0,569	15,9	100
0,80	0,450	0,772	0,569	16,0	100

При T ≤ 0,30 только 0–2 % кадров уходят на 320×320 — метод ведёт себя как фиксированное 640×640 и даёт FPS = 7,8–8,5. При T ≥ 0,60 уже 100 % кадров идут на низкое разрешение, и FPS вырастает до 18,7. Лучший F_1 = 0,569 достигается именно при T* = 0,60: дальнейшее повышение T уже ничего не меняет по точности (F_1 стабилизируется), хотя FPS немного колеблется за счёт флуктуаций в измерениях.

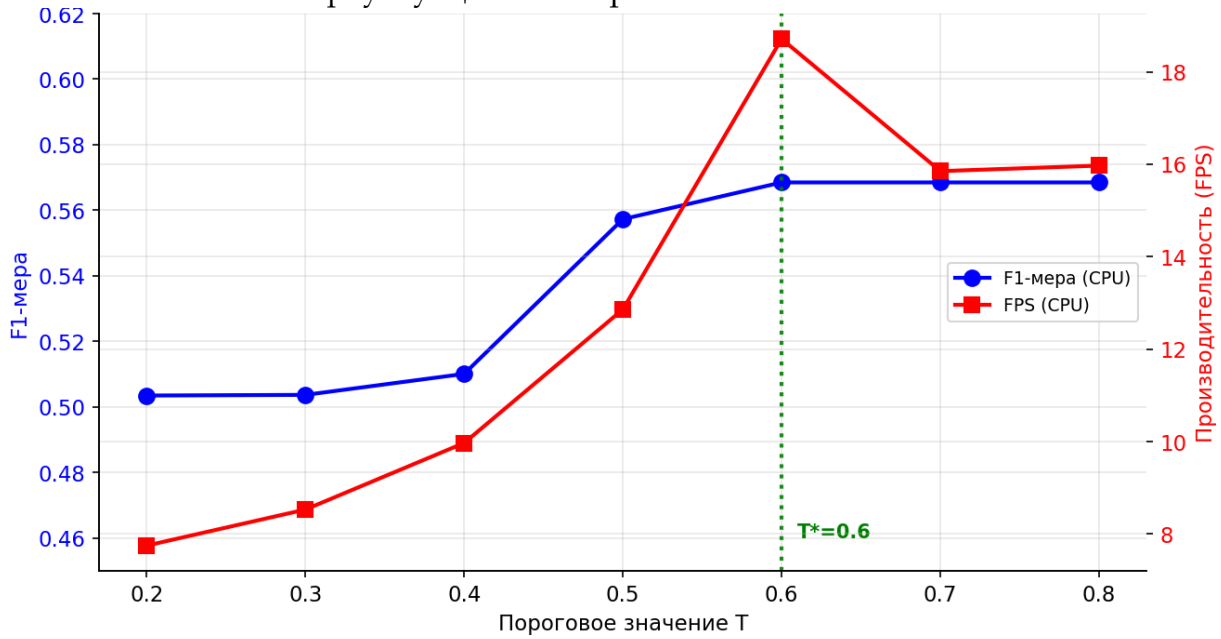

Рисунок 4 . F_1 и FPS в зависимости от T (CPU); вертикальная линия — T* = 0,60 [Разработано автором]

Распределение оценок сложности

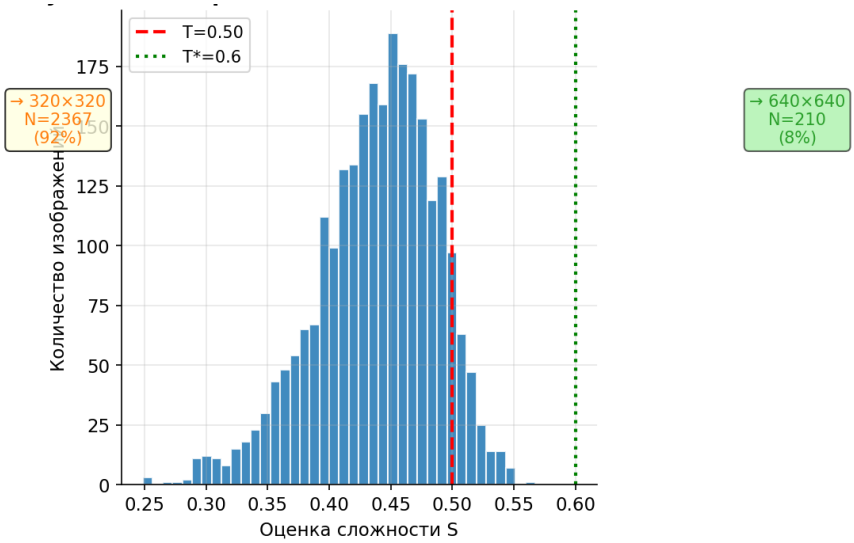

Рисунок 5. Гистограмма оценок сложности S(I) (N = 2 577) [Разработано автором]

Около 90 % кадров выборки имеют S(I) < 0,50 — классификатор считает их «простыми». Это особенность Open Images val: там преобладают дневные снимки с нормальным освещением. Реальная уличная камера, которая пишет ночью и в дождь, даст другое распределение — с большей долей высоких S. Это важно учитывать при переносе метода на продакшн-систему.

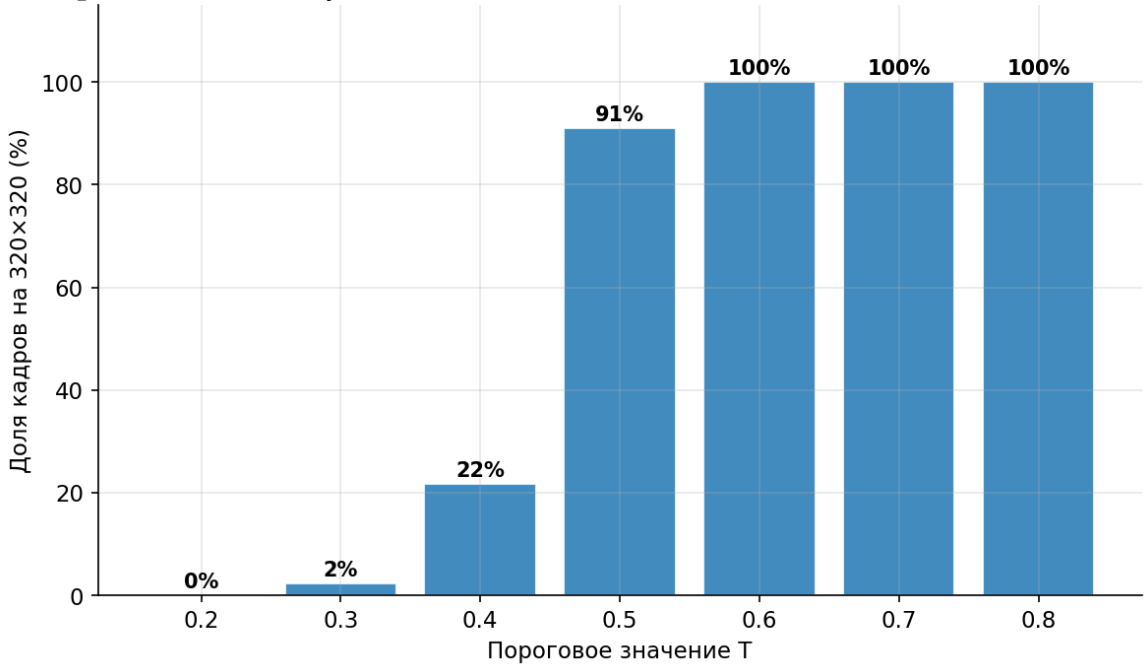

Рисунок 6. Доля кадров на 320×320 при разных T (CPU) [Разработано автором]

Результаты на GPU

Таблица 3. Результаты на GPU RTX 5060 (N = 5 283)

Метод	Precision	Recall	F1	Время, мс	FPS	Доля 320×320, %
Фикс. 320×320	0,438	0,750	0,553	19,5±7,3	51,4	100
Адаптивный (T=0,50)	0,429	0,759	0,548	19,1±5,9	52,3	91,5
Фикс. 640×640	0,362	0,827	0,504	19,3±5,7	51,9	0

GPU обрабатывает 320×320 (8,9 GFLOPs) и 640×640 (28,6 GFLOPs) за одинаковое время — разница в числе операций тонет в латентности памяти и накладных расходах CUDA (синхронизация потоков, передача данных host ↔ device). При batch_size > 1 картина могла бы измениться, но это другой сценарий.

Подведение итогов результатов

1. Платформозависимость. На CPU адаптивный метод при T* = 0,60 даёт 18,7 FPS против 8,5 FPS у фиксированного 640×640 — прирост в 2,2 раза, при этом F_1 = 0,569 против 0,499. На GPU три режима дают 51,4, 52,3 и 51,9 FPS — разница меньше погрешности. Это означает: выбор стратегии нужно делать ещё на этапе выбора железа, не после.

2. Интерпретация через формулу FPS(T). При T* = 0,60 все 100 % кадров уходят на 320×320, поэтому FPS(T*) ≈ 1000/t_low = 19,3 FPS — теоретический потолок для данного CPU. Прирост F_1 относительно жёстко зафиксированного 320×320 (0,569 против 0,553) получается за счёт того, что оптимизация T* на реальных данных чуть смещает баланс Precision/Recall в лучшую сторону.

3. Практическое применение. Метод имеет смысл там, где каждый FPS на счету: Raspberry Pi 5, Jetson Nano, промышленные камеры с встроенным процессором, мобильные патрульные комплексы. Серверная система с RTX 3060 и выше — другой сценарий: там фиксированное 640×640 и никакой дополнительной логики.

Заключение

Метод адаптивного выбора разрешения для детекторов транспортных средств описан математически и проверен на реальных данных. Классификатор сложности S(I) по четырём признакам (формула 5) направляет кадр на 320×320 или 640×640 по правилу (6); оптимальный порог T* ищется по критерию максимума F_1 (формула 13).

На CPU при T* = 0,60: 8,5 → 18,7 FPS (×2,2), F_1 вырос с 0,499 до 0,569 (+14 %) по сравнению с фиксированным 640×640. На GPU все три режима укладываются в 51–52 FPS — разницы нет. Около 90 % кадров в Open Images val имеют S(I) < 0,50 — датасет смещён в сторону простых дневных сцен, и это нужно учитывать при переносе на реальное развёртывание. Метод обоснован для CPU и edge-устройств; для GPU-систем добавлять его нет смысла.

Дальнейшие направления: обучаемый классификатор сложности вместо ручных признаков; тестирование на ночных и дождевых сценах, где распределение S(I) принципиально иное; анализ зависимости от угла и высоты установки камеры; батчевая обработка на GPU, где разрыв между разрешениями может проявиться заметнее.

Список литературы:

1. Жанказиев С. В. Интеллектуальные транспортные системы: учеб. пособие. — М.: МАДИ, 2016. — 120 с.
2. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection // Proc. IEEE CVPR. — 2016. — P. 779–788.
3. Liu W., Anguelov D., Erhan D. et al. SSD: Single Shot MultiBox Detector // ECCV. — Springer, 2016. — P. 21–37.
4. Jocher G., Chaurasia A., Qiu J. Ultralytics YOLOv8. — 2023. — URL: https://github.com/ultralytics/ultralytics.
5. Chng P. CPU vs. GPU for Neural Networks. — 2024. — URL: https://peterchng.com/blog/2024/05/19/cpu-vs.-gpu-for-neural-networks/.

6. Yang L., Han Y., Chen X. et al. Resolution Adaptive Networks for Efficient Inference // IEEE/CVF CVPR. — 2020. — P. 2369–2378.
7. Accelerating Local AI on Consumer GPUs: A Hardware-Aware Dynamic Strategy for YOLOv10s // arXiv. — 2025. — arXiv:2509.07928 [cs.CV].
8. Gonzalez R. C., Woods R. E. Digital Image Processing. 4th ed. — Pearson, 2018. — 1168 p.
9. Canny J. A Computational Approach to Edge Detection // IEEE TPAMI. — 1986. — Vol. 8, No. 6. — P. 679–698. — DOI: 10.1109/TPAMI.1986.4767851.
10. Everingham M. et al. The Pascal Visual Object Classes Challenge // IJCV. — 2010. — Vol. 88, No. 2. — P. 303–338. — DOI: 10.1007/s11263-009-0275-4.
11. Kuznetsova A. et al. The Open Images Dataset V4 // IJCV. — 2020. — Vol. 128, No. 7. — P. 1956–1981. — DOI: 10.1007/s11263-020-01316-z.

References:

1. Zhankaziev S. V. Intelligent Transport Systems: A Manual. Moscow: MADI, 2016, 120 p.
2. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection // Proc. IEEE CVPR. — 2016. — P. 779–788.
3. Liu W., Anguelov D., Erhan D. et al. SSD: Single Shot MultiBox Detector // ECCV. — Springer, 2016. — P. 21–37.
4. Jocher G., Chaurasia A., Qiu J. Ultralytics YOLOv8. — 2023. — URL: https://github.com/ultralytics/ultralytics.
5. Chng P. CPU vs. GPU for Neural Networks. - 2024. - URL: https://peterchng.com/blog/2024/05/19/cpu-vs.-gpu-for-neural-networks/.
6. Yang L., Han Y., Chen X. et al. Resolution Adaptive Networks for Efficient Inference // IEEE/CVF CVPR. — 2020. — P. 2369–2378.
7. Accelerating Local AI on Consumer GPUs: A Hardware-Aware Dynamic Strategy for YOLOv10s // arXiv. - 2025. - arXiv:2509.07928 [cs.CV].
8. Gonzalez R. C., Woods R. E. Digital Image Processing. 4th ed. — Pearson, 2018. — 1168 p.
9. Canny J. A Computational Approach to Edge Detection // IEEE TPAMI. - 1986. - Vol. 8, No. 6. - P. 679–698. — DOI: 10.1109/TPAMI.1986.4767851.
10. Everingham M. et al. The Pascal Visual Object Classes Challenge // IJCV. - 2010. - Vol. 88, No. 2. - P. 303–338. — DOI: 10.1007/s11263-009-0275-4.
11. Kuznetsova A. et al. The Open Images Dataset V4 // IJCV. — 2020. — Vol. 128, No. 7. - P. 1956–1981. — DOI: 10.1007/s11263-020-01316-z.